

# Improperly parametrized rational curves \*

Thomas W. SEDERBERG

*Brigham Young University, Provo, UT 84602, U.S.A.*

Received 19 December 1985

**Abstract.** A parametric curve is said to be improperly parametrized when, to every point on the curve, there corresponds more than one parameter value. It is always possible to reparametrize an improperly parametrized curve in such a way that it becomes properly parametrized – that is, to each non-singular point on the curve there corresponds exactly one parameter value. This reparametrization always reduces the degree of the rational polynomials which define a rational curve.

This paper presents an algorithm for detecting whether a rational curve is improperly parametrized, and also an algorithm for reparametrizing such a curve so as to make it properly parametrized. Examples are given of improperly parametrized rational Bézier curves.

The algorithm uses only rational arithmetic operations, and therefore can be implemented in exact integer arithmetic. A floating point implementation is also outlined.

**Keywords.** Rational curves, parametrization.

## 1. Introduction

A rational polynomial parametric curve (or, simply a rational curve) is any curve that can be expressed parametrically in terms of rational functions:

$$X = \frac{x(t)}{w(t)}, \quad Y = \frac{y(t)}{w(t)}, \quad Z = \frac{z(t)}{w(t)}$$

where  $x(t)$ ,  $y(t)$ ,  $z(t)$  and  $w(t)$  are polynomials. For clarity, this paper restricts its discussion to planar rational curves (i.e.,  $z \equiv 0$ ), but the extension to space curves is obvious.

Rational curves can be reparametrized using the bilinear parameter substitution

$$t = \frac{a_1 s + a_0}{b_1 s + b_0}, \quad (a_1, b_0 - b_1 a_0 \neq 0).$$

Note that there is a one-to-one correspondence between  $t$  and  $s$ . Also, a parameter substitution has no effect on the appearance of the curve – it only redefines the relationship between points on the curve and parameter values.

It is also possible to reparametrize a rational curve using the non-linear parameter substitution

$$t = \frac{a_n s^n + a_{n-1} s^{n-1} + \dots + a_0}{b_n s^n + b_{n-1} s^{n-1} + \dots + b_0}. \quad (1)$$

Once again, this parameter substitution does not alter the curve, but the correspondence between parameter values and points on the curve is altered. If there is a one-to-one

\* This work was sponsored by General Electric Company.

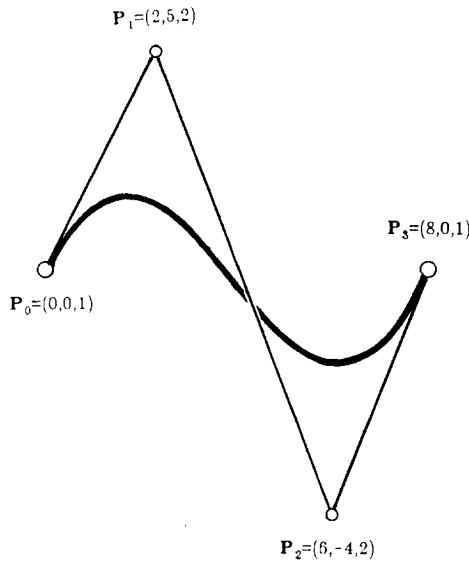


Fig. 1a. Cubic Bézier curve.

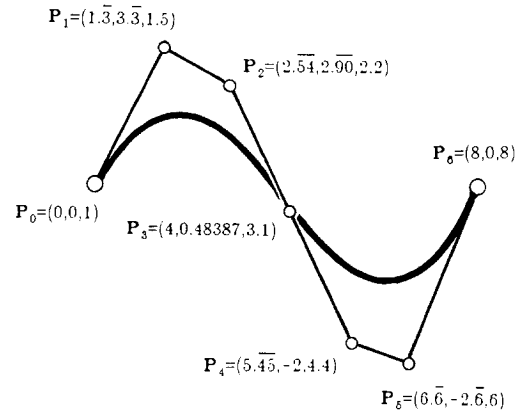


Fig. 1b. Degree six Bézier curve.

correspondence between parameter values  $t$  and points on the curve (with the exception of singular points), then there is an  $n$ -to-one correspondence between parameter values  $s$  and points on the curve. Curves which have a one-to-one relationship between parameter values and points on the curve are called *properly parametrized curves* – a term used occasionally in algebraic geometry. If the correspondence is  $n$ -to-one ( $n > 1$ ), the curve is *improperly parametrized*. Some or all of those  $n$  parameter values can be complex.

Consider, for example, the rational cubic Bézier curve in Fig. 1a. It is defined by the equations

$$X = \frac{-16t^3 + 12t^2 + 12t}{-3t^2 + 3t + 1}, \quad Y = \frac{54t^3 - 84t^2 + 30t}{-3t^2 + 3t + 1}.$$

We make the parameter substitution  $t = (s^2 + s)/(s^2 + 1)$  and multiply  $x(s)$ ,  $y(s)$ , and  $w(s)$  by  $(s^2 + 1)^3$  to get

$$X(s) = \frac{8s^6 - 12s^5 + 32s^3 + 24s^2 + 12s}{s^6 - 3s^5 + 3s^4 + 3s^2 + 3s + 1},$$

$$Y(s) = \frac{24s^5 + 54s^4 - 54s^3 - 54s^2 + 30s}{s^6 - 3s^5 + 3s^4 + 3s^2 + 3s + 1}$$

which are the power basis equations of the rational degree six Bézier curve in Fig. 1b. The triple next to each control point in the two figures expresses the Cartesian coordinates and the weight of the respective control points. Note that the curves in Fig. 1a and Fig. 1b are identical except for the parametrization. It may be useful to demonstrate that the curve in Fig. 1b is indeed doubly defined. This can be done, for example, by verifying that  $(X(0), Y(0)) = (X(-1), Y(-1)) = (0, 0)$ , or that in general,

$$\left( X(s), Y(s) \right) = \left( X\left(\frac{s+1}{s-1}\right), Y\left(\frac{s+1}{s-1}\right) \right) \quad \text{for any } s.$$

An improperly parametrized curve is not necessarily *multiply traced*, which means that at least a portion of the curve is traced more than once as the parameter sweeps between zero and

one. In fact, the example in Fig. 1 is *not* multiply traced. However, any multiply traced curve is improperly parametrized. In fact, it would seem that a multiply traced Bézier curve is much less likely to occur than a Bézier curve which is improperly parametrized but not multiply traced. Because of the variation diminishing property of Bézier curves, the control polygon would have to fold back on itself quite severely in order for the curve to be multiply traced. However, Fig. 1b illustrates how a seemingly innocuous Bézier curve may indeed be improperly parametrized. Finally, we note that most improperly parametrized curves would be multiply traced if the parameter were to vary over a large enough range. Multiply traced curves are mentioned in [Faux and Pratt '79].

This raises the question of how to detect if a rational curve is improperly parametrized, and if so, how to reparametrize it in terms of polynomials of lower degree. The contribution of this paper is to present such an algorithm.

It should also be noted that improper parametrization is less likely to occur in conics and cubics than curves of higher degree. If an improperly parametrized curve is formed by substituting equation (1) (which is of degree  $n$ ) into a properly parametrized curve of degree  $m$ , the resulting curve will be of degree  $mn$ . Thus, an improperly parametrized curve of degree two or three could only be a multiply defined straight line, whereas a quartic curve could be a doubly defined conic, etc.

A special case of an improperly parametrized curve occurs when a properly parametrized curve is reparametrized by

$$t = a_n s^n + a_{n-1} s^{n-1} + \dots + a_0.$$

This differs from the general case considered here in that there is no denominator polynomial. An algorithm for dealing with this special case of improperly parametrized curves is described in [Sederberg '84]. The algorithm discussed herein can handle this special case, and is a more numerically stable algorithm.

Section 2 develops an algorithm for detecting if a curve is improperly parametrized, and if so, for reparametrizing it to make it properly parametrized. Section 3 presents an example of this algorithm in exact integer arithmetic, and section 4 discusses floating point implementation.

## 2. Algorithm for detecting improper rational parametrization

Consider the plane rational curve

$$x = \frac{x(s)}{w(s)}, \quad y = \frac{y(s)}{w(s)}.$$

We wish to know if this curve is improperly parametrized. We know from Luroth's theorem (see [Archbold [48]]) that any improperly parametrized curve can be expressed as a properly parametrized curve

$$x = \frac{\bar{x}(t)}{\bar{w}(t)}, \quad y = \frac{\bar{y}(t)}{\bar{w}(t)}$$

where  $t = f(s)/g(s)$ . Let  $n$  be the degree of  $f(s)$  (or, the degree of  $g(s)$  if larger). If  $n = 1$ , there will *always* exist an  $\bar{x}(t)$ ,  $\bar{y}(t)$  and  $\bar{w}(t)$  for *any*  $f(s)$  and  $g(s)$  because that is simply a bilinear reparametrization. Therefore, only cases where  $n > 1$  are interest.

If  $(x', y')$  is any non-singular point on the curve, then there will be  $n$  values of  $s$  which simultaneously satisfy the two equations

$$x' = \frac{x(s)}{w(s)}, \quad y' = \frac{y(s)}{w(s)}.$$

Let  $x' = x(\alpha)/w(\alpha)$  and  $y' = y(\alpha)/w(\alpha)$  where  $\alpha$  is the parameter value of any non-singular point. We want to know whether there are any values of  $s$  other than  $s = \alpha$  which satisfy

$$x' = \frac{x(\alpha)}{w(\alpha)} = \frac{x(s)}{w(s)}, \quad y' = \frac{y(\alpha)}{w(\alpha)} = \frac{y(s)}{w(s)}$$

or, rearranging,

$$P_x(s, \alpha) = x(\alpha)w(s) - w(\alpha)x(s) = 0, \quad P_y(s, \alpha) = y(\alpha)w(s) - w(\alpha)y(s) = 0.$$

Clearly,  $s = \alpha$  satisfies both of these equations, and  $n - 1$  values of  $s$  will also satisfy them simultaneously (if  $n - 1 > 0$ ).

Consider  $P_x(s, \alpha)$  and  $P_y(s, \alpha)$  as polynomials in  $s$  whose coefficients are polynomials in  $\alpha$ . If  $\alpha$  is assigned a numerical value, then  $P_x(s, \alpha)$  and  $P_y(s, \alpha)$  are univariate polynomials in  $s$ . Denote by  $GCD_\alpha(s)$  the greatest common divisor of  $P_x(s, \alpha)$  and  $P_y(s, \alpha)$ . Clearly  $GCD_\alpha(s)$  is a polynomial in  $s$  for which  $\alpha$  is a root. There are  $n - 1$  other roots of  $GCD_\alpha(s)$ , each one of which is a parameter value which corresponds to the same point on the curve as does  $\alpha$ . If any of these other  $n - 1$  roots were selected as the value of  $\alpha$ , the same GCD would have resulted.

Equation  $GCD_\alpha(s) = 0$  defines a relationship between parameter values which correspond to the same point on the curve. Equation (1), the reparametrization equation, defines a one parameter family of such relationships:

$$t \times g(s) - f(s) = 0. \quad (2)$$

Equation (2) is called an *involution*, which means that it defines all possible relationships between parameter values which correspond to the same point on the curve (see [Archbold '48] for a proof of this). If  $g(s) = a_0 + a_1s + \dots + a_ns^n$  and  $f(s) = b_0 + b_1s + \dots + b_ns^n$ , then the involution can be written

$$(a_0t - b_0) + (a_1t - b_1)s + (a_2t - b_2)s^2 + \dots + (a_nt - b_n)s^n = 0. \quad (3)$$

Said another way, the involution on the curve is a one parameter family of univariate polynomials with  $f(s)$  and  $g(s)$  as a basis. Thus, we can always find values for  $\delta$  and  $\gamma$  such that  $GCD_\alpha(s) = \delta f(s) + \gamma g(s)$ . Likewise, we could use for a basis any two different polynomials that belong to the involution. For our purposes, a useful such pair of polynomials is  $GCD_\alpha(s)$  (which is the GCD of  $P_x(s, \alpha)$  and  $P_y(s, \alpha)$ ) and  $GCD_\beta(s)$  (which is the GCD of  $P_x(s, \beta)$  and  $P_y(s, \beta)$ ) where  $\beta$  and  $\alpha$  are two arbitrary constants. Thus, we can write

$$f(s) = a \times GCD_\alpha(s) + b \times GCD_\beta(s), \quad (4a)$$

$$g(s) = c \times GCD_\alpha(s) + d \times GCD_\beta(s). \quad (4b)$$

Thus, we can compute  $f(s)$  and  $g(s)$  by simply finding any pair of independent GCDs and performing two linear combinations on them. One would likely pick  $a$ ,  $b$ ,  $c$  and  $d$  such that  $t = 0$  when  $s = 0$  and  $t = 1$  when  $s = 1$ , because then the properly parametrized curve segment will have the same endpoints as the improperly parametrized curve segment.

Once we have determined  $f(s)$  and  $g(s)$ , it remains to compute the coefficients of the properly parametrized curve. If the maximum degree of  $f(s)$  and  $g(s)$  is  $n$ , and the degree of the improperly parametrized curve is  $p$ , then the degree of the properly parametrized curve will be  $m = p/n$ . Thus, the curve has the form:

$$\bar{x}(t) = x_m t^m + \dots + x_1 t + x_0,$$

$$\bar{y}(t) = y_m t^m + \dots + y_1 t + y_0,$$

$$\bar{w}(t) = w_m t^m + \dots + w_1 t + w_0.$$

The coefficients  $x_i$ ,  $y_i$ , and  $w_i$  can be easily found using the method of undetermined coefficients. Simply select  $m + 1$  values of  $s$ , compute  $t = f(s)/g(s)$ , compute  $x(s)$ ,  $y(s)$  and

$w(s)$  from the improperly parametrized equations (noting that  $\bar{x}(t) = x(s)/g(s)^m$ , etc.) and fit  $\bar{x}(t)$ ,  $\bar{y}(t)$  and  $\bar{w}(t)$  through the data points.

There are restrictions on what we can use for  $\alpha$  and  $\beta$ . For example, notice that if  $w(\alpha) = 0$ , then  $GCD_\alpha(s) \equiv w(s)$ , which is clearly not what we want. Another restriction is that  $\alpha$  and  $\beta$  cannot be members of the same involution, for then  $GCD_\alpha(s) \equiv GCD_\beta(s)$ . The solution is again to simply pick some other value of  $\beta$ .

It may occur that  $x(s)$  and  $w(s)$  or  $y(s)$  and  $w(s)$  may have a common factor. If so,  $\alpha$  or  $\beta$  should not be a root of that common factor. For similar reasons,  $\alpha$  or  $\beta$  should not be the parameter value of a multiple point.

Finally, a word on how to compute the GCD of two polynomials using Euclid's algorithm. Given two polynomials  $f(t)$  and  $g(t)$ , Euclid's algorithm proceeds by dividing  $f(t)$  by  $g(t)$  and finding the remainder polynomial  $R_1(t)$ . Then  $g(t)$  is divided by  $R_1(t)$  yielding a remainder  $R_2(t)$ . The process continues, dividing  $R_{i-1}(t)$  by  $R_i(t)$  to produce remainder  $R_{i+1}(t)$ . Since the degree of  $R_{i+1}(t)$  is always less than the degree of  $R_i(t)$ , eventually a remainder is produced  $R_k(t) = 0$ , and  $R_{k-1}(t)$  is the GCD of  $f(t)$  and  $g(t)$  (see any text on college algebra, such as [Kurosh '75]).

Euclid's algorithm is a standard tool in computer algebra system which deal in exact arithmetic wherein rational numbers are represented by an integer numerator and an integer denominator so that no floating point roundoff error is introduced. Our experience is that it is not very stable in floating point arithmetic. Section 4 discusses a floating point implementation.

### 3. Numerical example

We illustrate using the example we generated in the introduction:

$$x = \frac{8s^6 - 12s^5 + 32s^3 + 24s^2 + 12s}{s^6 - 3s^5 + 3s^4 + 3s^2 + 3s + 1},$$

$$y = \frac{24s^5 + 54s^4 - 54s^3 - 54s^2 + 30s}{s^6 - 3s^5 + 3s^4 + 3s^2 + 3s + 1}.$$

We arbitrarily select  $\alpha = 2$  and  $\beta = 4$ . For  $\alpha = 2$ , we obtain:

$$P_x(s, 2) = 224s^6 - 1092s^5 + 1512s^4 - 1120s^3 + 672s^2 + 1092s + 504,$$

$$P_y(s, 2) = 1044s^6 - 3972s^5 + 1242s^4 + 1890s^3 + 5022s^2 + 2082s + 1044.$$

The GCD of these two polynomials is:

$$GCD_\alpha(s) = s^2 - 5s + 6.$$

For  $\beta = 4$  we obtain

$$P_x(s, 4) = 8136s^6 - 46644s^5 + 68880s^4 - 59296s^3 + 24408s^2 + 46644s + 22960,$$

$$P_y(s, 4) = 34200s^6 - 147072s^5 + 2538s^4 + 100062s^3 + 202662s^2 + 47010s$$

$$+ 34200$$

and the GCD of these polynomials is

$$GCD_\beta(s) = 3s^2 - 17s + 20.$$

Our parameter substitution is of the form

$$t = \frac{a(s^2 - 5s + 6) + b(3s^2 - 17s + 20)}{c(s^2 - 5s + 6) + d(3s^2 - 17s + 20)}$$

and we would like to select  $a$ ,  $b$ ,  $c$  and  $d$  so that  $t = 0$  when  $s = 0$  and  $t = 1$  when  $s = 1$ . The first constraint establishes the relationship  $6a + 20b = 0$ , so we select  $a = 10$  and  $b = -3$ . The second constraint requires that  $c + 3d = 1$ , and we can freely choose either  $c$  or  $d$ . If we pick  $c = 4$  and  $d = -1$ , we obtain  $t = (s^2 + s)/(s^2 + 1)$ , which is the reparametrization used in creating this example problem.

#### 4. Floating point implementation

The algorithm we have just outlined works nicely in exact integer arithmetic, requiring only the arithmetic operations of addition, subtraction, multiplication and division in a closed form, non-iterative solution. We suggest a few modifications for using the algorithm in floating point arithmetic. These modifications have two motivations. First, we have observed Euclid's GCD algorithm to be unstable in floating point. Second, our algorithm would be more useful if it could detect whether a given curve is improperly parametrized *to within a tolerance*. In other words, if we allow small perturbations of the Bézier control points of a curve, can we find a non-linear reparametrization of the curve.

The following suggestions are heuristic approaches to implementing the algorithm in floating point. However, the reader should be aware that the last step in the floating point algorithm is an *a posteriori* check to detect how closely a candidate curve of lower degree approximates the given curve. This check acts as a safety net for the heuristics.

##### 4.1. Floating point GCD

It has been argued that the GCD of two polynomials is a concept foreign to floating point, because 'GCD' connotes infinite precision. This argument is well taken. We suggest that our current needs may be better served by taking the following approximation-theoretic view of GCD's. Consider two polynomials,  $f(t)$  and  $g(t)$  with real, floating point coefficients. For our purposes, we restrict the range of  $t$  to  $0 \leq t \leq 1$ , because that is the parameter range of a Bézier curve.

*Step 1.* Compute all roots of  $f$  and  $g$ .

*Step 2.* Express all real roots as linear factors in Bernstein polynomial form:  $a_0(1-t) + a_1t$ . Furthermore, normalize so that the largest Bernstein coefficient is 1. Thus, for real roots  $r$ , we have

$$a_0 = 1, \quad a_1 = \frac{r-1}{r}, \quad r \geq \frac{1}{2},$$

$$a_0 = \frac{r}{r-1}, \quad a_1 = 1, \quad r < \frac{1}{2},$$

*Step 3.* Express all pairs of complex conjugate roots as quadratic factors in Bernstein polynomial form:  $a_0(1-t)^2 + 2a_1t(1-t) + a_2t^2$ . Again, normalize so that the largest Bernstein coefficient is 1.

*Step 4.* Compare all linear factors of  $f$  to all linear factors of  $g$ . If  $a(t) = a_0(1-t) + a_1t$  is a factor of  $f(t)$  and  $b(t) = b_0(1-t) + b_1t$  is a factor of  $g(t)$ , then  $\max(|a_0 - b_0|, |a_1 - b_1|)$  is the maximum functional difference between  $a(t)$  and  $b(t)$  over the range  $0 \leq t \leq 1$ . If this difference is within prescribed tolerance, then we define  $\frac{1}{2}(a(t) + b(t))$  to be a floating point common factor of  $f$  and  $g$ . Note the flexibility of this definition. Suppose  $f(t)$  has a root of  $10^6$  and  $g(t)$  has a root of  $10^7$ . By precise GCD definition, there is no way to construe these as being common roots – they have no digits in common! However, over the interval  $[0, 1]$ , those normalized factors agree to six digits.

*Step 5.* Likewise, compare all quadratic factors of  $f$  to all quadratic factors of  $g$ .

*Step 6.* Multiply all common factors together (linear and quadratic) to arrive at the floating point GCD. To avoid confusion, we will refer to this as the FGCD.

Note that a factor can contribute only once to the FGCD. For example, if  $f$  has a doubly defined factor, and  $g$  contains the same factor three times, it can only appear twice in the FGCD.

This FGCD algorithm is clearly much slower than Euclid's algorithm. However, stability is valued more than speed. We have coded this FGCD algorithm, and find (nor surprisingly) that it behaves much more reliably, but much slower, than a floating point implementation of Euclid's algorithm. An interesting research problem is to devise a fast, stable FGCD algorithm in the spirit of Euclid's algorithm.

#### 4.2. Floating point involution

In the exact arithmetic algorithm, only two sample points were used to determine the involution. In the floating point algorithm, it seems wise to use several sample points and to compute the involution using a least squares fit. We could perform this either in the Bernstein polynomial basis, or in the standard power basis. For ease of discussion, we will explain in terms of the power basis.

Our goal is to compute an involution of the form of equation (3). This amounts to finding two polynomials which form a basis for the involution. For convenience, we can specify several of the coefficients of those basis polynomials without loss of generality. If the involution is of degree  $n$ , we define our basis polynomials to be of the form

$$a_1t + \cdots + a_{n-1}t^{n-1} + t^n$$

and

$$1 + b_1t + \cdots + b_{n-1}t^{n-1}.$$

Any polynomial which belongs to the basis can now be expressed

$$\begin{aligned} c_{0i} + c_{1i}t + \cdots + c_{ni}t^n \\ = \alpha_i(a_1t + \cdots + a_{n-1}t^{n-1} + t^n) + \beta_i(1 + b_1t + \cdots + b_{n-1}t^{n-1}). \end{aligned} \quad (5)$$

In our case, we generate several polynomials which belong to the involution to within a tolerance, and we compute the best fit involution.

Select  $nsamp$  values of  $s$ , evenly spaced in the interval  $[0, 1]$ ,  $s_0, \dots, s_{nsamp}$ . Let  $c_{0i} + c_{1i}t + \cdots + c_{ni}t^n$  be the FGCD of  $x(s_i)w(s) - w(s_i)x(s)$  and  $y(s_i) - w(s_i)y(s)$ . Note from equation (5) that  $\alpha_i = c_{ni}$  and  $\beta_i = c_{0i}$ . We then equate similar powers of  $t$  to obtain

$$\begin{bmatrix} \alpha_1 & \beta_1 \\ \alpha_2 & \beta_2 \\ \vdots & \vdots \\ \alpha_{nsamp} & \beta_{nsamp} \end{bmatrix} \begin{Bmatrix} \alpha_j \\ b_j \end{Bmatrix} = \begin{Bmatrix} c_{1j} \\ c_{2j} \\ \vdots \\ c_{nsamp,j} \end{Bmatrix}$$

or  $[\alpha]\{a_j\} = \{c_j\}$  from which the least squares solution is  $\{a_j\} = ([\alpha]^T[\alpha])^{-1}[\alpha]^T\{c_j\}$ . The most common solution is probably a linear involution, which means that the curve is not nearly improperly parametrized.

#### 4.3. Least squares curve coefficients

Having computed a basis for the involution, the final step is to compute the coefficients of the 'properly parametrized' curve – set in quotes because in our floating point algorithm, a

curve from which we can construct an involution may not be truly improperly parametrized, but the algorithm may succeed in lowering the degree of its parametric equation nonetheless. We will discuss how to solve for the coefficients of the  $\bar{x}$  polynomial, and the coefficients of  $\bar{y}$  and  $\bar{w}$  are found in like manner.

We have a polynomial function

$$x(s) = \gamma_p s^p + \cdots + \gamma_1 s + \gamma_0$$

and a non-linear reparametrization which we obtained from the involution:

$$t = \frac{f(s)}{g(s)} = \frac{a_n s^n + \cdots + a_1 s + a_0}{b_n s^n + \cdots + b_1 s + b_0}$$

We want to find the coefficients of

$$\bar{x}(t) = x_m t^m + \cdots + x_1 t + x_0$$

such that

$$\bar{x}(t(s)) = x_m f(s)^m + \cdots + x_1 f(s) g(s)^{m-1} + x_0 g(s)^m = x(s)$$

where  $m = p/n$ . This problem is over-constrained, having  $p + 1$  linear equations in  $m + 1$  unknowns. Again, the proper approach seems to be a least squares solution. The problem is formulated as

$$\begin{bmatrix} \lambda_{m,p} & \cdots & \lambda_{0,p} \\ \vdots & \ddots & \vdots \\ \lambda_{m,0} & \cdots & \lambda_{0,0} \end{bmatrix} \begin{Bmatrix} x_m \\ \vdots \\ x_1 \\ x_0 \end{Bmatrix} = \begin{Bmatrix} \gamma_p \\ \vdots \\ \gamma_1 \\ \gamma_0 \end{Bmatrix}$$

or  $[\lambda]\{\mathbf{x}\} = \{\boldsymbol{\gamma}\}$  where the  $i$ th column of  $[\lambda]$  contains the polynomial coefficients of the expansion of  $g(s)^{i-1} f(s)^{m-i+1}$ . The least squares solution to this problem is

$$\{\mathbf{x}\} = ([\lambda]^T [\lambda])^{-1} [\lambda]^T \{\boldsymbol{\gamma}\}.$$

The final step in the floating point algorithm is an *a posteriori* check on how well the resulting curve approximates the original curve. This can be accomplished by substituting the non-linear reparametrization into the resulting curve and expanding the expression to see how closely we can replicate the original curve. This comparison is most meaningfully performed in the Bernstein–Bézier form of the curve.

## 5. Conclusion

An algorithm is presented for detecting and correcting improperly parametrized rational curves in exact integer arithmetic. A floating point algorithm is also outlined.

## Acknowledgements

John Snively shared some valuable algebraic insights. Bob Barnhill offered some thoughts on approximation theory.

## References

- Archbold, J.W. (1948), *Introduction to the Algebraic Geometry of a Plane*, Edward Arnold & Co., London.
- Faux, I.E. and Pratt, M.J. (1979), *Computational Geometry for Design and Manufacture*, Ellis Horwood Ltd., Chichester.
- Kurosh, A. (1975), *Higher Algebra*, MIR Publishers, Moscow.
- Sederberg, T.W. (1984), Degenerate parametric curves, *Computer Aided Geometric Design* 1, 301–307.