Title

Algorithm 952: PHquintic: A library of basic functions for the construction and analysis of planar quintic pythagorean-hodograph curves

Journal

ACM Transactions on Mathematical Software, 41(4)

ISSN

0098-3500

Authors

Dong, B
Farouki, RT

Publication Date

2015

DOI

10.1145/2699467

Peer reviewed

`PHquintic`: A library of basic functions for the construction and analysis of planar quintic Pythagorean–hodograph curves

Bohan Dong and Rida T. Farouki
Department of Mechanical and Aerospace Engineering,
University of California, Davis, CA 95616, USA

Abstract

The implementation of a library of basic functions for the construction and analysis of planar quintic Pythagorean–hodograph (PH) curves is presented, based on the complex representation. The special algebraic structure of PH curves permits exact algorithms for the computation of key properties, such as arc length, elastic bending energy, and offset (parallel) curves. Single planar PH quintic segments are constructed as interpolants to first–order Hermite data (end points and derivatives), and this construction is then extended to open or closed C^2 PH quintic spline curves interpolating a sequence of points in the plane. The non–linear nature of PH curves incurs a multiplicity of formal solutions to such interpolation problems, and a key aspect of the algorithms is to efficiently single out the unique "good" interpolant among them.

Keywords: Pythagorean–hodograph curves, computer–aided design, Hermite interpolation, spline interpolation, Newton–Raphson method, arc length, bending energy, offset curves, complex polynomials.

e–mail addresses: dddong@ucdavis.edu, farouki@ucdavis.edu

1 Introduction

In Cartesian coordinates, a plane curve may be specified through an implicit equation $f(x, y) = 0$ or a parametric equation $\mathbf{r}(t) = (x(t), y(t))$. The former amounts to defining a curve through a *predicate* function, whose evaluation indicates whether or not a given point is on the curve. The latter corresponds to a *generating* function, whose evaluation furnishes a sequence of points on the curve as the parameter t increases. Parametric representations are almost universally preferred for applications such as computer graphics, computer–aided design, robotics, and real–time motion control, since the task of tracing a curve based upon the implicit equation is a non–trivial problem.

However, curves parameterized by "simple" (i.e., polynomial or rational) functions are not without shortcomings. In general, the variation of the curve parameter t bears no relation to the curve geometry. Ideally, one would like to use the arc length s as the curve parameter, but the only curve that admits rational arc–length parameterization is a straight line [30]. Valuable progress toward this impossible ideal can nevertheless can be achieved by considering curves whose *parametric speed* $\sigma = \mathrm{d}s/\mathrm{d}t$ — the rate of change of arc length with respect to the parameter — is a polynomial or rational function.

This is the distinctive feature of the *Pythagorean–hodograph* (PH) *curves*, whose derivatives $\mathbf{r}'(t) = (x'(t), y'(t))$ have coordinate components satisfying [29] the Pythagorean condition

$$x'^2(t) + y'^2(t) \;=\; \sigma^2(t) \tag{1}$$

for some polynomial or rational function $\sigma(t)$. This special property endows PH curves with many important computational advantages over "ordinary" polynomial/rational curves. For example, they have rational *offset curves* — i.e., loci at each fixed distance d from a given curve, in the normal direction. The arc length of any segment of a (polynomial) PH curve can be computed by simply evaluating a polynomial, an invaluable property in motion control. In the design of free–form shapes by interpolation of discrete data, PH curves yield "fairer" shapes (with less curvature variation) than ordinary polynomial curves. Spatial PH curves admit rational *rotation–minimizing frames*, useful in computer animation, robotics, and spatial path planning.

This study presents the implementation of a suite of key algorithms for constructing, analyzing, and manipulating planar PH curves. The algorithms have attained a high degree of maturity in their development, and are thus ready for systematic codification and dissemination. The focus at present is

on the *polynomial* PH curves, that satisfy (1) with $x'(t), y'(t), \sigma(t)$ in the ring of real polynomials, and extensive use is made of the complex representation [9], which affords compact and efficient algorithm formulations.

The construction of *rational* PH curves, satisfying (1) with $x'(t), y'(t), \sigma(t)$ in the field of real rational functions, employs an entirely different approach [33, 43] based on the *dual representation* — i.e., the interpretation of a plane curve as the envelope of its tangent lines. Moreover, rational PH curves are less useful than polynomial PH curves in certain applications, since in general they do not have rational arc lengths (the integral of a rational function may incur transcendental terms). A more detailed discussion of the relationships between polynomial and rational PH curves may be found in [27].

Algorithms for spatial PH curves are also not addressed at present. These curves were first discussed in [31], but their proper characterization requires more sophisticated algebraic models, based [7, 15] on quaternions or the Hopf map from $\mathbb{R}^4$ to $\mathbb{R}^3$. Although considerable progress on algorithms for their construction has been made [16, 17, 23, 25] — including special PH curves with rational "rotation–minimizing" frames [14, 18, 19] — the algorithms are inherently more complicated than for planar PH curves, because of the need to determine appropriate values for certain free parameters. Since algorithms for spatial PH curves are still under active investigation, it seems premature to attempt a systematic codification and implementation at this time.

The algorithm descriptions provided below specify the input and output in each context, and provide a brief synopsis of the computations required to derive the latter from the former. These descriptions are by necessity skeletal in nature, but in each case appropriate references are provided so the reader can consult complete derivations and computational details. The monograph [13] provides a comprehensive, up–to–date single source for planar PH curves, and a much briefer introduction to them may be found in [12].

In recent years PH curves have found application in diverse contexts, such as real–time interpolator algorithms for computer numerical control (CNC) machines [21, 22, 32, 35]; spatial rigid–body motion design for animation and robotics [18, 19]; vision–based motion tracking systems [4]; mobile robot path planning [3] and cooperative path planning for unmanned aerial vehicles [41, 44, 45, 46]; reconstruction of planar shapes from medial axis transforms [6, 37, 38]; and artistic or aesthetic design [36]. It is hoped that this software library will encourage further use in a broad spectrum of applications.

Since the simplest non–trivial PH curves, the cubics, have limited shape flexibility, we focus herein on the PH quintics. These are the lowest–order PH

curves capable of inflecting, and interpolating arbitrary first–order Hermite data, and are therefore suitable for free–form design applications.

The remainder of this paper is organized as follows. After a brief review of the complex representation for planar PH curves in Section 2, it is used to derive some basic properties in Section 3 — namely, the parametric speed and arc length polynomials, and the unit tangent, normal, and curvature. The rational offset curves to a planar PH quintic $\mathbf{r}(t)$, i.e., the loci at a given fixed distance d from $\mathbf{r}(t)$ in the normal direction, are discussed in Section 4. An exact algorithm to compute the bending energy (the integral of the square of curvature with respect to arc length) for planar PH quintics is then presented in Section 5. The first–order PH quintic Hermite interpolants, discussed in Section 6, highlight a characteristic feature of all PH curve constructions — a multiplicity of formal solutions (due to the non–linear defining equations), and the need to identify the "good" interpolant among them. In Section 7 the two–point Hermite interpolation problem is generalized to the interpolation of a sequence of $N+1$ points in the plane, under prescribed end conditions, by a C^2 PH quintic spline curve. This incurs a (sparse) system of quadratic equations in complex variables, which must be solved by iterative methods, and an appropriate identification of starting values is critical for efficient and robust convergence on the "good" interpolant. Section 8 describes an interactive implementation of the PH curve construction and analysis algorithms, based on the MFC and OpenGL libraries. Finally, Section 9 briefly describes the accompanying software packages, and Section 10 summarizes the key capabilities and potential applications of the `PHquintic` library.

2 Complex representation

We focus here on *primitive* PH curves with $\gcd(x'(t), y'(t)) = \text{constant}$. Then a sufficient and necessary condition for satisfaction of (1) is that $x'(t)$, $y'(t)$ should be expressible in terms of relatively prime real polynomials $u(t)$, $v(t)$ in the form

$$x'(t) \;=\; u^2(t) - v^2(t)\,, \quad y'(t) \;=\; 2u(t)v(t)\,,$$

and hence $\sigma(t) = u^2(t) + v^2(t)$. The *complex representation* [9] succinctly embodies this structure. Regarding plane curves as complex–valued functions $\mathbf{r}(t) = x(t) + \mathrm{i}\, y(t)$ of a real parameter t [50], the PH curves are those curves whose derivatives correspond to the *perfect square* $\mathbf{r}'(t) = \mathbf{w}^2(t)$ of a complex

polynomial[1] $\mathbf{w}(t) = u(t) + \mathrm{i}\, v(t)$ with $\gcd(u(t), v(t)) = \text{constant}$.

Given a degree–m complex polynomial

$$\mathbf{w}(t) \;=\; u(t) + \mathrm{i}\, v(t) \;=\; \sum_{k=0}^{m} \mathbf{w}_k \binom{m}{k} (1-t)^{m-k} t^k \tag{2}$$

with Bernstein coefficients $\mathbf{w}_k = u_k + \mathrm{i}\, v_k$ for $k = 0, \ldots, m$, a PH curve of degree $n = 2m + 1$ is obtained by integration of $\mathbf{r}'(t) = \mathbf{w}^2(t)$. The control points $\mathbf{p}_0, \ldots, \mathbf{p}_n$ in the Bézier representation

$$\mathbf{r}(t) \;=\; \sum_{k=0}^{n} \mathbf{p}_k \binom{n}{k} (1-t)^{n-k} t^k$$

of this PH curve are determined entirely by the complex values $\mathbf{w}_0, \ldots, \mathbf{w}_m$ (with $\mathbf{p}_0$ being an arbitrary integration constant). The Bernstein polynomial form is used exclusively here, on account of its inherent numerical stability [28] and its close relation to the Bézier curve representation [8]. A library of functions for Bernstein–form polynomials has been presented in [47].

A straight line is trivially a PH curve, corresponding to the choice $\mathbf{w}(t) =$ constant. The simplest non–trivial PH curves are cubics, defined by a linear polynomial $\mathbf{w}(t)$. The PH cubics can be characterized by simple geometrical constraints on their Bézier control polygons [29], and correspond to scaled, rotated, and re–parameterized segments of a unique curve — the *Tschirnhaus cubic*. Since PH cubics have limited shape freedoms, we focus henceforth on the PH quintics, the lowest–order PH curves that may inflect and interpolate first–order Hermite data. Choosing the quadratic polynomial

$$\mathbf{w}(t) \;:=\; \mathbf{w}_0(1-t)^2 + \mathbf{w}_1 2(1-t)t + \mathbf{w}_2 t^2 \tag{3}$$

and integrating $\mathbf{r}'(t) = \mathbf{w}^2(t)$, one finds that the PH quintics are specified by

[1] Bold symbols are used to denote both complex values and points or vectors in $\mathbb{R}^2$.

control points of the form

$$\begin{aligned}
\mathbf{p}_1 &= \mathbf{p}_0 + \frac{1}{5}\,\mathbf{w}_0^2\,, \\
\mathbf{p}_2 &= \mathbf{p}_1 + \frac{1}{5}\,\mathbf{w}_0\mathbf{w}_1\,, \\
\mathbf{p}_3 &= \mathbf{p}_2 + \frac{1}{5}\,\frac{2\mathbf{w}_1^2 + \mathbf{w}_0\mathbf{w}_2}{3}\,, \\
\mathbf{p}_4 &= \mathbf{p}_3 + \frac{1}{5}\,\mathbf{w}_1\mathbf{w}_2\,, \\
\mathbf{p}_5 &= \mathbf{p}_4 + \frac{1}{5}\,\mathbf{w}_2^2\,. \qquad (4)
\end{aligned}$$

The complex representation greatly simplifies many fundamental algorithms for planar PH curves, and is systematically used herein. The data defining a single PH quintic segment is encapsulated in the following `C` struct.

```
struct PHquintic {
complex p[6] ; /* Bezier control points of PH quintic */
complex w[3] ; /* Bernstein coefficients of w(t) polynomial */
double sigma[5] ; /* parametric speed Bernstein coefficients */
double s[6] ; /* arc length Bernstein coefficients */
} ;
```

Here the complex arrays `p[6]` and `w[3]` store the control points $\mathbf{p}_0, \ldots, \mathbf{p}_5$ and the coefficients $\mathbf{w}_0, \mathbf{w}_1, \mathbf{w}_2$ of the quadratic polynomial (3), while the real arrays `sigma[5]` and `s[6]` store the coefficients of the parametric speed and arc length polynomials (see Section 3 below). This specification is redundant, since the other data can be reconstructed from $\mathbf{p}_0$ and $\mathbf{w}_0, \mathbf{w}_1, \mathbf{w}_2$ alone. However, pre–computing and storing the full contents of a `PHquintic` struct can save considerable effort in subsequent usage of the PH quintic it defines.

3 Arc length, tangent, curvature

The parametric speed, tangent, and curvature of the PH curve $\mathbf{r}(t)$ defined by integrating $\mathbf{r}'(t) = \mathbf{w}^2(t)$ may be expressed [9] as

$$\sigma(t) \;=\; |\mathbf{w}(t)|^2\,, \quad \mathbf{t}(t) \;=\; \frac{\mathbf{w}^2(t)}{\sigma(t)}\,, \quad \kappa(t) \;=\; 2\,\frac{\mathrm{Im}(\overline{\mathbf{w}}(t)\mathbf{w}'(t))}{\sigma^2(t)}\,.$$

The parametric speed is a polynomial in t, while the tangent and curvature are rational functions of t. The unit normal, $\mathbf{n}(t) = -\,\mathrm{i}\,\mathbf{t}(t)$, is also rational. Since $\sigma(t)$ is a polynomial, the cumulative arc length

$$s(t) \;:=\; \int_0^t \sigma(\xi)\;\mathrm{d}\xi$$

is likewise a polynomial in t. For a PH quintic, the parametric speed is the quartic polynomial

$$\sigma(t) \;=\; |\mathbf{w}(t)|^2 \;=\; \sum_{k=0}^{4} \sigma_k \binom{4}{k} (1-t)^{4-k} t^k$$

with Bernstein coefficients

$$\begin{aligned} \sigma_0 &= |\mathbf{w}_0|^2\,, \\ \sigma_1 &= \mathrm{Re}(\mathbf{w}_0 \overline{\mathbf{w}}_1)\,, \\ \sigma_2 &= \frac{2\,|\mathbf{w}_1|^2 + \mathrm{Re}(\mathbf{w}_0 \overline{\mathbf{w}}_2)}{3}\,, \\ \sigma_3 &= \mathrm{Re}(\mathbf{w}_1 \overline{\mathbf{w}}_2)\,, \\ \sigma_4 &= |\mathbf{w}_2|^2\,, \end{aligned} \tag{5}$$

and the arc length is the quintic polynomial

$$s(t) \;=\; \sum_{k=0}^{5} s_k \binom{5}{k} (1-t)^{5-k} t^k$$

with Bernstein coefficients defined by $s_0 = 0$ and

$$s_k \;=\; \frac{1}{5} \sum_{j=0}^{k-1} \sigma_j\,, \quad k = 1, \ldots, 5\,.$$

The length of any segment $t \in [\,t_1, t_2\,]$ of a PH quintic is simply $s(t_2) - s(t_1)$. The polynomial nature of the parametric speed facilitates the formulation of real–time interpolator algorithms for motion control applications [21, 22, 32].

4 Offset curves

The *offset* (or *parallel*) *curve* at a fixed normal distance d from a given plane curve $\mathbf{r}(t)$ is the locus defined by

$$\mathbf{r}_d(t) \;:=\; \mathbf{r}(t) \;+\; d\,\mathbf{n}(t)\,,$$

where $\mathbf{n}(t)$ is the unit normal to $\mathbf{r}(t)$. For general polynomial/rational curves, the offset $\mathbf{r}_d(t)$ is not rational, because the unitization of $\mathbf{n}(t)$ involves division by $|\mathbf{r}'(t)| = \sqrt{x'^2(t) + y'^2(t)}$. For a PH curve, however, $\mathbf{r}_d(t)$ can be exactly specified as a rational curve for any d, since $\mathbf{n}(t)$ depends rationally on t.

In the case of a PH quintic $\mathbf{r}(t)$, the offsets are degree 9 rational curves. Let the control points of $\mathbf{r}(t)$ be expressed in homogeneous coordinates as

$$\mathbf{P}_k \;=\; (1, x_k + \mathrm{i}\, y_k)\,, \quad k = 0, \ldots, 5\,,$$

with forward differences

$$\Delta\mathbf{P}_k \;:=\; \mathbf{P}_{k+1} - \mathbf{P}_k \;=\; (0, \Delta x_k + \mathrm{i}\,\Delta y_k)\,, \quad k = 0, \ldots, 4\,.$$

The offset curve can then be expressed as

$$\mathbf{r}_d(t) \;=\; \frac{X(t) + \mathrm{i}\, Y(t)}{W(t)}$$

where $W(t)$, $X(t) + \mathrm{i}\, Y(t)$ are polynomials of degree 9, whose coefficients

$$\mathbf{O}_k \;=\; (W_k, X_k + \mathrm{i}\, Y_k)\,, \quad k = 0, \ldots, 9$$

define the Bézier control points of the rational offset curve. The homogeneous coordinates for the control points of $\mathbf{r}_d(t)$ may be expressed [29] in terms of those of $\mathbf{r}(t)$ and the parameteric speed coefficients (5) as

$$\mathbf{O}_k \;=\; \sum_{j=\max(0,k-5)}^{\min(4,k)} \frac{\binom{4}{j}\binom{5}{k-j}}{\binom{9}{k}} (\sigma_j \mathbf{P}_{k-j} - \mathrm{i}\, 5d\, \Delta\mathbf{P}_j)\,, \quad k = 0, \ldots, 9\,.$$

The binomial coefficient factor can be simplified by noting that

$$\frac{\binom{4}{j}\binom{5}{k-j}}{\binom{9}{k}} \;=\; \frac{\binom{k}{j}\binom{9-k}{4-j}}{\binom{9}{4}} \;=\; \frac{\binom{k}{j}\binom{9-k}{4-j}}{126}\,,$$

and the homogeneous coordinates for the offset curve control points can thus be expressed for $k = 0, \ldots, 9$ as

$$W_k = \frac{1}{126} \sum_{j=\max(0,k-5)}^{\min(4,k)} \binom{k}{j} \binom{9-k}{4-j} \sigma_j \,,$$

$$X_k = \frac{1}{126} \sum_{j=\max(0,k-5)}^{\min(4,k)} \binom{k}{j} \binom{9-k}{4-j} (\sigma_j X_{k-j} + 5\, d\, \Delta Y_j) \,,$$

$$Y_k = \frac{1}{126} \sum_{j=\max(0,k-5)}^{\min(4,k)} \binom{k}{j} \binom{9-k}{4-j} (\sigma_j Y_{k-j} - 5\, d\, \Delta X_j) \,.$$

The format of the function that computes the offset to a single PH quintic segment is as follows. The struct `curve` passes the data that defines the PH quintic to the function, the value `d` defines the (signed) offset distance, and the homogeneous coordinates of the control points for the degree 9 rational offset curve are returned in the arrays `W[]`, `X[]`, `Y[]`.

```
void PHquintic_offset( double d ,
                       struct PHquintic *curve ,
                       double W[] , double X[] , double Y[] )
```

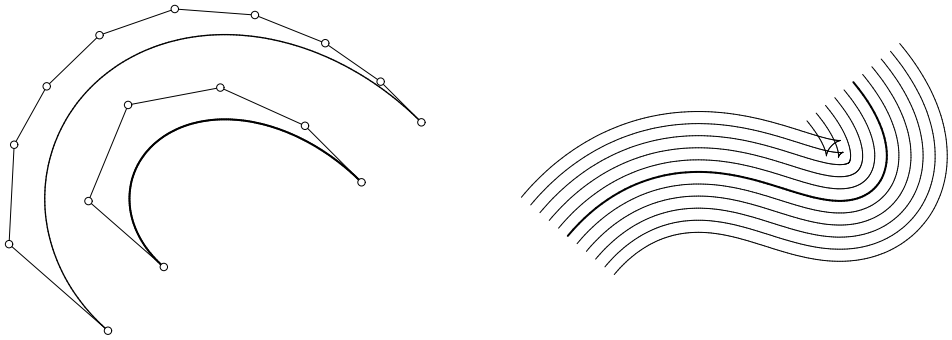

Figure 1: Left: the offset to a PH quintic as a degree 9 rational Bézier curve. Right: two–sided family of offsets to a PH quintic (the "swallowtails" develop when the offset distance exceeds the smallest concave radius of curvature).

In some cases it may be preferable to treat $\mathbf{r}_d(t)$ as simply the sum of the PH quintic $\mathbf{r}(t)$ and the rational vector $d\,\mathbf{n}(t)$ of degree 4, rather than

explicitly constructing the control points of the degree 9 rational Bézier form. Figure 1 illustrates the rational offset curves to a planar PH quintic.

5 Elastic bending energy

When an initially–straight elastic beam is bent into a curved shape, the strain energy is proportional to the integral of the squared curvature with respect to arc length. Minimizing this integral, under specified geometrical constraints, is often used as a criterion to ensure fair curve shapes [8]. Unlike "ordinary" polynomial curves, the PH curves admit closed–form evaluation of the strain energy integral through a partial fraction decomposition of the integrand.

For a PH curve of total arc length S, constructed by integrating $\mathbf{r}'(t) = \mathbf{w}^2(t)$, the strain energy integral may be expressed as

$$U \;:=\; \int_0^S \kappa^2 \, \mathrm{d}s \;=\; 4 \int_0^1 \frac{\mathrm{Im}^2(\overline{\mathbf{w}}(t)\mathbf{w}'(t))}{|\mathbf{w}(t)|^6} \, \mathrm{d}t \,. \tag{6}$$

Complete details on evaluating this integral for the case of the PH quintics, corresponding to the choice (3) of $\mathbf{w}(t)$, may be found in [10]. Here we simply summarize the steps required for implementation. It is convenient to re–write (3) in the form

$$\mathbf{w}(t) \;:=\; \mathbf{k}\,(t-\mathbf{a})(t-\mathbf{b})\,,$$

where $\mathbf{k} = \mathbf{w}_2 - 2\mathbf{w}_1 + \mathbf{w}_0$ and the roots of $\mathbf{w}(t)$ are

$$\mathbf{a}, \mathbf{b} \;=\; \frac{\mathbf{w}_0 - \mathbf{w}_1 \pm \sqrt{\mathbf{w}_1^2 - \mathbf{w}_0\mathbf{w}_2}}{\mathbf{w}_2 - 2\mathbf{w}_1 + \mathbf{w}_0} \,.$$

Defining $-\pi < \arg(\mathbf{z}) \leq +\pi$ for any complex number $\mathbf{z}$, the definite integral (6) may then be expressed [10] as

$$\begin{aligned} U \;=\; \frac{4}{|\mathbf{k}|^2} \Bigg\{ & 2\,\mathrm{Re}(\mathbf{a}_1) \ln \frac{|1-\mathbf{a}|}{|\mathbf{a}|} \;-\; 2\,\mathrm{Im}(\mathbf{a}_1)\,[\,\arg(1-\mathbf{a}) - \arg(-\mathbf{a})\,] \\ & + 2\,\mathrm{Re}(\mathbf{b}_1) \ln \frac{|1-\mathbf{b}|}{|\mathbf{b}|} \;-\; 2\,\mathrm{Im}(\mathbf{b}_1)\,[\,\arg(1-\mathbf{b}) - \arg(-\mathbf{b})\,] \\ & - \mathrm{Re}\left[\frac{2\,\mathbf{a}_2}{\mathbf{a}(1-\mathbf{a})} + \frac{2\,\mathbf{b}_2}{\mathbf{b}(1-\mathbf{b})} + \frac{(2\mathbf{a}-1)\,\mathbf{a}_3}{\mathbf{a}^2(1-\mathbf{a})^2} + \frac{(2\mathbf{b}-1)\,\mathbf{b}_3}{\mathbf{b}^2(1-\mathbf{b})^2} \right] \Bigg\} \,, \end{aligned}$$

where, with $\alpha = \mathrm{Im}(\mathbf{a})$ and $\beta = \mathrm{Im}(\mathbf{b})$, we define $\mathbf{a}_1, \mathbf{a}_2, \mathbf{a}_3$ and $\mathbf{b}_1, \mathbf{b}_2, \mathbf{b}_3$ by

$$
\begin{aligned}
\mathbf{a}_3 &= \frac{\mathrm{i}}{8\,\alpha\,(\mathbf{a}-\mathbf{b})(\mathbf{a}-\overline{\mathbf{b}})}\,, \\
\mathbf{b}_3 &= \frac{\mathrm{i}}{8\,\beta\,(\mathbf{a}-\mathbf{b})(\overline{\mathbf{a}}-\mathbf{b})}\,, \\
\mathbf{a}_2 &= \left[\,\frac{3\mathrm{i}}{2\alpha} + \frac{1}{\mathbf{a}-\mathbf{b}} - \frac{3}{\mathbf{a}-\overline{\mathbf{b}}}\,\right]\mathbf{a}_3\,, \\
\mathbf{b}_2 &= \left[\,\frac{3\mathrm{i}}{2\beta} - \frac{1}{\mathbf{a}-\mathbf{b}} + \frac{3}{\overline{\mathbf{a}}-\mathbf{b}}\,\right]\mathbf{b}_3\,, \\
\mathbf{a}_1 &= \frac{3\mathrm{i}}{2\alpha}\,\mathbf{a}_2 + \left[\,\frac{3}{4\alpha^2} - \frac{2}{(\mathbf{a}-\mathbf{b})^2} + \frac{6}{(\mathbf{a}-\overline{\mathbf{b}})^2} - \frac{1-2\beta/\alpha}{(\mathbf{a}-\mathbf{b})(\mathbf{a}-\overline{\mathbf{b}})}\,\right]\mathbf{a}_3\,, \\
\mathbf{b}_1 &= \frac{3\mathrm{i}}{2\beta}\,\mathbf{b}_2 + \left[\,\frac{3}{4\beta^2} - \frac{2}{(\mathbf{a}-\mathbf{b})^2} + \frac{6}{(\overline{\mathbf{a}}-\mathbf{b})^2} - \frac{1-2\alpha/\beta}{(\mathbf{a}-\mathbf{b})(\overline{\mathbf{a}}-\mathbf{b})}\,\right]\mathbf{b}_3\,.
\end{aligned}
$$

It is also possible to derive an indefinite integral expression $U(t)$, allowing the bending energy of any segment $t \in [\,t_1, t_2\,]$ to be determined as $U(t_2) - U(t_1)$ — see [10] for complete details.

The format of the function that computes the bending energy of a single PH quintic segment is as follows. The struct `curve` passes the data defining the PH quintic to the function, and the computed bending energy is returned as the value of the function.

```
double PHquintic_energy( struct PHquintic *curve )
```

6 First–order Hermite interpolants

The first–order Hermite interpolation problem entails construction of a PH quintic $\mathbf{r}(t)$, $t \in [\,0, 1\,]$ with prescribed initial and final points and derivatives: $\mathbf{r}(0) = \mathbf{p}_i$, $\mathbf{r}'(0) = \mathbf{d}_i$ and $\mathbf{r}(1) = \mathbf{p}_f$, $\mathbf{r}'(1) = \mathbf{d}_f$. On substituting (3) into $\mathbf{r}'(t) = \mathbf{w}^2(t)$, interpolation of the end derivatives yields

$$
\mathbf{w}_0^2 = \mathbf{d}_i \quad \text{and} \quad \mathbf{w}_2^2 = \mathbf{d}_f\,. \tag{7}
$$

Also, integrating $\mathbf{r}'(t) = \mathbf{w}^2(t)$ with $\mathbf{r}(0) = \mathbf{p}_i$ gives the equation

$$
\mathbf{w}_0^2 + \mathbf{w}_0\mathbf{w}_1 + \frac{2\mathbf{w}_1^2 + \mathbf{w}_0\mathbf{w}_2}{3} + \mathbf{w}_1\mathbf{w}_2 + \mathbf{w}_2^2 = 5\,\Delta\mathbf{p}\,, \tag{8}
$$

where $\Delta\mathbf{p} := \mathbf{p}_f - \mathbf{p}_i$. Writing $\mathbf{d}_i = d_i \exp(\mathrm{i}\phi_i)$ and $\mathbf{d}_f = d_f \exp(\mathrm{i}\phi_f)$, equations (7) have the solutions

$$\mathbf{w}_0 \;=\; \eta_0 \sqrt{d_i}\exp(\mathrm{i}\tfrac{1}{2}\phi_i) \quad \text{and} \quad \mathbf{w}_2 \;=\; \eta_2 \sqrt{d_f}\exp(\mathrm{i}\tfrac{1}{2}\phi_f)\,, \tag{9}$$

where $\eta_0 = \pm 1$ and $\eta_2 = \pm 1$. Regarding (8) as a quadratic equation

$$2\,\mathbf{w}_1^2 \;+\; 3\,(\mathbf{w}_0 + \mathbf{w}_2)\,\mathbf{w}_1 \;+\; 3\,(\mathbf{w}_0^2 + \mathbf{w}_2^2) + \mathbf{w}_0\mathbf{w}_2 - 15\,\Delta\mathbf{p} \;=\; 0$$

in $\mathbf{w}_1$, it has the solutions

$$\mathbf{w}_1 \;=\; -\,\frac{3}{4}\,(\mathbf{w}_0 + \mathbf{w}_2) \;+\; \frac{\eta_1}{4}\,\sqrt{120\,\Delta\mathbf{p} - 15\,(\mathbf{w}_0^2 + \mathbf{w}_2^2) + 10\,\mathbf{w}_0\mathbf{w}_2}\,, \tag{10}$$

where $\eta_1 = \pm 1$. The quantities η_0, η_1, η_2 reflect three independent sign choices arising in the solution, so one might expect eight distinct curves. However, if $\mathbf{w}_0, \mathbf{w}_1, \mathbf{w}_2$ is any particular solution, then $-\mathbf{w}_0, -\mathbf{w}_1, -\mathbf{w}_2$ is also a solution, defining exactly the same curve. Hence, there are in general just four distinct interpolants, which may be generated by choosing $\eta_1 = 1$ and exercising the four sign combinations represented by $\eta_0 = \pm 1$ and $\eta_2 = \pm 1$.

Since $\mathbf{p}_1 = \mathbf{p}_0 + \frac{1}{5}\,\mathbf{r}'(0)$ and $\mathbf{p}_4 = \mathbf{p}_5 - \frac{1}{5}\,\mathbf{r}'(1)$, one may regard the input to this problem as specifying the initial and final pairs of control points — $\mathbf{p}_0, \mathbf{p}_1$ and $\mathbf{p}_4, \mathbf{p}_5$ — and the algorithm then "fills in" the control points $\mathbf{p}_2, \mathbf{p}_3$ so as to obtain a PH quintic. This interpretation offers a control–polygon approach to the construction of PH quintics. Since it is perhaps more intuitive than specifying end derivatives, we adopt it in the implementation.

A number of strategies for selecting the η_0, η_2 combination that identifies the "good" Hermite interpolant have been proposed, including minimization of the *absolute rotation index* [26]; *absence of anti–parallel tangents* compared to the "ordinary" cubic interpolant [39]; and analysis of the *winding number* of the closed loop formed by the hodographs of the PH quintic and "ordinary" cubic [5]. Here we choose the "good" interpolant to be the solution with the least value of the absolute rotation index, defined by

$$R_{\mathrm{abs}} \;=\; \frac{1}{2\pi}\,\int_0^1 |\kappa(t)|\,\sigma(t)\;\mathrm{d}t\,. \tag{11}$$

R_{abs} measures the "total turning" of the curve tangent — i.e., clockwise and anti–clockwise tangent rotations do not cancel each other.

To compute $R_{\rm abs}$, the input points $\mathbf{p}_0, \mathbf{p}_1, \mathbf{p}_4, \mathbf{p}_5$ are first transformed [26] to *canonical form*[2] data $\mathbf{q}_0, \mathbf{q}_1, \mathbf{q}_4, \mathbf{q}_5$ defined by $\mathbf{q}_i = (\mathbf{p}_i - \mathbf{p}_0)/(\mathbf{p}_5 - \mathbf{p}_0)$, so that $\mathbf{q}_0 = 0$, $\mathbf{q}_5 = 1$. The Hermite interpolation problem is then solved using $\mathbf{q}_0, \mathbf{q}_1, \mathbf{q}_4, \mathbf{q}_5$, and for each solution $\mathbf{k}, \mathbf{a}, \mathbf{b}$ are defined in terms of $\mathbf{w}_0, \mathbf{w}_1, \mathbf{w}_2$ as in Section 5. We then have

$$R_{\rm abs} \;=\; \frac{1}{\pi}\,(\angle\, 0\,\mathbf{a}\,1 + \angle\, 0\,\mathbf{b}\,1)\,,$$

when $\mathbf{a}$ and $\mathbf{b}$ have imaginary parts of the same sign. If the imaginary part of $\mathbf{a}$ and $\mathbf{b}$ are of opposite sign, then

$$R_{\rm abs} \;=\; \frac{1}{\pi}\sum_{k=0}^{N} |\,\angle\, t_k\,\mathbf{a}\,t_{k+1} - \angle\, t_k\,\mathbf{b}\,t_{k+1}\,|\,,$$

where $t_0 = 0$, $t_{N+1} = 1$ and $t_1, \ldots, t_N$ $(N \le 2)$ are the ordered roots[3] on $t \in (0,1)$ of the (real) quadratic equation

$$\mathrm{Im}(\mathbf{a}+\mathbf{b})\,t^2 \;-\; 2\,\mathrm{Im}(\mathbf{ab})\,t \;+\; \mathrm{Im}(|\mathbf{a}|^2\mathbf{b} + |\mathbf{b}|^2\mathbf{a}) \;=\; 0\,. \qquad (12)$$

Here $\angle\, u\,\mathbf{z}\,v$ is the angle at the vertex $\mathbf{z}$ of a complex–plane triangle whose other vertices are at the values u, v on the real axis. To restore the canonical–form solution to the original coordinate system, the values $\mathbf{w}_0, \mathbf{w}_1, \mathbf{w}_2$ must be multiplied by $\sqrt{\mathbf{p}_5 - \mathbf{p}_0}$ (either complex root can be used). The control points of the PH quintic Hermite interpolant can then be computed from (4). Figure 2 shows some examples of PH quintics constructed in this manner.

The format of the function that constructs a single PH quintic segment is as follows. The complex variables `p0`, `p1`, `p4`, `p5` pass the initial and final pairs of control points to the function, and the data defining the constructed PH quintic is returned in the struct `curve`.

```
void construct_PHquintic( complex double p0 ,
                          complex double p1 ,
                          complex double p4 ,
                          complex double p5 ,
                          struct PHquintic *curve )
```

[2]Note that reduction to canonical form amounts to a scaling/rotation transformation, which does not alter the value of $R_{\rm abs}$.

[3]These roots identify the inflections of the PH quintic Hermite interpolant on $t \in (0,1)$.

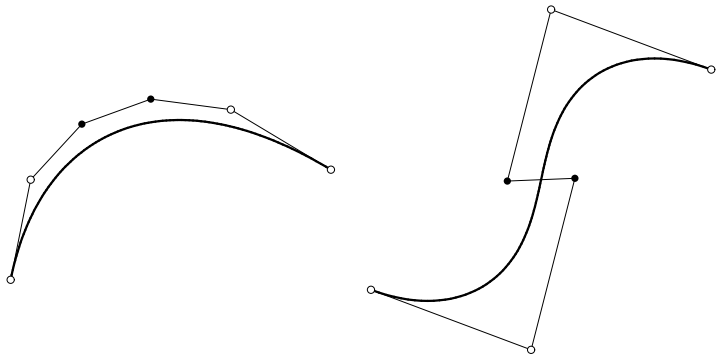

Figure 2: Some examples of PH quintic interpolants to first–order Hermite data — the user specifies the initial and final pairs of control points (circles), and the algorithm determines the interior pair of control points (solid dots).

Various generalizations of the first–order Hermite interpolation problem are possible, based on the additional degrees of freedom obtained by relaxing from *parametric* to *geometric* continuity (i.e., unit end tangents, rather than derivative vectors, are imposed). For example, this makes possible the design of *spiral segments*, with monotone curvature variation [11, 34, 48, 49].

7 C^2 PH quintic spline curves

In the construction of "ordinary" C^2 cubic splines interpolating given points $\mathbf{q}_0, \ldots, \mathbf{q}_N$, each segment $\mathbf{r}_i(t)$, $t \in [\,0, 1\,]$ for $i = 1, \ldots, N$ is expressed in the cubic Hermite basis with end points $\mathbf{r}_i(0) = \mathbf{q}_{i-1}$, $\mathbf{r}_i(1) = \mathbf{q}_i$ and (unknown) end derivatives $\mathbf{r}_i'(0) = \mathbf{d}_{i-1}$, $\mathbf{r}_i'(1) = \mathbf{d}_i$. The second derivative continuity condition $\mathbf{r}_i''(1) = \mathbf{r}_{i+1}''(0)$ for $i = 1, \ldots, N-1$ at the interior nodes, together with the prescribed end conditions, then defines a tridiagonal linear system of equations for the indeterminate nodal derivatives $\mathbf{d}_0, \ldots, \mathbf{d}_N$.

To construct a C^2 PH quintic spline interpolating the given points under specified end conditions, the derivative $\mathbf{r}_i'(t)$ of each segment is expressed in terms of complex unknowns $\mathbf{z}_{i-1}, \mathbf{z}_i, \mathbf{z}_{i+1}$ as the square of a complex quadratic polynomial, such that first and second derivative continuity, $\mathbf{r}_i'(1) = \mathbf{r}_{i+1}'(0)$ and $\mathbf{r}_i''(1) = \mathbf{r}_{i+1}''(0)$ for $i = 1, \ldots, N-1$, is *automatically* achieved. Choosing

integration constants $\mathbf{r}_i(0) = \mathbf{q}_{i-1}$, the displacement conditions

$$\int_0^1 \mathbf{r}_i'(t)\, \mathrm{d}t \;=\; \Delta\mathbf{q}_i \;:=\; \mathbf{q}_i - \mathbf{q}_{i-1} \tag{13}$$

for $i = 1, \ldots, N$ then define a system of N quadratic equations in the complex unknowns $\mathbf{z}_1, \ldots \mathbf{z}_N$, with only three consecutive variables in each equation (to eliminate the undefined variables $\mathbf{z}_0$ and $\mathbf{z}_N$, the first and last equations must be modified in accordance with the chosen end conditions).

The non–linear nature of the equations that define C^2 PH quintic splines implies a multiplicity of formal solutions. The number of distinct[4] solutions is 2^{N-1} for cubic end spans; 2^N periodic end conditions; and 2^{N+1} for prescribed end derivatives. An initial study [1] focused on investigating the entire family of formal solutions, computed using the homotopy method [2, 40]. A unique "good" solution is observed among them, while the others exhibit extreme curvature variations or undesired "looping" behavior. In practice, an efficient method to compute *only* the good solution is desired. This can be achieved using iterative methods [20], but a starting approximation close to the good solution is essential for rapid and reliable convergence to it.

7.1 C^2 PH quintic spline equations

Writing the derivative of the PH quintic segment $\mathbf{r}_i(t)$, $t \in [\,0, 1\,]$ of the spline curve, between $\mathbf{q}_{i-1}$ and $\mathbf{q}_i$, in the complex form

$$\mathbf{r}_i'(t) \;:=\; [\; \tfrac{1}{2}(\mathbf{z}_{i-1} + \mathbf{z}_i)(1-t)^2 + \mathbf{z}_i\, 2(1-t)t + \tfrac{1}{2}(\mathbf{z}_i + \mathbf{z}_{i+1})t^2 \;]^2 \tag{14}$$

ensures that successive spans i and $i+1$ satisfy the continuity conditions

$$\mathbf{r}_i'(1) = \mathbf{r}_{i+1}'(0) \qquad \text{and} \qquad \mathbf{r}_i''(1) = \mathbf{r}_{i+1}''(0)\,.$$

With $\mathbf{r}_i(0) = \mathbf{q}_{i-1}$, substituting (14) into (13) and evaluating the integral yields the equation

$$\begin{aligned} \mathbf{f}_i(\mathbf{z}_1, \ldots, \mathbf{z}_N) \;:=\; & 3\,\mathbf{z}_{i-1}^2 \;+\; 27\,\mathbf{z}_i^2 \;+\; 3\,\mathbf{z}_{i+1}^2 \;+\; \mathbf{z}_{i-1}\mathbf{z}_{i+1} \\ & +\; 13\,\mathbf{z}_{i-1}\mathbf{z}_i \;+\; 13\,\mathbf{z}_i\mathbf{z}_{i+1} \;-\; 60\,\Delta\mathbf{q}_i \;=\; 0\,. \end{aligned} \tag{15}$$

Such an equation arises from each span $i = 1, \ldots, N$ of the spline curve, but the first and last equations, $\mathbf{f}_1(\mathbf{z}_1, \ldots, \mathbf{z}_N) = 0$ and $\mathbf{f}_N(\mathbf{z}_1, \ldots, \mathbf{z}_N) = 0$, must be modified to reflect the chosen end conditions.

[4] If $\mathbf{z}_1, \ldots, \mathbf{z}_N$ is a solution, so is $-\mathbf{z}_1, \ldots, -\mathbf{z}_N$, that defines the same curve. To avoid replication we require $\mathrm{Re}(\mathbf{z}_m) > 0$, where m is the smallest index such that $\mathrm{Re}(\mathbf{z}_m) \neq 0$.

7.2 End conditions

End conditions are imposed to eliminate $\mathbf{z}_0$ and $\mathbf{z}_{N+1}$ from instances $i = 1$ and $i = N$ of (14). Cubic end spans and periodic end conditions, appropriate to open and closed C^2 spline curves, are discussed here. An alternative is the imposition of specified initial and final derivatives, $\mathbf{r}'_1(0) = \mathbf{d}_i$ and $\mathbf{r}'_N(1) = \mathbf{d}_f$ — see [20]. But this approach is generally not recommended unless there is *a priori* knowledge of suitable choices for $\mathbf{d}_i$ and $\mathbf{d}_f$.

With cubic end spans, the first and last spline segments $\mathbf{r}_1(t)$ and $\mathbf{r}_N(t)$ are actually PH cubics rather than quintics. The conditions $\mathbf{z}_0 - 2\,\mathbf{z}_1 + \mathbf{z}_2 = 0$ and $\mathbf{z}_{N-1} - 2\,\mathbf{z}_N + \mathbf{z}_{N+1} = 0$, characterizing this circumstance, are employed to eliminate $\mathbf{z}_0$ and $\mathbf{z}_{N+1}$. The first and last equations then assume the form

$$\begin{aligned} \mathbf{f}_1(\mathbf{z}_1, \ldots, \mathbf{z}_N) &:= 13\,\mathbf{z}_1^2 + \mathbf{z}_2^2 - 2\,\mathbf{z}_1\mathbf{z}_2 - 12\,\Delta\mathbf{q}_1 = 0\,, \\ \mathbf{f}_N(\mathbf{z}_1, \ldots, \mathbf{z}_N) &:= 13\,\mathbf{z}_N^2 + \mathbf{z}_{N-1}^2 - 2\,\mathbf{z}_N\mathbf{z}_{N-1} - 12\,\Delta\mathbf{q}_N = 0\,. \end{aligned} \tag{16}$$

For a C^2 closed curve with $\mathbf{r}_N(1) = \mathbf{r}_1(0)$ — i.e., $\mathbf{q}_N = \mathbf{q}_0$ — periodic end conditions imply that $\mathbf{r}'_N(1) = \mathbf{r}'_1(0)$ and $\mathbf{r}''_N(1) = \mathbf{r}''_1(0)$. To achieve this, $\mathbf{z}_1, \ldots, \mathbf{z}_N$ is viewed as a cyclical list, and on setting $\mathbf{z}_0 = \mathbf{z}_N$ and $\mathbf{z}_{N+1} = \mathbf{z}_1$, the first and last equations become

$$\begin{aligned} \mathbf{f}_1(\mathbf{z}_1, \ldots, \mathbf{z}_N) &:= 3\,\mathbf{z}_N^2 + 27\,\mathbf{z}_1^2 + 3\,\mathbf{z}_2^2 + \eta\,\mathbf{z}_N\mathbf{z}_2 \\ &\quad + 13\,\eta\,\mathbf{z}_N\mathbf{z}_1 + 13\,\mathbf{z}_1\mathbf{z}_2 - 60\,\Delta\mathbf{q}_1 = 0\,, \\ \mathbf{f}_N(\mathbf{z}_1, \ldots, \mathbf{z}_N) &:= 3\,\mathbf{z}_{N-1}^2 + 27\,\mathbf{z}_N^2 + 3\,\mathbf{z}_1^2 + \eta\,\mathbf{z}_{N-1}\mathbf{z}_1 \\ &\quad + 13\,\mathbf{z}_{N-1}\mathbf{z}_N + 13\,\eta\,\mathbf{z}_N\mathbf{z}_1 - 60\,\Delta\mathbf{q}_N = 0\,, \end{aligned} \tag{17}$$

where $\eta = \pm 1$ (to be determined in computing the starting approximation: see §7.4 below). In this case, the linear system is not strictly tridiagonal.

7.3 Iterative solution

The *Jacobian matrix* $\mathbf{M}$ for the system (15) is defined by the elements

$$\mathbf{M}_{ij} := \frac{\partial\,\mathbf{f}_i}{\partial\,\mathbf{z}_j}\,, \quad 1 \le i, j \le N\,. \tag{18}$$

For rows $i = 2, \ldots, N-1$, the only non–zero elements are

$$\begin{aligned} \mathbf{M}_{i,i-1} &= 6\,\mathbf{z}_{i-1} + 13\,\mathbf{z}_i + \mathbf{z}_{i+1}\,, \\ \mathbf{M}_{ii} &= 13\,\mathbf{z}_{i-1} + 54\,\mathbf{z}_i + 13\,\mathbf{z}_{i+1}\,, \\ \mathbf{M}_{i,i+1} &= \mathbf{z}_{i-1} + 13\,\mathbf{z}_i + 6\,\mathbf{z}_{i+1}\,. \end{aligned}$$

For cubic end spans, the non–zero elements on rows $i = 1$ and $i = N$ are

$$\mathbf{M}_{11} \;=\; 26\,\mathbf{z}_1 - 2\,\mathbf{z}_2\,, \quad \mathbf{M}_{12} \;=\; 2\,\mathbf{z}_2 - 2\,\mathbf{z}_1\,,$$

$$\mathbf{M}_{N,N-1} \;=\; 2\,\mathbf{z}_{N-1} - 2\,\mathbf{z}_N\,, \quad \mathbf{M}_{NN} \;=\; 26\,\mathbf{z}_N - 2\,\mathbf{z}_{n-1}\,,$$

and for periodic end conditions, they are

$$\begin{aligned}
\mathbf{M}_{11} &= 13\,\eta\,\mathbf{z}_N + 54\,\mathbf{z}_1 + 13\,\mathbf{z}_2\,,\\
\mathbf{M}_{12} &= \eta\,\mathbf{z}_N + 13\,\mathbf{z}_1 + 6\,\mathbf{z}_2\,,\\
\mathbf{M}_{1N} &= 6\,\mathbf{z}_N + 13\,\eta\,\mathbf{z}_1 + \eta\,\mathbf{z}_2\,,
\end{aligned}$$

$$\begin{aligned}
\mathbf{M}_{N1} &= \eta\,\mathbf{z}_{N-1} + 13\,\eta\,\mathbf{z}_N + 6\,\mathbf{z}_1\,,\\
\mathbf{M}_{N,N-1} &= 6\,\mathbf{z}_{N-1} + 13\,\mathbf{z}_N + \eta\,\mathbf{z}_1\,,\\
\mathbf{M}_{NN} &= 13\,\mathbf{z}_{N-1} + 54\,\mathbf{z}_N + 13\,\eta\,\mathbf{z}_1\,.
\end{aligned}$$

Writing $\mathbf{z} = (\mathbf{z}_1, \ldots, \mathbf{z}_N)^T$ and $\mathbf{f} = (\mathbf{f}_1, \ldots, \mathbf{f}_N)^T$, the Newton–Raphson iteration for the solution of the system (15) may be expressed as

$$\mathbf{z}^{(r+1)} \;:=\; \mathbf{z}^{(r)} \;+\; \delta\mathbf{z}^{(r)}\,, \quad r = 0, 1, 2, \ldots \tag{19}$$

where the increment vector $\delta\mathbf{z}^{(r)} = (\delta\mathbf{z}_1^{(r)}, \ldots, \delta\mathbf{z}_N^{(r)})^T$ is obtained by solving the linear system

$$\mathbf{M}^{(r)}\,\delta\mathbf{z}^{(r)} \;=\; -\;\mathbf{f}^{(r)}\,. \tag{20}$$

The superscripts on $\mathbf{M}^{(r)}$ and $\mathbf{f}^{(r)}$ indicate evaluation at $\mathbf{z}^{(r)} = (\mathbf{z}_0^{(r)}, \ldots, \mathbf{z}_N^{(r)})$. A suitable starting approximation $\mathbf{z}^{(0)} = (\mathbf{z}_1^{(0)}, \ldots, \mathbf{z}_N^{(0)})$ is required, and the iteration is truncated when the relative error measure

$$e_{r+1} \;:=\; \frac{\|\mathbf{z}^{(r+1)} - \mathbf{z}^{(r)}\|_2}{\|\mathbf{z}^{(r)}\|_2} \tag{21}$$

falls below a prescribed tolerance ϵ.

With cubic end spans, the Jacobian $\mathbf{M}$ for the system (15) is tridiagonal, and the increment $\delta\mathbf{z}^{(r)}$ in each Newton–Raphson step can be computed with $O(N)$ cost. For periodic end conditions, $\mathbf{M}$ is tridiagonal except for the non–zero elements $\mathbf{M}_{1N}$ and $\mathbf{M}_{N1}$, but $\delta\mathbf{z}^{(r)}$ can still be computed with $O(N)$ cost by simple modification[5] of the usual tridiagonal algorithm. Coupled with the

[5] A description of the standard tridiagonal solver, and its modification to accommodate periodic splines, may be found in Chapter 14 of [13].

quadratic convergence of the Newton iterations, this furnishes a very efficient approach to solving the system (15), provided a starting approximation $\mathbf{z}^{(0)} = (\mathbf{z}_1^{(0)}, \ldots, \mathbf{z}_N^{(0)})$ sufficiently close to the "good" solution is known.

Once the values $\mathbf{z}_1, \ldots, \mathbf{z}_N$ are determined, the Bézier control points (4) for each segment $\mathbf{r}_i(t)$ can be obtained by taking $\mathbf{p}_0 = \mathbf{q}_{i-1}$ and setting

$$\mathbf{w}_0 = \tfrac{1}{2}(\mathbf{z}_{i-1} + \mathbf{z}_i)\,, \quad \mathbf{w}_1 = \mathbf{z}_i\,, \quad \mathbf{w}_2 = \tfrac{1}{2}(\mathbf{z}_i + \mathbf{z}_{i+1})\,,$$

with appropriate modifications for the initial and final spans $\mathbf{r}_1(t)$ and $\mathbf{r}_N(t)$.

7.4 Starting approximation

The choice of the starting approximation $\mathbf{z}^{(0)} = (\mathbf{z}_1^{(0)}, \ldots, \mathbf{z}_N^{(0)})$ is critical in ensuring that the iterations (19) converge rapidly to the "good" solution of the system (15). The approach adopted here is based on equating mid–point derivatives of the PH quintic spline to those of the "ordinary" cubic spline interpolating the prescribed points with analogous end conditions. The latter derivatives satisfy the linear system

$$\mathbf{d}_{i-1} \,+\, 4\,\mathbf{d}_i \,+\, \mathbf{d}_{i+1} \,=\, 3\,(\mathbf{q}_{i+1} - \mathbf{q}_{i-1}) \quad i = 2, \ldots, N-1\,, \tag{22}$$

augmented by appropriate end conditions. Having solved for $\mathbf{d}_0, \ldots, \mathbf{d}_N$, the mid–point derivative matching condition yields the system of equations

$$\mathbf{z}_{i-1} \,+\, 6\mathbf{z}_i \,+\, \mathbf{z}_{i+1} \,=\, 4\,\sqrt{\mathbf{Q}_i}\,, \quad i = 2, \ldots, N-1\,, \tag{23}$$

where we introduce the quantities

$$\mathbf{Q}_i \,:=\, 6\Delta\mathbf{q}_i - (\mathbf{d}_{i-1} + \mathbf{d}_i)\,, \quad i = 1, \ldots, N\,. \tag{24}$$

For cubic end spans, this is augmented by

$$\mathbf{z}_1 \,=\, \tfrac{1}{2}\sqrt{\mathbf{Q}_1}\,, \qquad \mathbf{z}_N \,=\, \tfrac{1}{2}\sqrt{\mathbf{Q}_N}\,, \tag{25}$$

and in the case of periodic end conditions we use

$$\eta\,\mathbf{z}_N \,+\, 6\mathbf{z}_1 \,+\, \mathbf{z}_2 \,=\, 4\,\sqrt{\mathbf{Q}_1}\,, \qquad \mathbf{z}_{N-1} \,+\, 6\mathbf{z}_N \,+\, \eta\,\mathbf{z}_1 \,=\, 4\,\sqrt{\mathbf{Q}_N}\,. \tag{26}$$

The complex quantities (24) each have two square roots. Either root is chosen for $\sqrt{\mathbf{Q}_1}$, and each subsequent root is chosen by requiring $\sqrt{\mathbf{Q}_{i-1}}$ and $\sqrt{\mathbf{Q}_i}$,

regarded as vectors in $\mathbb{R}^2$, to possess a positive dot product for $i = 2, \ldots, N$. Finally, η is determined by the sign of the dot product of $\sqrt{\mathbf{Q}_N}$ and $\sqrt{\mathbf{Q}_1}$.

Numerical experiments show that using the solution to equations (23) and (25) or (26) as the starting approximation $(\mathbf{z}_1^{(0)}, \ldots, \mathbf{z}_N^{(0)})$ typically yields rapid and reliable convergence (within 4–5 Newton–Raphson iterations) to machine precision on the good interpolant, for "reasonable" point data $\mathbf{q}_0, \ldots, \mathbf{q}_N$. A discussion of the Kantorovich conditions, that guarantee convergence of the iteration (19) from any starting point inside a domain that contains a unique solution of the system (15), may be found in [20].

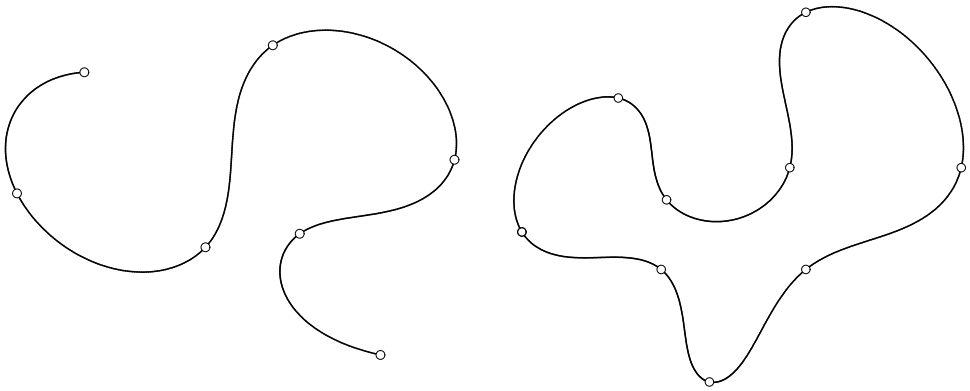

Figure 3: Left: an open C^2 PH quintic spline interpolating seven points with cubic end spans. Right: a closed C^2 PH quintic spline interpolating ten points under periodic end conditions (the first and last points being coincident).

Figure 3 shows examples of open and closed PH quintic splines, computed as described above with the convergence tolerance set at $\epsilon = 10^{-12}$. The rapid convergence of the Newton–Raphson iteration to machine precision is evident in the behavior of the error measure (21) in these two cases:

$$
\begin{array}{ll}
e_1 = 0.065195832156581 & e_1 = 0.064269028908590 \\
e_2 = 0.002028778363706 & e_2 = 0.001756440418895 \\
e_3 = 0.000002832072272 & e_3 = 0.000001570833518 \\
e_4 = 0.000000000011494 & e_4 = 0.000000000001339 \\
e_5 = 0.000000000000000 & e_5 = 0.000000000000000\,.
\end{array}
$$

7.5 Implementation

The format of the functions used to construct open and closed C^2 PH quintic splines is as follows. The integer `n` defines the number of points (namely, `n`+1) to be interpolated, passed to the function through the array of complex values `q[]` — the points are labelled `q[0]`,...,`q[n]`. The `n` PH quintic segments defining the constructed spline curve are returned through the struct array `spline[]` — these segments are labelled `spline[1]`,...,`spline[n]`.

```
void open_PHquintic_spline( int n ,
                            complex double q[] ,
                            struct PHquintic spline[] )

void closed_PHquintic_spline( int n ,
                              complex double q[] ,
                              struct PHquintic spline[] )
```

These call the functions `tridiag_open()` and `tridiag_closed()` to solve the tridiagonal system arising in each Newton–Raphson iteration. The inputs to these functions are the dimension `n` of the system, arrays `a[]`, `b[]`, `c[]` defining the lower, main, and upper diagonal matrix elements, and the array `d[]` of right–hand side values. The solutions are returned in the array `x[]`.

The function `beval()` is a further basic utility, that receives as input the degree `n` and array of Bernstein coefficients `b[]` of a polynomial, together with an independent variable value `t`, and returns the polynomial value computed by the de Castlejau algorithm. The maximum allowed degree `n` is 100.

7.6 Extensions and generalizations

A number of useful extensions can be implemented by simple modifications of the basic C^2 PH quintic spline. As described above, the algorithm assumes a *uniform parameterization* — i.e., each spline segment is defined on the unit parameter interval $t \in [\,0, 1\,]$. This is appropriate in cases where $\mathbf{q}_0, \ldots, \mathbf{q}_N$ have approximately uniform spacing, but when they are unevenly spaced it is well–known [8] that parameter intervals Δt_i proportional to the distances $|\,\mathbf{q}_i - \mathbf{q}_{i-1}|$ yield better interpolants. C^2 PH quintic splines with non–uniform parameterization correspond to solutions of a modification of the system (15), in which the terms of these equations are multiplied by certain real constants: complete details of the appropriate modification may be found in [20].

A further extension involves the *shape–preserving* PH splines [24], which can be made to preserve certain shape properties (monotonicity, convexity, etc.) of the point data by the introduction of "tension parameters." Finally, we mention a scheme for designing C^2 PH spline curves by means of control polygons [42] — the control polygon is used to determine certain nodal points that can serve as input to the interpolation algorithm described above. The resulting PH quintic splines closely resemble the "ordinary" cubic B–spline curves defined by the prescribed control polygon.

8 Interactive implementation

An interactive implementation of the various planar PH quintic construction and analysis functions was developed for the Windows environment, based on the Microsoft Foundation Class (MFC) Library and Open Graphics Library (OpenGL). For compatibility with these libraries, the implementation is in `C++`, and in some cases this required modifications of the interfaces to the `C` functions described in Sections 4–7. This interactive implementation allows the user to input the point data defining a single PH quintic segment or a C^2 PH quintic interpolating spline by mouse, and to modify the resulting curve in real time by using the mouse to move these points. Key properties of the resulting PH curves (arc length, bending energy, etc.) are reported, and the offset curves can also be constructed. For cases in which the point data must be precisely specified, the user can type in the coordinates.

Individual PH quintic curve segments are defined by the `PHquintic` struct described in Section 2, and C^2 PH quintic splines are specified as arrays of `PHquintic` structs. Class `PlanarPH` defines the basic functions for computing and analyzing these curves. To ensure a high level of precision, all complex variables in the class are type double.

```
PlanarPH(const CPointPH m_pt[]);
PlanarPH(const CPointPH m_pt[], int index);
```

The overloading constructors (for a single PH quintic Hermite interpolant and a PH quintic spline, respectively) convert the point coordinates specified by the user from type `CPointPH` to complex double. The `CPoint` type captured in the window display is long int, so the point type is redefined as `CPointPH`, which gives a representation of plane coordinates in type double.

The `beval()` function described in Section 7 is used to plot the PH curves. This function is reloaded to calculate the values of polynomials with *complex* Bernstein coefficients, to obtain the curve points directly. The friend function is a non–member function, but can access the private and protected members. Friend utility functions are typically used to allow mutual access to private or protected members of different classes. It also allows other classes to invoke functions using a concise syntax — e.g., `iter = getIter(pph)`, rather than `iter = pph.getIter()`.

```
friend double beval(int n, const double b[], double t);
friend complex<double> beval(int n, const complex<double> b[],
                                                    double t);
```

The function `Spline()` computes a C^2 PH quintic spline interpolating `num`+1 points labelled $0, \ldots,$ `num` under specified end conditions. The coordinates of these points are stored in the complex array `q[]`. The data that defines the `num` PH quintic segments of the constructed spline curve are returned in the `PHquintic` struct array `spline[]`. The Boolean parameter `closed` specifies the type of end conditions (cubic end spans or periodic end conditions, for open and closed curves, respectively). The two functions `tridiag_open()` and `tridiag_closed()`, described in Section 7, solve the tridiagonal systems appropriate to these cases.

```
void Spline(BOOL closed);
friend void tridiag_open(...);
friend void tridiag_closed(...);
```

The following functions define various interfaces for passing data. External functions must call `getCtrlPt` to obtain the complex values that define the control points. `getIter` returns the number of Newton–Raphson iterations employed in the spline construction, stored in the variable `iter`. Similarly, the functions `getParaSpeed`, `getArcLength`, and `getEnergy` compute the parametric speed, arc length, and bending energy.

```
friend complex<double> getCtrlPt(const PlanarPH &pph, int i, int j);
friend int getIter(const PlanarPH &pph);
friend double getParaSpeed(const PlanarPH &pph, int i, double t);
friend double getArcLength(const PlanarPH &pph, int i, double t);
friend double getEnergy(const PlanarPH &pph, int j);
```

In order to plot the constructed PH curves on the screen, OpenGL must first be initialized in the view class of MFC — this is named `PlanarPH_mfcView`, inherited from the base class `CView`.

```
int index, status, pointing;
CPointPH o_ActRF, o_ActRFTrans;
CPointPH ptActRF[Max+1];
CPointPH getAbsCoord(CPointPH pt);
CPointPH getActCoord(CPointPH pt);
double Distance(CPointPH p1, CPointPH p2);
```

The variable `index` (=`num`+1) records the number of input points, and `status` tracks the program mode (plotting, analyzing, translating, etc). The variable `pointing` identifies a point selected by the user after the curve is plotted, and the selected point is highlighted (`pointing` $= -1$ if no point is selected).

The program employs two coordinate systems, absolute and relative, to solve translation and display problems. For absolute coordinates, the window display selects the left upper window corner as the origin, with a downward ordinate direction. However, OpenGL selects the left lower corner as origin, with an upward ordinate direction. In both cases, the abscissa direction is to the right. The origin of the relative coordinates, in the absolute coordinate system, is defined by `o_ActRF`. All calculations are performed in the relative coordinates, including cursor location and PH curve computations. The array `ptActRF[]` stores the relative coordinates of the input points. By calling the functions `getabsCoord()` and `getActCoord`, the coordinates of any point can be transformed between the relative and absolute systems. The function `Distance()` returns the distance between two points.

```
PlanarPH pph;
void InitPH();
void DrawLine(int nIndex, int flag);
```

Before rendering, the function `InitPH()` should be called in most cases to initialize `pph`, which is an object of class `PlanarPH`. The plotting function `DrawLine` is called to plot a single point, or the whole curve, in accordance with the parameter `flag`. `nIndex` indicates the number of points that should be plotted when `flag` equals `plotPoint`. Otherwise, `nIndex` is the number of the last point which should be plotted.

```
BOOL isMenuPh, isMenuCubic, isMenuPhCtrl, isMenuCubicCtrl;
//Status of Menu
BOOL isLBDown;
CPointPH LBDownPt;
GLfloat margin;
```

The program also offers the ability to compare the C^2 PH quintic spline and the "ordinary" C^2 cubic spline with analogous end conditions, and the user may control the display status of each spline. The variables `isMenuPh`, `isMenuCubic`, `isMenuPhCtrl`, and `isMenuCubicCtrl` monitor the status of the menu. They are TRUE if the corresponding menu is checked, and the program then calls the `Drawline` function to plot the spline curves and their control polygons. `isLBDown` and `LBDownPt` specify the status of the left mouse button, and record the coordinates of the point where it is pressed down.

Color List offers an efficient and flexible means to render the scene in the window. With an array or pointer of different color values, OpenGL functions such as `glColor3fv` make the current brush color as needed. Each color in the Color List should be described in RGB format, with three floats between 0 and 1 defining the red, green, and blue values. In order to display the points and lines smoothly on the screen, the program uses OpenGL functions to enable the blending option and smooth option.

As the Graphical User Interface (GUI) for the program, the window has a title bar, menu bar, plotting area, and a status bar. The status bar shows the cursor coordinates before plotting, the iteration number, the arc length and bending energy, and the selected point after plotting. Each menu option has an accelerator (i.e., an alternative keyboard input: see Table 1). In the Edit menu, the user can plot a Hermite interpolant or spline curve, edit multiple points, translate the curve, and construct an offset. The program offers two approaches to editing points: right–clicking on a single point to change it, or selecting Edit Multiple Points in the menu to change all the points.

Figures 4–8 present examples of the program in use. Figure 4 illustrates the construction of a single PH quintic Hermite interpolant from initial and final pairs of control point specified by the user. Figure 5 shows a planar C^2 PH quintic spline interpolating a sequence of points freely selected by the user with the mouse. For this plot, the View menu options PH and PH(Ctrl) have been checked, so the control polygons of each PH quintic spline segment are also shown. Figure 6 presents the same C^2 PH quintic spline, but in this case the View menu options PH and Cubic are checked, to compare the PH spline

New Spline	N
Edit Multiple Points	ctrl+E
Translate Spline	T
Offset	O
Hermite	H
PH	P
PH(Ctrl)	ctrl+P
Cubic	C
Parametric Speed	shift+A
Arc Length	A

Table 1: Accelerator keyboard inputs for menu items.

with the ordinary cubic spline — note the rather poor shape of the latter, as compared to the former. In Figure 7, the parametric speed variation for the C^2 PH spline shown in Figure 4 is plotted. Finally, Figure 8 illustrates the construction of offset curves to a C^2 PH quintic spline, for several different (positive and negative) values of the offset distance d — note that the offset curves can be cleared from the display by setting $d = 0$.

9 Software packages

Two software packages have been prepared to accompany this paper. The first package, found in the file `PHquintic.c`, is written in plain `C` language and offers a set of modular functions that execute each of the basic PH curve construction and analysis computations described in Sections 4–7 (interfaces to these functions have been described in the corresponding sections of the paper). This package allows a software developer to "pick and choose" those functions of primary interest in a specific application context, and incorporate them in an existing software system with minimum effort. The main program accompanying these functions provides some sample data and function calls to test the various functions, but no graphics capability.

The second package, contained in the zip file `InteractivePHquintic`, is implemented in `C++` for compatibility with the MFC and OpenGL libraries, and provides interactive graphical construction, manipulation, and analysis capabilities (these libraries are required to compile and run the package). The unzipped package can be compiled and run by opening a `VC++` project file in

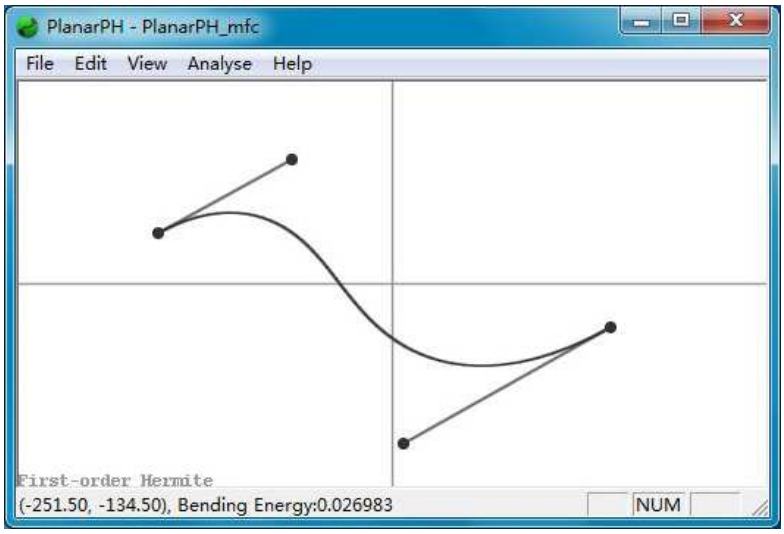

Figure 4: A single PH quintic segment, constructed as a Hermite interpolant from user mouse input specified as initial and final pairs of control points.

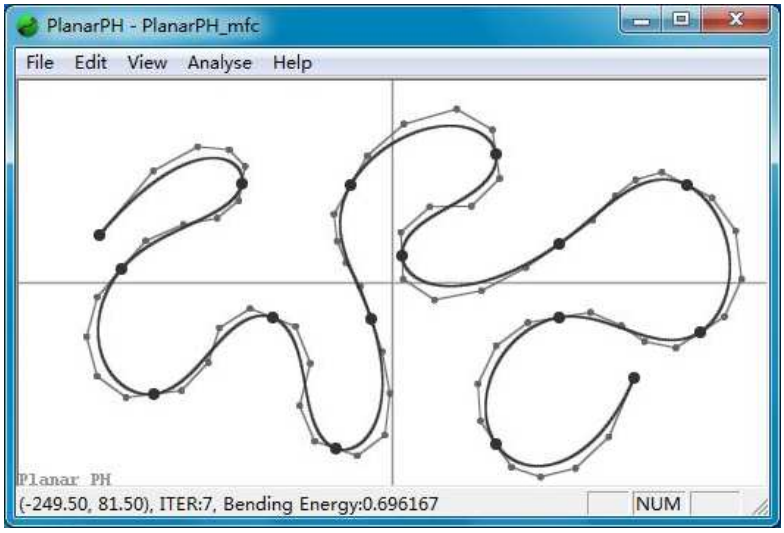

Figure 5: An example of an open C^2 PH quintic spline curve interpolating a sequence of points (large dots) specified interactively with the mouse. The control polygons for each of the PH quintic spline segments are also shown.

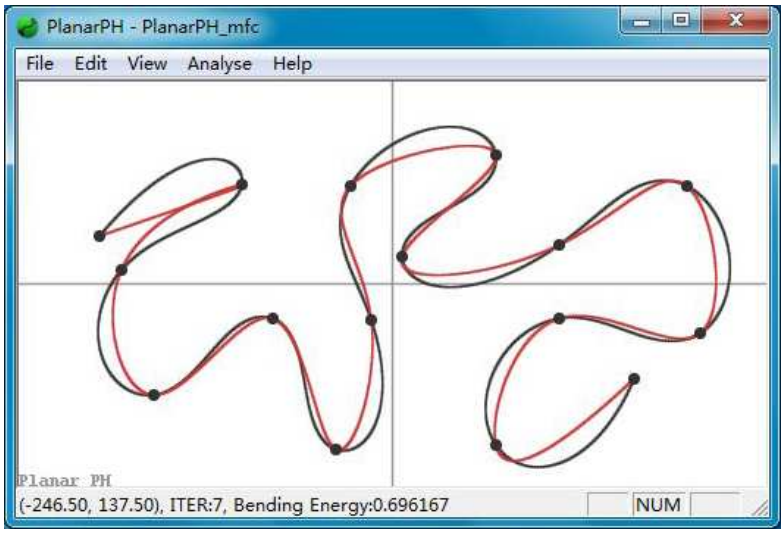

Figure 6: Comparison of the C^2 PH quintic spline shown in Figure 5 and the "ordinary" C^2 cubic spline interpolating the same points and end conditions.

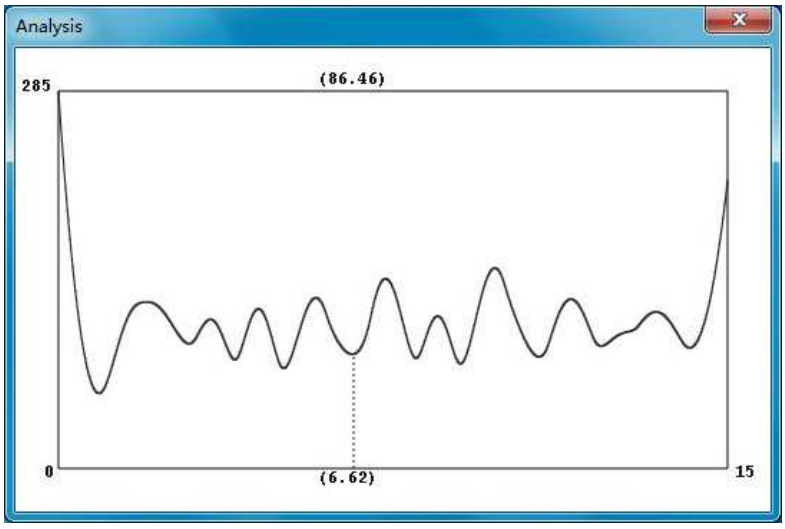

Figure 7: Parametric speed plot for the C^2 PH quintic spline in Figure 5.

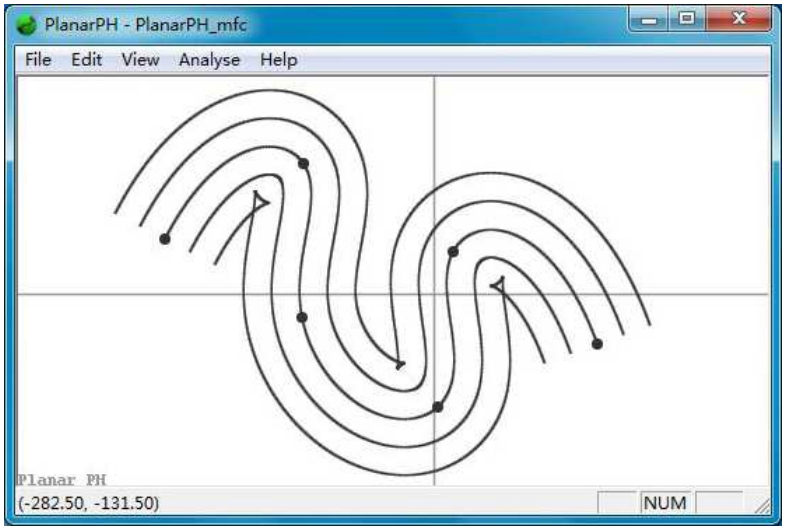

Figure 8: Construction of the rational offsets to a planar C^2 PH quintic spline curve for several — positive and negative — values of the offset distance d.

Microsoft Visual Studio. The basic functions in `PHquintic.c` are included in this package, but the interfaces are modified to meet the requirements of `C++` and the MFC and OpenGL libraries. This package offers a more visual and intuitive user interface, but is perhaps less well–suited to the purpose of porting individual functions to an existing software system.

10 Closure

The basic functions described herein, and implemented in the accompanying software packages, offer a suite of key utilities for the construction, analysis, and application of planar Pythagorean–hodograph quintic curves. Use of the complex representation for planar PH curves ensures compact, efficient, and robust algorithm formulations, allowing interactive manipulation capability. The software functions provide a basis for application–specific customization, tailored to the needs of particular application contexts. It is hoped that the availability of this package will further encourage the growing adoption of PH curves in diverse scientific and engineering computations.

References

[1] G. Albrecht and R. T. Farouki (1996), Construction of C^2 Pythagorean–hodograph interpolating splines by the homotopy method, *Adv. Comp. Math.* **5**, 417–442.

[2] E. L. Allgower and K. Georg (1990), *Numerical Continuation Methods: An Introduction*, Springer, Berlin.

[3] H. Bruyninckx and D. Reynaerts (1997), Path planning for mobile and hyper–redundant robots using Pythagorean–hodograph curves, *Proceedings, International Conference on Advanced Robotics* (ICAR 97), Monterey, CA, 595–600.

[4] C–Y. Chen, S–S. Shieh, M–Y. Cheng, and K–H. Su (2013), Vision–based Pythagorean–hodograph spline command generation and adaptive disturbance compensation for planar contour tracking, *Int. J. Adv. Manuf. Technol.* **65**, 1185–1199.

[5] H. I. Choi, R. T. Farouki, S. H. Kwon, and H. P. Moon (2008), Topological criterion for selection of quintic Pythagorean–hodograph Hermite interpolants, *Comput. Aided Geom. Design* **25**, 411–433.

[6] H. I. Choi, C. Y. Han, H. P. Moon, K. H. Roh, and N–S. Wee (1999), Medial axis transform and offset curves by Minkowski Pythagorean hodograph curves, *Comput. Aided Design* **31**, 59–72.

[7] H. I. Choi, D. S. Lee, and H. P. Moon (2002), Clifford algebra, spin representation, and rational parameterization of curves and surfaces, *Adv. Comp. Math.* **17**, 5–48.

[8] G. Farin (1997), *Curves and Surfaces for Computer Aided Geometric Design* (4th edition), Academic Press, San Diego.

[9] R. T. Farouki (1994), The conformal map $z \to z^2$ of the hodograph plane, *Comput. Aided Geom. Design* **11**, 363–390.

[10] R. T. Farouki (1996), The elastic bending energy of Pythagorean–hodograph curves, *Comput. Aided Geom. Design* **13**, 227–241.

[11] R. T. Farouki (1997), Pythagorean–hodograph quintic transition curves of monotone curvature, *Comput. Aided Design* **29**, 601–606.

[12] R. T. Farouki (2002), Pythagorean–hodograph curves, *Handbook of Computer Aided Geometric Design* (G. Farin, J. Hoschek, and M–S. Kim, eds.), Elsevier, 405–427.

[13] R. T. Farouki (2008), *Pythagorean–Hodograph Curves: Algebra and Geometry Inseparable*, Springer, Berlin.

[14] R. T. Farouki (2010), Quaternion and Hopf map characterizations for the existence of rational rotation–minimizing frames on quintic space curves, *Adv. Comp. Math.* **33**, 331–348.

[15] R. T. Farouki, M. al–Kandari, and T. Sakkalis (2002), Structural invariance of spatial Pythagorean hodographs, *Comput. Aided Geom. Design* **19**, 395–407.

[16] R. T. Farouki, M. al–Kandari, and T. Sakkalis (2002), Hermite interpolation by rotation–invariant spatial Pythagorean–hodograph curves, *Adv. Comp. Math.* **17**, 369–383.

[17] R. T. Farouki, C. Giannelli, C. Manni, and A. Sestini (2008), Identification of spatial PH quintic Hermite interpolants with near–optimal shape measures, *Comput. Aided Geom. Design* **25**, 274–297.

[18] R. T. Farouki, C. Giannelli, C. Manni, and A. Sestini (2012), Design of rational rotation–minimizing rigid body motions by Hermite interpolation, *Math. Comp.* **81**, 879–903.

[19] R. T. Farouki, C. Y. Han, P. Dospra, and T. Sakkalis (2013), Rotation–minimizing Euler–Rodrigues rigid–body motion interpolants, *Comput. Aided Geom. Design* **30**, 653–671.

[20] R. T. Farouki, B. K. Kuspa, C. Manni, and A. Sestini (2001), Efficient solution of the complex quadratic tridiagonal system for C^2 PH quintic splines, *Numer. Algor.* **27**, 35–60.

[21] R. T. Farouki, J. Manjunathaiah, D. Nicholas, G–F. Yuan, and S. Jee (1998), Variable–feedrate CNC interpolators for constant material

removal rates along Pythagorean–hodograph curves, *Comput. Aided Design* **30**, 631–640.

[22] R. T. Farouki, J. Manjunathaiah, and G–F. Yuan (1999), G codes for the specification of Pythagorean–hodograph tool paths and associated feedrate functions on open–architecture CNC machines, *Int. J. Mach. Tools Manuf.* **39**, 123–142.

[23] R. T. Farouki, C. Manni, F. Pelosi, and M. L. Sampoli (2012), Design of C^2 spatial Pythagorean–hodograph quintic splines by control polygons, in (J. D. Boissonnat et al., eds.), Lecture Notes in Computer Science Vol. 6920, 253–269, Springer.

[24] R. T. Farouki, C. Manni, and A. Sestini (2003), Shape–preserving interpolation by G^1 and G^2 PH quintic splines, *IMA J. Numer. Anal.* **23**, 175–195.

[25] R. T. Farouki, C. Manni, and A. Sestini (2003), Spatial C^2 PH quintic splines, in *Curve and Surface Design: Saint–Malo 2002* (T. Lyche, M.–L. Mazure, and L. L. Schumaker, eds.), Nashboro Press, 147–156.

[26] R. T. Farouki and C. A. Neff (1995), Hermite interpolation by Pythagorean–hodograph quintics, *Math. Comp.* **64**, 1589–1609.

[27] R. T. Farouki and H. Pottmann (1996), Polynomial and rational Pythagorean–hodograph curves reconciled, in *The Mathematics of Surfaces VI* (G. Mullineux, ed.), Oxford University Press, 355–378.

[28] R. T. Farouki and V. T. Rajan (1987), On the numerical condition of polynomials in Bernstein form, *Comput. Aided Geom. Design* **4**, 191–216.

[29] R. T. Farouki and T. Sakkalis (1990), Pythagorean hodographs, *IBM J. Res. Develop.* **34**, 736–752.

[30] R. T. Farouki and T. Sakkalis (1991), Real rational curves are not "unit speed," *Comput. Aided Geom. Design* **8**, 151–157.

[31] R. T. Farouki and T. Sakkalis (1994), Pythagorean–hodograph space curves, *Adv. Comp. Math.* **2**, 41–66.

[32] R. T. Farouki and S. Shah (1996), Real–time CNC interpolators for Pythagorean–hodograph curves, *Comput. Aided Geom. Design* **13**, 583–600.

[33] J. C. Fiorot and T. Gensane (1994), Characterizations of the set of rational curves with rational offsets, in *Curves and Surfaces II* (P. J. Laurent, A. Le Méhauté, and L. L. Schumaker, eds.) AK Peters, Boston, 153–160.

[34] Z. Habib and M. Sakai (2007), G^2 Pythagorean hodograph quintic transition between two circles with shape control, *Comput. Aided Geom. Design* **24**, 252–266.

[35] B. M. Imani and A. Ghandehariun (2011), Real–time PH–based interpolation algorithm for high speed CNC machining, *Int. J. Adv. Manuf. Technol.* **56**, 619–629.

[36] G. Klár and G. Valasek (2011), Employing Pythagorean hodograph curves for artistic patterns, *Acta Cybernetica* **20**, 101–110.

[37] J. Kosinka and M. Lávička (2011), A unified Pythagorean hodograph approach to the medial axis transform and offset approximation, *J. Comput. Appl. Math.* **235**, 3413-3424.

[38] H. P. Moon (1999), Minkowski Pythagorean hodographs, *Comput. Aided Geom. Design* **16**, 739–753.

[39] H. P. Moon, R. T. Farouki, and H. I. Choi (2001), Construction and shape analysis of PH quintic Hermite interpolants, *Comput. Aided Geom. Design* **18**, 93–115.

[40] A. P. Morgan (1987), *Solving Polynomial Systems Using Continuation for Engineering and Scientific Problems*, Prentice–Hall, Englewood Cliffs, NJ.

[41] A. A. Neto, D. G. Macharet, and M. F. M. Campos (2010), On the generation of trajectories for multiple UAVs in environments with obstacles, *J. Intell. Robot. Syst.* **57**, 123–141.

[42] F. Pelosi, M. L. Sampoli, R. T. Farouki, and C. Manni (2007), A control polygon scheme for design of planar C^2 PH quintic spline curves, *Comput. Aided Geom. Design* **24**, 28–52.

[43] H. Pottmann (1995), Rational curves and surfaces with rational offsets, *Comput. Aided Geom. Design* **12**, 175–192.

[44] M. A. Shah and N. Aouf (2010), 3D cooperative Pythagorean hodograph path planning and obstacle avoidance for multiple UAVs, 9th IEEE International Conference on Cybernetic Intelligent Systems, 1–6.

[45] M. Shanmugavel, A. Tsourdos, B. A. White, and R. Zbikowski (2007), Differential geometric path planning of multiple UAVs, *ASME J. Dyn. Sys. Meas. Contr.* **129**, 620–632.

[46] M. Shanmugavel, A. Tsourdos, B. A. White, and R. Zbikowski (2011), Path planning of UAVs in urban region using Pythagorean–hodograph curves, *Appl. Mech. Mater.* **110–116**, 4096–4100.

[47] Y–F. Tsai and R. T. Farouki (2001), Algorithm 812: BPOLY: An object–oriented library of numerical algorithms for polynomials in Bernstein form, *ACM Trans. Math. Software* **27**, 267–296.

[48] D. J. Walton and D. S. Meek (2002), Planar G^2 transition with a fair Pythagorean hodograph quintic curve, *J. Comput. Appl. Math.* **138**, 109–126.

[49] D. J. Walton and D. S. Meek (2004), A generalisation of the Pythagorean hodograph quintic spiral, *J. Comput. Appl. Math.* **172**, 271–287.

[50] C. Zwikker (1963), *The Advanced Geometry of Plane Curves and Their Applications*, Dover Publications (reprint), New York.