We describe an algorithm for the reparametrization of a closed curve defined by a sequence of m points while providing the user a high level of control over the frequency content of the resulting curve. Specifically, the algorithm views the tangential angle of the curve as a function of the arc-length, filters it as such, and adds a small analytic perturbation so that the curve passes through the input data. If the number of nodes in the initial discretization is n, the entire scheme has asymptotic complexity $\mathcal{O}(n \log n)$. The resulting curve is analytic and bandlimited. The performance of the scheme is illustrated with several numerical examples.

Fitting a Bandlimited Curve to Points in a Plane

Daniel Beylkin[†] and Vladimir Rokhlin[‡]
Technical Report YALEU/DCS/TR-1480
July 19, 2013

[†] This author was supported by the DOD, AFOSR NDSEG Fellowship Award FA9550-11-C-0028 and ONR Grant N00014-11-1-0718
[‡] This author was supported by the ONR Grant N00014-10-1-0570 and AFOSR Grant FA9550-09-1-0241

Program in Applied Mathematics, Yale University, New Haven, CT 06511

Approved for public release: distribution is unlimited.
Keywords: *Bandlimited functions, curve fitting, natural parametrization.*

1 Introduction

The need for resampling of discretized planar curves arises in many areas: CAD/CAM systems, computer graphics, numerical computation, etc. Due to their simplicity and highly developed underlying theory, splines and related methods are the standard approaches in these fields; in many situations, they work quite well. The idea is to use relatively low-degree polynomials (combinations of exponentials with polynomials, etc.) in a piecewise fashion while ensuring sufficient smoothness (i.e., differentiability) at connecting points (see, e.g., [1, 2, 4, 5]).

Unfortunately, there are several classes of problems where such techniques encounter difficulties. For example, rapidly convergent algorithms of numerical analysis tend to require data with many continuous derivatives. In other situations, the nodes at which the curves are specified are poorly spaced, often leading to undesirable behavior of spline-based algorithms.

Sometimes, it is desirable to have direct control over the frequency content of the curve. In such cases, it is tempting to view the coordinates of the curve as a function of the arc-length and to use standard Fourier domain filtering techniques. Unfortunately, the process of filtering the coordinates changes the arc-length in a non-uniform manner, so it generally fails to remove the high frequency components of the curve.

In this paper, we describe an algorithm for the reparametrization of a closed curve defined by a sequence of points $S = \{x_i, y_i\}_{i=1,\dots,m} \in \mathbb{R}^2$. The resulting curve is analytic (in the sense that the coordinates of a point on it are analytic functions of the arc-length) and the user has direct control over its frequency content. The algorithm views the tangential angle of the curve as a function of the arc-length and filters it as such. While the straightforward application of this approach leads to a curve that does not pass exactly through the original data S, it tends to pass very close. A small analytic perturbation of the curve eliminates the discrepancy while minimizing the impact on its Fourier content.

This paper is organized as follows. Section 2 summarizes various standard mathematical facts used in the remainder of the paper. Section 3 describes the algorithm for the construction of the closed, bandlimited curve. Section 4 contains the results of our numerical experiments.

2 Mathematical and Numerical Preliminaries

2.1 Parametrizations of a Curve

In this section, we summarize several facts from elementary differential geometry. We refer the reader to, e.g., [10, 12] for details.

A curve γ is a continuous mapping $\gamma : I = [a, b] \to \mathbb{R}^2$; we will use the notation

$$\gamma(t) = \{x(t), y(t)\}. \tag{2.1}$$

A curve is simple if $\gamma(t_1) = \gamma(t_2)$ implies that $t_1 = t_2$, i.e., it is not self-intersecting. It is closed if $\gamma(a) = \gamma(b)$. Unless otherwise stated, in this paper we only consider simple closed curves $\gamma \in C^2[a, b]$.

The length of a curve γ on the interval $[t_1, t_2] \subseteq I = [a, b]$ is given by the formula

$$\int_{t_1}^{t_2} \left\|\gamma'(t)\right\| dt = \int_{t_1}^{t_2} \sqrt{(x'(t))^2 + (y'(t))^2} dt, \tag{2.2}$$

so the length of the entire curve is

$$L = \int_a^b \left\|\gamma'(t)\right\| dt. \tag{2.3}$$

We define the function $s : [a, b] \to [0, L]$ via the formula

$$s(t) = \int_a^t \left\|\gamma'(\tau)\right\| d\tau \tag{2.4}$$

and observe that $s(t)$ is the length of γ on the interval $[a, t] \subseteq I = [a, b]$. We denote the inverse of s by $\xi : [0, L] \to [a, b]$ so that $\xi(s(t)) = t$ for all $t \in [a, b]$. Since s is the arc-length of the curve, the continuous mapping $\widetilde{\gamma} : [0, L] \to \mathbb{R}^2$, given by the formula

$$\widetilde{\gamma}(s) = \{\widetilde{x}(s), \widetilde{y}(s)\} = \gamma(\xi(s)), \tag{2.5}$$

is referred to as the arc-length or natural parametrization of the curve γ (s is called the natural parameter).

For planar curves, the tangential angle $\theta(s)$ of a curve is the angle between the tangent line to the curve and the x-axis. Given the curve $\widetilde{\gamma}$ in (2.5), the tangent line to the curve is $\widetilde{\gamma}'(s) = \{\widetilde{x}'(s), \widetilde{y}'(s)\}$. Hence,

$$\widetilde{\gamma}'(s) = \{\cos(\theta(s)), \sin(\theta(s))\}. \tag{2.6}$$

This is known as the natural equation of the curve. Integrating (2.6) yields

$$\begin{aligned} \widetilde{x}(s) &= \int_0^s \cos(\theta(\sigma))d\sigma + c_x \\ \widetilde{y}(s) &= \int_0^s \sin(\theta(\sigma))d\sigma + c_y, \end{aligned} \tag{2.7}$$

where c_x, c_y are arbitrary constants. Hence, in order to obtain the curve uniquely from $\theta(s)$, the point $(c_x, c_y) \in \mathbb{R}^2$ must be fixed *a priori*.

From (2.7), it follows that for a closed curve,

$$\begin{aligned} 0 = \widetilde{x}(L) - \widetilde{x}(0) &= \int_0^L \cos(\theta(s))ds \\ 0 = \widetilde{y}(L) - \widetilde{y}(0) &= \int_0^L \sin(\theta(s))ds. \end{aligned} \tag{2.8}$$

Moreover, since $\frac{d}{ds}\widetilde{x}(L) = \frac{d}{ds}\widetilde{x}(0)$ and $\frac{d}{ds}\widetilde{y}(L) = \frac{d}{ds}\widetilde{y}(0)$ for closed curves, the functions $\cos(\theta(s))$ and $\sin(\theta(s))$ are periodic with the period L.

Given a curve $\widetilde{\gamma}$ parametrized by its arc-length, the curvature κ is

$$\begin{aligned}\kappa &= \left\|\widetilde{\gamma}''(s)\right\| \\ &= \left\|\{-\sin\left(\theta(s)\right)\theta'(s), \cos\left(\theta(s)\right)\theta'(s)\}\right\| \\ &= \sqrt{\left(\sin^2\left(\theta(s)\right) + \cos^2\left(\theta(s)\right)\right)\left(\theta'(s)\right)^2} \\ &= \theta'(s).\end{aligned} \tag{2.9}$$

Hence, κ measures the rate at which the tangent line to the curve rotates.

2.2 Bandlimited Periodic Functions

The Discrete Fourier Transform (DFT) is a transformation $\mathcal{F} : \mathbb{C}^n \to \mathbb{C}^n$ defined by the formula

$$\hat{f}_k = \frac{1}{n}\sum_{j=0}^{n-1} f_j e^{-2\pi ikj/n}, \;\; k = 0, \ldots, n-1. \tag{2.10}$$

The inverse of the DFT is given by the formula

$$f_j = \sum_{k=0}^{n-1} \hat{f}_k e^{2\pi ikj/n}, \;\; j = 0, \ldots, n-1. \tag{2.11}$$

The n complex numbers $\{\hat{f}_k\}$ are referred to as the Fourier coefficients of the signal $\{f_j\}$. In many applications, such as those of this paper, it is convenient to view $\{f_j\}$ as n equally-spaced samples of a continuous, periodic real function f with a single period defined to be $[0, 1]$.

Observation 1. *The DFT and its inverse can also be described by the closely related formulae*

$$\hat{f}_k = \frac{1}{n}\sum_{j=0}^{n-1} f_j e^{-2\pi ikj/n}, \;\; k = -\frac{n-1}{2}, \ldots, \frac{n-1}{2} \tag{2.12}$$

$$f_j = \sum_{k=-\frac{n-1}{2}}^{\frac{n-1}{2}} \hat{f}_k e^{2\pi ikj/n}, \;\; j = 0, \ldots, n-1. \tag{2.13}$$

While the forms (2.10) and (2.11) are the standard representation for the DFT, the forms (2.12) and (2.13) are usually preferred in applications of DFT to analysis (see, e.g., [6]). If $t_j = \frac{j}{n}$, $j = 0, \ldots, n-1$ *are the equally-spaced samples on* $[0, 1]$, *i.e.,*

$$f_j = f(t_j), \tag{2.14}$$

then (2.13) can be restated as

$$f_j = \sum_{k=-\frac{n-1}{2}}^{\frac{n-1}{2}} \hat{f}_k e^{2\pi ikt_j}, \;\; j = 0, \ldots, n-1. \tag{2.15}$$

The corresponding trigonometric polynomial for the evaluation of $f(t)$ anywhere on $[0,1]$ is therefore

$$f(t) = \sum_{k=-\frac{n-1}{2}}^{\frac{n-1}{2}} \hat{f}_k e^{2\pi ikt}. \tag{2.16}$$

Obviously, straightforward computation of the DFT or its inverse costs $\mathcal{O}(n^2)$ operations. Algorithms that perform such computations in $\mathcal{O}(n \log n)$ are known as Fast Fourier Transforms (FFT) (see, e.g., [8, 11]).

The function f is referred to as bandlimited with band-limit $\hat{n}$ whenever $\hat{f}_k = 0$ for $k \notin [-\hat{n}, \hat{n}]$. However, approximate notions of band-limit are more useful for practical purposes, such as

$$\sum_{k \notin (-\hat{n}, \hat{n})} \left| \hat{f}_k \right| < \varepsilon \cdot \sum_{k=-\frac{n-1}{2}}^{\frac{n-1}{2}} \left| \hat{f}_k \right| \tag{2.17}$$

for some small ε.

2.3 Filtering a Function

For $\{f_j\}$ in (2.14), a finite impulse response (FIR) filter $\{g_j\}$ transforms the signal $\{f_j\}$ into $\{h_j\}$ by the formula

$$h_j = \sum_{k=0}^{j} g_k f_{j-k}, \;\; j = 0, \dots, n-1, \tag{2.18}$$

which is known as convolution. If $\{\hat{f}_k\}$, $\{\hat{g}_k\}$, and $\{\hat{h}_k\}$ for $k = 0, \dots, n-1$ are the Fourier coefficients of $\{f_j\}$, $\{g_j\}$, and $\{h_j\}$, respectively, then it is well known that

$$\hat{h}_k = \hat{f}_k \hat{g}_k, \;\; k = 0, \dots, n-1. \tag{2.19}$$

Thus, a signal which is not bandlimited can be made so by attenuating its high-order (e.g., those larger than some desired band-limit) Fourier coefficients. Such a filter is called a low-pass filter [8, 11].

An ideal low-pass filter, also referred to as a brick-wall filter, has Fourier coefficients

$$\hat{g}_k = \begin{cases} 1 & k \in [-\hat{n}, \hat{n}] \\ 0 & k \notin [-\hat{n}, \hat{n}] \end{cases} \tag{2.20}$$

for some cutoff frequency $\hat{n}$. In this case the filtered $\{f_j\}$ is

$$h_j = \sum_{k=-\hat{n}}^{\hat{n}} \hat{f}_k e^{2\pi ikj/n}, \;\; j = 0, \dots, n-1. \tag{2.21}$$

However, even if the signal $\{f_j\}$ is non-oscillatory, the output $\{h_j\}$ may oscillate due to the abrupt truncation of slowly decaying Fourier coefficients - the so-called Gibbs phenomenon [8, 11]. This is an affliction of many low-pass filters and it is often desirable to minimize the magnitude of this effect.

Certain low-pass filters do not exhibit the Gibbs phenomenon. One such example is the Gaussian filter, which has Fourier coefficients

$$\hat{g}_k = e^{-\frac{\ln(c)}{\hat{n}^2}k^2}, \ \ k = -\frac{n-1}{2}, \dots, \frac{n-1}{2}, \tag{2.22}$$

where c is chosen so that $\hat{g}_{\hat{n}} = \frac{1}{c}$. In many engineering applications, another popular filter is the Kaiser window [8, 11], which has Fourier coefficients

$$\hat{g}_k = \frac{I_0\left[\beta\sqrt{1-\left(\frac{2k}{n-1}\right)^2}\right]}{I_0\left[\beta\right]} \tag{2.23}$$

where I_0 is the modified Bessel function of order zero and β is computed so that $\hat{g}_{\hat{n}} = \frac{1}{c}$ for some desired c.

Observation 2. *Large curvature in (2.9) gives rise to high-order frequencies of θ when viewed in the Fourier domain. For the purpose of fitting smooth curves to a sequence of points in a plane, we would like to filter out these high-order frequencies. It is in this sense that we seek to fit a bandlimited curve to the data. Viewed in this context, the band-limit of θ may be understood to be a measure of smoothness of the curve.*

2.4 Spectral Integration

For $f(t)$ defined in (2.16), we would like to compute the integral

$$\begin{aligned} g(t) &= \int_0^t f(\tau)d\tau \\ &= \int_0^t \sum_{k=-\frac{n-1}{2}}^{\frac{n-1}{2}} \hat{f}_k e^{2\pi i k\tau} d\tau \\ &= \sum_{k\neq 0} \hat{f}_k \int_0^t e^{2\pi i k\tau} d\tau + \int_0^t \hat{f}_0 d\tau \\ &= \sum_{k\neq 0} \frac{\hat{f}_k}{2\pi i k}\left(e^{2\pi i k t} - 1\right) + \hat{f}_0 t \\ &= \sum_{k\neq 0} \frac{\hat{f}_k}{2\pi i k} e^{2\pi i k t} - \sum_{k\neq 0} \frac{\hat{f}_k}{2\pi i k} + \hat{f}_0 t. \end{aligned} \tag{2.24}$$

at equally-spaced $t = t_j = \frac{j}{n}$, $j = 0, \dots, n-1$ on $[0, 1]$.

Clearly, evaluating (2.24) at all $t_0, \dots, t_{n-1}$ directly costs $\mathcal{O}(n^2)$ operations. The following algorithm reduces the cost to $\mathcal{O}(n \log n)$ operations:

1. Compute the Fourier coefficients $\{\hat{f}_k\}$ via the FFT; denote the output as $\{\hat{g}_k\}$
2. Set $\hat{g}_0 = 0$

3. Divide $\{\hat{g}_k\}_{k\neq 0}$ by $2\pi ik$
4. Invert $\{\hat{g}_k\}$ via the (inverse) FFT to obtain $\{g_j\}$
5. At this point, $g_0 = \sum_{k\neq 0} \frac{\hat{f}_k}{2\pi ik}$, so subtract g_0 from all of the terms in $\{g_j\}$
6. Add in the linear term $\hat{f}_0 t$ by adding $\{\hat{f}_0 t_j\}$ to $\{g_j\}$.

The output of this algorithm is the sequence $\{g_j\} = \{g(t_j)\}$.

Observation 3. *As a consequence of (2.12) and (2.24), for $f(t)$ defined in (2.16), the integral*

$$\int_0^1 f(t)dt = \hat{f}_0 = \frac{1}{n}\sum_{j=0}^{n-1} f(t_j) \tag{2.25}$$

can be evaluated exactly by the n-point Trapezoidal rule [9].

2.5 Lagrange Interpolation

Given $\{f_j\}$ where $f_j = f(t_j)$ for $f(t)$ defined in (2.16) and $t_j = \frac{j}{n}$, $j = 0, \dots, n-1$, a single evaluation of the function $f(t)$ at a point other than t_j costs $\mathcal{O}(n)$ operations. As an alternative, we can use Lagrange polynomials to accurately interpolate the function at intermediate points and reduce the cost of evaluation to $\mathcal{O}(1)$ [3] - provided the sampling of the curve is sufficiently dense.

Since $f(t)$ is periodic with period 1,

$$f(t_{j+\lambda n}) = f_j \tag{2.26}$$

where $t_{j+\lambda n} = t_j + \lambda$ for any integer λ. For $t_k \leq t < t_{k+1}$, the Lagrange basis polynomial

$$\ell_j(t) = \prod_{\substack{i=k-\frac{n}{2} \\ i\neq j}}^{k+\frac{n}{2}} \frac{t-t_i}{t_j - t_i}, \;\; j = k - \frac{n}{2}, \dots, k + \frac{n}{2} \tag{2.27}$$

satisfies the property

$$\ell_j(t_i) = \delta_{ij} = \begin{cases} 1 & i = j \\ 0 & i \neq j. \end{cases} \tag{2.28}$$

Hence,

$$\psi(t) = \sum_{j=k-\frac{n}{2}}^{k+\frac{n}{2}} f_j \ell_j(t), \;\; t_k \leq t < t_{k+1} \tag{2.29}$$

is the Lagrange interpolating polynomial for $f(t)$, i.e., $\psi(t_j) = f_j$.

In practical implementations, the interpolation is constructed locally using low-degree polynomials. For a given t, this means that the interpolation is restricted to the nearest $m \ll n$ samples. For $t_k \leq t < t_{k+1}$,

$$\widetilde{\psi}(t) = \sum_{j=k-\frac{m}{2}}^{k+\frac{m}{2}} f_j \widetilde{\ell}_j(t) \tag{2.30}$$

where

$$\widetilde{\ell}_j(t) = \prod_{\substack{i=k-\frac{m}{2} \\ i\neq j}}^{k+\frac{m}{2}} \frac{t-t_i}{t_j-t_i}, \;\; j = k-\frac{m}{2}, \dots, k+\frac{m}{2} \tag{2.31}$$

In our experience, $m \approx 20$ works well for double precision computations.

To see that the evaluation of (2.30) costs only $\mathcal{O}(1)$ operations, we write

$$\widetilde{\ell}_j(t) = \frac{w_j}{t-t_j}\phi(t) \tag{2.32}$$

where

$$w_j = \frac{1}{\prod_{i=k-\frac{m}{2}}^{k+\frac{m}{2}} (t_j - t_k)} \tag{2.33}$$

and

$$\phi(t) = \prod_{\substack{i=k-\frac{m}{2} \\ i\neq j}}^{k+\frac{m}{2}} (t-t_k). \tag{2.34}$$

Thus,

$$\widetilde{\psi}(t) = \phi(t) \sum_{j=k-\frac{m}{2}}^{k+\frac{m}{2}} \frac{w_j}{t-t_j} f_j, \;\; t_k \le t < t_{k+1}, \tag{2.35}$$

which is known as the modified form of Lagrange's interpolation formula. The computation of each w_j costs $\mathcal{O}(m)$. Since $\{t_j\}$ are equally spaced samples, the set of coefficients $\{w_j\}$ will remain the same regardless of the value of k. Hence, $\{w_j\}$ can be pre-computed and the total cost is $\mathcal{O}(m^2)$ operations. Once $\{w_j\}$ have been computed, the cost of evaluating $\widetilde{\psi}(t)$ at any point $t \in [0,1]$ is $\mathcal{O}(m)$. Since m is $\mathcal{O}(1)$, the cost of evaluating the function $f(t)$ for any $t \in [0,1]$ is $\mathcal{O}(1)$.

2.6 Method of Conjugate Gradients

The method of conjugate gradients (see, e.g., [14]) is an iterative scheme for the solution of a symmetric, positive-definite $m \times m$ linear system

$$Ax = y. \tag{2.36}$$

A single iteration of the method costs $\mathcal{O}(m+q)$ operations where q is the cost of matrix-vector multiplication.

If x is the solution of the linear system, the relative error at step k is bounded by

$$\frac{\|x - x_k\|_A}{\|x\|_A} < 2\left(\frac{\sqrt{\kappa}-1}{\sqrt{\kappa}+1}\right)^k \tag{2.37}$$

where

$$\|x\|_A = x^\dagger A x \tag{2.38}$$

and κ is the condition number of the matrix A, i.e., ratio of the largest eigenvalue of A to the smallest. In finite precision, additional complications arise whenever the condition number of the matrix A is large; in this paper, we will only deal with well-conditioned matrices.

2.7 Adaptive Gaussian Integration

Quadrature formulae are expressions of the form

$$\sum_{j=0}^{n-1} w_j f(t_j) \tag{2.39}$$

where the points $t_j \in \mathbb{R}$ and coefficients $w_j \in \mathbb{R}$ are the nodes and weights of the quadrature, respectively [3, 13]. The nodes and weights of Gaussian quadratures are chosen so that polynomials of degree less than or equal to $2n-1$, $\{\phi_0, \dots, \phi_{2n-1}\}$, are integrated exactly, that is,

$$\sum_{j=0}^{n-1} w_j \phi_i(t_j) = \int_a^b \phi_i(t)dt \tag{2.40}$$

for all $i = 0, \dots, 2n-1$. Provided that the function f is well approximated by linear combinations of such polynomials, the Gaussian quadrature formulae provide good approximations to integrals of the form

$$\int_a^b f(x)dx. \tag{2.41}$$

An adaptive implementation of Gaussian quadrature formulae approximates (2.41) on repeatedly subdivided intervals until the approximation of the integral over a subinterval achieves a desired level of accuracy ε. Specifically, we compute

$$I \approx \int_a^b f(x)dx \tag{2.42}$$

as well as

$$I_1 \approx \int_a^{\frac{a+b}{2}} f(x)dx \tag{2.43}$$

and

$$I_2 \approx \int_{\frac{a+b}{2}}^b f(x)dx \tag{2.44}$$

using Gaussian quadrature formulae. If $|I - (I_1 + I_2)| < \varepsilon$, then I is declared to be a sufficiently accurate approximation to the integral. Otherwise, the interval is halved and this procedure is repeated on both subintervals separately.

3 Description of the Algorithm

3.1 Initial Discretization

In the remainder of the paper, we assume that all curves are discretized in the counterclockwise direction; if the discretization is in the clockwise direction, some of the formulas will have to be changed slightly. Given $S = \{x_i, y_i\}_{i=1,\dots,m} \in \mathbb{R}^2$, the initial construction of the curve is the polygon obtained by connecting the points in S with straight lines, which we denote γ_{poly}. Using the obtained polygon, we construct a mapping from the arc-length to the tangent angles in the following manner: if the arc-length of the polygon from (x_1, y_1) to (x_i, y_i) is denoted s_i ($s_1 = 0$), then $\theta_{poly}(s_i)$ is the angle between the line segment connecting (x_i, y_i) to (x_{i+1}, y_{i+1}) and the x-axis. Since the curve is closed and non-self-intersecting, we introduce the notation $(x_{m+1}, y_{m+1}) = (x_1, y_1)$ and hence

$$\theta_{poly}(s_{m+1}) = \theta_{poly}(s_1) + 2\pi \tag{3.1}$$

where $L = s_{m+1}$ is the arc-length of the entire polygon.

Obviously, the function $\theta_{poly} : [0, L] \to \mathbb{R}^1$ is a step function given by the formula

$$\theta_{poly}(s) = \theta_{poly}(s_i), \quad for\, all\, s \in (s_i, s_{i+1}); \tag{3.2}$$

in other words, θ_{poly} is the tangential angle of the polygon as a function of its arc-length. We then resample $\theta_{poly}(s)$ at n equally-spaced points $t_j = \frac{j}{n} \cdot L$, $j = 0, \dots, n-1$ to obtain $\{\theta_j\}$, where $\theta_j = \theta_{poly}(t_j)$.

3.2 Filtering $\{\theta_j\}$

Since $\cos(\theta_{poly}(s))$ and $\sin(\theta_{poly}(s))$ are periodic with the period L and the curve is non-self-intersecting, the function

$$\theta_{poly}(s) - \frac{2\pi}{L}s \tag{3.3}$$

is also periodic with period L. Due to corners in the polygon, the function (3.3) is not bandlimited. Thus, we construct a bandlimited approximation to (3.3) via a fairly standard filtering procedure.

Using $\{\theta_j\}$ obtained from the discretization of the polygon, we apply the DFT to obtain the Fourier coefficients of (3.3) with respect to arc-length on γ_{poly}:

$$\hat{\theta}_k = \frac{1}{n}\sum_{j=0}^{n-1}\left(\theta_j - \frac{2\pi}{n}j\right)e^{-2\pi ikj/n}, \;\; k = -\frac{n-1}{2}, \dots, \frac{n-1}{2}. \tag{3.4}$$

We filter the Fourier coefficients using the Gaussian filter (2.22) (with c and $\hat{n}$ to be chosen later), so the filtered Fourier coefficients $\hat{\phi}_k$ become

$$\hat{\phi}_k = \hat{\theta}_k e^{-\frac{\ln(c)}{\hat{n}}k^2}, \;\; k = -\frac{n-1}{2}, \dots, \frac{n-1}{2}. \tag{3.5}$$

To reconstruct the bandlimited version of $\{\theta_j\}$, we apply the inverse of the DFT to obtain

$$\phi_j = \sum_{k=-\frac{n-1}{2}}^{\frac{n-1}{2}} \hat{\phi}_k e^{2\pi ikj/n} + \frac{2\pi}{n}j, \;\; j = 0, \dots, n-1. \tag{3.6}$$

Due to (2.16), $\{\phi_j\}$ are equally-spaced samples on $[0, L]$ of the trigonometric polynomial

$$\phi(s) = \sum_{k=-\frac{n-1}{2}}^{\frac{n-1}{2}} \hat{\phi}_k e^{2\pi ik\frac{s}{L}} + \frac{2\pi}{L}s. \tag{3.7}$$

Observation 4. *While in our codes we used the Gaussian filter, other low-pass filters (such as the Kaiser window) can be used. In our experiments, the results were not particularly sensitive to the choice of filter.*

Observation 5. *Obviously, the filtered curve obtained in this manner will not necessarily be closed even if γ_{poly} is. Indeed, for any closed curve defined in (2.7),*

$$\int_0^L \cos(\theta(s))ds = \int_0^L \sin(\theta(s))ds = 0. \tag{3.8}$$

While $\cos(\phi(s))$ *and* $\sin(\phi(s))$ *for* $\phi(s)$ *defined in (3.7) are still periodic, there are no assurances that their integrals will be zero.*

3.3 Closing the Curve

Since the process of filtering, generally speaking, will open the curve, we need to enforce the constraint

$$\int_0^L \cos(\phi(s))ds = \int_0^L \sin(\phi(s))ds = 0, \tag{3.9}$$

while changing the curve as little as possible otherwise. Given the expansion (3.7), these integrals are

$$\begin{aligned}\int_0^L \cos(\phi(s))ds &= \int_0^L \cos(\sum_{k=-\frac{n-1}{2}}^{\frac{n-1}{2}} \hat{\phi}_k e^{2\pi i k\frac{s}{L}} + \frac{2\pi}{L}s)ds \\ \int_0^L \sin(\phi(s))ds &= \int_0^L \sin(\sum_{k=-\frac{n-1}{2}}^{\frac{n-1}{2}} \hat{\phi}_k e^{2\pi i k\frac{s}{L}} + \frac{2\pi}{L}s)ds.\end{aligned} \tag{3.10}$$

Therefore, we need to change the Fourier coefficients $\hat{\phi}_k$ to ensure that the constraint (3.9) is satisfied with minimal impact on the higher order coefficients.

In practice, we evaluate these two integrals using the Trapezoidal rule. The constraint (3.9) becomes

$$\sum_{j=0}^{n-1} \cos(\phi_j) = \sum_{j=0}^{n-1} \sin(\phi_j) = 0. \tag{3.11}$$

Hence, we have two functions which we want to simultaneously set to zero:

$$\begin{aligned} f_x(\phi_0, \ldots, \phi_{n-1}) &= \sum_{j=0}^{n-1} \cos(\phi_j) \\ f_y(\phi_0, \ldots, \phi_{n-1}) &= \sum_{j=0}^{n-1} \sin(\phi_j). \end{aligned} \tag{3.12}$$

Since there are n parameters $\phi_0, \ldots, \phi_{n-1}$ which can be used to solve these equations, there are many ways to enforce (3.11). However, in many applications it is desirable (or necessary) to preserve straight lines. In other words, when the user resamples a polygon, the result should look like a smoothed version of the original polygon. Yet another interpretation is to observe that any smoothing process will have to change the curvature of the user-specified curve - but a large region with zero curvature should stay that way. The iterative procedure below has been constructed with this in mind. Given the values of $\{\phi_j\}$ in (3.6), we compute

$$\phi_j^{new} = \phi_j + \delta_x \cos(\phi_j) + \delta_y \sin(\phi_j). \tag{3.13}$$

In practice, we filter $\cos(\phi_j)$ and $\sin(\phi_j)$ in order to minimize the impact on the higher order Fourier coefficients.

The objective of the iteration is to choose δ_x, δ_y so that

$$\begin{aligned} f_x(\phi_0^{new}, \dots, \phi_{n-1}^{new}) &= 0 \\ f_y(\phi_0^{new}, \dots, \phi_{n-1}^{new}) &= 0. \end{aligned} \tag{3.14}$$

To evaluate (3.12) at (3.13), we compute its Taylor expansion:

$$\begin{aligned} f_x(\phi_0^{new}, \dots, \phi_{n-1}^{new}) &= \sum_{j=0}^{n-1} \cos(\phi_j) - \sum_{j=0}^{n-1} \sin(\phi_j)(\phi_j^{new} - \phi_j) + \mathcal{O}(\delta_x^2 + \delta_y^2) \\ f_y(\phi_0^{new}, \dots, \phi_{n-1}^{new}) &= \sum_{j=0}^{n-1} \sin(\phi_j) + \sum_{j=0}^{n-1} \cos(\phi_j)(\phi_j^{new} - \phi_j) + \mathcal{O}(\delta_x^2 + \delta_y^2). \end{aligned} \tag{3.15}$$

We approximate f_x, f_y by ignoring the higher order terms and find δ_x, δ_y so that these approximations of f_x, f_y are zero. That is, at every step of the iteration, we compute δ_x, δ_y so that

$$\begin{aligned} 0 &= \sum_{j=0}^{n-1} \cos(\phi_j) - \sum_{j=0}^{n-1} \sin(\phi_j)(\phi_j^{new} - \phi_j) \\ 0 &= \sum_{j=0}^{n-1} \sin(\phi_j) + \sum_{j=0}^{n-1} \cos(\phi_j)(\phi_j^{new} - \phi_j). \end{aligned} \tag{3.16}$$

Substituting (3.13) into (3.16) yields

$$\begin{aligned} 0 &= \sum_{j=0}^{n-1} \cos(\phi_j) - \delta_x \sum_{j=0}^{n-1} \sin(\phi_j)\cos(\phi_j) - \delta_y \sum_{j=0}^{n-1} \sin^2(\phi_j) \\ 0 &= \sum_{j=0}^{n-1} \sin(\phi_j) + \delta_x \sum_{j=0}^{n-1} \cos^2(\phi_j) + \delta_y \sum_{j=0}^{n-1} \cos(\phi_j)\sin(\phi_j). \end{aligned} \tag{3.17}$$

This can be written as the 2×2 linear system

$$\begin{pmatrix} \sum_{j=0}^{n-1} \sin(\phi_j)\cos(\phi_j) & \sum_{j=0}^{n-1} \sin^2(\phi_j) \\ -\sum_{j=0}^{n-1} \cos^2(\phi_j) & -\sum_{j=0}^{n-1} \cos(\phi_j)\sin(\phi_j) \end{pmatrix} \begin{pmatrix} \delta_x \\ \delta_y \end{pmatrix} = \begin{pmatrix} \sum_{j=0}^{n-1} \cos(\phi_j) \\ \sum_{j=0}^{n-1} \sin(\phi_j) \end{pmatrix}, \tag{3.18}$$

which we solve to obtain δ_x, δ_y. We iterate the procedure in (3.13) until f_x, f_y is sufficiently small. Being a version of the Newton algorithm, this procedure converges rapidly.

3.4 Reconstructing the Curve

Given $\theta(s)$ on $[0, L]$, we reconstruct the curve $\gamma(s) = \{x(s), y(s)\}$ using (2.7). The constants c_x, c_y will be determined later, so we first compute

$$\begin{aligned} \widetilde{x}(s) &= x(s) - c_x = \int_0^s \cos(\phi(\sigma))d\sigma \\ \widetilde{y}(s) &= y(s) - c_y = \int_0^s \sin(\phi(\sigma))d\sigma. \end{aligned} \tag{3.19}$$

Using the procedure described in Section 2.4, we obtain $\widetilde{x}(s)$ and $\widetilde{y}(s)$ at n equally-spaced points $\{\widetilde{x}_j, \widetilde{y}_j\}_{j=0,\ldots,n-1}$, where $\widetilde{x}_j = \widetilde{x}(s_j)$ and $\widetilde{y}_j = \widetilde{y}(s_j)$ for $s_j = \frac{j}{n} \cdot L$ and $j = 0, \ldots, n-1$. To evaluate the curve at intermediate points, we use Lagrange interpolation as described in Section 2.5.

Clearly, there are many ways in which c_x and c_y can be chosen so that some selected properties of the location of γ coincide with such properties of γ_{poly}, the polygon obtained from the input data. We choose c_x, c_y so that the center of mass of γ

$$\begin{aligned} \overline{x} &= \frac{1}{L}\int_0^L x(s)ds \\ \overline{y} &= \frac{1}{L}\int_0^L y(s)ds, \end{aligned} \tag{3.20}$$

coincides with the center of mass $(\overline{x}_{poly}, \overline{y}_{poly})$ of the polygon γ_{poly} viewed as a curve; a simple calculations shows that

$$\begin{aligned} \overline{x}_{poly} &= \sum_{i=1}^{m} \frac{x_{i+1} + x_i}{2}\sqrt{(x_{i+1} - x_i)^2 + (y_{i+1} - y_i)^2} \\ \overline{y}_{poly} &= \sum_{i=1}^{m} \frac{y_{i+1} + y_i}{2}\sqrt{(x_{i+1} - x_i)^2 + (y_{i+1} - y_i)^2}. \end{aligned} \tag{3.21}$$

Evaluating (3.20) via the Trapezoidal rule (see Observation 3) yields

$$\begin{aligned} \overline{x} &= \frac{1}{n}\sum_{j=0}^{n-1} x(s_j) = c_x + \frac{1}{n}\sum_{j=0}^{n-1}\widetilde{x}_j \\ \overline{y} &= \frac{1}{n}\sum_{j=0}^{n-1} y(s_j) = c_y + \frac{1}{n}\sum_{j=0}^{n-1}\widetilde{y}_j. \end{aligned} \tag{3.22}$$

Hence,

$$\begin{aligned} c_x &= \overline{x}_{poly} - \frac{1}{n}\sum_{j=0}^{n-1}\widetilde{x}_j \\ c_y &= \overline{y}_{poly} - \frac{1}{n}\sum_{j=0}^{n-1}\widetilde{y}_j. \end{aligned} \tag{3.23}$$

Observation 6. *Since the tangential angles θ_{poly} have been filtered, the reconstructed curve will not pass through the original data $S = \{x_i, y_i\}_{i=1,\dots,m}$. We denote by t_i the parameter value for which $\gamma(t_i) = \{x(t_i), y(t_i)\}$ is the point on the curve γ closest to (x_i, y_i). We compute t_i by minimizing the function*

$$f(s) = (x(s) - x_i)^2 + (y(s) - y_i)^2 \tag{3.24}$$

for $s \in [t_{i-1}, t_{i-1} + L]$ (thus, $t_1 < \cdots < t_m$).

3.5 Perturbations of the Curve

As a consequence of the filtering step, the reconstructed curve $\gamma(s) = (x(s), y(s))$, $s \in [0, L]$ does not pass through the original data $S = \{x_i, y_i\}_{i=1,\dots,m}$ (see Observation 6). To construct a curve $\tilde{\gamma}$ that passes through the points in S, we add a small analytic perturbation to γ.

We denote by $t_1 < \cdots < t_m$ the parameter values for which $\gamma(t_1), \dots, \gamma(t_m)$ are the points on the curve γ closest to S. We define the functions

$$g_j(s) = e^{-\sigma_j (\frac{\nu_j(s) - t_j}{L})^2} \tag{3.25}$$

where

$$\nu_j(s) = \begin{cases} s & t_j - \frac{L}{2} \le s \le t_j + \frac{L}{2} \\ s - L & s > t_j + \frac{L}{2} \\ s + L & s < t_j - \frac{L}{2} \end{cases} \tag{3.26}$$

for $s \in [0, L]$ and $j = 1, \dots, m$. Obviously, $|\nu_j(s) - t_j| \le \frac{L}{2}$ for all $s \in [0, L]$, since $\gamma(s)$ is a periodic function with period L.

While the curve γ does not pass through the points in S, the distance between them is usually small. To construct a curve $\widetilde{\gamma}(s) = \{\widetilde{x}(s), \widetilde{y}(s)\}$ so it passes through the points in S, we add the functions $g_j(s)$ to the curve $\gamma(s)$ at the points $t_1, \dots, t_m$ so that

$$\begin{aligned} \widetilde{x}(s) &= x(s) + \sum_{j=1}^{m} c_j \cdot g_j(s) \\ \widetilde{y}(s) &= y(s) + \sum_{j=1}^{m} d_j \cdot g_j(s), \end{aligned} \tag{3.27}$$

where the coefficients c_j, d_j for $j = 1, \dots, m$ are determined by solving the linear system

$$\begin{aligned} x_i &= x(t_i) + \sum_{j=1}^{m} c_j \cdot g_j(t_i) \\ y_i &= y(t_i) + \sum_{j=1}^{m} d_j \cdot g_j(t_i). \end{aligned} \tag{3.28}$$

By construction, the curve $\widetilde{\gamma}$ passes through the points in S. We define the matrix G via the formula

$$G_{ij} = g_j(t_i). \tag{3.29}$$

For small-scale problems (perhaps $m \le 200$), this linear system can be solved directly. For larger ones, iterative schemes like the method of conjugate gradients discussed in Section 2.6 are more appropriate.

3.5.1 Selection of the Parameters σ_i and the Solution of the Linear System

Clearly, the selection of $\{\sigma_j\}$ in (3.25) is critical. In particular, we have to achieve three seemingly contradictory goals:

1. The linear system must be solvable and, ideally, well-conditioned
2. The coefficients c_j, d_j should be small
3. The functions $g_j(t)$ should be smooth, so their Fourier coefficients decay fast

Clearly, we can choose functions $g_j(s)$ with very large σ_j so that the resulting linear system in (3.28) is nearly diagonal. Each function $g_j(s)$ is a Gaussian, so its Fourier coefficients have magnitude

$$|\hat{g}_k| = \sqrt{\frac{\pi}{\sigma_j}} e^{-\pi^2 \frac{k^2}{\sigma_j}}. \tag{3.30}$$

Hence, as σ_j becomes large, the rate of decay of $|\hat{g}_k|$ decreases. On the other hand, choosing σ_j to be too small can result in an ill-conditioned linear system, leading to large coefficients c_j, d_j.

Observation 7. *Once the functions $g_j(s)$ in (3.25) are added, the resulting curve $\widetilde{\gamma}(s)$, $s \in [0, L]$ in (3.27) is no longer parametrized by its arc-length, so the equally-spaced arc-length discretization of $\widetilde{\gamma}(s)$ will not correspond to the equally-spaced discretization of $g_j(s)$. While this change in sampling of $g_j(s)$ will impact the Fourier coefficients in (3.30), such "second order" effects can usually be ignored, except for the need to resample the curve once more (see Section 6.1).*

Observation 8. *The following selection of $\{\sigma_j\}$ has worked well in our experience.*

We choose $\sigma_1, \ldots, \sigma_m$ so that the linear system being solved is diagonally dominant to ensure that G is well-conditioned. Specifically, we choose $\sigma_1, \ldots, \sigma_m$ so that

$$2 \cdot \sum_{i \neq j} g_j(t_i) < g_j(t_j) = 1. \tag{3.31}$$

In this case, due to the Gershgorin circle theorem [13], the magnitude of all the eigenvalues of G will be in the range $[\frac{1}{2}, \frac{3}{2}]$ and hence G will be well-conditioned.

We also choose σ_j to be sufficiently large so that the function $g_j(s)$, a Gaussian centered at t_j, is negligible for $\nu_j(s) \notin [t_j - b, t_j + b]$ for some $b \leq \frac{L}{2}$. This allows the matrix G to be applied to an arbitrary vector for a cost of no more than $\mathcal{O}(m^2)$ operations.

To solve the linear system in (3.28), we observe that the matrix G is well-conditioned and, for large-scale problems, apply the standard method of conjugate gradients to the corresponding normal equations. Needless to say, for small-scale problems, the linear system (3.28) is solved directly.

Observation 9. *The straightforward evaluation of the curve $\widetilde{\gamma}(s) = \{\widetilde{x}(s), \widetilde{y}(s)\}$ in (3.27) at a single point costs $\mathcal{O}(m)$ operations if the curve $\gamma(s) = \{x(s), y(s)\}$ evaluated at a single point costs $\mathcal{O}(1)$ operations (due to Lagrange interpolation). Likewise, the sums*

$$\sum_{j=1}^{m} c_j \cdot g_j(s), \ \ \sum_{j=1}^{m} d_j \cdot g_j(s) \tag{3.32}$$

can be evaluated at a single point in $\mathcal{O}(1)$ operations with Lagrange interpolation, but requires a preprocessing step where these sums are first evaluated at many (e.g., n) equally-spaced points in the parameter s. This approach is particularly advantageous if the curve $\widetilde{\gamma}(s)$ is to be evaluated at a large number of points N, especially with $N \gg n$.

3.6 Summary of the Algorithm and its Cost

The steps of the algorithm are:

1. Given $S = \{x_i, y_i\}_{i=1,\ldots,m} \in \mathbb{R}^2$, construct the polygonal curve by connecting these points with straight lines.

2. Compute the tangential angles along the polygonal curve, $\theta_{poly}(s)$ in (3.2), and sample it at $n \gg m$ equally-spaced points to obtain $\{\theta_j\}$

3. Apply the DFT to obtain the Fourier coefficients $\{\hat{\theta}_k\}$, filter $\{\hat{\theta}_k\}$ to obtain $\{\hat{\phi}_k\}$, and apply the inverse of the DFT to obtain $\{\phi_j\}$

4. Close the curve using the iteration in (3.13)

5. Reconstruct the curve from its tangent angles via spectral integration

6. Solve the linear system (3.28)

The cost of linearly interpolating the data and discretizing it at n points is $\mathcal{O}(n)$. The cost of filtering $\{\theta_j\}$ is the cost of the FFT, which requires $\mathcal{O}(n \log n)$ operations. Clearly, closing the curve iteratively costs $\mathcal{O}(n)$ operations. Reconstruction of the curve via spectral integration is the cost of applying the FFT.

The cost of computing the coefficients of the Gaussians directly is $\mathcal{O}(m^3)$. If the method of conjugate gradients is used to solve the linear system, the cost is $\mathcal{O}(m^2)$. However, for most large-scale problems, the matrix G is banded (see Observation 8), so the cost is further reduced.

Observation 10. *In fact, evaluation of sums of Gaussians at multiple points is a well-studied problem (see [7]). The techniques of [7] can be used to reduce the cost of Step 6 above to $\mathcal{O}(m)$ operations. However, in our experiments, the cost of this calculation had not been excessive and this last step has not been implemented.*

4 Numerical Experiments

We applied the algorithm of Section 3.6 to several different data sets. The first set of data came from a sampling of the curve

$$\gamma(t) = \{\cos(t), \sin(t) + \alpha \cos^2(t)\} \tag{4.1}$$

where α is a tunable parameter. The other two are discretizations of curves with sharp corners, which are potentially challenging examples for this algorithm. The results are shown below.

Figure 4.1 shows the output of the algorithm for the curve in (4.1) with $\alpha = 1$ sampled at 128 points. For visual clarity, only some of the points are shown. In this case, the curve was upsampled to $n = 4096$ points and filtered with the Gaussian filter (2.22) with $\hat{g}_{32} = 0.1$. The magnitude of the Fourier coefficients of the tangential angle of the curve is displayed in Figure 4.2.

Figure 4.3 shows the output of the algorithm for the curve in (4.1) with $\alpha = 2$ sampled at 512 points. Again, only some of them are shown. In this case, the curve was upsampled to $n = 16384$ points and filtered with the Gaussian filter with $\hat{g}_{1024} = 0.1$. The magnitude of the Fourier coefficients of the tangential angle of the curve is displayed in Figure 4.4.

Figure 4.5 is the input data of a sparsely sampled contour with corners that need to be smoothed. In Figure 4.6, the curve was upsampled to $n = 16384$ points and filtered with a Gaussian filter with $\hat{g}_{256} = 0.1$. In this case, the filter was too strong resulting in a curve that is (arguably) too rounded or "cartoonish." The magnitude of the Fourier coefficients of the tangential angle of the curve is displayed in Figure 4.7. In Figure 4.8, the curve was upsampled to $n = 65536$ points and filtered with a Gaussian filter with $\hat{g}_{4096} = 0.1$. To the naked eye, the output appears to be the input data connected with straight lines. However, by zooming in on the corner in Figure 4.9, we can see that it is actually smoothed. The magnitude of the Fourier coefficients of the tangential angle of the curve is displayed in Figure 4.10. In both cases, the Fourier coefficients have converged, but the appropriate strength of the filter is clearly problem-dependent.

Finally, Figure 4.11 is contour with many corners that is heavily sampled. In this case, the curve was upsampled to $n = 32768$ points and filtered with a Gaussian filter with $\hat{g}_{4096} = 0.1$. The magnitude of the Fourier coefficients of the tangential angle of the curve is displayed in Figure 4.12.

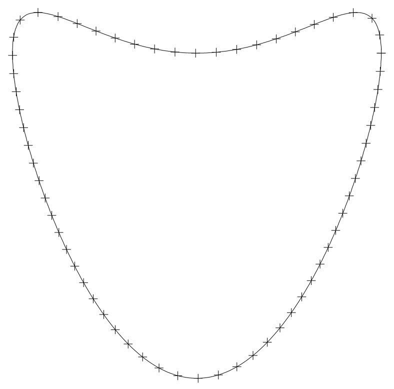

Figure 4.1: The crosses mark the input sampling of (4.1) with $\alpha = 1$. Not all input points are shown.

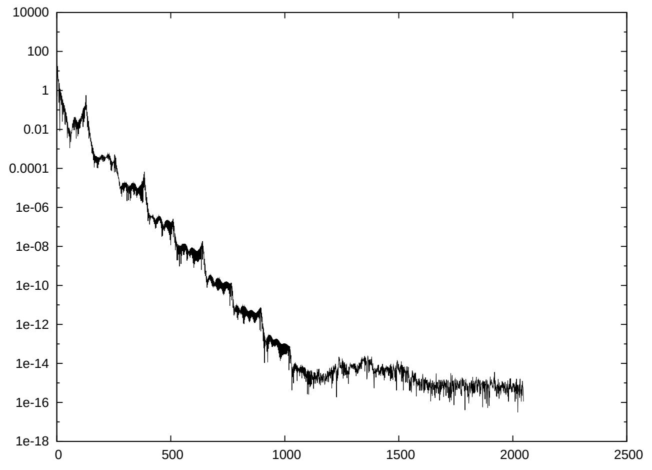

Figure 4.2: Fourier series of the tangential angle for Figure 4.1

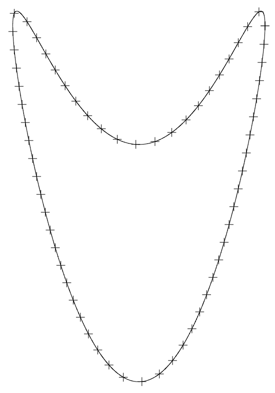

Figure 4.3: The crosses mark the input sampling of (4.1) with $\alpha = 2$. Not all input points are shown.

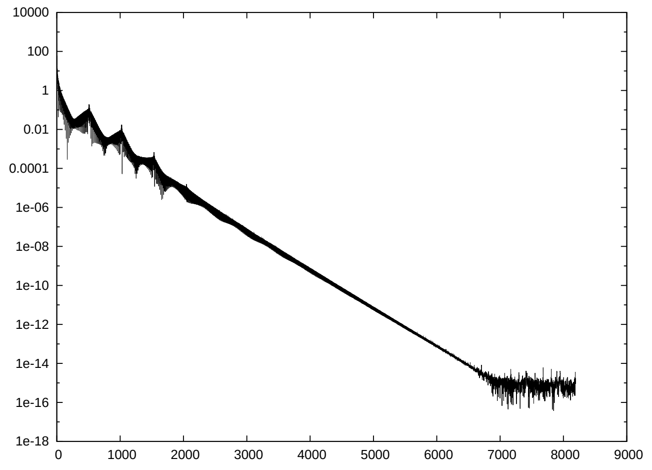

Figure 4.4: Fourier series of the tangential angle for Figure 4.3

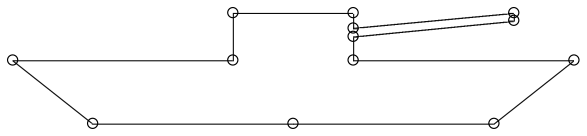

Figure 4.5: The circles mark the input data of the contour.

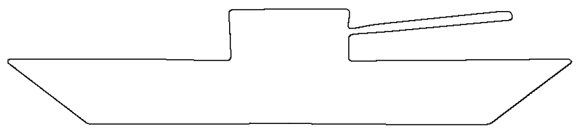

Figure 4.6: Overfiltered tank.

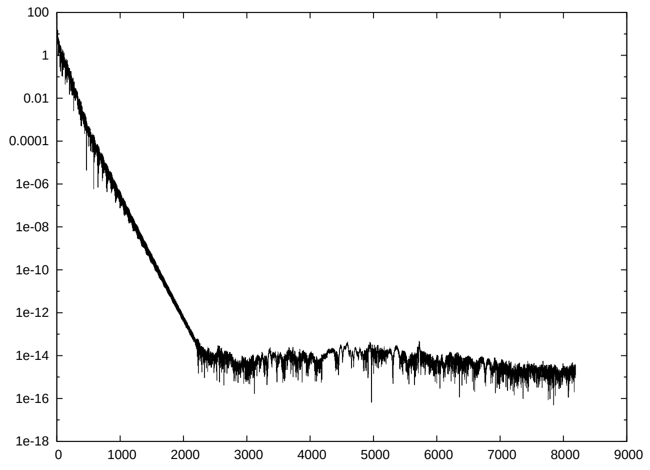

Figure 4.7: Fourier series of the tangential angle for Figure 4.6

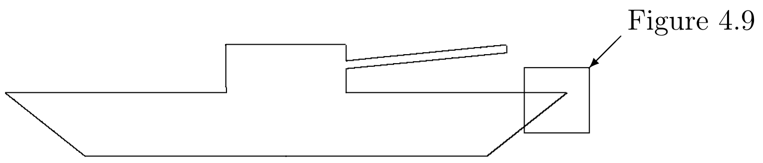

Figure 4.8: Mildly filtered tank - the filtering is not visible to the naked eye.

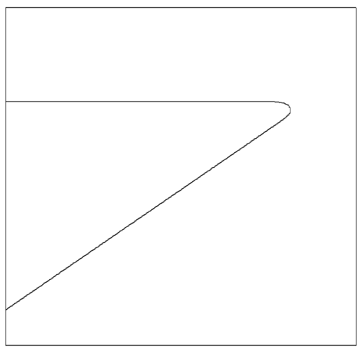

Figure 4.9: A closer look at a corner of the tank.

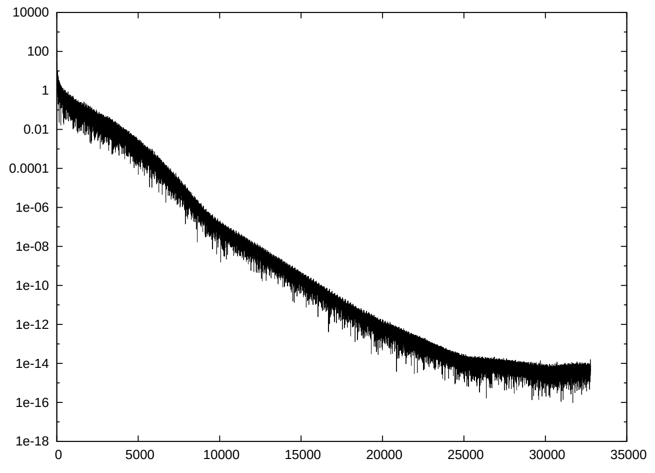

Figure 4.10: Fourier series of the tangential angle for Figure 4.8

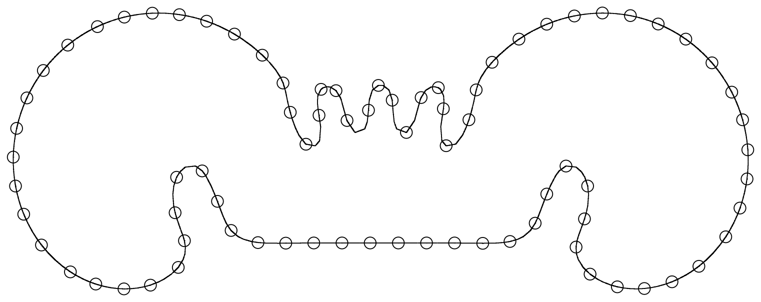

Figure 4.11: The circles mark the input sampling of a contour with many corners. Not all input points are shown.

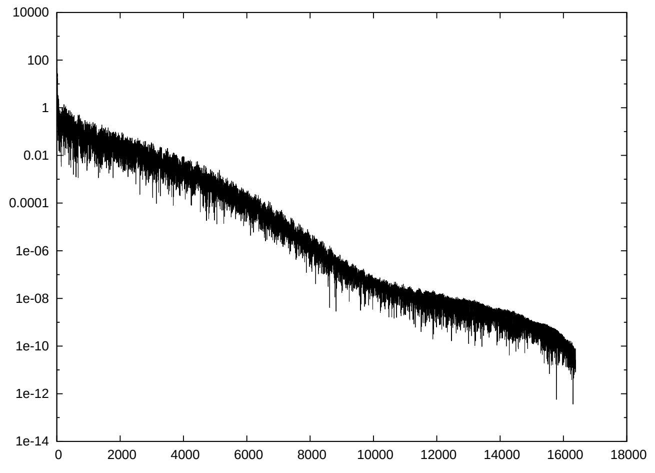

Figure 4.12: Fourier series of the tangential angle for Figure 4.11

Observation 11. *All of the above experiments have been run with the Kaiser window (2.23) under identical conditions. The results are virtually indistinguishable.*

5 Conclusions and Extensions

We have described an algorithm for the reparametrization of a planar closed curve defined by a sequence of m points. The algorithm constructs an analytic curve that passes through the m points while providing the user a high level of control over its frequency content. If n is the number of points in the initial upsampling of the curve, the asymptotic cost of the algorithm is $\mathcal{O}(n \log n)$; however, in practice, if an arc-length parametrization is demanded, the constants can be fairly large.

There are two obvious extensions of this algorithm: to curves with corners (i.e., planar curves that are not closed) and to surfaces in $\mathbb{R}^3$. For curves with corners with the desired tangential angles specified at their corner points, the trigonometric polynomials are replaced by bandlimited functions, often represented by linear combinations of prolate spheroidal wave functions [15]. The extension of this algorithm to surfaces in $\mathbb{R}^3$ using the fundamental forms of surfaces is more involved. The work on this is being vigorously pursued.

6 Appendix

6.1 A Simple Algorithm for the Resampling of Analytically Specified Curves

Given $\gamma(t) = \{x(t), y(t)\}$ and $\gamma'(t) = \{x'(t), y'(t)\}$, $t \in [0,1]$, where $x(t), y(t), x'(t), y'(t)$ are specified by exact formulas, we describe an algorithm to reparametrize the curve by its arc-length s.

1. Compute the arc-length of the entire curve (via adaptive Gaussian quadratures as described in Section 2.7)

$$L = \int_0^1 \left\|\gamma'(t)\right\| dt = \int_0^1 \sqrt{\left(x'(t)\right)^2 + \left(y'(t)\right)^2} dt. \tag{6.1}$$

2. Find the (rather large) n points in the parameter t which correspond to equally spaced points in the arc-length parameter. That is, starting with $t_0 = 0$, we find the points

$$t_1, \dots, t_n \tag{6.2}$$

such that for each $i = 1, \dots, n$

$$h = \frac{L}{n} = \int_{t_{i-1}}^{t_i} \sqrt{\left(x'(t)\right)^2 + \left(y'(t)\right)^2} dt. \tag{6.3}$$

We solve for each t_i via Newton's method. To initialize the Newton iteration, we use the Taylor expansion of the integral

$$\begin{aligned} \int_{t_{i-1}}^{t_{i-1}+\Delta} \left\|\gamma'(t)\right\| dt &= \int_{t_{i-1}}^{t_{i-1}} \left\|\gamma'(t)\right\| dt + \left\|\gamma'(t_{i-1})\right\| \Delta + \mathcal{O}(\Delta^2) \\ &= \left\|\gamma'(t_{i-1})\right\| \Delta + \mathcal{O}(\Delta^2) \end{aligned}, \tag{6.4}$$

which we would like to equal h. By ignoring the higher order terms, we obtain an initial approximation for $t_i = t_{i-1} + \Delta$, where

$$\Delta = \frac{h}{\left\|\gamma'(t_{i-1})\right\|}. \tag{6.5}$$

Alternatively, after the first few t_i have been computed, one can use the preceding results and extrapolate forward to obtain an initial guess. As a result of this step, we have a discretization of the mapping ξ from the arc-length s to the user-specified parameter t, i.e., we have

$$\xi(s_i) = t_i \tag{6.6}$$

for $i = 0, \dots, n$ where

$$s_i = i \cdot h \tag{6.7}$$

and t_i is given in (6.2).

3. To evaluate γ corresponding to a given arc-length parameter s, we find τ such that

$$s = \int_0^\tau \sqrt{\left(x'(t)\right)^2 + \left(y'(t)\right)^2} dt \tag{6.8}$$

via Newton's method. To initialize $\tau = \xi(s)$, we construct an approximation to the mapping ξ. Specifically, we determine the nearest points among $s_0, \dots, s_n$ to s and the corresponding $t_0, \dots, t_n$ to construct a polynomial approximation for $\xi(s)$. For example, if $s_j < s < s_{j+1}$ for some $0 \leq j < n$, the linear approximation of $\xi(s)$ is given by

$$\tau = \xi(s) \approx \frac{t_{j+1} - t_j}{s_{j+1} - s_j}(s - s_j) + t_j. \tag{6.9}$$

In our experience, a cubic approximation of $\xi(s)$ provides a good trade-off between the accuracy of the approximation and the cost of computing it.

Observation 12. *The above algorithm requires repeated integrations of* $\sqrt{(x'(t))^2 + (y'(t))^2}$ *over small intervals, which can naively be computed via adaptive Gaussian quadratures. In this case, in order to evaluate*

$$I = \int_a^b \sqrt{(x'(t))^2 + (y'(t))^2} dt, \tag{6.10}$$

the integral must be approximated on $[a, b]$ *as well as* $[a, \frac{a+b}{2}]$ *and* $[\frac{a+b}{2}, b]$ *(see Section 2.7), thus requiring at least three evaluations of the integral - although the interval* $[a, b]$ *might be small.*

In order to improve the speed of the computation of these integrals, we replace the adaptive Gaussian quadratures with non-adaptive Gaussian quadratures by only integrating over sufficiently small subintervals. Specifically, during the initial computation of the arc-length of the entire curve (in Step 1), we store the subintervals over which non-adaptive Gaussian quadratures are sufficiently accurate. We also store the corresponding value of the integral over this subinterval. If we denote the boundaries of these subintervals by $z_0 = 0 < z_1 < \cdots < z_p = 1$ *and the value of the integrals over these subintervals by* $r_1, r_2, \ldots, r_p$*, then*

$$r_i = \int_{z_{i-1}}^{z_i} \sqrt{(x'(t))^2 + (y'(t))^2} dt. \tag{6.11}$$

To compute I *where* $z_i < a < z_{i+1}$ *and* $z_j < b < z_{j+1}$*, we have*

$$I = \int_a^{z_{i+1}} \sqrt{(x'(t))^2 + (y'(t))^2} dt + \sum_{k=i+1}^{j-1} r_{k+1} + \int_{z_j}^b \sqrt{(x'(t))^2 + (y'(t))^2} dt \tag{6.12}$$

where the two integrals are computed via non-adaptive Gaussian quadratures to the desired accuracy. In fact, most often, $z_i < a < b < z_{i+1}$*, in which case*

$$I = \int_a^b \sqrt{(x'(t))^2 + (y'(t))^2} dt \tag{6.13}$$

can be computed non-adaptively. Hence, compared with evaluating all of the integrals adaptively, this approach reduces the cost of integration by nearly $\frac{2}{3}$.

References

[1] J.H. Ahlberg and E.N. Nilson. *The Theory of Splines and Their Applications.* Academic Press, 1967.

[2] Richard H. Bartels, John C. Beatty, and Brian A. Barsky. *An Introduction to Splines for Use in Computer Graphics and Geometric Modeling.* Morgan Kaufmann, 1987.

[3] Germund Dahlquist and Åke Björck. *Numerical Methods in Scientific Computing*, volume I. Society for Industrial and Applied Mathematics, 2008.

[4] Carl de Boor. *A Practical Guide to Splines.* Springer, revised edition, 1994.

[5] Paul Dierckx. *Curve and Surface Fitting with Splines.* Oxford University Press, 1993.

[6] A. Dutt and V. Rokhlin. Fast Fourier Transforms for Nonequispaced Data. *SIAM J. Sci. Comput.*, 14(6):1368–1393, 1993.

[7] Leslie Greengard and John Strain. The Fast Gauss Transform. *SIAM J. Sci. Stat. Comput.*, 12(1):79–94, 1991.

[8] R.W. Hamming. *Digital Filters.* Courier Dover Publications, third edition, 1989.

[9] J.E. Hansen. *Spherical Near-Field Antenna Measurements.* Peter Peregrinus Ltd., 1988.

[10] Erwin Kreysig. *Differential Geometry.* Courier Dover Publications, 1959.

[11] Alan V. Oppenheim and Ronald W. Schafer. *Discrete-Time Signal Processing.* Prentice Hall, third edition, 2010.

[12] D. Somasundaram. *Differential Geometry - A First Course.* Alpha Science International Ltd, 2005.

[13] J. Stoer and R. Bulirsch. *Introduction to Numerical Analysis.* Spring-Verlag, second edition, 1993.

[14] Lloyd N. Trefethen and David Bau, III. *Numerical Linear Algebra.* Society for Industrial and Applied Mathematics, 1997.

[15] H. Xiao, V. Rokhlin, and N. Yarvin. Prolate spheroidal wavefunctions, quadrature and interpolation. *Inverse Problems*, 17(4):805–838, 2001.