The standard of correctness and completeness necessary to get a computer program to work at all is a couple of orders of magnitude higher than the mathematical community's standard of valid proofs. Nonetheless, large computer programs, even when they have been very carefully written and very carefully tested, always seem to have bugs.

— *William P. Thurston*

Chapter 1
INTRODUCTION

In this thesis we tell a story of how the theory of nonlinear waves shares a deep connection with the field of algebraic geometry. In this chapter we summarize this path, highlighting the primary objects at play.

1.1 The Story

The Kadomtsev–Petviashvili equation is a nonlinear partial differential equation used to describe two-dimensional surface waves,

$$(-4u_t + 6uu_x + u_{xxx})_x + 3\sigma^2 u_{yy} = 0. \tag{1.1.1}$$

The KP equation is a generalization of the Korteweg-de Vries (KdV) equation,

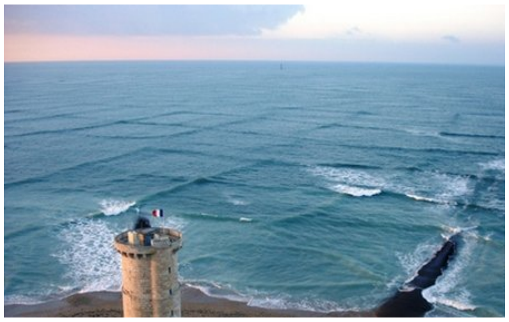
$$4u_t = 6uu_x + u_{xxx}, \tag{1.1.2}$$

and in fact the equation appears as a sub-expression in the KP equation. The water waves modeled by the KP equation are typically of long wavelength and are weakly two-dimensional in that there is a dominant direction of propagation with smaller-scale oscillations ocurring in the transverse direction.

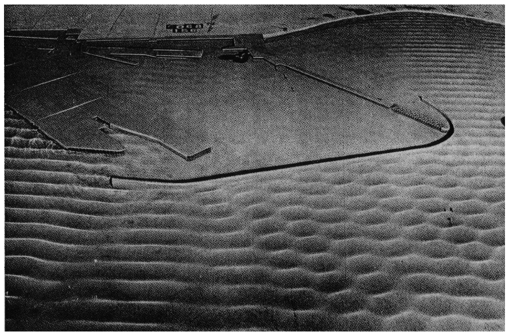
The KP equation admits a large family of quasi-periodic solutions of the form,

$$u(x,y,t) = 2\partial_x^2 \theta(z,\Omega), \quad z_j(x,y,t) = U_j x + V_j y + W_j t + \phi_j \tag{1.1.3}$$

where $\theta : \mathbb{C}^g \times \mathbb{C}^{g\times g} \to \mathbb{C}$ is the *Riemann theta function of genus g*. This space of solutions is dense in the space of all periodic solutions and, therefore, fundamental to the study of the equation.



(a) *Île de Ré, France*



(b) *Model of San Diego Harbor*

Before continuing with the story it is important to mention that KP-like waves certainly appear in nature. In Figure **??** we see two examples of the periodic structure emerging from chaotic conditions. The photograph in Figure 1.1a was captured on a particularly windy day off the coast of Île de Ré in France. Figure 1.1b was actually taken in the lab where a conference room-sized model of the proposed San Diego harbor was constructed [TODO REF] before direct numerical simulation was feasible. In this image we can see two phases of waves: on the bottom left and top right there are long, KdV-like one-dimensional waves. In the bottom right is the characteristic hexagonal wave structure of a "typical" KP solution. We will later see that these structures correspond to different genera of quasi-periodic KP solution.

Miraculously, the path to deriving and computing the finite-genus KP solutions takes us through the field of algebraic geometry beginning in complex analysis with algebraic curves. We give a brief glimpse into this path, here, and invite the reader to examine the details in Chapter **??**. An algebraic curve is the complex solution set to a bi-variate polynomial equation,

$$C_f : \left\{ (\lambda, \mu) \in \mathbb{C}^2 \;\middle|\; f(\lambda,\mu) = 0, f \in \mathbb{C}[x,y] \right\}. \tag{1.1.4}$$

Algebraic curves are closely related to Riemann surfaces in that, modulo minor details, a holomorphic mapping takes every algebraic curve to a Riemann surafce and vice-versa.

Riemann surfaces are connected complex manifolds homeomorphic to a sphere with g

handles (or a torus with g holes) where g is called the *genus*. One can consider if there are any holomorphic one-forms defined on a given Riemann surface and what kind of cycles (simple closed paths) they could be integrated on. It turns out that the space of holomorphic one-forms is g-dimensional and the space of homologous paths is $2g$-dimensional. Lets label the bases elements of these spaces $\{\omega_1, \ldots, \omega_g\}$ and $\{a_1, \ldots, a_g, b_1, \ldots, b_g\}$, respectively. Naturally, we can then determine the $2g^2$ resulting integrals, or *periods*,

$$A_{ij} = \oint_{a_j} \omega_i, \quad B_{ij} = \oint_{b_j} \omega_i. \tag{1.1.5}$$

These two quanties together form the *period matrix* of the Riemann surface / algebraic curve. After normalizing the holomorphic one-forms such that $A_{ij} = \delta_{ij}$, the Kronecker delta function, we obtain the period matrix $\tau = [I; \Omega]$. The quantity Ω appears in the finite-genus solutions given in Equation 1.1.3, above.

The other parameters, $U, V, W, \phi \in \mathbb{C}^g$ are also determined from the Riemann surface from a specified collection of *places* on the Riemann surface also called a *divisor*. The parameters themselves are computed by evaluating two functions, the Abel map and the Riemann constant vector, at the pdivisor. In particular, this divisor encodes information about the initial condition of the desired finite-genus solution if one were solving the KP initial value problem.

Finally, this mathematical path from algebraic curves to KP solutions can be made extraordinarily concrete: each step outlined above can be computationally realized. As Field's Medalist William Thurston recognizes, without attention to detial this is a challenging feat. By developing algorithms along the way we accomplish two things. First, we provide an experimental tool for the study of periodic KP waves. Second, and more importantly, with the use of sound software design principles we create a general framework for performing analytic calculations on Riemann surfaces that both is computationally efficient and is easily extendible by future researchers. The development of such a framework extends the reach of the code to beyond the application to non-linear waves and serves the mathematical community.

1.2 Outline

Chapter 2 presents a succinct introduction to the field of complex algebraic geometry and Riemann surfaces. A student approaching the subject need only the basics of complex analysis to get started. It begins with an introduction to algebraic curves and their geometry. We then connect these curves to the theory of Riemann surfaces and demonstrate, at least in the compact and connected case, that the two are synonymous. The bulk of the chapter dives deep into the study of Riemann surfaces from this perspective of algebraic curves building up the necessary machinery needed to define the *Jacobian* of a Riemann surface and its associated *Period Matrix*; central objects of focus for much of the work in this thesis as well as current research in the field. The chapter concludes with two key objects defined on the Jacobian, the *Abel Map* and *Riemann Constant Vector*, as well the *Riemann theta function*. Each of these are primary ingredients in the connection to and calculation of solutions to the Kadomtsev-Petviashvili equation.

If Chapter 2 presents the theory then Chapter 3 contains the corresponding algorithms which allow us to experiment with Riemann surfaces on a computer. For each key concept described in the previous chapter we present efficient algorithms for their computation along with a collection of examples. These algorithms are gathered together in a Sage package called *Abelfunctions*. We spend some time in this chapter discussing the high-level design of this package including how important principles in software design allows the package to be easily extendible.

Chapters 4 and 5 showcase the use of these algorithms to compute the finite-genus solutions to the Kadomtsev-Petviashvili equation as well as determinantal representation of algebraic curves. Previous approaches to computing periodic solutions to KP are either too restrictive (the resulting solution space is not dense in the space of periodic solutions) or rely on direct numerical simulation where the accuracy of the solution degrades over time. In contrast, the finite-genus solutions derived by the Riemann surface machinery is valid for all space and time. The work on determinantal representations presented here is the computational realization of the theory developed by Helton and Vinnikov, the importance of which lies in the application to polynomial optimization algorithms.

1.3 Acknowledgements

First and foremost I would like to thank my wife, Megan Karalus. Thank you for your love and unending patience. You stood by me when things got tough, celebrated with me when things got better, and encouraged me to finish what I started. Thank you to the many people in the Applied Mathematics department at the Univeristy of Washington for shaping me into the mathematican, computer scientist, and human being that I am today. Bernard for his mentorship and comraderie, Randy for encouraging me to keep one toe in the HPSC world, Lauren for her friendship and counsel; and Daniel, Lowell, and Alan for their friendship and the many hours spent at the College Inn. Thank you to Amazon AI for believing that I can make a difference in this world by creating great things and by providing a rich an supportive environment where the interface between mathematical research and high performance computation find a natural home. Finally, this work was generously supported bpy the National Science Foundation under grant NSF-DMS-1008001. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the funding sources.

Chapter 2

COMPLEX ALGEBRAIC GEOMETRY AND RIEMANN SURFACES

In this chapter we will give a brief overview of Riemann surfaces from the perspective of complex algebraic geometry. Among the primary references used in this thesis, Griffiths [**?**] provides one of the most succinct introductions to the subject without requiring background in algebraic geometry or algebraic curves and gives solid foundation from which to further explore the topic. Ueno's [37] presentation is similar but with more emphasis on the algebraic perspectives whereas Bliss [4] focuses on an analytic approach, instead. Dubrovin's [14] classical manuscript includes connections to the Kadomtsev-Petviashvili equation introduced in Chapter 4.

2.1 *Complex Algebraic Geometry*

We begin with some preliminary constructions and facts from complex algebraic geometry. This section culminates in the Normalization Theorem which connects the subject to the study of Riemann surfaces. Although other computational approaches to Riemann surfaces exist, this connection makes the computational approach possible later.

2.1.1 The Projective Line

The primary motivation behind complex projective geometry is to make concrete the way in which we analyze the behavior of functions, such as polynomials, at infinity without having to resort to techniques separate from those used at finite points. For example, in applications we may need to integrate a differential along a path on an algebraic curve going to infinity. Knowing the geometry of the curve at infinity makes such an operation computationally feasible.

In fact, anyone with an elementary complex analysis background has seen an example of projective geometry. The Riemann sphere is the complex plane $\mathbb{C}$ with a "point at infinity" added. Let z denote the coordinate in $\mathbb{C}$ (i.e., the point $z = 0$ represents the origin of the complex plane). In order to discuss the point at infinity we introduce the coordinate $w = 1/z$. The analysis of some function at ∞ is equivalent to rewriting the problem in terms of the coordinate w and examining its behavior in a neighborhood of $w = 0$. This explains why, for example, the exponential function

$$e^z = \sum_{n=0}^{\infty} z^n/n!,$$

though entire in the complex plane, has an essential singularity on the Riemann sphere since the exponential function in the coordinate w centered at $w = 0$ is expressed by the series

$$\sum_{n=0}^{\infty} \frac{w^{-n}}{n!}.$$

This point at infinity is not rigorously defined because it does not make sense to *equate* $z = \infty$. The definition of the Riemann sphere is made explicit by the following construction: consider the set $U = \mathbb{C}^2 - \{(0,0)\}$. Define the equivalence relation

$$(a_0, a_1) \sim (\lambda a_0, \lambda a_1), \quad \forall \lambda \in \mathbb{C} - \{0\}.$$

Thus two points (a_0, a_1) and (b_0, b_1) in U are considered the same if the ratios $a_0 : a_1$ and $b_0 : b_1$ are equal. The set of all points (b_0, b_1) equal to (a_0, a_1) is called the *equivalence class* of (a_0, a_1) and the *complex projective line* $\mathrm{P}^1\mathbb{C}$ is the set of all such equivalence classes. That is,

$$\mathrm{P}^1\mathbb{C} := \mathbb{C}^2 \,/ \sim .$$

The equivalence class of (a_0, a_1), called a "point" in $\mathrm{P}^1\mathbb{C}$, is written $(a_0 : a_1) \in \mathrm{P}^1\mathbb{C}$. $\mathrm{P}^1\mathbb{C}$ is precisely the Riemann sphere. To see this, consider the two subsets

$$U_0 = \{(a_0 : a_1) \in \mathrm{P}^1\mathbb{C} \mid a_0 \neq 0\},$$

$$U_1 = \{(a_0 : a_1) \in \mathrm{P}^1\mathbb{C} \mid a_1 \neq 0\}.$$

For any $(a_0 : a_1) \in U_0$ we have, by the equivalence property,

$$(a_0 : a_1) = (1 : a_1/a_0) = (1 : a).$$

Similarly, $(b_0 : b_1) = (b : 1)$ for every point in U_1. Every point in the intersection $U_0 \cap U_1$ can be written in either of these two forms. Each of these subspaces are isomorphic to $\mathbb{C}$ since the maps

$$\phi_0 : U_0 \to \mathbb{C}, \quad \phi_0\left((a_0 : a_1)\right) = a_1/a_0, \quad \text{and}$$
$$\phi_1 : U_1 \to \mathbb{C}, \quad \phi_1\left((a_0 : a_1)\right) = a_0/a_1,$$

are continuous bijections with inverses

$$\phi_0^{-1}(a) = (1 : a), \tag{2.1.1}$$
$$\phi_1^{-1}(b) = (b : 1). \tag{2.1.2}$$

Finally, note that $(0 : 1)$ is the only projective point in U_1 which is not in U_0. Therefore, we identify U_0 with the complex plane (in the coordinate z) and the point $P_\infty = (0 : 1)$ with the point at infinity and set

$$\mathrm{P}^1\mathbb{C} = U_0 \cup \{(0 : 1)\} \cong \mathbb{C} \cup P_\infty. \tag{2.1.3}$$

Indeed P_∞ is considered the point at infinity on the Riemann sphere for if one considers the image of $(0 : 1)$ under ϕ_0, though undefined since $(0 : 1) \notin U_0$, it maps to $z = 1/0$ "$=$" ∞. Again, this does not make sense without the complex projective space construction above but is merely used to illustrate the point. The coordinate transformation from z to w at the beginning of this section is equivalent to identifying U_1 with the complex plane $\mathbb{C}$ and $\{(1 : 0)\}$ with the point at infinity, instead.

2.1.2 The Projective Plane

The natural environment we use in the sequel is not the complex projective line but the complex projective plane. In this section we construct the projective plane and examine its geometric properties. The construction is similar to that of the projective line.

Let $U = \mathbb{C}^3 - \{(0,0,0)\}$. Following the strategy of the previous section, consider the set of all ratios $(a_0 : a_1 : a_2)$, that is, the collection of all equivalence classes under the equivalence relation $(a_0 : a_1 : a_2) \sim (\lambda a_0 : \lambda a_1 : \lambda a_2), \forall \lambda \in \mathbb{C} - \{0\}$. The space of all such equivalence classes is called the two-dimensional complex projective space or *the projective plane* and is denoted $\mathrm{P}^2\mathbb{C}$.

Define the subsets U_0, U_1, U_2 by

$$U_j = \{(a_0 : a_1 : a_2) \in \mathrm{P}^2\mathbb{C} \mid a_j \neq 0\},$$

and note that all $(a_0 : a_1 : a_2) \in U_0$ satisfy $(a_0 : a_1 : a_2) = (1 : a_1/a_0 : a_2/a_0)$. We define the bijective mapping

$$\phi_0 : U_0 \to \mathbb{C}^2,$$
$$\phi_0\left((a_0 : a_1 : a_2)\right) = \left(\frac{a_1}{a_0}, \frac{a_2}{a_0}\right),$$
$$\phi_0^{-1}\left((x, y)\right) = (1 : x : y).$$

The mappings ϕ_1 and ϕ_2 are similarly defined on U_1 and U_2, respectively. Therefore, we can identify U_0 with the complex plane $\mathbb{C}^2$.

Consider the space $U_0^c = \mathrm{P}^2\mathbb{C} - U_0$. By definition, every point in U_0^c is of the form $(0 : a_1 : a_2)$. By definition, every point in U_0^c determines a point on the complex projective line $\mathrm{P}^1\mathbb{C}$. The converse is true as well, resulting in the bijection

$$(0 : a_1 : a_2) \in \mathrm{P}^2\mathbb{C} \;\leftrightarrow\; (a_1 : a_2) \in \mathrm{P}^1\mathbb{C}\,.$$

By identifying U_0^c with $\mathrm{P}^1\mathbb{C}$ we may write

$$\mathrm{P}^2\mathbb{C} = U_0 \cup U_0^c \cong \mathbb{C}^2 \cup \mathrm{P}^1\mathbb{C} \tag{2.1.4}$$

where $U_0^c \cong \mathrm{P}^1\mathbb{C}$ is called the *line at infinity*, denoted l_∞, and $U_0 \cong \mathbb{C}^2$ is called the *complex affine plane*. We may also identify the complex affine plane with the sets U_1 or U_2 and the line at infinity with their complements.

We saw a natural geometric interpretation of $\mathrm{P}^1\mathbb{C}$ in the previous section. Does such an interpretation exist for $\mathrm{P}^2\mathbb{C}$? Consider a line in the complex affine plane $\mathbb{C}^2$ which can be written in the form

$$\alpha + \beta x + \gamma y = 0, \quad \text{where } (\beta, \gamma) \neq 0, \alpha, \beta, \gamma \in \mathbb{C}\,.$$

Using the inverse mapping ϕ_0^{-1} on $\mathbb{C}^2$ we have

$$x = \frac{x_1}{x_0} \text{ and } y = \frac{x_2}{x_0},$$

where $(x_0 : x_1 : x_2)$ are the coordinates of $\mathrm{P}^2\mathbb{C}$, and we get the line

$$\alpha x_0 + \beta x_1 + \gamma x_2 = 0.$$

This equation, called the *homogenization* of the affine curve, makes sense in all of $\mathrm{P}^2\mathbb{C}$. Setting $x_0 = 1$ gives the original affine line. On the other hand, setting $x_0 = 0$ gives the equation

$$\beta x_1 + \gamma x_2 = 0,$$

which is the equation of the line in l_∞. However, this implies $x_1/x_2 = -\gamma/\beta$. Hence the projective point $(0 : -\gamma : \beta)$ satisfies the equation

$$\alpha x_0 + \beta x_1 + \gamma x_2 = 0$$

and is, in fact, the only projective point in l_∞ on the line.

This means that the line "intersects" l_∞ at the point $(0 : -\gamma : \beta)$ and that this intersection point depends only on the slope of the affine portion of the line. Hence, the line at infinity has the geometric meaning that each point on it is the intersection point of an entire family of parallel lines in $\mathbb{C}^2$. This leads to a generalization of a theorem from classical planar geometry: *any two, distinct lines in* $\mathrm{P}^2\mathbb{C}$ *intersect at exactly one point.*

2.1.3 Projective Plane Curves

The set of all points (x_0, x_1, x_2) satisfying

$$\alpha x_0 + \beta x_1 + \gamma x_2 = 0$$

is called a *projective line* and is a simple example of a projective algebraic curve (of degree one). In this section we introduce various properties of general projective curves.

A *complex plane algebraic curve* is the zero locus of the homogenization of a polynomial $f \in \mathbb{C}[x, y]$. That is, given a polynomial $f(x, y) = \alpha_n(x)y^n + \alpha_{n-1}(x)y^{n-1} + \cdots + \alpha_0(x)$ its homogenization is the polynomial $F \in \mathrm{P}^2\mathbb{C}[x_0, x_1, x_2]$ where

$$F(x_0, x_1, x_2) = x_0^d f(x_1/x_0, x_2/x_0),$$

and where d is the degree of F. The homogeneity of F means that we can write,

$$F(x_0, x_1, x_2) = \sum_{i+j+k=d} \alpha_{ijk} x_0^i x_1^j x_2^k.$$

In terms of the projective polynomial F, its affine part can be written $F(x, y) = F(1, x, y)$. As in the case of a projective line, f can be thought of as a projection of the polynomial F onto $\mathbb{C}^2$ and there is always a one-to-one correspondence between an affine polynomial and its homogenization. Therefore, a *complex plane algebraic curve* is the set

$$C = \left\{(x_0 : x_1 : x_2) \in \mathrm{P}^2\mathbb{C} : F(x_0, x_1, x_2) = 0\right\}.$$

Important to the study of projective curves, and specifically in the computational work described here, are singular points.

Definition 2.1.1. *A point $a = (a_0 : a_1 : a_2) \in C$ is a* **singular point of** C*, or a multiple point of* C*, if*

$$\left(\frac{\partial F}{\partial x_0}, \frac{\partial F}{\partial x_1}, \frac{\partial F}{\partial x_2}\right)(a) = (0, 0, 0).$$

Consider the case when $a = (1 : 0 : 0)$ (corresponding to the point $(0, 0)$ in the affine plane $\mathbb{C}^2$) is a singular point of F. The affine poirtion of the curve is

$$f(x, y) = \sum_{i+j\geq 2}^{d} c_{ij} x^i y^j.$$

Note that the constant term is zero since $(0, 0)$ is a point on the affine curve and the linear term vanishes since $(0, 0)$ is a singular point. We write

$$f(x, y) = f_m(x, y) + f_{m+1}(x, y) + \cdots + f_d(x, y), \quad m \geq 2,$$

where each f_n is the sum of all terms of f of degree n; that is, terms of the form $c_{ij}x^iy^j$ such that $i+j=n$. The smallest such m with non-zero term f_m appearing in f is called the *multiplicity* of the singular point $(1:0:0)$. Singularities with multiplicity two are called *double points*, those with multiplicity three are called *triple points*, and so on.

The homogeneous term f_m can be factored into linear factors,

$$f_m(x,y) = \prod_{j=1}^{m}(\alpha_j x - \beta_j y).$$

We call the space $C_m = \{(x,y) \in \mathbb{C}^2 : f_m(x,y) = 0\}$ the *tangent cone of the plane curve C* at $a = (1:0:0)$ which consists of a finite number of intersecting lines $L_j : \alpha_j x - \beta_j y$.

When a generic affine point $a = (1:c:d)$ is a singular point we write the affine curve in the form,

$$f(x,y) = \sum_{i+j\geq 2}^{d} \tilde{c}_{ij}(x-c)^i(y-d)^j,$$

which we can write as a sum of polynomials $g_n(x-c, y-d)$ homogenous in $x-c$ and $y-d$.

In the case when the singular point $a = (0:1:b) \in l_\infty$ we repeat the above process with the affine curve,

$$g(u,v) = \frac{1}{x_1^d}F(x_0,x_1,x_2) = F(u,1,v), \quad u = \frac{x_0}{x_1}, v = \frac{x_2}{x_1},$$

which is a projection of F onto $U_1 \cong \mathbb{C}$ instead of U_0. We write g as a sum of terms of the form $g_{ij}u^i(v-b)^j$. Finally, in the case $a = (0:0:1) \in l_\infty$ we use the affine curve

$$h(w,z) = \frac{1}{x_2^d}F(x_0,x_1,x_2) = F(w,z,1), \quad w = \frac{x_0}{x_2}, z = \frac{x_1}{x_2},$$

and write h as a sum of terms of the form $h_{ij}w^iz^j$.

Example 2.1.2. Consider the cubic curve

$$C : F(x_0,x_1,x_2) = x_0^4x_2^3 + 2x_0^3x_1^3x_2 - x_1^7$$

In complex affine space $x_0 = 1$ this curve is

$$f(x,y) = F(1,x,y) = y^3 + 2x^3y - x^7.$$

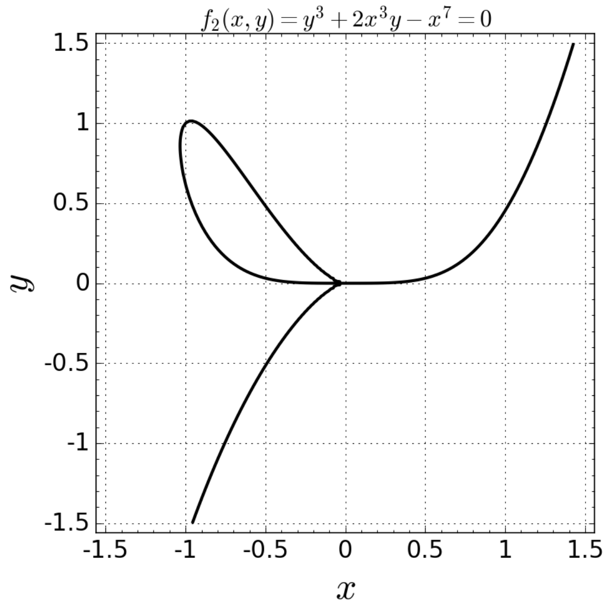


Figure 2.1: A real plot of the curve $C : f(x,y) = y^3 + 2x^3y - x^7$. The plot suggests that $(x_0 : x_1 : x_2) = (1 : 0 : 0)$, corresponding to $(x,y) = (0,0)$, is a singular point of C.

A plot of f for x, y real is shown in Figure 2.1. For $a = (a_0 : a_1 : a_2)$ we have

$$\begin{aligned}
\frac{\partial F}{\partial x_0}(a) &= 4a_0^3a_2^3 + 6a_0^2a_1^3a_2, \\
\frac{\partial F}{\partial x_1}(a) &= 6a_0^3a_1^2a_2 - 7a_1^6, \\
\frac{\partial F}{\partial x_2}(a) &= 3a_0^4a_2^2 + 2a_0^3a_1^3. \qquad (2.1.5)
\end{aligned}$$

First, we find the finite singular points of C. Setting $a_0 = 1$ and solving the above equations for a_1 and a_2 we see that $p = (1 : 0 : 0)$ is the only finite singular point of F. Note that

$$f(x,y) = f_3(x,y) + f_4(x,y) + f_7(x,y),$$

$$f_3(x,y) = y^3, \quad f_4(x,y) = 2x^3y, \quad f_7(x,y) = -x^7,$$

and that f_3, f_4, and f_7 are homogeneous of degrees 3, 4, and 7, respectively. Therefore, p is a singular point of multiplicity 3 with

$$f_3(x, y) = y^3 = 0,$$

as the equation for the tangent cone at p. These properties are suggested by Figure 2.1 where, near the point p, the real curve looks like the intersection of three curves well approximated by the line $y = 0$ near the point $x = 0$.

Setting $a_0 = 0$, the only expression in Equation (2.1.5) that does not reduce to zero is

$$\frac{\partial F}{\partial x_1}((0, a_1, a_2)) = -7a_1^6 = 0,$$

implying that the point $a = (0 : 0 : 1)$ is the only singular point at infinity. The curve at infinity centered at $(0 : 0 : 1)$ is

$$h(w, z) = F(w, z, 1) = w^4 + 2w^3z^3 - z^7.$$

The order of this singularity is four since this is the degree of the lowest degree homogeneous term. The tangent cone at a is $g_4(w, z) = w^4$.

2.1.4 Connection to Riemann Surfaces

There is a close relationship between the study of compact Riemann surfaces and that of algebraic curves. Recall that a Riemann surface X is a complex manifold of complex dimension one endowed with an *atlas*: an open covering $\{U_\alpha\}_{\alpha \in A}$ of X together with a collection of homeomorphisms $\{z_\alpha : U_\alpha \to \mathbb{C}\}_{\alpha \in A}$, called *local parameters*, such that every pair of *transition functions*

$$f_{\beta,\alpha} := z_\beta \circ z_\alpha^{-1} : z_\alpha \left(U_\alpha \cap U_\beta\right) \to z_\beta \left(U_\alpha \cap U_\beta\right),$$

is holomorphic. The pairs (U_α, z_α) are called *coordinate charts.* In other words, a Riemann surface is a topological space such that for all $P \in X$ there is a neighborhood of P homeomorphic to an open subset of the complex plane and one can analytically continue from any $P \in X$ to any $Q \in X$ via transition functions.

The Riemann sphere $X = \mathbb{C}^*$ is an example of a Riemann surface. Its atlas consists of two coordinate charts (U_1, z_1) and (U_2, z_2) with

$$U_1 = \mathbb{C}, \quad z_1 = z,$$
$$U_2 = (\mathbb{C} - \{0\}) \cup \{\infty\}, \quad z_2 = 1/z.$$

This is a valid atlas since the transition functions

$$f_{1,2}, f_{2,1} : (\mathbb{C} - \{0\}) \to (\mathbb{C} - \{0\})$$
$$f_{1,2} = z_1 \circ z_2^{-1} = 1/z$$
$$f_{2,1} = z_2 \circ z_1^{-1} = 1/z$$

are holomorphic on $U_1 \cap U_2 = \mathbb{C} - \{0\}$.

These relationships between curves and Riemann surfaces are embodied by the following two theorems

Theorem 2.1.3. (Normalization Theorem.) *For any irreducible algebraic curve $C \subset \mathrm{P}^2\mathbb{C}$ there exists a compact Riemann surface X and a holomorphic mapping*

$$\sigma : X \to \mathrm{P}^2\mathbb{C},$$

such that $\sigma(X) = C$ and σ is injective on the inverse image of the set of smooth points of C.

A Riemann surface together with the mapping σ is called the *normalization of C.* Loosely speaking, the normalization theorem states that an algebraic curve is a Riemann surface except at the singular points.

Conversely, every compact Riemann surface can be represented by an algebraic curve.

Theorem 2.1.4. *Any compact Riemann surface X can be obtained through the normalization of a certain plane algebraic curve C with at most ordinary double points. That is, there exists a holomorphic mapping*

$$\sigma : X \to \mathrm{P}^2\mathbb{C}$$

such that $\sigma(X)$ is an algebraic curve possessing at most ordinary double points.

Many of the geometric algorithms presented in this document are designed to avoid singular points. Except, for example, when we want to integrate a 1-form along a path leading to a singular point in which case we "unwrap" the singularity using Puiseux series. This is discussed in more detail in the following section. However, because of this we use the terms "curve" and "Riemann surface" interchangably.

Additionally, the algorithms presented in this document primarily work with the affine part $f(x, y)$ of the curve $F(x_0, x_1, x_2)$. If analysis on the line at infinity is necessary, for example, when computing the singular points of a curve, we consider an affine projection g of F onto l_∞,

$$g(u, y) = u^d f(1/u, y) = 0.$$

Thus, the surface considered here is the branched algebraic y-covering of the complex x-Riemann sphere, the set of all (x, y)-solutions to the affine polynomial equation

$$C = \{(x, y) \in \mathbb{C} \mid f(x, y) = \alpha_d(x)y^d + \alpha_{d-1}y^{d-1} + \cdots + \alpha_1(x)y + \alpha_0(x) = 0\},$$

as x varies along all of $\mathbb{C}$. We treat x and y as the independent and dependent variables of the equation, respectively.

A *point* $\alpha \in \mathbb{C}$ is called a *regular point of* C if

$$f(\alpha, y) = 0$$

has d distinct y-roots $y_0, \ldots, y_{d-1}$. A point $\alpha \in \mathbb{C}$ is called a discriminant point if it is not regular. The point $\alpha = \infty$ is a regular point of C if

$$g(0, y)$$

has d disctinct roots.

A *place* P is an element of X. For all but finitely many places, P is given by a pair (α, β) such that $f(\alpha, \beta) = 0$. However, some places, particularly those where α is a discriminant point, instead needs to be represented by a pair of series $(x(t), y(t))$ in some local coordinate t. This will be discussed in more detail in the following section.

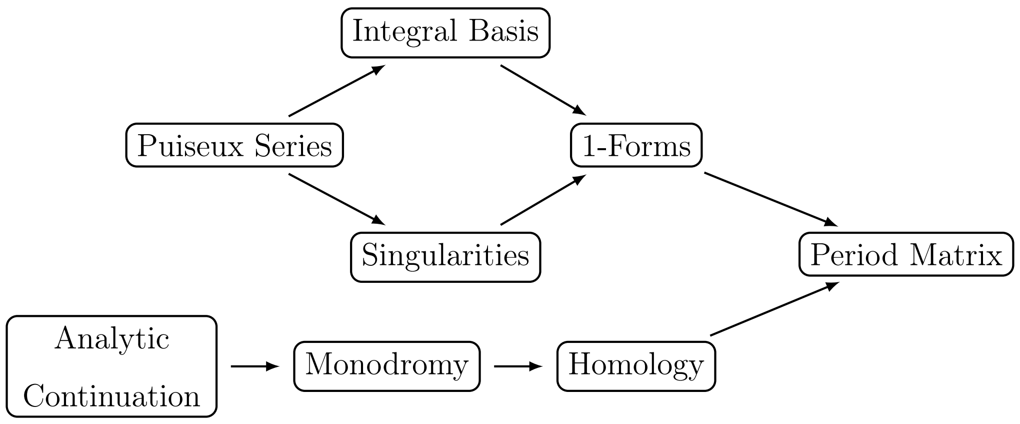


Figure 2.2: The major computations performed by `abelfunctions` *and their dependencies on one another.*

2.2 Algebraic Curves and Riemann Surfaces

In the previous section we introduced the Normalization Theorem relating complex algebraic curves to Riemann surfaces. The goal of this section is to derive the period matrix of a Riemann surface, the Abel Map, and the Riemann theta function. These three objects are the primary ingredients in constructing a large class of solutions to the Kadomtsev–Petviashvili equation, discussed in Chapter 4, and determinantal representations of plane curves, discussed in Chapter 5.

Throughout this section we present computational examples using the Sage software package Abelfunctions [34]. Abelfunctions is a library which provides general and easy to use framework for computing with Abelian functions, Riemann surfaces, and algebraic curves. The reader may follow along in the computations by downloading and installing Sage [13] from `www.sagemath.org` and following the instructions on `github.com/abefunctions/abelfunctions` for downloading and installing Abelfunctions. See Chapter 3 for information on the algorithms used and design of the software. In particular, an algorithmic perspective on the below components can be found in Section 3.4.

The examples used in this chapter use the curves,

$$C_2 : f_2(x,y) = y^3 + 2x^3y - x^7 = 0, \tag{2.2.1}$$

$$C_4 : f_4(x,y) = x^2y^3 - x^4 + 1 = 0. \tag{2.2.2}$$

In order to execute the code examples used in the rest of this chapter we begin by starting Sage, importing Abelfunctions, and defining these two curves in Sage,

```
sage: from abelfunctions import *
sage: R.<x,y> = QQ['x,y']; R
Multivariate Polynomial Ring in x, y over Rational Field
sage: f2 = y^3 + 2*x^3*y - x^7
sage: X2 = RiemannSurface(f2)
sage: f4 = x^2*y^3 - x^4 + 1
sage: X4 = RiemannSurface(f4)
```

2.2.1 Puiseux Series

Victor Puiseux's modesty was intimidating, and his patience and politeness were admirable. To a student blundering at some test, he just used to say, with a very sweet tone: "I don't know whether I heard well or whether I am mistaken, but it seems to me that what you said is not completely true." — Émile Picard

Every analytic function $f \in C^1(\mathbb{R})$ admits a local Taylor series representation in a neighborhood about $x = \alpha$. If the function is meromorphic it still admits a local series representation in the form of a Laurent series,

$$f(x) = \sum_{n=N}^{\infty} c_n (x-\alpha)^n$$

for some $N \in \mathbb{Z} \cup \{-\infty\}$ depending on α. In both of these situations the variable x is a *local coordinate* of f near the point α.

With algebraic curves, local coordinates are given in terms of Puiseux series which can be thought of as an extension of Laurent series.

Definition 2.2.1. *A* **Puiseux series** *expansion of a curve $C : f(x, y) = 0$ at the point $x = \alpha$ is a collection of $j = 1, \ldots, m$ series, $m \le d = deg_y f$, of the form*

$$P_j(t) = \begin{cases} x_j(t) = \alpha + \lambda_j t^{e_j}, \\ y_j(t) = \sum_{k=N}^{\infty} \beta_{jk} t^{n_{jk}}, \end{cases}$$

where $N \in \mathbb{Z} \cup \{-\infty\}$, $\alpha, \lambda_j, \beta_{jk} \in \mathbb{C}$, and $e_j, n_{jk} \in \mathbb{Z}$. Each Puiseux series P_j is called a **place** *on C. The variable, t, is called the* **local parameter** *of the series and is centered at $t = 0$.*

A place $P_j = P_j(t)$ satisfies,

$$f\big(P_j(t)\big) := f\big(x_j(t), y_j(t)\big) = 0.$$

When a Puiseux series $P_j(t)$ represents an expansion about a non-singular (α, β_j) on the curve then $P(0) = (\alpha, \beta)$. This is not necessarily the case with singular places. We list some additional important facts and properties of Puiseux series.

- The integer $|e_j|$ is called the *branching number* or *ramification index* of the series expansion at that place: $|e_j| > 1$ when $x = \alpha$ is a branch point of the curve. If $e_j = 1$ we say that the place is *unramified.*
- The number of Puiseux series m at a branch point $x = \alpha$ is less than or equal to $d = \deg_y f$. In particular, $d = \sum_{j=1}^{m} |e_j|$. If α is not a branch point of the curve (see Section 2.2.4) then $e_j = 1$ for all $j = 1, \ldots, d$; that is, there are d unramified places above $x = \alpha$.
- The field of Puiseux series is a splitting field for $\mathbb{C}[x, y] = \mathbb{C}[x][y]$. That is, given any $f \in \mathbb{C}[x, y]$ and Puiseux series expansions about any $x = \alpha$ we can write

  $$f(x, y) = \prod_{j=1}^{m} \prod_{k=1}^{e_j} \left(y - y_j\left((e/\lambda_j)^{2\pi i k/e_j}(x - \alpha)\right)\right),$$

  where the first product ranges over all Puiseux series P_j at $x = \alpha$ and the second product ranges over all y-components $y_j(x_j)$ when solving for t in terms of x. Note that a Puiseux series P_j with ramification index $|e_j| > 1$ splits into $|e_j|$ y-series in x. Note that each unramified place has only one x-series representation; that is P_j and y_j are considered equivalent and in 1-1 correspondence.

Example 2.2.2. Consider the example curve

$$C_2 : f_2(x, y) = y^3 + 2x^3y - x^7 = 0.$$

As seen in Example 2.1.2, the point $(x, y) = (0, 0)$ is a singular point of C. The Puiseux series expansions lying above $x = 0$ are,

$$P_1(t) = \begin{cases} x_1(t) = t, \\ y_1(t) = \frac{t^4}{2} - \frac{t^9}{16} + \frac{3t^{14}}{128} + O\left(t^{15}\right) \end{cases}$$
$$P_2(t) = \begin{cases} x_2(t) = -\frac{t^2}{2}, \\ y_2(t) = -\frac{t^3}{2} - \frac{t^8}{64} + \frac{3t^{13}}{4096} + O\left(t^{14}\right) \end{cases} \tag{2.2.3}$$

We see that P_1 is unramified and P_2 has ramification index of 2. The sum of ramification indices of these two places is indeed $3 = \deg_y f_2$. P_2 splits into two Puiseux x-series: solving for x_2 we get,

$$t = \pm\sqrt{2}ix^{1/2}, \tag{2.2.4}$$

which, when substituted into the expression for y_2 gives us,

$$y_{2,+}(x) = -\sqrt{2}ix^{3/2} - \tfrac{1}{4}x^4 - \tfrac{3\sqrt{2}}{64}x^{13/2} + O(x^7)$$
$$y_{2,-}(x) = \sqrt{2}ix^{3/2} - \tfrac{1}{4}x^4 + \tfrac{3\sqrt{2}}{64}x^{13/2} + O(x^7) \tag{2.2.5}$$

Substituting these truncated Puiseux series into affine expression of the curve we get,

$$f_2\left(x_1(t), y_1(t)\right) = \tfrac{3}{128}t^{22} - \tfrac{19}{4096}t^{27} + O\left(t^{32}\right)$$
$$f_2\left(x_2(t), y_2(t)\right) = -\tfrac{3}{8192}t^{19} - \tfrac{1}{262144}t^{24} + O\left(t^{39}\right). \tag{2.2.6}$$

Again, the local parameter is always centered at $t = 0$. Therefore, for small values of $|t|$

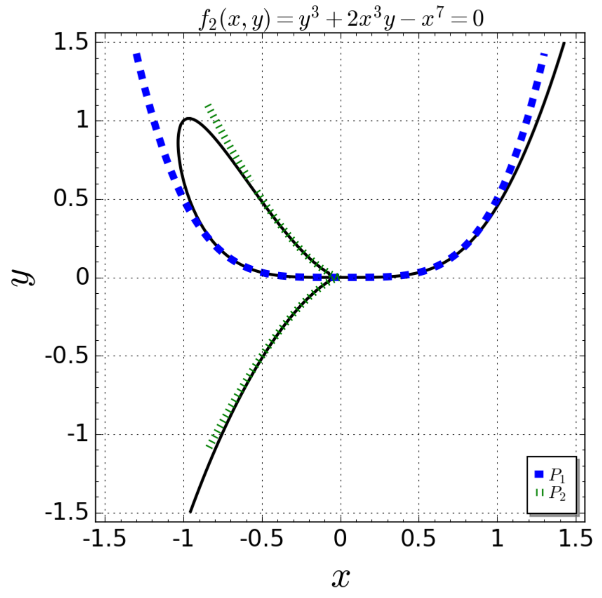


Figure 2.3: A real plot of the curve $C_2 : f_2(x,y) = y^3 + 2x^3y - x^7$. The blue and green dashed lines are plots of the places $P_1 = \left(t, t^4/2 + O(t^9)\right)$ and $P_2 = \left(-t^2/2, -t^3/2 + O(t^8)\right)$ for $t \in (-1.3, 1.3)$.

we see that $f_2\left(x(t), y(t)\right) = O(t^{19})$ is extraordinarily small in magnitude. As more terms are determined in the expansions of y_1 and y_2 we expect $N \to \infty$ in $f_2\left(x(t), y(t)\right) = O(t^N)$. We see that these series well-approximate the curve at $x = 0$ in Figure 2.3.

We can verify these Puiseux series expansions above $x = 0$ and their properties using `abelfunctions`.

```
sage: P1, P2 = X2(0)  # the places above x=0 on the surface
sage: P1
(t, 1/2*t^4 + O(t^7))
sage: P1.puiseux_series.extend(14)  # extend the curve to at least O(t^14)
sage: P1
(t, 1/2*t^4 - 1/16*t^9 + 3/128*t^14 + O(t^19))
sage: P2
(-1/2*t^2, -1/2*t^3 + O(t^5))
sage: P2.puiseux_series.extend(13)
```

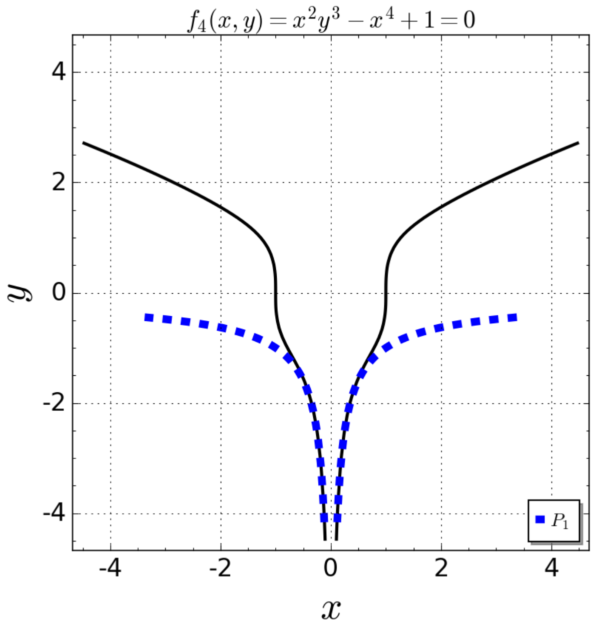


Figure 2.4: A real plot of the curve $C_4 : f_4(x,y) = x^2y^3 - x^4 + 1$. The dashed line is a plot of the place $P = \left(-t^3, -t^{-2} + O(t^{10})\right)$ for $t \in (-1.5, 1.5)$.

```
sage: P2  # may include higher order terms
(-1/2*t^2, -1/2*t^3 - 1/64*t^8 + 3/4096*^13 - 1/16384*t^18 + O(t^20))
```

Example 2.2.3. Now consider the example curve,

$$C_4 : f_4(x,y) = x^2y^3 - x^4 + 1 = 0.$$

A plot of the curve in the real x, y-plane is shown in Figure 2.4. There is a single place lying above $x = 0$,

$$P(t) = \begin{cases} x(t) = -t^3, \\ y(t) = -t^{-2} + \frac{1}{3}t^{10} + O(t^{15}) \end{cases} \tag{2.2.7}$$

Unlike the places shown in Example 2.2.2 the place P has a component at infinity. The figure illustrates how P acts as a local chart of the curve: the $O(t^{15})$ series expansion well-approximates the curve at $x = 0$. In Abelfunctions, this place is treated like any other place.

```
sage: P, = X4(0)
sage: P
(-t^3, -t^-2 + O(t^0))
sage: P.puiseux_series.extend(14)
sage: P
(-t^3, -t^-2 + 1/3*t^10 + O(t^15))
```

Puiseux series expansions allow us to locally examine the behavior of an algebraic curve or Riemann surface. In particular, they provide a local coordinate chart centered at a point $(x, y) = (\alpha, \beta)$ on the curve in the sense discussed in Section 2.1.4. In the next section we describe a particular class of point/place on a Riemann surface: the singular point.

2.2.2 Singularities

Recall from Definition 2.1.1 that a point a on a projective curve C is a singular point if

$$\left(\frac{\partial F}{\partial x_0}, \frac{\partial F}{\partial x_1}, \frac{\partial F}{\partial x_2}\right)(a) = (0, 0, 0).$$

For singular points of the form $a = (1 : \alpha, \beta)$, this is equivalent to

$$\frac{\partial f}{\partial x}(\alpha, \beta) = 0, \quad \frac{\partial f}{\partial y}(\alpha, \beta) = 0,$$

where f is the affine portion of the curve. The singular points of a curve need to be determined not only for the numerical analytic continuation and integration methods discussed below, so we can appropriately desingularize the curve C and obtain a Riemann surface X, but they are also an essential ingredient to computing a basis of holomorphic 1-forms on X.

For singular points of the form $a = (1 : \alpha : \beta)$ the Puiseux series expansion P_j of $f = f(x, y)$ such that $P_j(0) = (\alpha, \beta)$ is a coordinate chart centered at $(x, y) = (\alpha, \beta)$. P_j tells us how to approach and pass through $(x, y) = (\alpha, \beta)$ on the curve. Similarly, singular points at infinity have coordinate charts defined by the Puiseux series with $e_j < 0$. With these coordinate charts we can desingularize the curve and thus create an appropriate atlas for the corresponding Riemann surface X.

For the purposes of later determining the genus of X as well as the space of holomorphic 1-forms $\Gamma(X, \Omega^1_X)$ on X we need to compute the *delta invariant* and the *multiplicity* of a singularity. In the following discussion we assume for simplicity that the singularity is finite. To analyze infinite singular points we project the curve C onto the line at infinity l_∞ using the method described in Section 2.1.3.

- **branching number** — The branching number R of a singular point (α, β) is the sum of the branching numbers/ramification indices of the Puiseux series centered at $(x, y) = (\alpha, \beta)$. That is,
$$R = \sum_{\substack{j \\ P_j(0)=(\alpha,\beta)}} |e_j|. \tag{2.2.8}$$
- **multiplicity** — As given in Section 2.1.3, the multiplicity of a singular point is the degree of the lowest degree non-zero homogeneous term appearing in the polynomial expression for the curve centered at (α, β).
- **delta invariant** — The delta invariant δ_P of a singularity P is the number of double points concentrated at the singularity. This is equal to the number of quadratic factors $(\alpha_i x - \beta_i y)^2$ appearing in the tangent cone at the singularity. Let S be the set of all singular points, finite and infinity, of C. Then the genus is given by
$$g = \frac{(d-1)(d-2)}{2} - \sum_{P \in S} \delta_P. \tag{2.2.9}$$

Example 2.2.4. We compute the singularities of the curve,

$$C_2 : f_2(x, y) = y^3 + 2x^3 y - x^7 = 0.$$

Note that to match the notation used in Sage the finite and infinite singularities are presented in the format $(\alpha : \beta : 1)$ and $(\gamma : 1 : 0)$, respectively. The order of the projective components is different than that presented in Section 2.1.2.

```
sage: S = singularities.singularities(f2)
sage: for s, (m, delta, r) in S:
....:     print 'Singularity:', s
....:     print '\tmultiplicity     =', m
....:     print '\tdelta invariant  =', delta
....:     print '\tbranching number =', r
....:
Singularity: (0, 0, 1)
        multiplicity     = 3
        delta invariant  = 4
        branching number = 2
Singularity: (0, 1, 0)
        multiplicity     = 4
        delta invariant  = 9
        branching number = 1
```

As seen in Equation (2.2.3) the places P_1 and P_2 above $x = 0$, which have $y-$component $y(0) = 0$, have ramification indices 1 and 2, respectively, which matches the expected multiplicity. HOLY SHIT! THERE MAY BE AN ISSUE, HERE.

The genus is computed using the formula in Equation (2.2.9). Later, performs additional checks on the genus using the geometric properties discussed in 2.2.4.

```
sage: singularities.genus(f,x,y)
2
```

Example 2.2.5. We compute the singularities of the curve,

$$C : f(x, y) = (y^2 - x^2)(x - 1)(2x - 3) - 4(x^2 + y^2 - 2x)^2.$$

A plot of the curve in the real x, y-plane is shown in Figure 2.5. The gradient of the curve in the affine plane is given by,

$$\nabla f(x, y) = \left(-(12x - 11)y^2 - 24x^3 + 63x^2 - 38x, \quad -16y^3 - 2(6x^2 - 11x - 3)y\right).$$

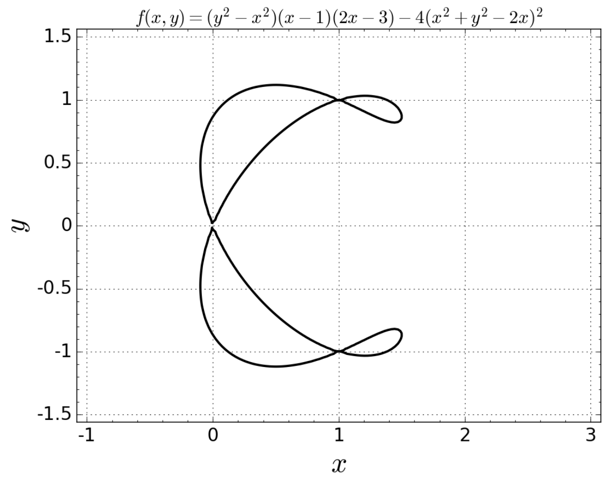


Figure 2.5: A real plot of the curve $C : f(x,y) = (y^2 - x^2)(x-1)(2x-3) - 4(x^2+y^2-2x)^2$*. Note the singularities at* $(0,0), (1,1),$ *and* $(1,-1)$ *where the gradient vanishes.*

The gradient indeed vanishes at the points $(x,y) = (0,0), (1,1)$, and $(1,-1)$. In fact, these are the only singular points of this curve.

```
sage: f = (y^2-x^2)*(x-1)*(2*x-3) - 4*(x^2+y^2-2*x)^2
sage: S = singularities.singularities(f4)
sage: for s, (m, delta, r) in S:
....:     print 'Singularity:', s
....:     print '\tmultiplicity     =', m
....:     print '\tdelta invariant  =', delta
....:     print '\tbranching number =', r
....:
Singularity: (0, 0, 1)
        multiplicity     = 2
        delta invariant  = 1
        branching number = 2
Singularity: (1, -1, 1)
        multiplicity     = 2
        delta invariant  = 1
```

```
          branching number = 2
Singularity: (1, 1, 1)
          multiplicity     = 2
          delta invariant  = 1
          branching number = 2
```

This curve is a rational curve: the total degree of the curve is $d = 4$ and the three singularities have a delta invariant equal to one.

2.2.3 Analytic Continuation

The description of right lines and circles, upon which geometry is founded, belongs to mechanics. Geometry does not teach us to draw these lines, but requires them to be drawn. — Isaac Newton

The root of our geometric approach toward the period matrix of a Riemann surface are paths on the Riemann surface. A *path on a Riemann surface, X,* is a continuous map $\gamma : [0,1] \to X$. On the normalization of a curve $C : f(x,y) = 0$ a path can be defined as $\gamma : [0,1] \to C \subset \mathbb{C}^2$. In particular, if $\gamma(t) = (x_\gamma(t), y_\gamma(t))$ then $f(x(t), y(t)) = 0$ for all $t \in [0,1]$ including points at infinity. Because the roots of a polynomial are continuous as a function of the coefficients [23], an x-path $x_\gamma : [0,1] \to \mathbb{C}_x$ and an initial y-root $y_0 \in \mathbb{C}_y$ are sufficient for defining a path on C. Given these information, the resulting y-path $y_\gamma : [0,1] \to \mathbb{C}_y$ is completely determined by the curve

$$f\big(x_\gamma(t), y_\gamma(t)\big) = 0, \quad y_\gamma(0) = y_0.$$

The process of deriving this y-path from the data provided is called *analytic continuation.*

Example 2.2.6. A picture helps demonstrate this concept. Consider the curve,

$$C_2 : f_2(x,y) = y^3 + 2x^3y - x^7 = 0.$$

Above $x = 2$ are three unramified places, $y_0 = 4$, $y_1 = TODO$, and $y_2 = TODO$. Consider the complex x-path $x_\gamma(t) = 2e^{i\pi t}$: the counter-clockwise semi-circle of radius 2 centered at the origin beginning at $x = 2$. As x continuously varies along this path for $t \in [0, 1]$ the roots $y_0 = y_{\gamma_0}(t)$, $y_1 = y_{\gamma_1}(t)$, and $y_2 = y_{\gamma_2}(t)$ vary continuously as well. Figure 2.6a

Consider, instead, the path $x_\gamma(t) = 2 - 4t$. Recall that $x = 0$ has three places lying above it each of them at $y = 0$. To properly analytically continue through this point along x_γ we use the Puiseux series expansions of C at $x = 0$ given in Equation (2.2.3),

$$P_1(t) = \begin{cases} x_1(t) = t, \\ y_1(t) = \frac{t^4}{2} - \frac{t^9}{16} + \frac{3t^{14}}{128} + O\left(t^{15}\right) \end{cases}$$

$$P_2(t) = \begin{cases} x_2(t) = -\frac{t^2}{2}, \\ y_2(t) = -\frac{t^3}{2} - \frac{t^8}{64} + \frac{3t^{13}}{4096} + O\left(t^{14}\right) \end{cases}$$

Figure **??** shows how these three roots continuously "pass through" the branch point.

A *closed path γ on a Riemann surface* is one such that $\gamma(0) = \gamma(1)$. That is, a path is closed when $x_\gamma(0) = x_\gamma(1)$ and $y_\gamma(0) = y_\gamma(1)$. When constructing a path using an x-path it may be the case that the x-path $x_\gamma(t)$ is closed in $\mathbb{C}_x$ but the derived y-path $y_\gamma(t)$ may not satisfy $y_\gamma(0) = y_\gamma(1)$. This situation is described in more detail in Section **??** on monodromy groups of algebraic curves. Finally, analytic continuation can be extended to piecewise continuous path in $\mathbb{C}_x$.

2.2.4 Monodromy

At a generic point $x = \alpha_0 \in \mathbb{C}$ on a curve $C : f(x, y) = 0$ has d distinct ordered y-roots $(y_0, \ldots, y_{d-1})$ at α_0. This collection of y-roots is sometimes called the *lift of* or the *fibre above* $x = \alpha_0$. However, at a point $x = b$ where both $f(x, y) = 0$ and $\partial_y f(x, y) = 0$ the number of distinct roots in the lift is strictly less than d. Such a point $x = b$ is called a *discriminant point* of f. That is, a discriminant point has at least one ramified place lying above it.

A *branch point* $x = b$ is a discriminant point having the property that if one were to analytically continue a fibre around some closed path encircling b then the elements of the fibre are permuted.

images/livekp.jpg

images/livekp.jpg

images/livekp.jpg

images/livekp.jpg

(a) $x_\gamma(t) = 2\exp(i\pi t)$

(b) $x_\gamma(t) = 2 - 4t$

Figure 2.6: Analytic continuation of the roots $y_0 = 4, y_1 = TODO$, and $y_1 = TODO$ of the curve C_2 along various paths in $\mathbb{C}_x$.

Specifically, let $x_\gamma : [0,1] \to \mathbb{C}$ be a piecewise differentiable oriented closed path in the complex x-plane encircling a branch point $x = b$ exactly once in the positive direction and let $(y_0, \ldots, y_{d-1})$ be a fixed ordering of the fibre at $x_\gamma(0) = b$. Then, after analytically continuing the fibre around x_γ and returning to $x_\gamma(1) = b$, the fibre is equal to

$$(y_{\pi_b(0)}, \ldots, y_{\pi_b(d-1)}), \tag{2.2.10}$$

where $\pi_b \in S_d$ is a permutation on d elements. With this notation in hand, a *branch point* is a discriminant point with $\pi_b \neq \mathrm{id}$.

Example 2.2.7. We see from the example in the previous section that the fibre $(y_0, y_1, y_2) = (2, TODO, TOO)$ above $x = 2$ is permuted

To analyze the permutation behavior of multiple branch points $\{b_1, \ldots, b_n\}$ we start by fixing some *base point* $x = \alpha_0$ in the complex plane such that α_0 is not a branch point and we fix an ordering $(y_0, \ldots, y_{d-1})$ of the fibre above α_0. Let $x_{\gamma_k} : [0,1] \to \mathbb{C}$ be a path encircling only the branch point b_k in the positive direction which does not cross the other paths. Such a path is called a *monodromy path* of b_k. In the case when $x = \infty$ is a branch point a monodromy path for ∞ is taken to be a circle going around all of the finite branch points in the negative direction. See Figure 2.7 for an illustration of these paths.

Analytically continuing the ordered fibre $(y_0, \ldots, y_{d-1})$ around each of the branch points results in $n + 1$ permutations

$$\pi_{b_1}, \ldots, \pi_{b_n}, \pi_\infty \in S_d$$

The group generated by these permutations is called the *fundamental group* of $\mathbb{C} \setminus \{b_1, \ldots, b_n\}$. It is denoted $\pi_1(\mathrm{P}^1\mathbb{C} \setminus \{b_1, \ldots, b_n\}, \alpha_0)$. Observe that, by the disjoint path condition on the monodromy paths, moving the base point α_0 corresponds to conjugation of the generators of the fundamental group by some $\pi \in S_d$. Hence, the monodromy group has explicit dependence on the base point.

We compute the monodromy group of the curve

$$C : f(x,y) = y^3 + 2x^3y - x^7 = 0,$$

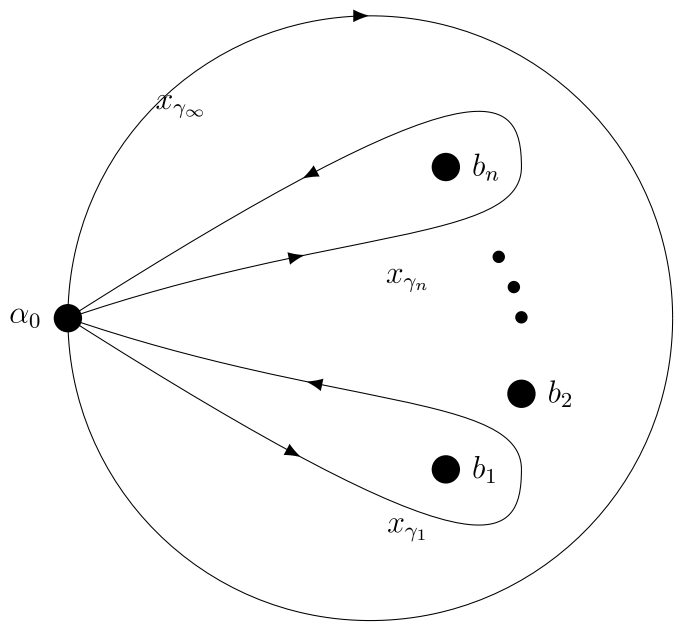


Figure 2.7: The discriminant points $b_1, \ldots, b_n$ with their respective monodromy paths $x_{\gamma_1}, \ldots, x_{\gamma_n}$ and the path x_{γ_∞} around the point at infinity.

where the permutations $\pi_j \in \pi_1(\mathrm{P}^1\mathbb{C} \setminus \{b_1, \ldots, b_n\}, \alpha_0)$ are presented in disjoint cycle notation.

```
from abelfunctions import *
from sympy.abc import x,y,t

f = y**3 + 2*x**3*y - x**7
X = RiemannSurface(f,x,y)

b = X.branch_points()
pi_1 = X.monodromy_group()

for bj,pi_1j in zip(b,pi_1):
    print 'branch point:', bj
    print 'permutation: ', pi_1j
    print
```

```
branch point: (-0.31969776999-0.983928563571j)
permutation:  [(0, 2), (1,)]

branch point: (0.836979627962-0.608101294789j)
permutation:  [(0,), (1, 2)]

branch point: (-1.03456371594+0j)
permutation:  [(0,), (1, 2)]

branch point: 0j
permutation:  [(0, 2), (1,)]

branch point: (0.836979627962+0.608101294789j)
permutation:  [(0,), (1, 2)]

branch point: (-0.31969776999+0.983928563571j)
permutation:  [(0, 1), (2,)]

branch point: oo
permutation:  [(0, 2, 1)]
```

The method `RiemannSurface.show_paths()` plots all the monodromy paths x_{γ_j} in the complex x-plane. The base point $x = \alpha_0$ is marked in red.

```
X.show_paths()
```

```
<matplotlib.figure.Figure object at 0x107d60810>
```

2.2.5 Homology

A compact Riemann surface X of genus g is homeomorphic to a sphere with g handles or, equivalently, a doughnut with g holes. A cycle on X is a closed, oriented, piecewise smooth curve $\gamma : [0,1] \to X$ such that $\gamma(0) = \gamma(1)$. The first homology group $H_1(X,\mathbb{Z})$ of X is the collection of all cycles on X modulo homologous transformations. In this document we do not state precisely what it means for two cycles to be homologous since it involves presenting the basic theory of simplicial complexes which is beyond the scope of this document.

However, in brief, two cycles on X are homologous if they can be deformed to each other where the process of deformation not only allows continuous transformations but the splitting of one cycle into two via "pinching". A demonstration of this procedure is shown in Figure 2.8. Two cycles can be added together by "reversing" the pinching process and negation of a path corresponds to reversing its orientation. The *first homology group* $H_1(X,\mathbb{Z})$ is the set of all cycles on X with the addition operation described. The equivalence of cycles on a Riemann surface is the same as that of closed paths on the complex plane (specifically, the Riemann sphere) upon which one integrate a fixed meromorphic function g. Closed paths not encircling a pole of g are homologous to the zero path since they can be contracted to a point. The set of all paths encircling a single, given pole are all homologous to each other.

$H_1(X,\mathbb{Z})$ has a basis of cycles $\{a_1,\ldots,a_g,b_1,\ldots,b_g\}$. That is, every cycle on X can be written as a finite, integer linear combination of the a- and b-cycles. These cycles can be chosen such that they satisfy the intersection properties

$$\begin{aligned} a_i \circ a_j &= 0, \quad \forall i \neq j \\ b_i \circ b_j &= 0, \quad \forall i \neq j \\ a_i \circ b_j &= \delta_{ij}, \quad \forall i,j = 1,\ldots,g \end{aligned}$$

where δ_{ij} is the Kronecker delta. That is, the only cycles that intersect are a_i and b_i. A basis of cycles fulfilling these intersection requirements is called a *canonical basis of cycles.* Figure 2.9 illustrates the canonical basis for a genus two Riemann surface.

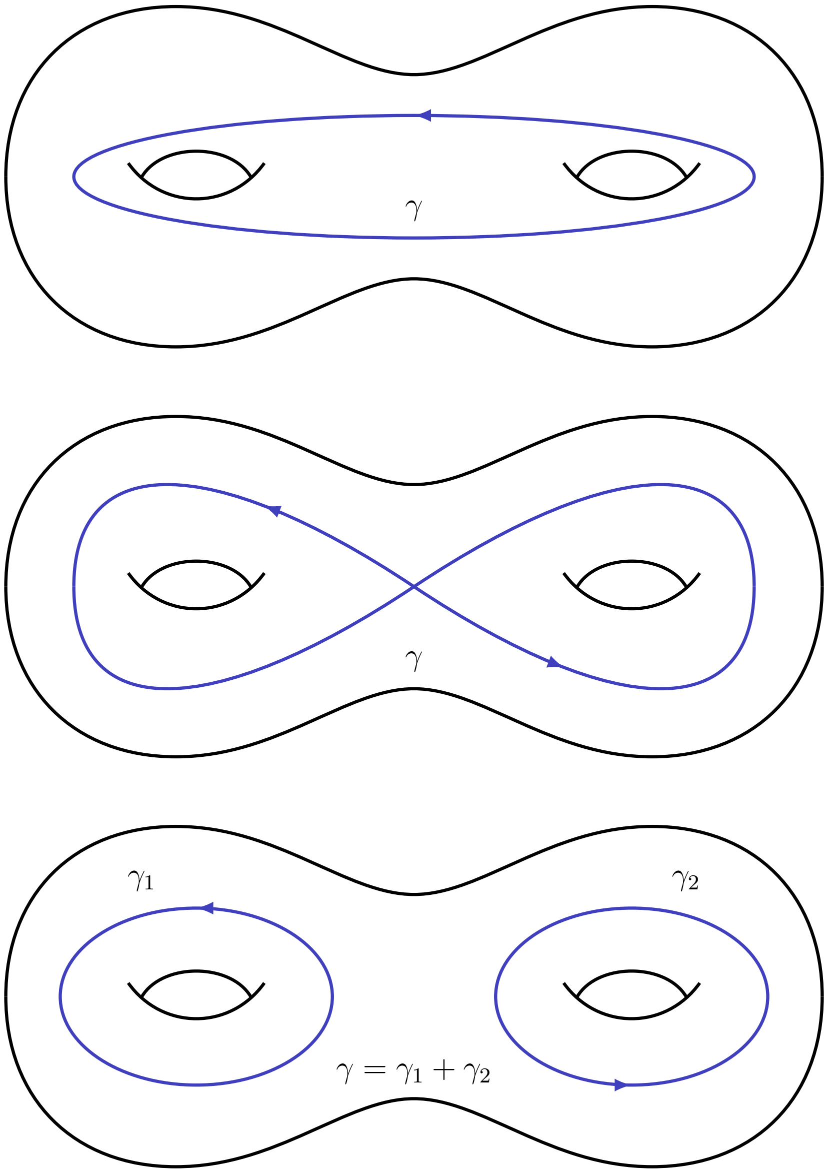


Figure 2.8: A genus $g = 2$ Riemann surface X with three homologous cycles. The process of "pinching" and separating a cycle γ into two cycle is allowed. Cycles can be added together by reversing this pinching process. Negation of a cycle corresponds to reversing its orientation.

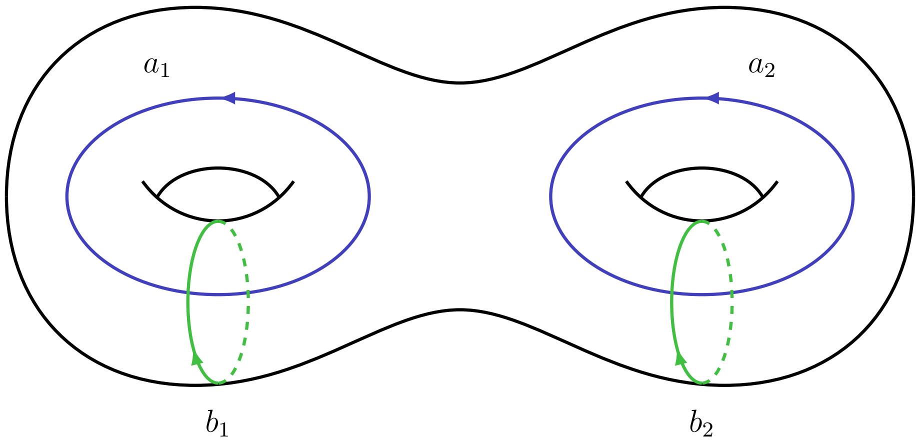


Figure 2.9: A genus $g = 2$ Riemann surface X with the basis cycles $\{a_1, a_2, b_1, b_2\}$ for the first homology group $H_1(X, \mathbb{Z})$.

We compute a homology basis for the Riemann surface X obtained by desingularizing and compactifying the curve

$$C : f(x,y) = y^3 + 2x^3y - x^7 = 0.$$

The a- and b- cycles are presented as a list $[\ldots, s_i, (b_i, r_i), s_{i+1}, \ldots]$ where s_i is the current sheet number, b_i is a branch point of C, $r_i \in \mathbb{Z}$, and s_{i+1} the the sheet reached after rotating r_i times around b_i and returning to the base point α_0.

```
from abelfunctions import *
from sympy.abc import x,y,t

f = y**3 + 2*x**3*y - x**7
X = RiemannSurface(f,x,y)
a,b = X.homology()

# print the a-cycles
for i in range(g):
    print 'a_%d:'%(i+1)
    print a[i]
    print

# print the b-cycles
for i in range(g):
    print 'b_%d:'%(i+1)
    print b[i]
    print


a_1:
[0, ((-0.31969776999025984-0.9839285635706635j), 1), 2,
 ((-1.0345637159435732+0j), -1), 1,
 ((-0.31969776999025984+0.9839285635706635j), -1), 0]

a_2:
[0, (0j, 1), 2, ((-0.31969776999025984-0.9839285635706635j), -1), 0]

b_1:
[0, ((-0.31969776999025984+0.9839285635706635j), 1), 1,
 ((0.8369796279620464-0.6081012947885316j), 1), 2,
 ((-0.31969776999025984-0.9839285635706635j), -1), 0]



b_2:
[0, ((-0.31969776999025984-0.9839285635706635j), 1), 2,
 ((-1.0345637159435732+0j), -1), 1,
 ((-0.31969776999025984+0.9839285635706635j), -1), 0, (oo, 1), 2,
 ((-0.31969776999025984-0.9839285635706635j), -1), 0,
 ((-0.31969776999025984+0.9839285635706635j), 1), 1,
 ((0.8369796279620464-0.6081012947885316j), 1), 2,
 ((-0.31969776999025984-0.9839285635706635j), -1), 0]
```

We can plot the projection of the cycle in the complex x- and y-planes. In this example, we plot the cycle a_1 by computing 512 interpolating points on the path. The x-projection

x_γ is in blue and the y-projection y_γ is in green.

```
alpha = X.base_point()
betas = X.base_lift()
P0 = alpha, betas

gamma = RiemannSurfacePath((f,x,y), P0, cycle = a[0])
gamma.plot(512)
```

```
<matplotlib.figure.Figure at 0x106e9cd90>
```

2.2.6 Holomorphic and Meromorphic Differentials

1-forms on a Riemann surface X are objects that can be integrated along piecewise smooth paths on X.

Definition 2.2.8. (1-Form) *Let X be a Riemann surface with atlas $\{(U_\alpha, z_\alpha)\}$. A 1-form ω on X, also called a differential, is such that in each local coordinate $z_\alpha : U_\alpha \subset X \to \mathbb{C}$,*

$$\omega\Big|_{U_\alpha} = f_\alpha(z_\alpha)dz_\alpha,$$

and the appropriate compatibility conditions are satisfied under the action of transition functions on $U_\alpha \cup U_\beta$ where (U_β, z_β) is another local coordinate. The space of all 1-forms on X is denoted Ω^1_X.

The space of all *holomorphic 1-forms* is of particular interest.

Definition 2.2.9. (Holomorphic 1-Forms) *The space of holomorphic 1-forms $\Gamma(X, \Omega^1_X)$ on X is the space of 1-forms ω such that in each local coordinate (U_α, z_α),*

$$\omega\Big|_{U_\alpha} = h_\alpha(z_\alpha)dz_\alpha$$

where $h_\alpha : U_\alpha \to \mathbb{C}$ is a holomorphic function.

For a compact genus g Riemann surface X, $\Gamma(X, \Omega^1_X)$ is a finite-dimensional vector space of dimension g over $\mathbb{C}$. Thus, it has a basis of g holomorphic 1-forms $\{\omega_1, \ldots, \omega_g\}$.

For Riemann surfaces obtained by desingularizing and compactifying an algebraic curve $C : f(x, y) = 0$ these basis holomorphic 1-forms can be written as

$$\omega_k(x, y) = \frac{p_k(x, y)}{\partial_y f(x, y)} dx,$$

where $p_k \in \mathbb{C}[x, y]$ is of degree at most $d - 3$ in x and y. The polynomials p_k are called the *adjoint polynomials of f*. Note that since y has explicit dependence on x due to the equation $f(x, y) = 0$, we can use x as the local coordinate of the differential.

We compute a basis of holomorphic 1-forms on the Riemann surface X given by the desingularization and compactification of the algebraic curve

$$C : f(x, y) = y^3 + 2x^3y - x^7 = 0.$$

```
from sympy.abc import x,y,t

f = y**3 + 2*x**3*y - x**7
X = RiemannSurface(f,x,y)
oneforms = X.holomorphic_differentials()

for omega in oneforms:
    print 'omega(x,y) =\n'
    sympy.pprint(omega, use_unicode=False)
    print
```

```
omega(x,y) =

    x*y
-----------
   3      2
2*x  + 3*y
```

```
omega(x,y) =


         3
        x
-----------
   3       2
2*x   + 3*y
```

From this we can infer that the adjoint polynomials of the curve are $p_1(x, y) = xy$ and $p_2(x, y) = x^3$.

Divisors

The *valuation divisor* associated with a meromorphic one-form on X is a direct ingredient in the determination of finite-genus solutions to the KP-equation. Before we define the valuation divisor we take a brief detour into the realm of general divisors on a Riemann surface.

Definition 2.2.10. *A* **divisor** *$\mathcal{D}$ on the Riemann surface X is a finite formal linear combination of places P_i with multiplicities n_i:*

$$\mathcal{D} = \sum_i n_i P_i. \tag{2.2.11}$$

The sum,

$$\deg \mathcal{D} = \sum_i n_i \tag{2.2.12}$$

is called the **degree of** *$\mathcal{D}$.*

The set of all divisors on a Riemann surface forms an Abelian group $\mathrm{Div}(X)$ under addition. A divisor with all $n_i \geq 0$ is called *positive* or *effective*.

2.2.7 Jacobian and Period Matrices

Period matrices are matrices obtained by integrating the holomorphic differentials $\omega_1, \ldots, \omega_g$ along the a-cycles $a_1, \ldots, a_g$ and b-cycles $b_1, \ldots, b_g$. Define the $g \times g$ matrices

$$A = (A_{ij})_{i,j=1}^{g}, \quad A_{ij} = \oint_{a_j} \omega_i,$$
$$B = (B_{ij})_{i,j=1}^{g}, \quad B_{ij} = \oint_{b_j} \omega_i.$$

A *period matrix* of X is the $g \times 2g$ matrix

$$\tau = [A\ B].$$

We often normalize the differentials ω_i such that $A_{ij} = \delta_{ij}$ which results in the period matrix

$$\tau = [I_{g\times g}\ \Omega]. \tag{2.2.13}$$

This is equivalent to setting $\Omega = A^{-1}B$. The matrix $\Omega \in \mathbb{C}^{g\times g}$ is a *Riemann matrix*: an invertible, symmetric complex matrix with positive definite imaginary part. The columns of τ define a lattice

$$\Lambda = \{Im + \Omega n \mid m, n \in \mathbb{Z}^g\} \subset \mathbb{C}^g.$$

This lattice plays an important role in the theory of algebraic curves since the quotient space

$$J(C) = \mathbb{C}^g / \Lambda \cong \mathbb{T}^{2g} \tag{2.2.14}$$

is the *Jacobian* or *Jacobian variety* of the curve C. Jacobian varieties play a central role in the theory of algebraic curves. For example, the Torelli theorem [27] states that a non-singular projective curve is completely determined by its Jacobian. The Schottky problem establishes a link between the Jacobian and the Kadomtsev–Petviashvili equation by providing conditions on when a given Riemann matrix is a period matrix of some algebraic curve.

The `RiemannSurface.period_matrix()` method returns the matrices A and B defined above. The Riemann matrix Ω is obtained by computing $\Omega = A^{-1}B$

```
from abelfunctions import *
from sympy.abc import x,y,t
from scipy import dot
from scipy.linalg import inv

f = -x**7 + 2*x**3*y + y**3
X = RiemannSurface(f, x, y)
A,B = X.period_matrix()
Omega = dot(inv(A), B)

print 'A =\n', A
print 'B =\n', B
print 'Omega =\n', Omega
```

```
A =
[[ -1.38142275e-12-1.20192474j   1.84957199e+00+0.60096237j]
 [  9.22903420e-12+1.97146395j   7.16176201e-01-0.98573197j]]
B =
[[-0.70647363+2.17430227j -1.84957199+2.54571744j]
 [-1.87497364-1.36224808j -0.71617620+0.23269975j]]
Omega =
[[-1.30901699+0.95105652j -0.80901699+0.58778525j]
 [-0.80901699+0.58778525j -1.00000000+1.1755705j ]]
```

We numerically verify that Ω is a Riemann matrix by computing $\|\Omega - \Omega^T\|$ as well as the eigenvalues of $\operatorname{Im}\Omega$.

```
print norm(Omega.T - Omega)
print
print eigvals(Omega.imag)
```

```
9.303308740879998e-11
```

```
[ 0.46490467  1.66172235]
```

2.2.8 The Abel Map

Definition 2.2.11. *Let $P \in X$ be a fixed place. The Abel Map $\boldsymbol{A} : X \to J(X)$ is defined by*

$$\boldsymbol{A}(P,Q) = \big(A_1(P,Q), \dots, A_g(P,Q)\big), \tag{2.2.15}$$

where

$$A_j(P,Q) = \int_P^Q \omega_j, \tag{2.2.16}$$

and the path chosen from P to Q is the same for each A_j. The Abel map is written in vector form as

$$\boldsymbol{A}(P,Q) = \int_P^Q \boldsymbol{\omega}. \tag{2.2.17}$$

The definition of the Abel map can be extended to divisors: let $\mathcal{D} = \sum_i n_i P_i$. We define

$$\boldsymbol{A}(P,\mathcal{D}) = \sum_i n_i \boldsymbol{A}(P,P_i). \tag{2.2.18}$$

The Abel Map is independent of the path γ from P to Q chosen on X for if γ and η are two such paths then their difference is a linear combination of homology basis cycles. The integral of $\boldsymbol{\omega}$ along this closed path is a lattice element and therefore is congruent to zero in $J(X)$.

An algorithm for computing the Abel map is described in [10]. The implementation in ABELFUNCTIONS is based on this algorithm.

Example 2.2.12. ABELFUNCTIONS selects a regular "base place" P_0 from which to construct all paths on X. When given a single argument `AbelMap()` returns $\boldsymbol{A}(P_0, P)$.

```
sage: J = Jacobian(X)       # reduces vectors modulo lattice ZZ^g + Omega ZZ^g
sage: z1 = AbelMap(P); z1   # Abel map from P0 to P
[-0.5261+0.0864j  0.0669+0.6392j -0.7495+1.1037j -1.5030+1.0356j]
sage: z2 = AbelMap(Q); z2   # Abel map from P0 to Q
```

```
[-0.3875+0.1157j -0.0290+0.4437j -0.4532+0.7774j -0.9721+0.6732j]
sage: z3 = AbelMap(P,Q); z3 # Abel map from P to Q
[ 0.1468-0.0985j  0.8467+0.6989j  0.0996+1.0083j -1.1003+0.8159j]
sage: numpy.linalg.norm( J((z2-z1) - z3) ) # numerically verify that A(P,Q) = A(P0,Q) - A
3.80631643473e-16
```

The Abel map accepts divisors as well.

```
sage: w = AbelMap(D); w
[ 0.0670-0.1361j  0.9421+0.7429j -0.4887+0.7663j -1.5057+0.6992j]
sage: z = J(3*z1 + z2) # verify that w = 3*z1 + z2 mod period lattice
sage: numpy.linalg.norm(w-z)
1.57107346e-15
```

2.2.9 The Riemann Constant Vector

Definition 2.2.13. *Let X be a genus g Riemann surface with associated Riemann matrix Ω. The Riemann constant vector $\boldsymbol{K} : X \to J(X)$ is defined as*

$$\boldsymbol{K}(P) = \big(K_1(P), \ldots, K_g(P)\big), \tag{2.2.19}$$

where

$$K_j(P) = \frac{1 + \Omega_{jj}}{2} - \sum_{k \neq j}^{g} \oint_{a_k} \omega_k(Q) A_j(P, Q). \tag{2.2.20}$$

Once we know the value of $\boldsymbol{K}(P_0)$ the value of $\boldsymbol{K}(P)$ is determined using only a shift by the Abel map:

Theorem 2.2.14. *Let P_0, P be places on a genus g Riemann surface X. Then*

$$\boldsymbol{K}(P) = \boldsymbol{K}(P_0) + (g-1)\,\boldsymbol{A}(P_0, P). \tag{2.2.21}$$

Proof. Let Q be an arbitrary place on X. By the definition of the Abel map, $\boldsymbol{A}(P, Q) =$

$\boldsymbol{A}(P, P_0) + \boldsymbol{A}(P_0, Q)$. Using this identity in the definition of the RCV we obtain

$$\begin{aligned}
K_j(P) &= \frac{1+\Omega_{jj}}{2} - \sum_{k\neq j}^{g} \oint_{a_k} \omega_k(Q) A_j(P,Q)\,\mathrm{d}Q \\
&= \frac{1+\Omega_{jj}}{2} - \sum_{k\neq j}^{g} \oint_{a_k} \omega_k(Q) \Big(A_j(P,P_0) + A_j(P_0,Q) \Big)\,\mathrm{d}Q \\
&= \frac{1+\Omega_{jj}}{2} - \sum_{k\neq j}^{g} \oint_{a_k} \omega_k(Q) A_j(P,P_0)\,\mathrm{d}Q - \sum_{k\neq j}^{g} \oint_{a_k} \omega_k(Q) A_j(P_0,Q)\,\mathrm{d}Q. \qquad (2.2.22)
\end{aligned}$$

The jth-component of the Abel map appearing in the first sum has no dependence on the variable of integration Q nor on the summation index k. Therefore,

$$\begin{aligned}
K_j(P) &= \frac{1+\Omega_{jj}}{2} - A_j(P,P_0) \sum_{k\neq j}^{g} \oint_{a_k} \omega_k(Q)\,\mathrm{d}Q - \sum_{k\neq j}^{g} \oint_{a_k} \omega_k(Q) A_j(P_0,Q)\,\mathrm{d}Q \\
&= \frac{1+\Omega_{jj}}{2} - A_j(P,P_0)(g-1) - \sum_{k\neq j}^{g} \oint_{a_k} \omega_k(Q) A_j(P_0,Q)\,\mathrm{d}Q \\
&= K_j(P_0) + (g-1) A_j(P_0,P). \qquad (2.2.23)
\end{aligned}$$

□

The primary computational benefit to using the result of Theorem 2.2.14 is that most of the work in evaluating $\boldsymbol{K}$ comes from evaluating it at a fixed place P_0. Once this is done, we only need the Abel map to determine $\boldsymbol{K}$ for all other places $P \in X$. In ABELFUNCTIONS a fixed place of the Riemann surface is automatically chosen.

The inspiration behind the algorithm for computing the RCV described in the following section comes from the following two theorems. Theorem 2.2.15 characterizes a certain class of divisors in terms of the RCV, a proof of which is found in [17].

Theorem 2.2.15. *Let $\mathcal{C}$ be a divisor on a genus g Riemann surface X of degree $2g-2$. Then $\mathcal{C}$ is a canonical divisor if and only if*

$$2\,\boldsymbol{K}(P) \equiv -\,\boldsymbol{A}(P, \mathcal{C}). \qquad (2.2.24)$$

Theorem 2.2.17 establishes a connection between the Riemann theta function and the RCV, a proof of which is also found in [17].

Definition 2.2.16. *The Riemann theta function* $\theta : J(X) \times \mathfrak{h}_{\mathrm{g}} \to \mathbb{C}$ *is defined by*

$$\theta(z, \Omega) = \sum_{n \in \mathbb{Z}^g} e^{2\pi i \left(\frac{1}{2} n \cdot \Omega n + n \cdot z\right)}. \tag{2.2.25}$$

This series converges absolutely and uniformly on compact sets in $J(X) \times \mathfrak{h}_{\mathrm{g}}$ *where* $\mathfrak{h}_{\mathrm{g}}$ *is the space of all Riemann matrices.*

Theorem 2.2.17. *Let* Ω *be the Riemann matrix associated with the Riemann surface* X *and* $P_0 \in X$ *an arbitrary place. Then a vector* $\boldsymbol{W} \in J(X)$ *satisfies*

$$\theta(\boldsymbol{W}, \Omega) = 0, \tag{2.2.26}$$

if and only if there exists a divisor $\mathcal{D} = P_1 + \cdots P_{g-1}$ *such that*

$$\boldsymbol{W} = \boldsymbol{A}(P_0, \mathcal{D}) + \boldsymbol{K}(P_0). \tag{2.2.27}$$

Note that $\mathcal{D}$ may contain a place of multiplicity greater than one. The primary requirement of $\mathcal{D}$ is that it is of degree $g-1$ and is effective. The set $\Theta := \{\boldsymbol{W} \in J(X) : \theta(\boldsymbol{W}, \Omega) = 0\}$ is known as the *theta divisor* of the Riemann surface X. It is a $(g-1)$ complex–dimensional subvariety of $J(X)$. Theorem 2.2.17 states that $\Theta = \boldsymbol{A}\left(P_0, SX^{g-1}\right) + \boldsymbol{K}(P_0)$ where SX^{g-1} is the $(g-1)$–fold symmetric product of the Riemann surface.

2.3 The Riemann Theta Function

Chapter 3

ABELFUNCTIONS

The primary result of this thesis is the Sage software library Abelfunctions [34]. The goal of Abelfunctions is to provide general and easy to use framework for computing with Abelian functions, Riemann surfaces, and algebraic curves.

Abelfunctions was inspired by the Maple software package Algcurves developed by Deconinck, van Hoeij, and Patterson [10]. The need for Abelfunctions grew out of the difficulty in keeping Algcurves up to date within Maple's proprietary software. Because of the need to resolve issues as they are encountered as well as allowing the ability to improve upon the software when new algorithms are developed, Abelfunctions was created using a completely open-source model within completely open-source software.

Abelfunctions differs from Algcurves in two main ways. First, the opportunity write a new a software package for computing with Riemann surfaces invites the opportunity to investigate better software design. The philosophy of design is first influenced by the choice of the Python/Sage language which offers features not availble in Maple. Furthermore, the use of object-oriented design offers the ability to more easily improve upon Abelfunctions in the future. Following these well-studied object-oriented design patterns [21] is in line with the reproducibility movement spreading in the computational sciences [32] which encourages the release of well-designed and well-documented code used in any scientific endeavor which makes use of computational results.

Second, the opportunity to implement the functionality of Algcurves in a different environment provided the opportunity to improve upon existing algorithms. The path from plane algebraic curve to period matrix, Abel map, and Riemann constant vector draws from a diverse set of algorithms used to realize the intermediate steps leading to these goals.

In this chapter we will take a closer look at the software design and algorithm implementation details of the package. In Section 3.1 we begin with a discussion on reproducible research and its motivating role in the creation of Abelfunctions. In Section 3.2 we give a short introduction to object-oriented design and highlight the key patterns and techniques used to improve reproducibility. Section 3.3 is the "developer's guide" where we explore the design and organization of Abelfunctions. Finally, in Section **??** we highlight the algorithmic improvements made on previous work in the field.

The Abelfunctions source code can be found on GitHub at `http://github.com/abelfunctions`.

3.1 Reproducibility and Scientific Software Design

Reproducibility is standard practice in the physical sciences. When sharing research with the scientific community one should provide sufficient detail of the experimental setup, performance metrics, and observations for a second party to repeat the experiment. Whatever the state of reproducibility is in physical science some have observed that field of computational science is lagging behind. From the generation of figures to the demonstration of newly developed algorithms many papers display these results without providing the means of generating these data: the source code.

During the development of WaveLab, Buckheit and Donoho created the following slogan,

> An article about computational science in a scientifc publication is not the scholarship itself, it is merely advertising of the scholarship. The actual scholarship is the complete software development environment and the complete set of instructions which generated the figures [7].

The [XXX][FINISH]

To read more about the reproducibility movement in computational science see [29], [32], and [33].

3.2 A Brief Look at Object-Oriented Design

Many applied mathematician write code but do not necessarily write object-oriented code. Sometimes this is a restriction imposed by the language of choice. For example, Lisp and Scala, programming languages popular in the artificial intelligence communities, are not object-oriented languages. Other times, it is due to an unawareness of its existence or utility. This section provides an introduction to object-oriented programming and design for computational scientists and why it is useful in scientific code. In particular, we will see examples where this software design tool is used to produce reusable scientific software and avoid rotting design. It is suggested that readers of this section have some programming experience, preferrably in Python or Sage.

Robert Martin succinctly addresses what goes wrong with software once it achieves non-trivial levels of complexity in a list of four symptoms of rotting design [25]. The common element across these symptoms is that rotting design adds developer time and reduces manager and customer trust.

- **Rigidity** - the tendency for software to be difficult to change. When a simple fix should take several days but instead takes weeks, managers and advisors are reluctant to let developers address non-critical issues.
- **Fragility** - the tendency for software to break, potentially in many places, every time it is changed. Fragile software creates connections between parts of code that are supposed to be logically distinct.
- **Immobility** - the inability to reuse software from parts of other projects or the same project. The more extraneous components a software component has the less potential it has to be generalized. When immobility is high the amount of work required to separate out potentially resuable parts of code is greater than the work it takes to rewrite the code entirely.
- **Viscosity** - the situation where, when making a code change, the approach that preserves the software design is more expensive than the approach that does not. Viscous

software encourages “hacks” as solutions to problems rather than coherent modifications.

Object-oriented languages, such as Python and Sage, provide the ability to form abstractions of data and functionality for the purposes of reusability and modularity. When used correctly one avoids rotting design traps. The main concepts of object-oriented programming are objects, classes, inheritance, and polymorphism.

Design patterns are methodologies for writing object-oriented code so as to avoid rotting design. A complete overview of design patterns is beyond the scope of this text. The curious reader is encouraged to read [21] for an encyclopedia of design patterns for many common design situations and [25] for a list of general strategies to avoid introducing rot into your system.

3.2.1 Objects and Classes

Objects are the basic entities in an object oriented system. They represent the combination of data and operations acting on these data. When a program is executed objects interact with each other by sending messages. Objects have two components: state and behavior. The state of an object is any data that it may encapsulate and the behavior consists of functions, often called methods, that use the state to communicate information to the user and to other objects.

Objects are instances of classes which define the state format and available behaviors. For example, a `Person` class may consist of two data fields, `name:string` and `age:int`, and a method `speak()` which returns the string `'My name is <name> and I am <age> years old.'`. A class diagram [XXX] can be used to graphically represent this class.

Two objects, `Alice` and `Bob`, are instances of the `Person` class with the `name` field defined as `'Alice'` and `'Bob'`, repectively, and given definitions for `age` as well. Note that the `speak()` method does not differ between the two objects. Instead, it uses the data defined by the objects to return the appropriate string message.

In the remainder of this section we will use a more interesting running example of building an numerical integrator package in order to motivate the use of object-oriented design principles. The implementation details will be omitted we instead focuson the software structure.

3.2.2 Inheritance

Suppose we wanted to write a piece of scientific software implementing several numerical integration techniques. The use provides a function `func` and the software package will numerically integrate the function using a chosen algorithm. If an algorithm is not provided then a default algorithm is chosen for the user.

The novice programmer might write a function along the following lines,

```
def integrate(f, method_name):
    if method_name = 'newton_cotes':
        # integrate func using the Newton Cotes method
    else if method_name = 'gaussian_quadrature':
        # integrate func using the Gaussaian Quadrature method
    else if method_name = 'clenshaw_curtis':
        # integrate func using the Clenshaw Curtis Quadrature method
    else
        # integrate using default technique (Newton Cotes)

    return integral
```

This approach would work, and a certain class of engineer may deem this acceptable, but may be difficult to work on. One practical issue is that the `integrate` function may become too long and thus increasingly difficult to debug. The developer would have to keep track of which lines of code corresponded to which numerical integration technique.

One obvious improvement is to break out each of the integration methods into a separate function.

```
def integrate(func, method_name):
    if method_name = 'newton_cotes':
        integral = integrate_newton_cotes(func)
    else if method_name = 'gaussian_quadrature':
        integral = integrate_gaussian_quadrature(func)
    else if method_name = 'clenshaw_curtis':
        integral = integrate_clenshaw_curtis(func)
    else
        integral = integrate_newton_cotes(func)

    return integral

def integrate_newton_cotes(func):
    # integrate func using the Newton Cotes method
    return integral

def integrate_gaussian_quadrature(func):
    # integrate func using the Gaussian Quadrature method
    return integral

def integrate_clenshaw_curtis(func):
    # integrate func using the Clenshaw Curtis method
    return integral
```

This is an improvement from an organizational perspective. However, imagine the following situation: a contributor wants to implement a new numerical integrator. They write a function `integate_my_technique`. But to integrate it into the primary `integrate()` function they would need to modify the source code of `integrate()`.

In this simple example this may not be an issue. However, moving beyond this toy situation reveals potential issues. If the definition of `integate()` existed in a different source file than the contributor would need to find it and add an `else..if` conditional to the growing list. An ad hoc solution would be for the developer to provide documentation of this situation.

One class of numerical integrators are of the form

$$\int_{-1}^{1} f(x)dx \approx \sum_{i=0}^{n} w_i f(x_i)$$

where the weights w_i and evaluation locations x_i are chosen based on the technique used. For example,

- **Degree 2 Newton-Cotes (Trapezoid Rule)** - $w_i = 1$ if $i = 0$ or $i = n$ and $w_i = 2$ otherwise, and $x_i = -1 + 2i/n$.
- **Gaussian Quadrature** - $w_i = 1/\sqrt{1 - x_i^2}$

```
class NumericalIntegrator(object):
    def __init__(self, n):
        self.n = n

    def weights(self):
        raise NotImplementedError()

    def integrate(self, func):
        w, x = self.weights_and_locations()
        return sum(w[i]*func(x[i]) for i in range(n))


class NewtonCotesQuadrature(NumericalIntegrator):
    def weights(self):
        # return the newton coates quadrature weights, w
        return w

class GaussianQuadrature(NumericalIntegrator):
    def weights(self):
        # return the gaussian quadrature weights, w
        return w
```

```
class ClenshawCurtisQuadrature(NumericalIntegrator):
    def weights(self):
        # return the clenshaw curtis quadrature weights, w
        return w
```

Such a system is easily extendible: if a user or developer devises a new numerical integration technique they need only to create a new subclass of `NumericalQuadrature` and pass instances of that subclass to the appropriate functions.

This design avoids the four rotting design symptoms:

- **Rigidity** - adding a new integrator does not require changing code in multiple locations.
- **Fragility** - although this example is relatively simple, as long as the new integrator adheres to the data contract of `NumericalQuadrature.integrate()` it will not break the software. Furthermore, the design makes it easy to test the new integrator directly.
- **Immobility** - the integration logic is separated from its uses in other parts of the code.
- **Viscosity** - separating the integration logic from the rest of the code allows the developer to isolate and better control the interface to these components. `NumericalQuadrature` makes it clear how functions are integrated.

3.2.3 Composition

3.2.4 Strategy Design Pattern

3.2.5 Dependency Inversion

3.3 A Tour of Abelfunctions

Abelfunctions several top-level functions and objects. The user takes advantage of the Abelfunctions functionality primarily through these objects.

- `RiemannSurface` - principle object, defines a Riemann surface given a bivariate Sage polynomial.

- `AbelMap` - defines the Abel map function from one place on a Riemann surface to another. Can accept divisors as well.
- `RiemannConstantVector` - defined the Riemann constant vector function as a function of an arbitrary place on a Riemann surface.
- `RiemannTheta` - the Riemann theta function as a function from $\mathbb{C}^g \times \mathfrak{h}_g$ to $\mathbb{C}$.

We begin by importing this functionality and constructing a Riemann surface corresponding to the plane algebraic curve,

$$C : x^2y^3 - x^4 + 1 = 0, \quad x, y \in \mathbb{C}^* .$$

```
sage: from abelfunctions import *  # import the main Abelfunctions functionality
sage: R.<x,y> = QQ[]  # construct a Sage polynomial ring and the above curve
sage: f = y**3 - 2*x**3*y + x**7
sage: X = RiemannSurface(f); X  # construct the corresponding Riemann surface
Riemann surface defined by f = x^7 - 2*x^3*y + y^3
```

The genus of the curve is determined using the singularity structure. In this case, the Riemann surface is of genus four.

```
sage: g = X.genus(); g
4
```

We can verify this with the singularity data.

```
sage: from abelfunctions.singularities import singularities
sage: for point, (m, delta, r) in singularities(f):
 ...:     print point, '\tdelta invariant =', delta
(0, 1, 0)    delta invariant = 2
sage: d = f.total_degree()
sage: g = (d-1)*(d-2)/2 - 2  # (-2) from the delta inv. above
sage: g
4
```

3.3.1 Places and Divisors

3.4 Algorithms

This section is the algorithmic equivalent to Section 2.2. Here we present the algorithms used to computationally realize Riemann surfaces, the Abel map, and the Riemann theta function.

3.4.1 Puiseux Series

The algorithm used in Abelfunctions for computing truncations of Puiseux series expansions is based on that of Duval [16]. The main ingredient of the algorithm are Newton polygons of algebraic curves. We give a brief outline of the process here.

- The goal of the algorithm is to compute a list of tuples $\pi = (\tau_1, \tau_2, \dots, \tau_R)$ where $\tau_i = (q_i, \mu_i, m_i, \beta_i, \eta_i)$. These tuples define the relations

  $$\begin{aligned} X_{i-1} &= \mu_i X_i^{q_i}, \\ Y_{i-1} &= (\beta_i + \eta_i Y_i) X_i^{m_i}, \end{aligned}$$

  where $i = 1, \dots, R$. To obtain the desired Puiseux series we set $x = X_0$, $y = Y_0$, and $t = X_R$ and eliminate the intermediate variables $X_1, \dots, X_{R-1}$ and $Y_1, \dots, Y_{R-1}$.

- The first set of τ_i computed are those representing the *singular part* of P, that is, the part of the series if the Puiseux series has a ramification index $|e_j| > 1$. This is done using the Newton polygon method. The output of this stage of the algorithm provides enough information to distinguish the Puiseux series expansions at $x = \alpha$.

- Finally the algorithm computes the *regular* terms of the Puiseux series using a standard Taylor series techniques.

3.4.2 Singularities

First we determine the finite singularities of $C : f(x, y) = 0$. Let $R(x)$ be the resultant of $f(x, y)$ and $\partial_y f(x, y)$ [22]. We compute the roots

$$\begin{aligned} S &= \{x \in \mathbb{C} \mid R(x) = 0, \partial_x R(x) = 0\} \\ &= \{x_1, \ldots, x_s\}. \end{aligned}$$

For each $x_j \in S$ we compute the y-roots $\{y_{j1}, \ldots, y_{jd}\}$ where $d = \deg_y f$. The places (x_j, y_{jk}), $j = 1, \ldots, s$, $k = 1, \ldots, d$ satisfy

$$f(x_j, y_{jk}) = 0, \quad \partial_y f(x_j, y_{jk}) = 0,$$

by the definition of the resolvent. Therefore, the singular places (x_k, y_{jk}) are those that satisfy

$$\partial_x f(x_j, y_{jk}) = 0.$$

By definition, these remaining places are the finite singular points of the curve C. For the infinite case we use the projection of the curve on the line at infinity. (See Section 2.1.3.)

The methods used to compute the branching number, multiplicity, and delta invariant of a singularity rely on examining the leading order behavior of the Puiseux series expansions such that $P_j(0) = (\alpha, \beta)$. For the details of these methods see [10]. In brief the branching number R of singularity is computed using the above formula

$$R \quad = \sum_{\substack{j \\ P_j(0)=(\alpha,\beta)}} |e_j|.$$

The multiplicity is the sum of the minimum of e_j and n_{jN} over all Puiseux series P_j such that $P_j(0) = (\alpha, \beta)$ where $\beta_{jN} t^{n_{jN}}$ is the first non-zero, non-constant term appearing the in y_j. The delta invariant is equal to

$$\delta = \frac{1}{2} \sum_{j=1}^{m} r_j \mathrm{Int}_{P_j} - r_j + 1,$$

where

$$\mathrm{Int}_{P_j} = \sum_{k=1^d, k \neq j} \mathrm{val}_x\big(y_j(x) - \tilde{y}_k(x)\big),$$

with $\mathrm{val}_x(g(x))$ equal to the lowest exponent of x appearing in $g(x)$, and $y_j(x)$ is the y-part of P_j when solving for $t = t(x)$. The sum appearing in Int_{P_j} is taken over *all* Puiseux series expansions P_k at $x = \alpha$, not just the ones with $P_k(0) = (\alpha, \beta)$.

3.4.3 Analytic Continuation

To compute a path γ on a Riemann surface C we provide a continuous x-path $x_\gamma(t)$ and an initial y-value y_0 as input and we wish to receive the resulting y-path $y_\gamma(t)$ as output. Analytic continuation of y_0 along γ is a fundamental operation in Abelfunctions since evaluation of and integration along paths is done frequently. Therefore, it is important to make the construction and evaluation along paths as fast and efficient as possible.

We use numerical methods to estimate values along $y_\gamma(t)$. In general, the problem is phrased as given $\gamma(t_i) = (x_i, y_i)$ as well as some later $t_{i+1} = t_i + \Delta t$ and $x_{i+1} = x_\gamma(t_{i+1})$ determine the value $y_{i+1} = y_\gamma(t_{i+1})$.

A first and natural approach to solving this problem is to use a root finder. Given x_{i+1} we numerically or symbolically solve the equation

$$f(x_{i+1}, y) = 0.$$

This produces n y-roots $y_{i+1,1}, \ldots, y_{i+1,d}$ over x_{i+1}. However, even if one finds an effective and fast method for doing this with arbitrary degree polynomials f, the main problem with this approach is determining which $y_{i+1,k}$ is equal to the desired root y_{i+1}. One could argue that the desired root is the one minimizing $|y_{i+1,k} - y_i|$ (the root closest to the previous y-root) but it is conceivable that this closest can change as a function of Δt, especially if Δt is too large.

Another approach could be to use Newton iteration: given x_{i+1} and an *initial guess* y_i at x_i use Newton iteration on the function $g(y) = f(x_{i+1}, y)$ to determine y_{i+1}. However, this approach suffers from the same problem, namely, the root $y_{i+1,k}$ produced by Newton iteration may change as a function of Δt. Too large of a Δt may result in *branch jumping*, where we converge to the incorrect y-root. Too small of a Δt gives an inefficient numerical algorithm. Further, the definition of "small enough" may change as a function of the curve C and the x-points x_i and x_{i+1}.

To solve the problem of selecting an appropriate Δt we use Smale's α-theory [**?**]. The purpose of Smale's α-theory is to answer the following questions about $g(y) = f(x_{i+1}, y)$ for some finite set of points $Y \subset \mathbb{C}$:

1. From which points in Y will Newton's method converge quadratically to some solution to g?
2. From which points in Y will Newton's method converge quadratically to distinct solutions to g?
3. If g is real ($\bar{g} = g$), from which points of Y will Newton's method converge quadratically to real solutions to g?

See [24] for an excellent summary of Smale's α-theory. Using the notation of Hauenstein and Sottile, we outline the analytic continuation algorithm here.

1. Assume we know the *y-fibre* $y_i = \{y_{i,1}, \ldots, y_{i,d}\}$ of $g_i(y) := f(x_i, y) = 0$. Fix some initial Δt and let $x_{i+1} = x_\gamma(t_i + \Delta t)$. We wish to compute the y-fibre y_{i+1} of $g_{i+1}(y) := f(x_{i+1}, y)$ such that each element $y_{i+1,j}$ is the analytic continuation of $y_{i,j}$ to x_{i+1}.
2. We first determine if the y-fibre y_i is an *approximate solution* to $g_{i+1}(y) = 0$. Does each element of y_i lie in the quadratic convergence region of Newton's method on g_{i+1}?

   If not, return to Step (1) with $\Delta t \mapsto \Delta t/2$.
3. Next, determine if the approximate solutions $y_{i,j}$ will converge to distinct associated solutions $y_{i+1,j}$: will the approximate solutions jump branches or stay on their respective branches?

   If not, return to Step (1) with $\Delta t \mapsto \Delta t/2$.

4. The y-fibre satisfies the necessary conditions for Newton iteration to converge to the appropriate analytic continuations $y_{i+1,j}$ at x_{i+1}. Newton iterate and output this solution y-fibre.

Note that this algorithm requires analytically continuing all of the y-roots along an x-path in the complex plane since we cannot determine an appropriate step size for continuing a given root without knowing the locations of the other roots. Although this impacts the performance of the algorithm since we have to perform d sets of Newton iterations at each step, Smale's α-theory provides a rigorous method for determining an appropriate step size.

3.4.4 Monodromy

The algorithm implemented in Abelfunctions for computing the monodromy group of a curve is based on the one described in [20]. Due to the technical nature of the algorithm only a summary is provided here.

- We require that the monodromy paths constructed stay sufficiently far from the branch points due to the numerical accuracy of Newton's method when used in the analytic continuation process. For each branch point, b_i we let

  $$\rho_i = \min_{\substack{j=1,\ldots,n \\ j\neq i}} |b_i - b_j|.$$

  The minimal distance that any path x_γ be from the branch point b_i is

  $$R_i = \frac{\rho_i \kappa}{2}$$

  where $\kappa \in (0, 1]$ is a chosen relaxation factor. The implementation of this algorithm in Abelfunctions uses $\kappa = 3/5$.

- Let $b = b_i$ be the branch point where $\operatorname{Re} b_i < \operatorname{Re} b_j$ for all $j \neq i$. The point b is referred to as the *base branch point*. Choose the base point α_0 to be the point $\alpha_0 = b - R_b$, the point on the minimal distance circle encircling b.

- Order the remaining branch points $\{b_j\}_{j\neq i}$ by increasing argument with b. This ordering determines the ordering of the monodromy paths x_{γ_j}.
- Construct a complete graph G with the branch points $b_1, \ldots, b_n$ as nodes and compute the minimal spanning tree T of this graph with b as the parent node.
- Using line segments and semi-circles, construct the path x_{γ_j} encircling b_j once in the positive direction by starting at the base point, following the minimal spanning tree to b_j and using semicircles to traverse over or under the branch points along the way depending on the ordering. That is, the path x_{γ_j} should be constructed in such a way so that the branch points $b_1, \ldots, b_{j-1}$ lie below the path.
- Fix an ordering of the base fibre $(y_0, \ldots, y_{d-1})$ above α_0 and analytically continue around each x_{γ_j} to determine the permutations π_j.

3.4.5 Homology

Tretkoff and Tretkoff [36] provide an algorithm for determining a canonical cycle basis for $H_1(X, \mathbb{Z})$ given the monodromy group of a curve. We omit the details of the algorithm here. At its core, it computes a graph which encodes how to travel from the *base place* of X, chosen to be

$$P_0 = (\alpha_0, \beta_0)$$

where α_0 is the base point of the monodromy group of the curve and $\beta_0 = y_0$ is a fixed root lying above $x = \alpha_0$, to the other places (α_0, y_i) lying above $x = \alpha_0$ via traversal around one or more branch points. (These places are sometimes called the *sheets* of the surface X with the ith sheet referring to the place (α_0, y_i)). A minimal spanning tree of this graph is computed where each removed edge corresponds to a cycle in $H_1(X, \mathbb{Z})$. The is because the addition of an edge to the minimal spanning tree forms a cycle in the graph. This graph cycle in turn represents a possibly a non-zero cycle on X. A separate part of the algorithm is then used to compute a canonical basis of cycles by taking appropriate linear combinations of these intermediate cycles.

3.4.6 Holomorphic and Meromorphic Differentials

One condition on the p_k's is immediately apparent: to preserve holomorphicity p_k must have a zero, with sufficient multiplicity, at the places $P = (\alpha, \beta)$ where $\partial_y f(x,y)$ vanishes. More precisely, Noether showed that if a singular place P has multiplicity m_P then P_k must have a zero of order at least $m_P - 1$ at P [28].

The technique we use to determine the adjoint polynomials p_k uses a theorem of Mñuk relying on computing an integral basis for the algebraic function field of the curve [26]. Let $A(C)$ be the *coordinate ring*

$$A(C) = \mathbb{C}[x,y]/(f)$$

of the curve $C : f(x,y) = 0$. The coordinate ring can be though of as the ring of all functions $g \in \mathbb{C}[x,y]$ vanishing on the curve f. Note that $A(C)$ is a subset of the *algebraic function field* $\mathbb{C}(x,y)$.

Given a ring R and a field S such that $R \subset S$, the *integral closure* $\bar{R}$ *of* R *in* S is the ring of all elements $s \in S$ such that

$$s^n + r_{n-1}s^{n-1} + \cdots + r_1 s + r_0 = 0,$$

for some choice of $n > 0$ and $r_0, \ldots, r_{n-1} \in R$. That is $\bar{R}$ consists of all elements in S satisfying some monic polynomial equation with coefficients in R. Here, we consider the integral closure $\overline{A(C)}$ of $A(C)$ in $\mathbb{C}(x,y)$.

Again, we wish to find the set of all adjoint polynomials of C. By [26], these are the set of all $p \in \mathbb{C}[x,y]$ such that

$$\overline{A(C)}p(x,y) \subset \mathbb{C}[x,y].$$

That is, all of the polynomials p such that every element of $\overline{A(C)}$, when multiplied by p and reduced modulo $f(x,y)$, results in a polynomial. Now, $\overline{A(C)}$ is a finite extension of $A(C)$. Therefore,

$$\overline{A(C)} = \beta_1 A(C) + \cdots + \beta_m A(C), \quad \beta_k \in \mathbb{C}(x,y).$$

for an appropriate choice of β_k's in $\mathbb{C}(x,y)$. The set $\{\beta_1,\dots,\beta_m\}$ is called the *integral basis* of the integral closure of the coordinate ring. Thus, finding the set of adjoint polynomials is equivalent to finding polynomials $p \in \mathbb{C}[x,y]$ such that

$$\beta_k(x,y)p(x,y) \in \mathbb{C}[x,y], \quad \forall j = 1,\dots,m.$$

To compute the adjoint polynomials we write

$$p(x,y) = \sum_{i+j\leq d-3} c_{ij}x^i y^j$$

where $d = \deg_y f$. We compute an integral basis $\{\beta_1,\dots,\beta_m\}$ for $\overline{A(C)}$ using the algorithm of van Hoeij [38]. The requirement that

$$\beta_k(x,y)\sum_{i+j\leq d-3} c_{ij}x^i y^j \in \mathbb{C}[x,y]$$

imposes a number of conditions on the coefficients c_{ij} appearing in the expression for $p(x,y)$. The set of all possible c_{ij}'s satisfying these conditions for every $\beta_k, k=1,\dots,m$ gives us the adjoint polynomials we need.

3.4.7 Computing on the Period Lattice

To compute A_{ij}, B_{ij} we first a method for numerically integrating holomorphic differentials over any given path $\gamma \subset C$. We only consider finite paths on the curve: paths with finite x- and y-components. Any such path can be parameterized by some parameter $t \in [0,1]$. Let

$$\gamma : [0,1] \to C, \quad \gamma(t) = \Big(x_\gamma(t), y_\gamma\big(x_\gamma(t)\big)\Big).$$

(Recall that we treat y as the dependent variable in $f(x,y)=0$.) Given this parameterization, we compute the integral of a holomorphic differential ω. Letting x and y represent the local coordinates in $\mathbb{C}^2$ we have

$$\begin{aligned}\int_\gamma \omega &= \int_\gamma \omega\big(x, y(x)\big)dx \\ &= \int_0^1 \omega\Big(x_\gamma(t), y_\gamma\big(x_\gamma(t)\big)\Big)\frac{dx_\gamma}{dt}(t)dt. \end{aligned} \tag{3.4.1}$$

In Abelfunctions we construct paths $\gamma \subset C$ where x_γ either parameterizes a line from some $z_0 \in \mathbb{C}$ to $z_1 \in \mathbb{C}$

$$x_\gamma(t) = z_0(1-t) + z_1 t,$$

or arcs on a circle of radius R with center $w \in \mathbb{C}$

$$x_\gamma(t) = w + Re^{i(\theta + t\Delta\theta)}$$

where θ is the starting angle and $\theta + \Delta\theta$ is the ending angle on the circle. We use the analytic continuation methods described in Section **??** to compute $y_\gamma(x_\gamma(t))$. Finally, to compute the integral in Equation (3.4.1) we use a numerical integrator of choice. Abelfunctions allows one to use any numerical integrator provided by the `scipy` Python package that can integrate complex-valued functions. The Romberg method [1] is chosen by default.

3.4.8 The Abel Map

3.4.9 Riemann Constant Vector

In this section we present an algorithm for computing the Riemann constant vector as well as a demonstration of its implementation in ABELFUNCTIONS. First we present an overview of and the motivation behind the algorithm. We describe the two primary components of the algorithm later in this section.

Theorem 2.2.15 suggests an approach to computing the RCV provided we can compute a canonical divisor of the Riemann surface. However, even with such a divisor the theorem only makes a statement about the value of $2\,\boldsymbol{K}(P_0)$. That is, one would like to say

$$\boldsymbol{K}(P_0) \equiv -\tfrac{1}{2}\,\boldsymbol{A}(P_0, \mathcal{C}), \tag{3.4.2}$$

but division is not unique in this equivalence class. In general, there are 2^{2g} *half-lattice vectors* $\boldsymbol{h} \in \frac{1}{2}\Lambda$ such that $\boldsymbol{K}(P_0) \equiv \boldsymbol{h} - \frac{1}{2}\,\boldsymbol{A}(P_0, \mathcal{C})$. Therefore, a second objective is to find an appropriate half-lattice vector.

Algorithm 3.4.1 `riemann_constant_vector`

```
Require: Riemann surface $X$ given by the desingularization and compactification of complex plane algebraic curve $C : f(x, y) = 0$
Require: place $P \in X$
Ensure: Riemann constant vector $\boldsymbol{K}(P)$
1: compute the Riemann matrix $\Omega$
2: $\mathcal{C} \leftarrow$ canonical_divisor()          ▷ Algorithm 3.4.2
3: $\boldsymbol{h} \leftarrow$ half_lattice_vector()      ▷ Algorithm 3.4.4
4: $\boldsymbol{K}_0 \leftarrow \boldsymbol{h} - \frac{1}{2}\boldsymbol{A}(P_0, \mathcal{C}) \mod \Lambda$
5: $\boldsymbol{K} \leftarrow \boldsymbol{K}_0 + (g-1)\,\boldsymbol{A}(P_0, P)$
6: return $\boldsymbol{K}$
```

The rest of this section presents in detail the subroutines `canonical_divisor` and `half_lattice_vector` which provide the necessary remaining ingredients for the above algorithm.

Computing a Canonical Divisor

Determining the zeros and poles of a meromorphic one-form

$$\nu = \frac{p(x,y)}{q(x,y)}\,\mathrm{d}x \tag{3.4.3}$$

is not as straightforward as finding the roots of the polynomials p and q. One challenge comes from analyzing the local behavior of $\mathrm{d}x$. Furthermore, it may occur that the numerator and denominator have the same order of vanishing at some place $P \in X$ in which case P is neither a root nor pole of ν.

Let $P \in X$ be a place with Puiseux series representation $(x_P(t), y_P(t))$ where t is a local parameter as given in Definition **??**. A necessary condition for P to be a root or pole of ν is that

$$p\big(x_P(t), y_P(t)\big)\Big|_{t=0} = 0, \qquad q\big(x_P(t), y_P(t)\big)\Big|_{t=0} = 0, \qquad \text{or} \qquad \frac{\mathrm{d}x_P}{\mathrm{d}t}\big(0\big) = 0. \tag{3.4.4}$$

In particular, to determine if $P \in (\nu)_{\text{val}}$ we substitute the Puiseux series representation into ν and expand as a Laurent series in t:

$$\begin{aligned}\nu\Big|_P &= \frac{p\big(x_P(t), y_P(t)\big)}{q\big(x_P(t), y_P(t)\big)}\,\mathrm{d}x_P(t) \\ &= \frac{p(t)}{q(t)} x_P'(t)\,\mathrm{d}t \\ &= \Big(ct^{\mathrm{val}(\nu,P)} + \cdots\Big)\,\mathrm{d}t, \end{aligned} \tag{3.4.5}$$

where $\mathrm{val}(\nu, P)$ is the leading order behavior of ν at P. This gives us a test for determining if P is a member of the set of places appearing in $(\nu)_{\text{val}}$: if $\mathrm{val}(\nu, P) < 0$ then P is a pole, if $\mathrm{val}(\nu, P) > 0$ then P is a zero, otherwise P does not appear in the valuation divisor of ν. The multiplicity of the zero or pole is equal to $|\mathrm{val}(\nu, P)|$.

With the above membership test what remains is to construct a set of places $\mathcal{P}_\nu$ guaranteed to contain the places appearing in $(\nu)_{\text{val}}$. We obtain the valuation divisor by applying the membership test to each $P \in \mathcal{P}_\nu$. Consider the resultant $R(f, p)(x)$ of f and p with respect to y. By definition, the roots of R are the points $\alpha \in \mathbb{C}$ such that

$$f(\alpha, y) = 0 \quad \text{and} \quad p(\alpha, y) = 0 \tag{3.4.6}$$

have simultaneous solutions. Therefore, for a place P to be a zero of p it must be the case that the x-projection of P, $x_P(0)$, is a root of the resultant R. Similarly, for P to be a zero of q its x-projection $x_P(0)$ must be a root of the resultant $R(f, q)(x)$. We also need to include the places P which cause $\mathrm{d}x$ to vanish. This occurs when $(\mathrm{d}x_P/\mathrm{d}t)(0) = x_P'(0) = 0$. That is, when P lies above a branch point of f.

Define the sets

$$\begin{aligned} \mathcal{X}_\nu^{(1)} &= \{\alpha \in \mathbb{C} \mid R(f, p)(\alpha) = 0\}, \\ \mathcal{X}_\nu^{(2)} &= \{\alpha \in \mathbb{C} \mid R(f, q)(\alpha) = 0\}, \\ \text{and} \quad \mathcal{X}_\nu^{(3)} &= \{\alpha \in \mathbb{C} \mid \alpha \text{ is a branch point of } f\}. \end{aligned} \tag{3.4.7}$$

Since the representation of the one-form in (3.4.3) only captures its affine behavior it is necessary to examine its behavior at all places lying above $x = \infty$. Define

$$\mathcal{X}_\nu = \mathcal{X}_\nu^{(1)} \cup \mathcal{X}_\nu^{(2)} \cup \mathcal{X}_\nu^{(3)} \cup \{\infty\}. \tag{3.4.8}$$

This consists of all x-points above which there may be a place P where ν vanishes. That is, the only places we need to check are those with x-projections in $\mathcal{X}_\nu$. Therefore, the set

$$\mathcal{P}_\nu = \{P \in X \mid x_P(0) \in \mathcal{X}_\nu\}, \tag{3.4.9}$$

is guaranteed to contain the places appearing in $(\nu)_{\text{val}}$. The Puiseux algorithm is well-designed to compute this set.

This procedure is simplified when computing the valuation divisor of an Abelian differential of the first kind. Recall that every such differential is trivially a meromorphic one-form and therefore can be used to compute a canonical divisor. We could use one of the normalized basis elements $\{\omega_1, \ldots, \omega_g\}$ to obtain a canonical divisor on X but it is preferred to use the non-normalized differentials $\{\tilde{\omega}_1, \ldots, \tilde{\omega}_g\}$ returned by ABELFUNCTIONS. This is done for several performance-related reasons:

- We already compute these differentials for the purposes of determining the period matrix of X as well as in defining the Abel map.
- Fewer resolvent sets need to be determined. The denominator of every Abelian differential of the first kind is $\partial_y f(x, y)$ [6, 28] so one can compute the resolvent set of f with $\partial_y f$ once and use the results for any given basis element $\tilde{\omega} = \tilde{\omega}_i$. This particular resolvent set consists of the discriminant points of f and is already used in the period matrix calculations.
- The non-normalized differentials usually have simple, often monomial, numerators making the set $\mathcal{X}_{\tilde{\omega}}^{(1)}$ easier to compute and have smaller cardinality.
- The set of P such that $x_P(0)$ is a branch point of f is contained in the set of discriminant points of f. Therefore, the computation of the set $\mathcal{X}_{\tilde{\omega}}^{(3)}$ is a redundant calculation and is omitted.

- In general, the valuation divisors of Abelian differentials of the first kind consist of fewer distinct places. The degree of every canonical divisor is $2g-2$. Therefore, there must always be $2g-2$ more zeros than poles, counting multiplicities. Since Abelian differentials of the first kind have no poles, no negative degree places appear in the valuation divisor thus minimizing the total number of places to check.

- Algorithm 3.4.2 iteratively checks each $P \in \mathcal{P}_{\tilde{\omega}}$ for membership in the set of places in $\mathcal{C} = (\tilde{\omega})_{\text{val}}$. By using Abelian differentials of the first kind we can terminate this procedure the moment $\deg \mathcal{C}$ reaches $2g-2$ since each $P \in \mathcal{P}_{\tilde{\omega}}$ contributes a non-negative amount to the degree. For this reason, we distinguish between the set of $\mathcal{X}_{\tilde{\omega}}$ and corresponding places $\mathcal{P}_{\tilde{\omega}}$ in order to avoid unnecessarily computing Puiseux series expansions.

An algorithm for computing the valuation divisor of an Abelian differential of the first kind is given below.

Algorithm 3.4.2 `canonical_divisor` - canonical divisor of a Riemann surface

```
Require: Riemann surface $X$ given by the desingularization and compactification of complex plane algebraic curve $C : f(x,y) = 0$
Require: an Abelian differential of the first kind $\tilde{\omega} = p(x,y)/\partial_y f(x,y)\,\mathrm{d}x$ on $X$
Ensure: canonical divisor $\mathcal{C} = (\tilde{\omega})_{\mathrm{val}}$
 1: $\mathcal{C} \leftarrow$ zero divisor
 2: $\mathcal{X}_{\tilde{\omega}}^{(1)} \leftarrow$ roots of resolvent $R(f,p)(x) = 0$
 3: $\mathcal{X}_{\tilde{\omega}}^{(2)} \leftarrow$ discriminant points of $f$
 4: $\mathcal{X}_{\tilde{\omega}} \leftarrow \mathcal{X}_{\tilde{\omega}}^{(1)} \cup \mathcal{X}_{\tilde{\omega}}^{(2)} \cup \{\infty\}$
 5: for $\alpha \in \mathcal{X}_{\tilde{\omega}}$ do
 6:     $\mathcal{P}_{\tilde{\omega}}^{\alpha} \leftarrow \{P \in X \mid x_P(0) = \alpha\}$
 7:     for $P \in \mathcal{P}_{\tilde{\omega}}^{\alpha}$ do
 8:         $n \leftarrow \mathrm{val}\,(\tilde{\omega}, P)$
 9:         $\mathcal{C} \leftarrow \mathcal{C} + nP$
10:         if $\deg \mathcal{C} = 2g - 2$ then
11:             return $\mathcal{C}$
12:         end if
13:     end for
14: end for
15: raise error(“Not enough places found.”)
```

Some notes about the algorithm:

- Only the leading order behavior of the Puiseux series of each place $P \in \mathcal{P}_{\omega}^{\alpha}$ determining $\mathrm{val}(\tilde{\omega}, P)$ is needed implying that computing only the “singular part” of these expansions using the method of Duval is sufficient [16].
- Since any Abelian differential of the first kind is sufficient for computing a canonical divisor we can choose the basis element $\tilde{\omega}_i$ with lowest total degree numerator $p =$

$p(x, y)$ to reduce the number of places to check and amount of symbolic arithmetic to perform.

- Algorithm 3.4.2 terminates once the target degree is met and will spend no further effort computing places and valuations. Since the numerators of $\tilde{\omega}_i$ are often monomial, a significant gain in efficiency is observed when $\mathcal{X}_{\tilde{\omega}}$ is ordered such that places over $x \in \{0, \infty\}$ are checked first. For testing purposes, the algorithm can be modified to verify that $\operatorname{val}(\tilde{\omega}, P) = 0$ for all remaining $P \in \mathcal{P}_{\tilde{\omega}}$ after the degree requirement is met.
- If the main loop in Algorithm 3.4.2 terminates before the requisite degree is achieved then an error is reported. This is included as an additional check for the algorithm. Yet another test is to verify that $\deg(\tilde{\omega}_i)_{\text{val}} = 2g - 2$ for all basis elements $\tilde{\omega}_i$. Because this is an expensive calculation it is not performed by default. Instead it is relegated to the ABELFUNCTIONS test suite.

Example 3.4.3. For each non-normalized Abelian differential of the first kind from Example **??**,

$$\tilde{\omega}_1 = \frac{\mathrm{d}x}{3x^2y^2}, \quad \tilde{\omega}_2 = \frac{x\,\mathrm{d}x}{3x^2y^2}, \quad \tilde{\omega}_3 = \frac{xy\,\mathrm{d}x}{3x^2y^2}, \quad \tilde{\omega}_4 = \frac{x^2\,\mathrm{d}x}{3x^2y^2}, \tag{3.4.10}$$

we compute its corresponding canonical divisor. First, we verify that

$$(\tilde{\omega}_1)_{\text{val}} = 6P_{x=\infty} \qquad \text{where} \qquad P_{x=\infty} = \left(t^{-3}, t^{-2} + O\left(t^2\right)\right). \tag{3.4.11}$$

That is, the valuation divisor consists of the single place, $P_{x=\infty}$, of multiplicity six.

```
sage: C0 = omega[0].valuation_divisor()
sage: for place,multiplicity in C0:
....:     print multiplicity, place
....:
6 (t**(-3), t**(-2) + O(t**2))
sage: C0.degree
6
```

On the other hand, $(\tilde{\omega}_2)_{\text{val}}$ consists of two distinct places each of multiplicity three:

$$(\tilde{\omega}_2)_{\text{val}} = 3P_{x=\infty} + 3P_{x=0} \qquad \text{where} \qquad P_{x=0} = \left(-t^3, -t^{-2} + O(t^2)\right). \tag{3.4.12}$$

```
sage: C1 = omega[1].valuation_divisor()
sage: for place,multiplicity in C1:
....:     print multiplicity, place
....:
3 (t**(-3), t**(-2) + O(t**2))
3 (-t**3, -1/t**2 + O(t**2))
sage: C1.degree
6
```

For the canonical divisor obtained from $(\tilde{\omega}_3)_{\text{val}}$ we have

$$\mathcal{X}_{\tilde{\omega}_3} = \{0, \infty, 1, -1, i, -i\}, \tag{3.4.13}$$

which consists of the discriminant points of f and the point at infinity. The divisor $(\tilde{\omega}_3)_{\text{val}}$ happens to have non-zero valuation at the places lying above each of these points:

$$(\tilde{\omega}_3)_{\text{val}} = \sum_{\alpha \in \mathcal{X}_{\tilde{\omega}_3}} P_{x=\alpha}. \tag{3.4.14}$$

```
sage: C2 = omega[2].valuation_divisor()
sage: for place,multiplicity in C2:
....:     print multiplicity, place
....:
1 (-t**3, -1/t**2 + O(t**2))
1 (-t**3/4 - 1, t + O(t**2))
1 (-t**3*RootOf(_x**2 + 1, 0)/4 + RootOf(_x**2 + 1, 0), t + O(t**2))
1 (t**(-3), t**(-2) + O(t**2))
1 (t**3/4 + 1, t + O(t**2))
1 (-t**3*RootOf(_x**2 + 1, 1)/4 + RootOf(_x**2 + 1, 1), t + O(t**2))
sage: C2.degree
6
```

Finally, we verify that $(\tilde{\omega}_4)_{\text{val}} = 6P_{x=0}$.

```
sage: C3 = omega[3].valuation_divisor()
sage: for place,multiplicity in C3:
....:     print multiplicity, place
....:
6 (-t**3, -1/t**2 + O(t**2))
sage: C3.degree
6
```

Each of these canonical divisors satisfy the degree requirement $\deg \mathcal{C} = 2g-2 = 6$.

3.4.10 Computing a Half-Lattice Vector

Now that we have a canonical divisor $\mathcal{C}$ it remains to determine $\boldsymbol{K}(P_0)$ knowing that $2\,\boldsymbol{K}(P_0) \equiv -\boldsymbol{A}(P_0,\mathcal{C})$. For now, consider $\boldsymbol{K}(P_0)$ and $\boldsymbol{A}(P_0,\mathcal{C})$ to be vectors in $\mathbb{C}^g$ and set $\boldsymbol{K}_0 := \boldsymbol{K}(P_0)$ and $\boldsymbol{A}_0^{\mathcal{C}} := \boldsymbol{A}(P_0,\mathcal{C})$, for notational convenience. In $\mathbb{C}^g$ we have

$$2\,\boldsymbol{K}_0 + \boldsymbol{A}_0^{\mathcal{C}} = \boldsymbol{\lambda}, \tag{3.4.15}$$

where $\boldsymbol{\lambda} \in \mathbb{C}^g$ is unknown and $\boldsymbol{\lambda} \equiv \boldsymbol{0} \bmod \Lambda$, *i.e.* the vector $\boldsymbol{\lambda}$ is one of the 2^{2g} lattice vectors lying in the fundamental region of Λ. Division by two is now legal: setting $\boldsymbol{h} = \boldsymbol{\lambda}/2$ yields

$$\boldsymbol{K}_0 = \boldsymbol{h} - \tfrac{1}{2}\,\boldsymbol{A}_0^{\mathcal{C}}\,. \tag{3.4.16}$$

Reducing this expression modulo Λ gives the corresponding equivalence in $J(X)$

$$\boldsymbol{K}_0 \equiv \boldsymbol{h} - \tfrac{1}{2}\,\boldsymbol{A}_0^{\mathcal{C}}, \tag{3.4.17}$$

where the half-lattice vector $\boldsymbol{h}$ is unknown.

To determine which of the 2^{2g} half-lattice vectors $\boldsymbol{h}_j, j = 1,\ldots,2^{2g}$ is the correct half-lattice vector we use Theorem 2.2.17. The theorem requires a degree $g-1$ effective divisor. Consider the divisor

$$\mathcal{D} = (g-1)P_0. \tag{3.4.18}$$

Then

$$\begin{aligned}\theta\big(\boldsymbol{A}(P_0,\mathcal{D})+\boldsymbol{K}(P_0),\Omega\big) &= \theta\Big(\boldsymbol{A}\left(P_0,(g-1)P_0\right)+\boldsymbol{K}(P_0),\Omega\Big)\\ &= \theta\big(\mathbf{0}+\boldsymbol{K}(P_0),\Omega\big)\\ &= \theta\big(\boldsymbol{K}(P_0),\Omega\big)=0. \end{aligned}\tag{3.4.19}$$

Therefore, it is necessary that

$$\theta\left(\boldsymbol{h}_j-\tfrac{1}{2}\boldsymbol{A}_0^{\mathcal{C}},\Omega\right)=0 \tag{3.4.20}$$

for *at least* one of the 2^{2g} half lattice vectors $\boldsymbol{h}_j$. The choice of divisor $\mathcal{D}$ above simplifies the computations. However, any appropriate divisor can be used in conjunction with Theorem 2.2.17 to obtain the RCV.

The idea behind Algorithm 3.4.4 is to perform a number of "filter passes" to eliminate incorrect half-lattice vectors using the subroutine described in Algorithm 3.4.5. Each pass uses a different effective degree $g-1$ divisor beginning with the one defined in (3.4.18). This heuristic approach is unfortunately needed due to the numerical approximation inherent in evaluating the Riemann theta function as well as in numerically integrating the differentials along the Riemann surface paths. Future work would offer verifiable techniques, similar to those used in Smale's alpha theory, so as to avoid the need for heuristic methods.

Some notes on Algorithms 3.4.4 and 3.4.5:

- If a candidate vector $\boldsymbol{\kappa}_j \in J(X)$ is, in fact, the sought-after vector such that $\theta(\boldsymbol{\kappa}_j)=0$ then it must be so for all effective degree $g-1$ divisors $\mathcal{D}$. That is, for every such divisor

  $$\theta\big(\boldsymbol{\kappa}_j+\boldsymbol{A}(P_0,\mathcal{D}),\Omega\big)=0. \tag{3.4.21}$$

  This can be used for additional verification of our results.

- Care must be used when setting the numerical accuracy for the computation of the Riemann theta function. The default accuracy of $\epsilon=10^{-8}$ used in ABELFUNCTIONS may be insufficient for finding a unique solution. All numerical computations are

Algorithm 3.4.4 `half_lattice_vector`$(X,\mathcal{C})$

```
Require: a Riemann surface $X$
Require: a canonical divisor $\mathcal{C}$
Ensure: a half lattice vector $\boldsymbol{h}$
 1: $\mathcal{J} \leftarrow \{1,\ldots,2^{2g}\}$
 2: $\mathcal{D} \leftarrow (g-1)P_0$
 3: $\mathcal{J} \leftarrow$ half_lattice_filter($\mathcal{J},\mathcal{C},\mathcal{D}$)        ▷ filter pass #1
 4: if $\mathcal{J} = \{j^*\}$ return $\boldsymbol{h}_{j^*}$
 5: $\mathcal{D}_0 \leftarrow P_1 + \cdots P_{g-1}$ where the $P_i$'s are distinct regular places
 6: $\mathcal{J} \leftarrow$ half_lattice_filter($\mathcal{J},\mathcal{C},\mathcal{D}_0$)        ▷ filter pass #2
 7: if $\mathcal{J} = \{j^*\}$ return $\boldsymbol{h}_{j^*}$
 8: for each $m_1,\ldots,m_{g-1} \geq 0$ with $m_1 + \cdots + m_{g-1} = g-1$ do        ▷ filter pass #3
 9:     $\mathcal{D}_k \leftarrow m_1P_1 + \cdots + m_{g-1}P_{g-1}$
10:     $\mathcal{J} \leftarrow$ half_lattice_filter($\mathcal{J},\mathcal{C},\mathcal{D}_k$)
11:     if $\mathcal{J} = \{j^*\}$ return $\boldsymbol{h}_{j^*}$
12: end for
13: raise error("Could not find appropriate half-lattice vector.")
```

performed using native double precision so one must not set ϵ too close to 10^{-16} or else numerical round-off error may affect results as well.

- In practice it is observed that *only one* of the 2^{2g} half-lattice vectors $\boldsymbol{h}_j$ yields $\theta(\boldsymbol{K}(P_0),\Omega) = 0$. We have yet to find an example where the solution is not unique and expect that uniqueness can be shown mathematically.

Example 3.4.6. Finally, we provide examples of computing with the RCV.

```
sage: K = RiemannConstantVector   # alias the RCV function for brevity
sage: P0 = X.base_place()
sage: K(P0)
```

Algorithm 3.4.5 `half_lattice_filter`$(\mathcal{J}, \mathcal{C}, \mathcal{D})$

```
Require: index set $\mathcal{J} \subset \{1, \ldots, 2^{2g}\}$
Require: a canonical divisor $\mathcal{C}$
Require: effective, degree $g-1$ divisor $\mathcal{D}$
Ensure: filtered index set $\tilde{\mathcal{J}}$
 1: $\tilde{\mathcal{J}} \leftarrow \mathcal{J}$
 2: $\theta(\cdot, \Omega) \leftarrow$ the Riemann theta function uniformly accurate to order $\epsilon$
 3: $\boldsymbol{Z} \leftarrow \boldsymbol{A}(P_0, \mathcal{D}) - \frac{1}{2}\boldsymbol{A}(P_0, \mathcal{C})$
 4: for $j \in \mathcal{J}$ do
 5:     $\boldsymbol{\kappa}_j \leftarrow \boldsymbol{h}_j + \boldsymbol{Z} \bmod \Lambda$
 6:     if $||\theta(\boldsymbol{\kappa}_j)|| > \epsilon$ then
 7:         remove $j$ from $\tilde{\mathcal{J}}$
 8:     end if
 9: end for
10: return $\tilde{\mathcal{J}}$
```

```
[ 0.8488+0.7203j -0.5941-0.1146j -0.7432+0.8913j -0.8189+1.1381j]
```

We computationally verify that the RCV satisfies Theorems 2.2.15 and 2.2.17. First, we demonstrate that

$$2\,\boldsymbol{K}(P_0) + \boldsymbol{A}(P_0, \mathcal{C}) \equiv \boldsymbol{0}. \tag{3.4.22}$$

```
sage: z = J(2*K(P0) + AbelMap(C3))
sage: z
[ 0.+0.j  0.+0.j -0.-0.j -0.-0.j]
```

Next, we verify that $\boldsymbol{K}(P_0)$ belongs to the theta divisor. To test for membership we factor the Riemann theta function into its exponential and oscillatory parts [8, 9],

$$\theta(z, \Omega) = e^u v. \tag{3.4.23}$$

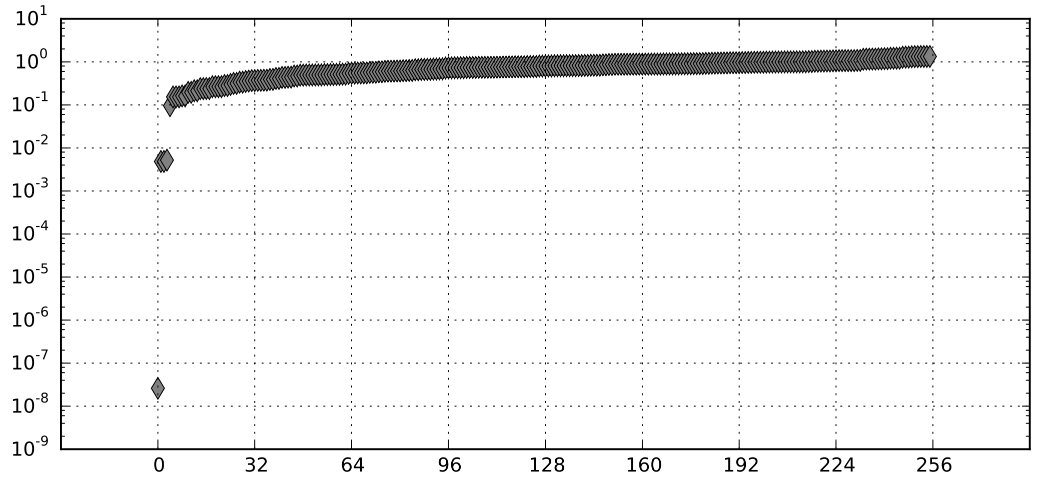


Figure 3.1: The sorted magnitudes of the oscillatory parts of $\theta\big(\boldsymbol{h}_j - \frac{1}{2}\boldsymbol{A}_0^{\mathcal{C}}, \Omega\big)$ *for each of the 256 half-lattice vectors* $\boldsymbol{h}_j$*. Note that the half-lattice vector resulting in the correct RCV produces a theta value approximately five orders of magnitude closer to zero than the others.*

Since the exponential part never vanishes we only examine the vanishing of the oscillatory part when determining $\boldsymbol{K}(P_0)$.

```
sage: W = K(P0)
sage v = RiemannTheta.oscillatory_part(W,Omega)
sage: abs(v)
2.60180216631e-08
```

Even with the default numerical integration and Riemann theta accuracy of 10^{-8} the choice of half-lattice vector producing the above RCV results in a Riemann theta value that is several orders of magnitude closer to zero than with the incorrect choices of half-lattice vector, as shown in Figure 3.1.

Let $\mathcal{D}$ be the divisor consisting of the three places lying above $x = 2$ each of multiplicity one. $\mathcal{D}$ is an effective divisor of degree $g - 1 = 3$. Therefore, $\boldsymbol{K}(P_0) + \boldsymbol{A}(P_0, \mathcal{D})$ is also a member of the theta divisor.

```
sage: D = sum(places_above_two)
sage: W = J(AbelMap(D) + K(P0))
sage: v = RiemannTheta.oscillatory_part(W, Omega)
sage: abs(v)
1.09506634962e-10
```

Chapter 4

APPLICATION: FINITE-GENUS SOLUTIONS OF THE KADOMTSEV–PETVIASHVILI EQUATION

The Kadomtsev–Petviashvili (KP) equation is a natural two-dimensional generalization of the Korteweg de-Vries (KdV) equation describing two-dimensional shallow water wave propagation.

Other approaches exist for computing with Riemann surfaces. Bobenko and collaborators [2, 5] compute solutions of integrable equations using a Schottky group representation for the associated surface. To our knowledge, the only paper dealing with all Riemann surfaces represented by algebraic curves is by Frauendiener, Klein, and Shramchenko who compute the homology of a Riemann surface from the monodromy of an underlying algebraic curve, following [12]. Otherwise, authors have restricted themselves to specific families of Riemann surfaces such as hyperelliptic ones [18, 19] or low genus ones [15, 30, 35]. Our aim throughout is the development of algorithms capable of dealing with arbitrary compact connected Riemann surfaces, as is required for the investigation of solutions of, for instance, the KP equation [11, 31].

4.1 Finite Genus Solutions

There is a large class of so-called "finite-genus" solutions to the KP equation. These solutions are quasiperiodic – functions f such that $f(z+T) \approx f(z)$ for some quasiperiod T and some meaning of "approximate". For example, a function satisfying $f(z+T) = Cf(z)$ is called geometric quasiperiodic. Similarly, a function satisfying $f(z+T) = f(z) + C$ is called arithmetic quasiperiodic. In fact, we have already seen an example of quasiperiodicity in the Riemann theta function $\theta(z, \Omega)$. The quasiperiod is $T = m + \Omega n$ for $m, nin \mathbb{Z}^g$ and the

$$\begin{tikzcd}
u(x,y,0) \arrow[r, "\text{Deconinck}"] \arrow[d] & X, \mathcal{D} \arrow[d, "\text{A) Krichever, Deconinck, van Hoeij}" near start, "\text{B) Frauendiener, Klein}" near end] \arrow[dl, dashed, "\text{Bobenko}"'] \\
u(x,y,t) & \boldsymbol{U}, \boldsymbol{V}, \boldsymbol{W}, \boldsymbol{D}, \Omega \arrow[l, "\text{Krichever}"] \arrow[l, dashed, bend left, "\text{Dubrovin, Flickinger, Segur}"]
\end{tikzcd}$$

Figure 4.1: A map of contributions to the solution to the initial value problem for the KP equation. The path taken by this work is represented by the solid arrows. Alternate approaches are shown as dashed arrows.

Riemann theta function satisfies the functional equation,

$$\theta(z + m + \Omega n, \Omega) = e^{-2\pi i\left(\frac{1}{2}n\cdot\Omega n + n\cdot z\right)}\theta(z, \Omega). \tag{4.1.1}$$

In order to define the class of finite-genus solutions to the KP equation we must first derive some components.

Given a genus g Riemann surface X with normalized basis of holomorphic one-forms $\omega = \{\omega_1, \ldots, \omega_g\}$

Theorem 4.1.1. *The KP equation*

$$u = c + 2\partial_x^2 \log\theta\Big(\boldsymbol{U}x + \boldsymbol{V}y + \boldsymbol{W}t + \boldsymbol{A}\left(P^\infty, \mathcal{D}\right) - \boldsymbol{K}(P^\infty), \Omega\Big), \tag{4.1.2}$$

4.2 Symmetric Period Bases

The approach discussed above provides solutions to the KP equation but make no restrictions on smoothness. However, for general solutions of the form

$$u = c + 2\partial_x^2 \log\theta\Big(\boldsymbol{U}x + \boldsymbol{V}y + \boldsymbol{W}t + \boldsymbol{D}, \Omega\Big), \tag{4.2.1}$$

there are certain restrictions imposed on the Riemann surface itself as well as the choice of phase shift $\boldsymbol{D}$ in order to obtain solutions to the KP equation that are real and smooth. In this section we will describe a technique due to Kalla and Klein used to derive such a solution as well as the algorithmic design in Abelfunctions. Note that this approach makes no direct contribution to the solution to the initial value problem but is nonetheless an important perspective of the study of solutions to the KP equation.

We begin with a central theorem.

Theorem 4.2.1. *[3] For smoothness and reality of solutions to the KP equation it is necessary and sufficient that, for a given Riemann surface X, place at infinity P_∞, and local parameter $p = k^{-1}$ at P_∞, that the following conditions are satisfied for the vector $\boldsymbol{D}$:*

- *X admits and antiholomorphic involution $\tau : X \to X, \tau^2 = 1$, where $\tau P_\infty = P_\infty$ and $\tau^* k = \bar{k}$.*
- *The set of all fixed ovals of the involution τ decomposes X into two connected components X^+ and X^-.*
- *If $P_\infty \in$*

The algorithms discussed in Chapter 2 compute the period matrix of a Riemann surface using the canonical basis of of cycles determined by the Tretkoff and Tretkoff algorithm [36]. However, these cycles may not be invariant under the antiholomorphic involution defined on a real plane algebraic curve.

Definition 4.2.2. *The* **topological** *type of a real Riemann surface X is given by the tuple (g, k, a) where g is the genus of the surface, k is the number of real ovals, and $a = 0$ if the curve is dividing and $a = 1$ if the curve is non-dividing. The* **real ovals** *of a surface are the connected components of $X(\mathbb{R})$. X is called* **dividing** *if and only if $X \setminus X(\mathbb{R})$ has two connected components. Otherwise, it has one connected component and is called* **non-dividing**.

Definition 4.2.3. *Let τ be an antiholomorphic involution on a real Riemann surface X. A canonical homology basis $(\mathcal{A}, \mathcal{B})$ is called a* **symmetric homology basis** *if it satisfies,*

$$\begin{pmatrix} \tau\mathcal{A} \\ \tau\mathcal{B} \end{pmatrix} = \begin{pmatrix} \mathbb{I}_g & 0 \\ \mathbb{H} & -\mathbb{I}_g \end{pmatrix} \begin{pmatrix} \mathcal{A} \\ \mathcal{B} \end{pmatrix}, \tag{4.2.2}$$

where $\mathbb{H}$ is a block diagonal matrix depending on the topological type (g, k, a) of X and has one of the following forms:

$$\mathbb{H} = \begin{pmatrix} 0 & 1 & & & & & & \\ 1 & 0 & & & & & & \\ & & \ddots & & & & & \\ & & & 0 & 1 & & & \\ & & & 1 & 0 & & & \\ & & & & & 0 & & \\ & & & & & & \ddots & \\ & & & & & & & 0 \end{pmatrix}, \qquad \text{if } k > 0 \text{ and } a = 0, \tag{4.2.3}$$

$$\mathbb{H} = \begin{pmatrix} \mathbb{I}_{g+1-k} & \\ & 0 \end{pmatrix}, \qquad \text{if } k > 0 \text{ and } a = 1, \tag{4.2.4}$$

$$\mathbb{H} = \begin{pmatrix} 0 & 1 & & & \\ 1 & 0 & & & \\ & & \ddots & & \\ & & & 0 & 1 \\ & & & 1 & 0 \end{pmatrix}, \qquad \text{if } k = 0 \text{ and } g \text{ is even}, \tag{4.2.5}$$

$$\mathbb{H} = \begin{pmatrix} 0 & 1 & & & & \\ 1 & 0 & & & & \\ & & \ddots & & & \\ & & & 0 & 1 & \\ & & & 1 & 0 & \\ & & & & & 0 \end{pmatrix}, \qquad \text{if } k = 0 \text{ and } g \text{ is odd}. \tag{4.2.6}$$

Recall that

The goal is to construct a symplectic transformation

$$\Gamma = \begin{pmatrix} A & B \\ C & D \end{pmatrix} \tag{4.2.7}$$

mapping an arbitrary canonical homology basis $(\tilde{\mathcal{A}}, \tilde{\mathcal{B}})$ to an equivalent, in the sense of Siegel, symmetric homology basis $(\mathcal{A}, \mathcal{B})$ satisfying,

$$\begin{pmatrix} \tau\mathcal{A} \\ \tau\mathcal{B} \end{pmatrix} \begin{pmatrix} \mathbb{I}_g & 0 \\ \mathbb{H} & -\mathbb{I}_g \end{pmatrix} \begin{pmatrix} \mathcal{A} \\ \mathcal{B} \end{pmatrix}, \tag{4.2.8}$$

and, therefore, the necessary and sufficient conditions to produce real and smooth solutions to the KP equation. Vinnikov proves [39] that such a symplectic transformation exists,

$$\begin{pmatrix} A & B \\ C & D \end{pmatrix} \begin{pmatrix} \tau\mathcal{A} \\ \tau\mathcal{B} \end{pmatrix} = \begin{pmatrix} \mathcal{A} \\ \mathcal{B} \end{pmatrix}. \tag{4.2.9}$$

We summarize the work of Kalla and Klein

4.3 Computations and Results

4.4 Future Work

Chapter 5

APPLICATION: DETERMINANTAL REPRESENTATIONS OF ALGEBRAIC CURVES

5.1 Background

5.2 Computations and Results

5.3 Future Work

BIBLIOGRAPHY

[1] Romberg's method. `http://en.wikipedia.org/wiki/Romberg's_method`, March 2014. [Online; accessed 11-March-2014].

[2] E. Belokolos, A. Bobenko, V. Enol'skii, A. Its, and V. Matveev. *Algebro-geometric approach to nonlinear integrable problems*. Springer Series in Nonlinear Dynamics. Springer Berlin, Heidelberg, 1994.

[3] Eugene D Belokolos and Alexander I Bobenko. Algebro-geometric approach to nonlinear integrable equations. 1994.

[4] G. A. Bliss. *Algebraic functions*. Dover Publications, Inc., New York, 1966.

[5] A. I. Bobenko and L. A. Bordag. Periodic multiphase solutions of the Kadomsev-Petviashvili equation. *J. Phys. A*, 22(9):1259–1274, 1989.

[6] E. Brieskorn and H. Knörrer. *Plane Algebraic Curves*. Birkhäuser Basel, 1986.

[7] Jonathan B Buckheit and David L Donoho. *Wavelab and reproducible research*. Springer, 1995.

[8] B. Deconinck. Multidimensional theta functions. In *NIST handbook of mathematical functions*, pages 537–547. U.S. Dept. Commerce, Washington, DC, 2010.

[9] B. Deconinck, M. Heil, A. Bobenko, M. van Hoeij, and M. Schmies. Computing Riemann theta functions. *Math. Comp.*, 73(247):1417–1442, 2004.

[10] B. Deconinck and M. S. Patterson. Computing with plane algebraic curves and Riemann surfaces: the algorithms of the Maple package "algcurves". In *Computational approach to Riemann surfaces*, volume 2013 of *Lecture Notes in Math.*, pages 67–123. Springer, Heidelberg, 2011.

[11] B. Deconinck and H. Segur. The KP equation with quasiperiodic initial data. *Phys. D*, 123(1-4):123–152, 1998. Nonlinear waves and solitons in physical systems (Los Alamos, NM, 1997).

[12] B. Deconinck and M. van Hoeij. Computing Riemann matrices of algebraic curves. *Phys. D*, 152/153:28–46, 2001. Advances in nonlinear mathematics and science.

[13] The Sage Developers. *SageMath, the Sage Mathematics Software System (Version x.y.z)*, YYYY. `http://www.sagemath.org`.

[14] B. A. Dubrovin. Theta-functions and nonlinear equations. *Uspekhi Mat. Nauk*, 36(2(218)):11–80, 1981. With an appendix by I. M. Krichever.

[15] B. A. Dubrovin, R. Flickinger, and H. Segur. Three-phase solutions of the Kadomtsev-Petviashvili equation. *Stud. Appl. Math.*, 99(2):137–203, 1997.

[16] D. Duval. Rational Puiseux expansions. *Compositio Math.*, 70(2):119–154, 1989.

[17] H. M. Farkas and I. Kra. *Riemann surfaces*, volume 71 of *Graduate Texts in Mathematics*. Springer-Verlag, New York, second edition, 1992.

[18] J. Frauendiener and C. Klein. Hyperelliptic theta-functions and spectral methods: KdV and KP solutions. *Lett. Math. Phys.*, 76(2-3):249–267, 2006.

[19] J. Frauendiener and C. Klein. Computational Approach to Hyperelliptic Riemann Surfaces. *Lett. Math. Phys.*, 105(3):379–400, 2015.

[20] J. Frauendiener, C. Klein, and V. Shramchenko. Efficient computation of the branching structure of an algebraic curve. *Comput. Methods Funct. Theory*, 11(2):527–546, 2011.

[21] Erich Gamma. *Design patterns: elements of reusable object-oriented software*. Pearson Education India, 1995.

[22] P. A. Griffiths. *Introduction to algebraic curves*, volume 76 of *Translations of Mathematical Monographs*. American Mathematical Society, Providence, RI, 1989. Translated from the Chinese by Kuniko Weltin.

[23] Gary Harris and Clyde Martin. The roots of a polynomial vary continuously as a function of the coefficents. *Proceedings of the American Mathematical Society*, 100(2), 1987.

[24] J. D. Hauenstein and F. Sottile. Algorithm 921: alphaCertified: certifying solutions to polynomial systems. *ACM Trans. Math. Software*, 38(4):Art. ID 28, 20, 2012.

[25] Robert C Martin. Design principles and design patterns. *Object Mentor*, 1(34), 2000.

[26] M. Mňuk. An algebraic approach to computing adjoint curves. *J. Symbolic Comput.*, 23(2-3):229–240, 1997. Parametric algebraic curves and applications (Albuquerque, NM, 1995).

[27] D. Mumord. Appendix: Curves and their jacobians. In *The Red Book of Varieties and Schemes*, volume 1358 of *Lecture Notes in Mathematics*, pages 225–291. Springer Berlin, Heidelberg, 1999.

[28] M. Noether. Rationale ausführungen der operationen in der theorie der algebraischen funktionen. *Math. Ann.*, 23:311–358, 1983.

[29] Roger D Peng. Reproducible research in computational science. *Science*, 334(6060):1226–1227, 2011.

[30] H. Segur and A. Finkel. An analytical model of periodic waves in shallow water. *Stud. Appl. Math.*, 73(3):183–220, 1985.

[31] T. Shiota. Characterization of Jacobian varieties in terms of soliton equations. *Invent. Math.*, 83(2):333–382, 1986.

[32] Victoria Stodden. Reproducible research for scientific computing: Tools and strategies for changing the culture. *Computing in Science & Engineering*, 14(4):13–17, 2012.

[33] Victoria Stodden and Sheila Miguez. Best practices for computational science: Software infrastructure and environments for reproducible and extensible research. 2013.

[34] C. Swierczewski. Abelfunctions: A library for computing with Abelian functions, Riemann surfaces, and algebraic curves., 2017. http://github.com/abelfunctions/abelfunctions.

[35] C. Swierczewski and B. Deconinck. Computing Riemann theta functions in Sage with applications. *Mathematics and Computers in Simulation*, 2013. http://www.sciencedirect.com/science/article/pii/S0378475413000888.

[36] C. L. Tretkoff and M. D. Tretkoff. Combinatorial group theory, Riemann surfaces and differential equations. In *Contributions to group theory*, volume 33 of *Contemp. Math.*, pages 467–519. Amer. Math. Soc., Providence, RI, 1984.

[37] K. Ueno. *An introduction to algebraic geometry*, volume 166 of *Translations of Mathematical Monographs*. American Mathematical Society, Providence, RI, 1997. Translated from the 1995 Japanese original by Katsumi Nomizu.

[38] M. van Hoeij. An algorithm for computing an integral basis in an algebraic function field. *J. Symbolic Comput.*, 18(4):353–363, 1994.

[39] Victor Vinnikov. Self-adjoint determinantal representations of real plane curves. *Mathematische Annalen*, 296(1):453–479, 1993.