ELSEVIER

Computer Aided Geometric Design 19 (2002) 275–289

COMPUTER AIDED GEOMETRIC DESIGN

www.elsevier.com/locate/comaid

Total least squares fitting of Bézier and B-spline curves to ordered data

Carlos F. Borges *, Tim Pastva

Naval Postgraduate School, Code NA/BC, 93943 Monterey, CA, USA

Received 1 October 2000; received in revised form 1 November 2001

Abstract

We begin by considering the problem of fitting a single Bézier curve segment to a set of ordered data so that the error is minimized in the total least squares sense. We develop an algorithm for applying the Gauss–Newton method to this problem with a direct method for evaluating the Jacobian based on implicitly differentiating a pseudo-inverse. We then demonstrate the simple extension of this algorithm to B-spline curves. We present some experimental results for both cases.

Keywords: Data fitting; Total least-squares; B-splines

1. Introduction

A degree n Bézier curve segment is a parametric polynomial curve whose shape is determined by an ordered set of control points (x_j, y_j) for $j = 0, 1, 2, \ldots, n$. In particular, for $0 \leqslant t \leqslant 1$, the components of the curve are given by

$$x(t) = \sum_{j=0}^{n} B_j^n(t) x_j \quad \text{and} \quad y(t) = \sum_{j=0}^{n} B_j^n(t) y_j,$$

where $B_j^n(t)$ is the jth Bernstein polynomial of degree n, that is

$$B_j^n(t) = \binom{n}{j} t^j (1-t)^{n-j}.$$

* Corresponding author.
E-mail address: borges@nps.navy.mil (C.F. Borges).

0167-8396/02/$ – see front matter
PII: S0167-8396(02)00088-2

The problem we are interested in solving can be stated as follows. Given an ordered set of data points $\{(u_i, v_i)\}_{i=1}^m$ and a non-negative integer $n < m$, find nodes $\{t_i\}_{i=1}^m$ and control points $\{(x_j, y_j)\}_{j=1}^n$ that minimize

$$\sum_{i=1}^{m}\left\{\left(u_i - \sum_{j=1}^{n} B_j^n(t_i)x_j\right)^2 + \left(v_i - \sum_{j=1}^{n} B_j^n(t_i)y_j\right)^2\right\}. \tag{1}$$

In short, we are looking for the best total least-squares fit of a Bézier curve segment to a set of ordered data points in the plane.

We will find it much easier to proceed if we recast the problem in a more convenient linear algebra setting. To this end we introduce the following definition:

Definition 1. Given a vector $\mathbf{t} \in \mathbb{R}^m$ and a non-negative integer n we define the *Bernstein matrix* of degree n, which we denote by $\mathcal{B}_n(\mathbf{t})$, to be the real $m \times (n+1)$ matrix whose i, j element is $B_j^n(t_i)$. In particular

$$\mathcal{B}_n(\mathbf{t}) = \begin{bmatrix} B_0^n(t_1) & B_1^n(t_1) & \dots & B_n^n(t_1) \\ B_0^n(t_2) & B_1^n(t_2) & \dots & B_n^n(t_2) \\ \vdots & \vdots & & \vdots \\ B_0^n(t_m) & B_1^n(t_m) & \dots & B_n^n(t_m) \end{bmatrix}.$$

Be warned that for ease of notation we will generally omit the arguments of matrix functions when no confusion is possible.

This definition allows us to rewrite the objective function in expression (1) as follows

$$\left\| \begin{bmatrix} \mathbf{u} \\ \mathbf{v} \end{bmatrix} - \begin{bmatrix} \mathcal{B}_n & \\ & \mathcal{B}_n \end{bmatrix} \begin{bmatrix} \mathbf{x} \\ \mathbf{y} \end{bmatrix} \right\|_2^2, \tag{2}$$

where $\mathbf{x}$, $\mathbf{y}$, $\mathbf{u}$, and $\mathbf{v}$ are vectors whose elements are the x_i, y_i, u_i, and v_i respectively.

It is of particular interest that the variables in this problem are of two distinct types—those that appear linearly (the control points) and those that appear non-linearly (the nodes). Note that for a fixed $\mathbf{t}$ the optimal associated control points are found by solving a linear least squares problem. Formally we may write

$$\begin{bmatrix} \mathbf{x} \\ \mathbf{y} \end{bmatrix} = \begin{bmatrix} \mathcal{B}_n^- & \\ & \mathcal{B}_n^- \end{bmatrix} \begin{bmatrix} \mathbf{u} \\ \mathbf{v} \end{bmatrix}, \tag{3}$$

where $\mathcal{B}_n^-$ is a *generalized inverse* (see (Rao and Mitra, 1971)) of $\mathcal{B}_n$. Specifically, $\mathcal{B}_n^-$ is any $(n+1) \times m$ matrix function satisfying the following equations

$$\mathcal{B}_n \mathcal{B}_n^- \mathcal{B}_n = \mathcal{B}_n, \tag{4}$$

$$(\mathcal{B}_n \mathcal{B}_n^-)^{\mathrm{T}} = \mathcal{B}_n \mathcal{B}_n^-. \tag{5}$$

We note that the Moore–Penrose generalized inverse, $\mathcal{B}_n^+$, is one such matrix although there can be others.[1]

[1] This is a formal derivation and it is important to note that one should not form generalized inverses as a method for *computing* solutions to linear least-squares problems.

This structure leads to a useful approach to solving the problem. In particular, one can substitute (3) into the objective function (2) and formally *uncouple* the variables. It is then possible to solve for the optimal $\mathbf{t}$ independently of the control points by minimizing the *variable projection functional*

$$r(\mathbf{t}) = \left\| \begin{bmatrix} \mathbf{u} \\ \mathbf{v} \end{bmatrix} - \begin{bmatrix} \mathcal{B}_n \mathcal{B}_n^- & \\ & \mathcal{B}_n \mathcal{B}_n^- \end{bmatrix} \begin{bmatrix} \mathbf{u} \\ \mathbf{v} \end{bmatrix} \right\|_2^2. \tag{6}$$

It is important to note that $r(\mathbf{t}) = \mathbf{r}^{\mathrm{T}}\mathbf{r}$, where the residual vector, $\mathbf{r}$, is also a function of $\mathbf{t}$ and is given by

$$\mathbf{r}(\mathbf{t}) = \begin{bmatrix} P_{\mathcal{B}_n}^{\perp}\mathbf{u} \\ P_{\mathcal{B}_n}^{\perp}\mathbf{v} \end{bmatrix}, \tag{7}$$

where $P_{\mathcal{B}_n} = \mathcal{B}_n\mathcal{B}_n^-$ is the orthogonal projector onto the range of $\mathcal{B}$ and $P_{\mathcal{B}_n}^{\perp} = I - P_{\mathcal{B}_n}$ is the orthogonal projector onto the null-space of $\mathcal{B}_n^{\mathrm{T}}$. Note that both of these are also matrix functions although we have suppressed their arguments.

1.1. Numerical evaluation of $r(\mathbf{t})$

From a computational standpoint, we may reliably evaluate the variable projection functional by exploiting the QR factorization (see (Golub and Van Loan, 1996)) of the Bernstein matrix. In particular, there exists a permutation matrix Π such that

$$\mathcal{B}_n\Pi = [Q_1 \quad Q_2] \begin{bmatrix} R_{1,1} & R_{1,2} \\ 0 & 0 \end{bmatrix},$$

where $[Q_1 \quad Q_2]$ is an $m \times m$ orthogonal matrix and $R_{1,1}$ is $r \times r$, upper-triangular, and invertible where $r = \operatorname{rank}(\mathcal{B}_n)$. Note that $R_{1,2}$ vanishes whenever $\mathcal{B}_n$ is full rank. It is easily verified that

$$\mathcal{B}_n^- = \Pi \begin{bmatrix} R_{1,1}^{-1}Q_1^{\mathrm{T}} \\ 0 \end{bmatrix} \tag{8}$$

satisfies Eqs. (4) and (5), from which it follows directly that

$$P_{\mathcal{B}_n} = Q_1 Q_1^{\mathrm{T}},$$
$$P_{\mathcal{B}_n}^{\perp} = Q_2 Q_2^{\mathrm{T}},$$

whence

$$r(\mathbf{t}) = \left\| Q_2^{\mathrm{T}}\mathbf{u} \right\|_2^2 + \left\| Q_2^{\mathrm{T}}\mathbf{v} \right\|_2^2.$$

2. Minimizing the variable projection functional

In this section we will concern ourselves with minimizing the variable projection functional which amounts to solving a non-linear least-squares problem. Although a variety of methods exist for such problems we will explore the Gauss–Newton approach.

The Gauss–Newton approach involves using an affine model at the current point $\mathbf{t}_k$ as follows

$$\mathbf{m}_k(\mathbf{t}) = \mathbf{r}(\mathbf{t}_k) + J(\mathbf{t}_k)(\mathbf{t} - \mathbf{t}_k),$$

where the i, j element of the Jacobian matrix is

$$J(\mathbf{t})_{i,j} = \frac{\partial r_i(\mathbf{t})}{\partial t_j}.$$

Minimizing $\mathbf{m}_k(\mathbf{t})$ yields

$$\mathbf{t}_{k+1} = \mathbf{t}_k - J^+(\mathbf{t}_k)\mathbf{r}(\mathbf{t}_k).$$

For a variety of reasons it is prudent to use some form of stepsize control which leads to the following algorithm

$$\mathbf{t}_{k+1} = \mathbf{t}_k - \alpha_k J^+(\mathbf{t}_k)\mathbf{r}(\mathbf{t}_k),$$

where $-J^+(\mathbf{t}_k)\mathbf{r}(\mathbf{t}_k)$ gives a descent direction and α_k is usually chosen to guarantee that the step does in fact decrease the residual.

The critical step for us is computing the Jacobian since this involves differentiating an expression involving a generalized inverse of the Bernstein matrix. In particular, the jth column of the Jacobian matrix is given by

$$\frac{\partial \mathbf{r}(\mathbf{t})}{\partial t_j} = \begin{bmatrix} \frac{\partial P^{\perp}_{\mathcal{B}_n}}{\partial t_j}\mathbf{u} \\ \frac{\partial P^{\perp}_{\mathcal{B}_n}}{\partial t_j}\mathbf{v} \end{bmatrix}.$$

Notice that the Jacobian has an upper and a lower half, and furthermore, that to evaluate them we need to be able to evaluate

$$\frac{\partial P^{\perp}_{\mathcal{B}_n}}{\partial t_j}\mathbf{x},$$

where $\mathbf{x}$ can be either $\mathbf{u}$ or $\mathbf{v}$. It is essential therefore that we are able to differentiate a generalized inverse and we will find the following theorem will quite useful in this regard (see also (Hanson and Lawson, 1969) or (Golub and Pereyra, 1973)).

Theorem 1. *Let A be an $m \times n$ matrix function and let A^- be an $n \times m$ matrix function such that $AA^-A = A$ and $(AA^-)^{\mathrm{T}} = AA^-$. Then*

$$\frac{\partial P^{\perp}_A}{\partial t_k} = -P^{\perp}_A \frac{\partial A}{\partial t_k} A^- - \left(P^{\perp}_A \frac{\partial A}{\partial t_k} A^- \right)^{\mathrm{T}}. \tag{9}$$

Proof. First we note that by the product rule

$$\frac{\partial P_A A}{\partial t_k} = \frac{\partial P_A}{\partial t_k} A + P_A \frac{\partial A}{\partial t_k}$$

and since $P_A A = A$ we have

$$\frac{\partial A}{\partial t_k} = \frac{\partial P_A}{\partial t_k} A + P_A \frac{\partial A}{\partial t_k},$$

whence

$$\frac{\partial P_A}{\partial t_k} A = \frac{\partial A}{\partial t_k} - P_A \frac{\partial A}{\partial t_k} = P^{\perp}_A \frac{\partial A}{\partial t_k}.$$

Multiplying both sides on the right by A^- and using the fact that $P_A = AA^-$ yields

$$\frac{\partial P_A}{\partial t_k} P_A = P_A^\perp \frac{\partial A}{\partial t_k} A^-.$$

Transposing both sides and invoking the symmetry of P_A we also have

$$P_A \frac{\partial P_A}{\partial t_k} = \left(P_A^\perp \frac{\partial A}{\partial t_k} A^- \right)^{\mathrm{T}}.$$

Now, since P_A is idempotent

$$\begin{aligned} \frac{\partial P_A}{\partial t_k} &= \frac{\partial P_A^2}{\partial t_k} = \frac{\partial P_A}{\partial t_k} P_A + P_A \frac{\partial P_A}{\partial t_k} \\ &= P_A^\perp \frac{\partial A}{\partial t_k} A^- + \left(P_A^\perp \frac{\partial A}{\partial t_k} A^- \right)^{\mathrm{T}}. \end{aligned}$$

Finally, we note that $P_A^\perp = I - P_A$ and hence

$$\frac{\partial P_A^\perp}{\partial t_k} = -\frac{\partial P_A}{\partial t_k}$$

from which the result follows. □

Applying this to our case yields

$$\frac{\partial P_{\mathcal{B}_n}^\perp}{\partial t_j} = -Q_2 Q_2^{\mathrm{T}} \frac{\partial \mathcal{B}_n}{\partial t_j} \Pi \begin{bmatrix} R_{1,1}^{-1} Q_1^{\mathrm{T}} \\ 0 \end{bmatrix} - \begin{bmatrix} R_{1,1}^{-1} Q_1^{\mathrm{T}} \\ 0 \end{bmatrix}^{\mathrm{T}} \Pi^{\mathrm{T}} \frac{\partial \mathcal{B}_n^{\mathrm{T}}}{\partial t_j} Q_2 Q_2^{\mathrm{T}}.$$

It is clear that $\frac{\partial \mathcal{B}_n}{\partial t_j}$ is all zeros with the exception of its jth row which is

$$\frac{\partial}{\partial t_j} \begin{bmatrix} B_0^n(t_j) & B_1^n(t_j) & \dots & B_n^n(t_j) \end{bmatrix}$$

and this structure may be exploited to reduce the total number of operations required to evaluate the Jacobian. In particular, we can simultaneously compute all of the columns in the following way. Let $\mathcal{B}_n'(\mathbf{t})$ be the derivative of the Bernstein matrix which can be easily computed with

$$\mathcal{B}_n'(\mathbf{t}) = n\left\{ \begin{bmatrix} \underline{0} & \mathcal{B}_{n-1}(\mathbf{t}) \end{bmatrix} - \begin{bmatrix} \mathcal{B}_{n-1}(\mathbf{t}) & \underline{0} \end{bmatrix} \right\},$$

where $\underline{0}$ is a single column of zeros. Then the Jacobian is given by

$$\begin{bmatrix} Q_2 Q_2^{\mathrm{T}} \operatorname{diag}(P\mathbf{u}) + P^{\mathrm{T}} \operatorname{diag}(Q_2 Q_2^{\mathrm{T}} \mathbf{u}) \\ Q_2 Q_2^{\mathrm{T}} \operatorname{diag}(P\mathbf{v}) + P^{\mathrm{T}} \operatorname{diag}(Q_2 Q_2^{\mathrm{T}} \mathbf{v}) \end{bmatrix}, \tag{10}$$

where

$$P = \mathcal{B}_n' \Pi \begin{bmatrix} R_{1,1}^{-1} Q_1^{\mathrm{T}} \\ 0 \end{bmatrix}.$$

Careful implementation of this formula is important to maximize economies.

3. Implementation details

As we are fitting a single Bézier segment the problem may be simplified, without loss of generality, by fixing the nodes associated with the first and last data points to occur at the ends of the curve segment, in particular we set $t_1 = 0$ and $t_m = 1$ respectively. This has the effect of preventing the curve segment from growing excessively outside the region of the data set. Although it does not force the endpoints of the curve segment to coincide with the first and last points of the data set, it does restrict them to be tangents to circles centered at these points.

Indeed, experience shows that failure to enforce this restriction generally leads to serious problems as the curve segment tends to expand without bound. With this restriction it is useful to reduce the dimension of the search vector to $m - 2$ by deleting the first and last elements (t_1 and t_m). The Jacobian matrix becomes $(2m) \times (m - 2)$ and may be computed by replacing the first and last rows of $\mathcal{B}'_n$ with zeros, then evaluating formula (10), and finally deleting the first and last columns from the resulting matrix.

In our implementation we have used the stepsize control parameter α_k to enforce the condition that the nodal values satisfy $0 \leqslant t_i \leqslant 1$ for all i. If taking a full step would violate this constraint, then we set α_k to a value that will preserve it. Having done this, we then use a primitive line search before taking the step. In particular, we check to see if the current step will in fact decrease the residual. If not, then we repeatedly halve α_k until it does.

As with any iterative method, a good initial guess can be most valuable. The simplest is the chord-length parameterization. In particular, we let $\mathbf{d}_i = [u_i \; v_i]^{\mathrm{T}}$ and make our initial guess for the nodal values

$$t_i = \frac{\sum_{j=1}^{i} \|\mathbf{d}_i - \mathbf{d}_{i-1}\|_2}{\sum_{j=1}^{m} \|\mathbf{d}_i - \mathbf{d}_{i-1}\|_2}$$

for $i = 1, 2, \ldots, m$ with $t_0 = 0$.

This works reasonably well but we have noted better results using an affine invariant norm (see (Nielson, 1987)) in place of the 2-norm in the formula given above. In particular, we use the norm defined by

$$\|\mathbf{x}\|_V^2 = \mathbf{x}^{\mathrm{T}} V \mathbf{x},$$

where V is the inverse of the data covariance matrix, which is given by

$$V = \left(\frac{1}{m} \sum_{i=1}^{m} \mathbf{d}_i \mathbf{d}_i^{\mathrm{T}} \right)^{-1}.$$

Another excellent method is the *affine invariant angle* metric which is developed in the very elegant paper of Foley and Nielson (1989). It is somewhat more involved in its computation but worked very well in our experiments.

Finally, we use a traditional test for convergence and stop when

$$\frac{r_{\text{current}} - r_{\text{last}}}{r_{\text{last}}} < \text{tol},$$

where r_{current} is the squared residual at the current step, r_{last} is the squared residual at the last step, and tol is an arbitrary tolerance. In order to push the algorithm to the extreme we set $\text{tol} = 10^{-12}$ for the examples presented in the next section.

4. Bézier curve examples

In our first example we consider a fitting a data set of 23 points taken from an experiment on a reacting chemical system which may have multiple steady states (see (Marin and Smith, 1994)). The experiment samples the steady state oxidation rate R achieved by a catalytic system for an input concentration of carbon monoxide C_{CO}. The data is in log-log format and we show the results of fitting with segments of different degree in Figs. 1–6. The sixth-degree fitting (Fig. 6) is excellent and shows the proper ordering of the data from the first data point, d_1 at the left to the final data point, d_{23} at the lower right. Note that even with this very tight tolerance the convergence is fast in all but the fifth degree fittings (Fig. 5). If tol is reduced to 10^{-6} then the fifth degree fit converges in only 65 iterations with a final squared residual of 0.331×10^{-2}.

Of course, there may be local minima as is evidenced in the fitting of the fourth degree curve. In this case the algorithm converges to a local minimum that does not preserve the nodal ordering of the points (see Fig. 3). If, however, we use the affine invariant angle metric (see (Foley and Nielson, 1989)) to generate our initial guess then we converge to a different fit (Fig. 4) which does preserve the nodal ordering and has a squared residual that is about half as large.

The second set of fittings are to a Dillner 20-32-C low Reynolds number airfoil (all of the airfoil data used in this paper can be found at the UIUC Airfoil Data Site which can be accessed via world-wide web at http://amber.aae.uiuc.edu/~m-selig/ads.html). The results of this experiment appear in Figs. 7 and 8. The airfoil is represented by 35 points ordered counterclockwise starting from the top of the trailing edge. This airfoil has a relatively simple geometry and is easily fitted by the algorithm.

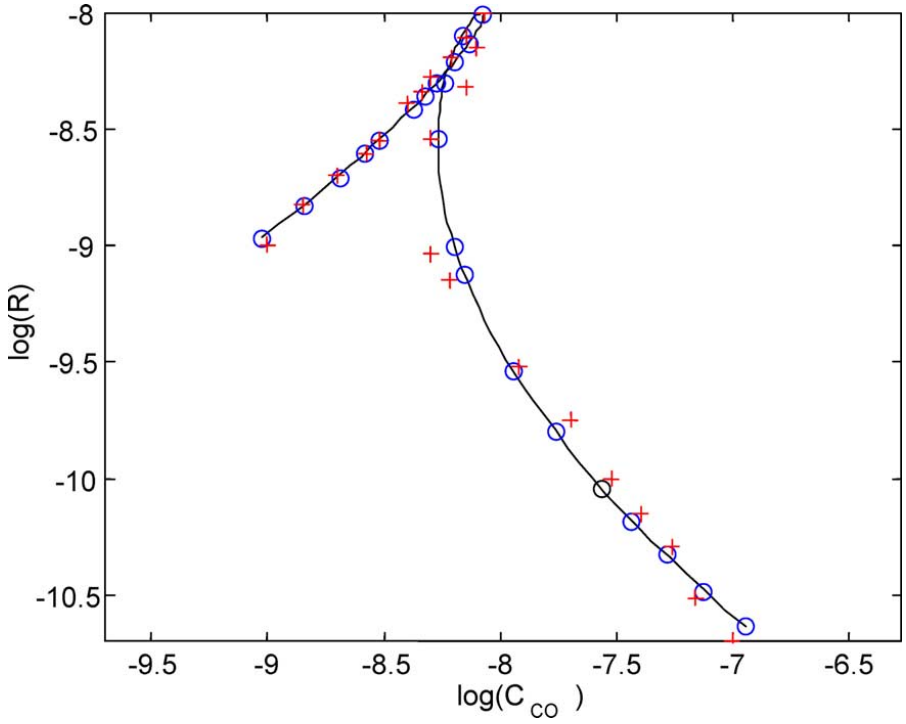

Fig. 1. Reaction rate data fitted with a third degree Bézier curve. The final squared residual was 0.567×10^{-1} which took 22 iterations. Note that a loop has appeared and the nodal points do not preserve the ordering of the data.

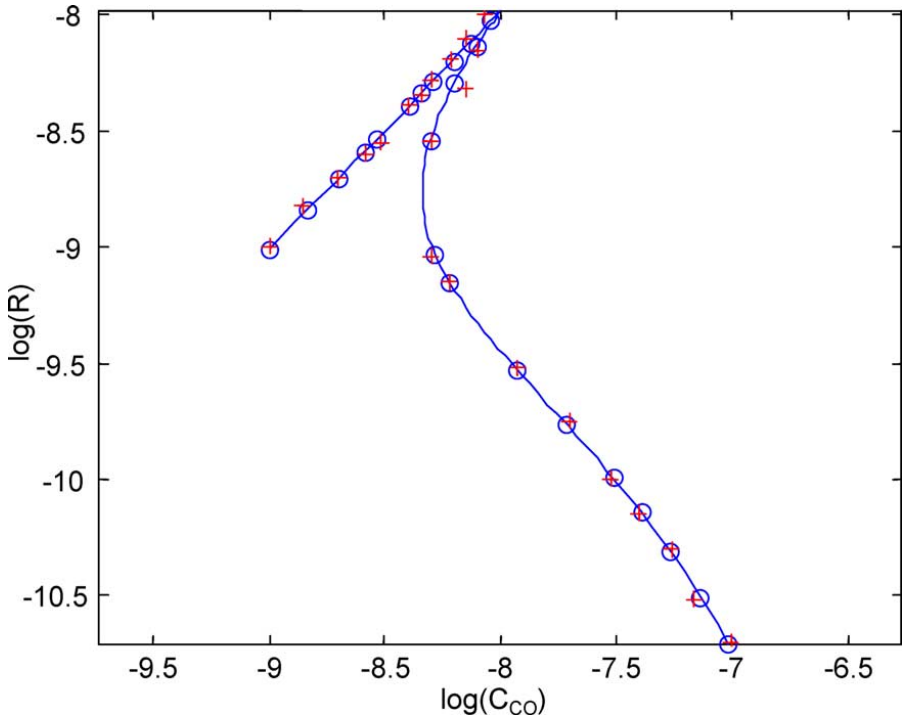

Fig. 2. Reaction rate data fitted with a fourth degree Bézier curve. The final squared residual was 0.136×10^{-1} which took 40 iterations. Note that the nodal points do not preserve the ordering of the data.

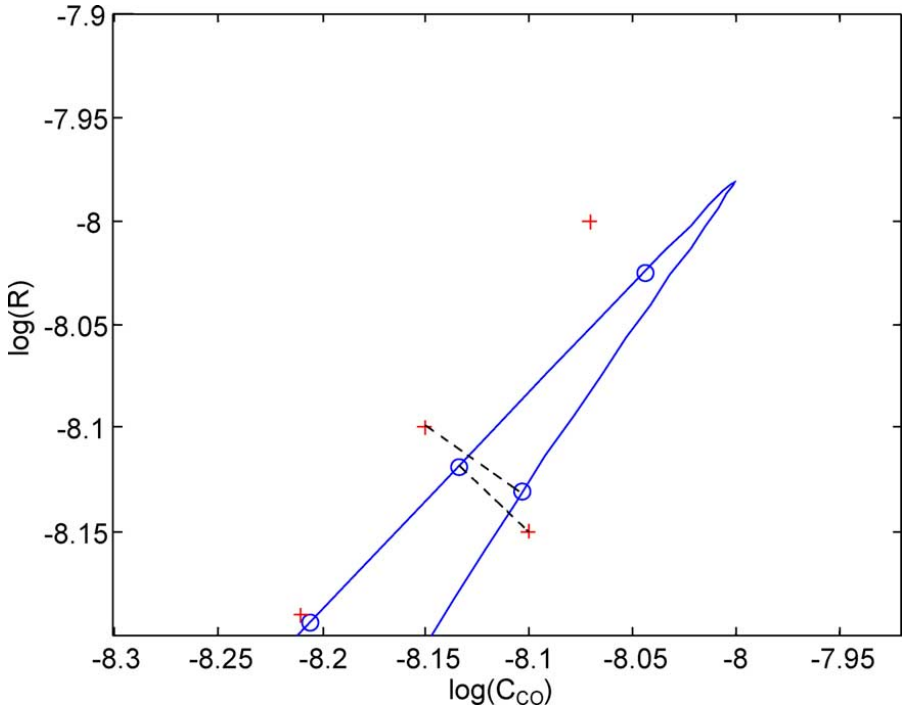

Fig. 3. A close-up of the fourth degree fitting. You can see that we have come to a local minimum as the nodal points associated with the d_{10} and d_{12} are not the closest points on the curve.

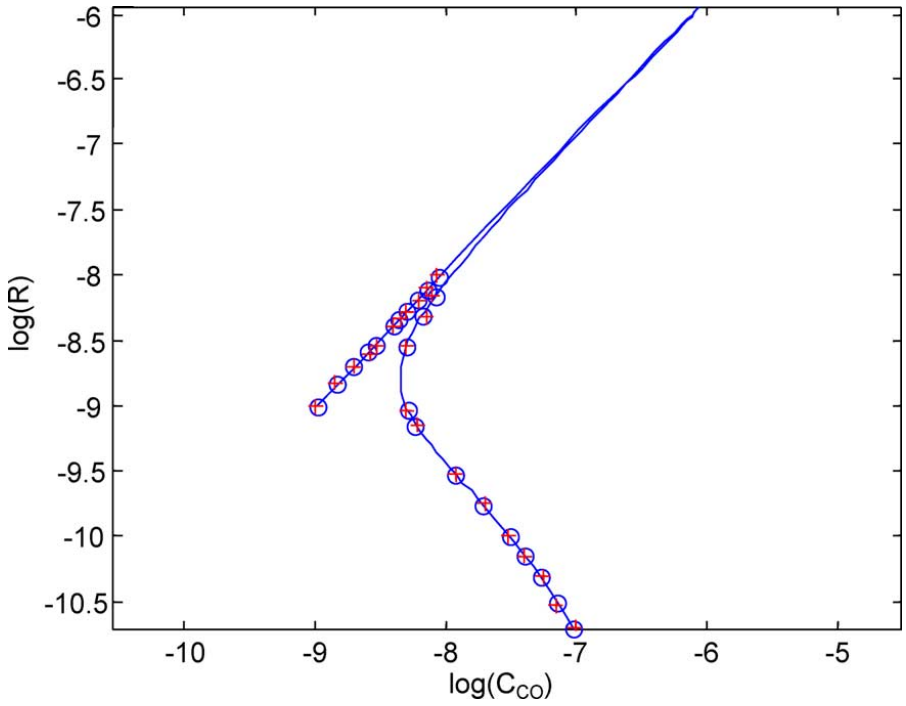

Fig. 4. Reaction rate data fitted with a fourth degree Bézier curve using a different starting guess yields a final squared residual of 0.602×10^{-2} and does preserve the nodal ordering.

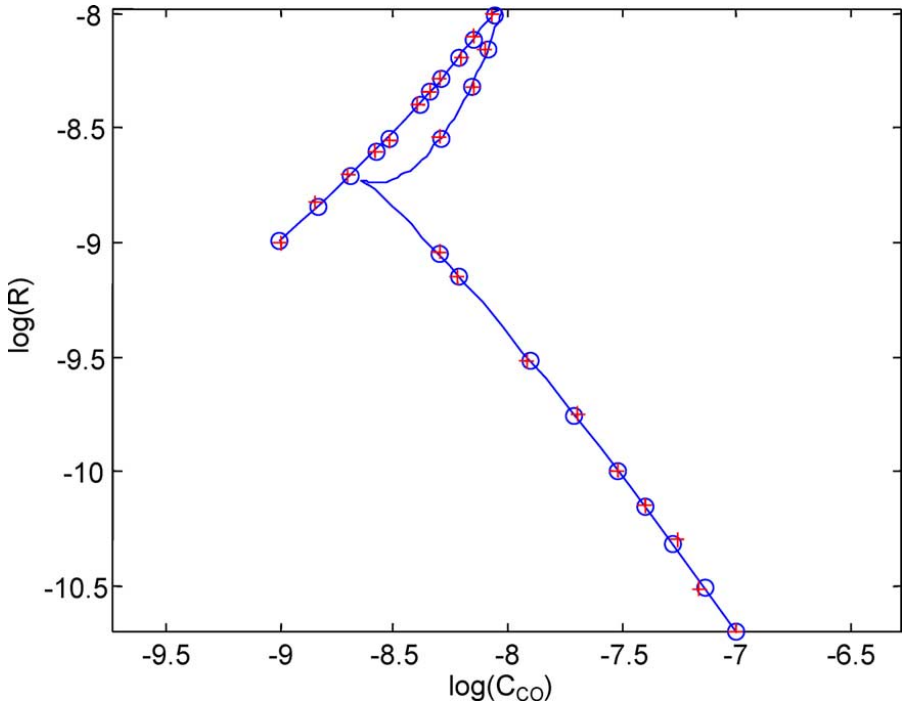

Fig. 5. Reaction rate data fitted with a fifth degree Bézier curve. The final squared residual was 0.330×10^{-2} which took 2371 iterations. Note the appearance of a cusp.

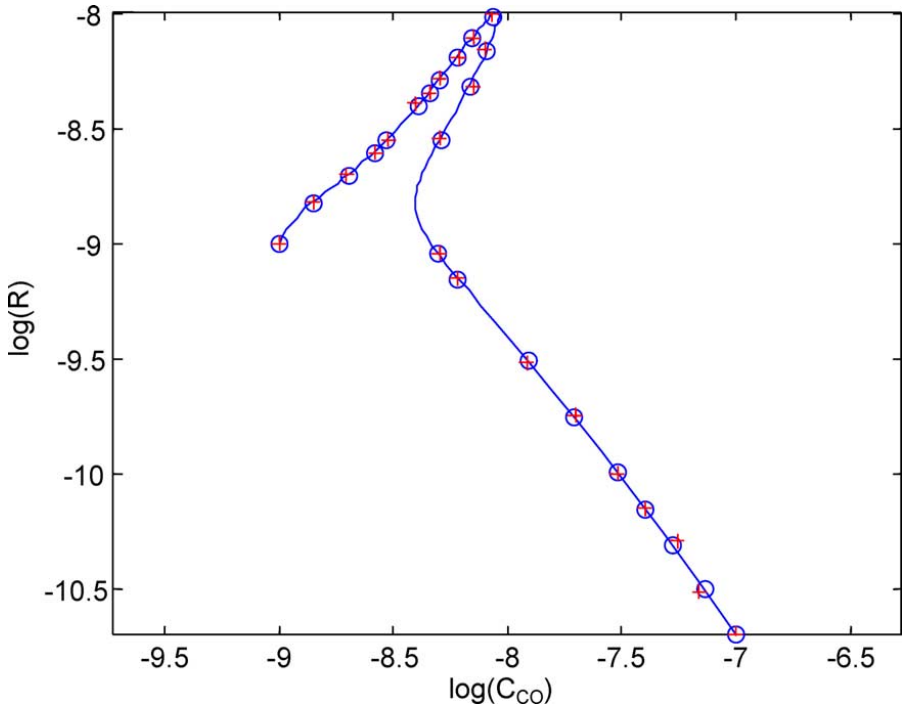

Fig. 6. Reaction rate data fitted with a sixth degree Bézier curve. The final squared residual was 0.234×10^{-2} which took 35 iterations.

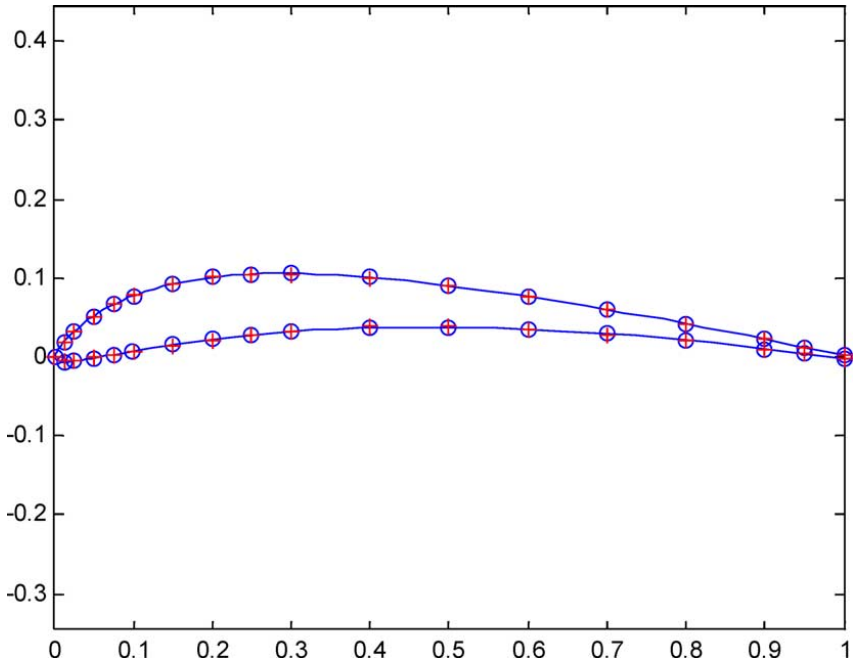

Fig. 7. Dillner 20-32-C low Reynolds number airfoil fitted with a single fifth degree Bézier curve segment. The final squared residual was 0.210×10^{-4} which took 22 iterations.

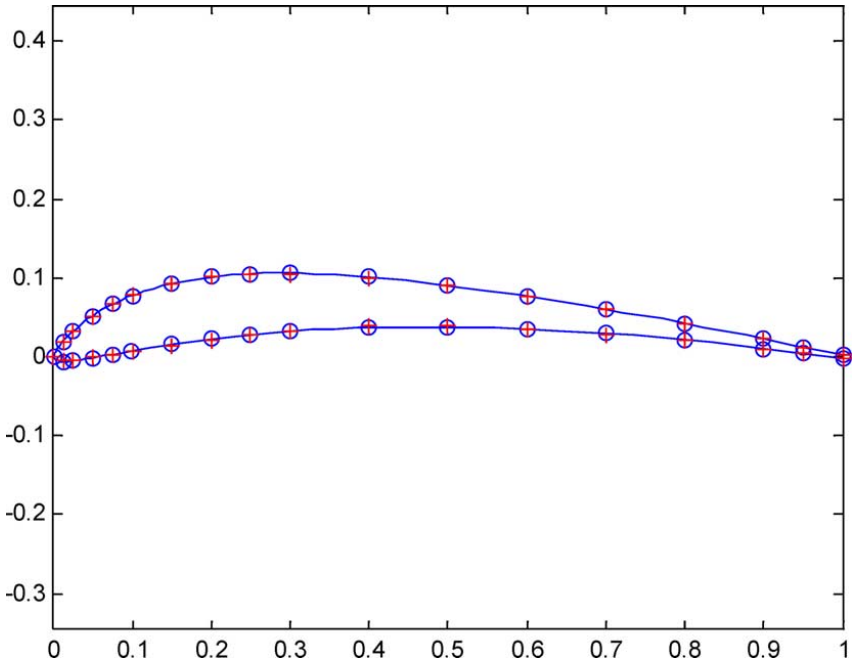

Fig. 8. Dillner 20-32-C low Reynolds number airfoil fitted with a single sixth degree Bézier curve segment. The final squared residual was 0.113×10^{-4} which took 44 iterations.

The final set of fittings are to a NACA/Munk M-27 airfoil. This airfoil is represented by 33 points also ordered counterclockwise starting from the top of the trailing edge. This airfoil has a more involved geometry than the first with more changes in concavity. Although the fifth-degree fitting has a small residual, it is not a good representation of the airfoil (Fig. 9). We can also see that convergence is relatively slow. However, when we use a sixth-degree segment (which does have sufficient flexibility to model the changes in concavity) we get a far more appealing fit (Fig. 10) and much faster convergence.

The speed of convergence can be improved by using a more sophisticated line search approach before taking each step. In some cases the higher per step costs are offset by substantial reduction in the number of iterations. We tested this by changing the step strategy to the following: If the Newton step decreases the residual, take it as is. If not then perform a line search to find the optimal stepsize in current direction and take that step. For the majority of the experiments presented here, this change in strategy causes little or no change in speed of convergence because it seldom comes into play (the full Newton step usually gives a decrease). However, when applied to the fifth degree fit of the reaction rate data it reduces the number of iterations from 2371 to 180. For the NACA/Munk M-27 airfoil fifth degree fit, this approach reduces the number of iterations from 689 to 201.

5. Extension to B-spline curves

It is very straightforward to extend preceding algorithm to the more general class of B-spline curves. Given a positive integer n and a non-decreasing knot sequence $k_0, k_1, \ldots, k_L$, where $L \geqslant 2n - 1$ we let

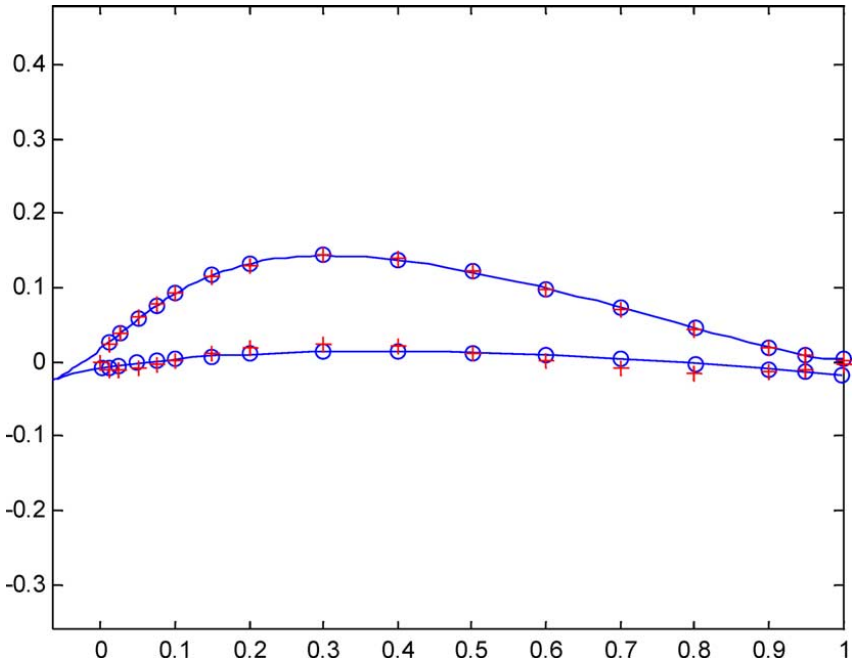

Fig. 9. NACA/Munk M-27 airfoil fitted with a single fifth degree Bézier curve segment. The final squared residual was 0.114×10^{-2} which took 689 iterations. This is a rather unappealing fit as a cusp has appeared at the leading edge.

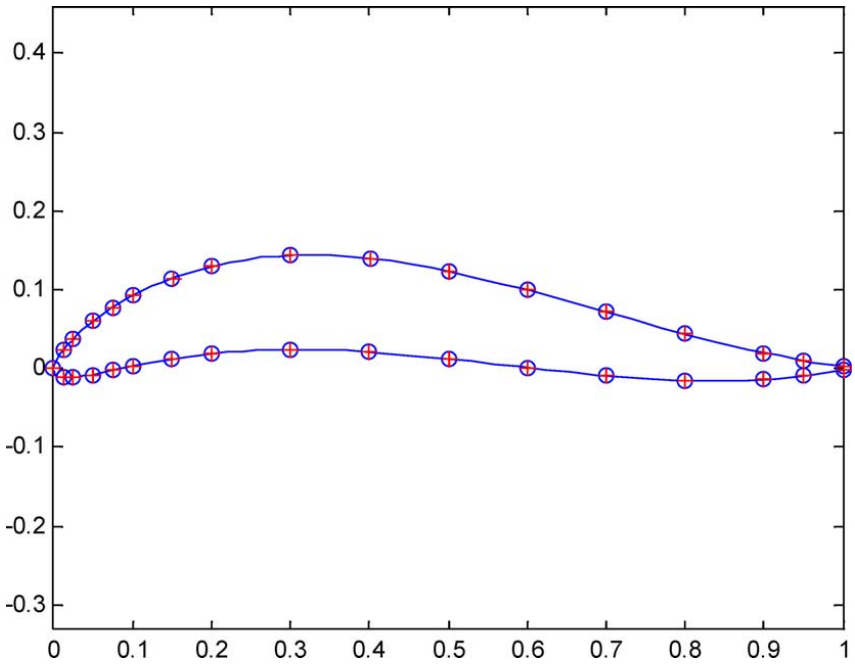

Fig. 10. NACA/Munk M-27 airfoil fitted with a single sixth degree Bézier curve segment. The final squared residual was 0.745×10^{-6} which took 13 iterations.

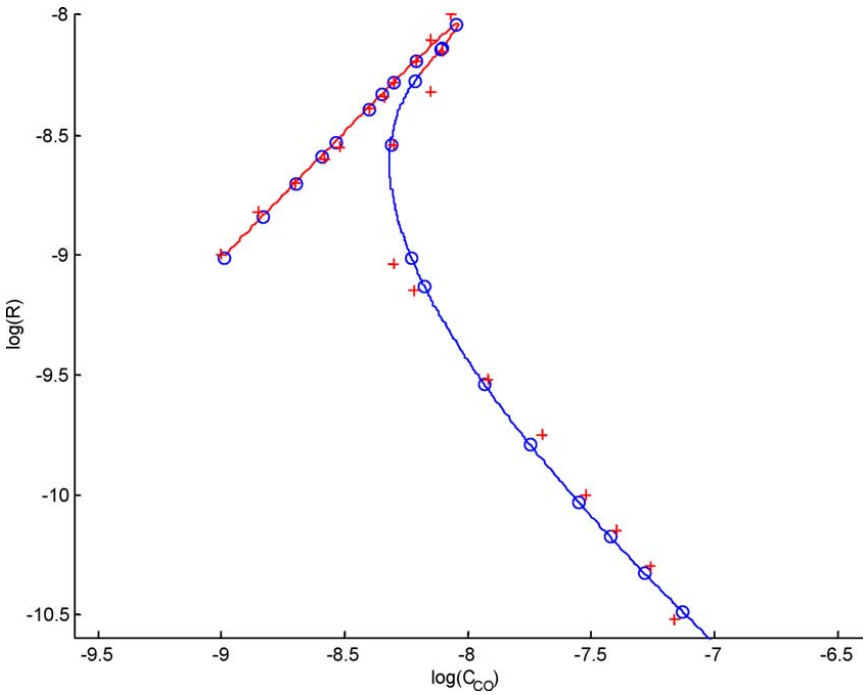

Fig. 11. Reaction rate data fitted with a third order B-spline curve with knot sequence $0, 0, 0, 1, 2, 2, 2$. The final squared residual was 0.361×10^{-1} which took 28 iterations.

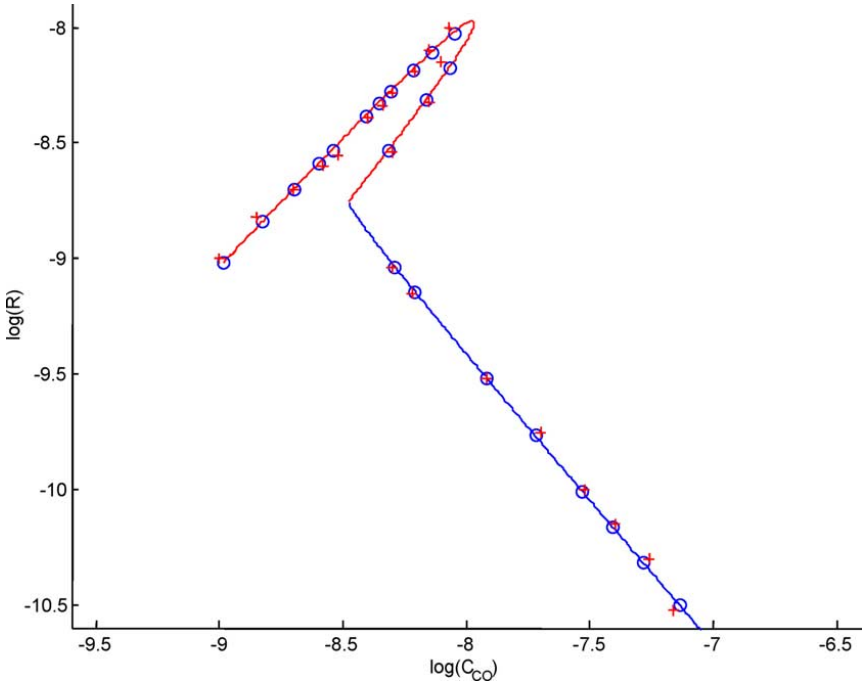

Fig. 12. Reaction rate data fitted with a third order B-spline curve with knot sequence $0, 0, 0, 1.95, 2, 2, 2$. The final squared residual was 0.102×10^{-1} which took 19 iterations.

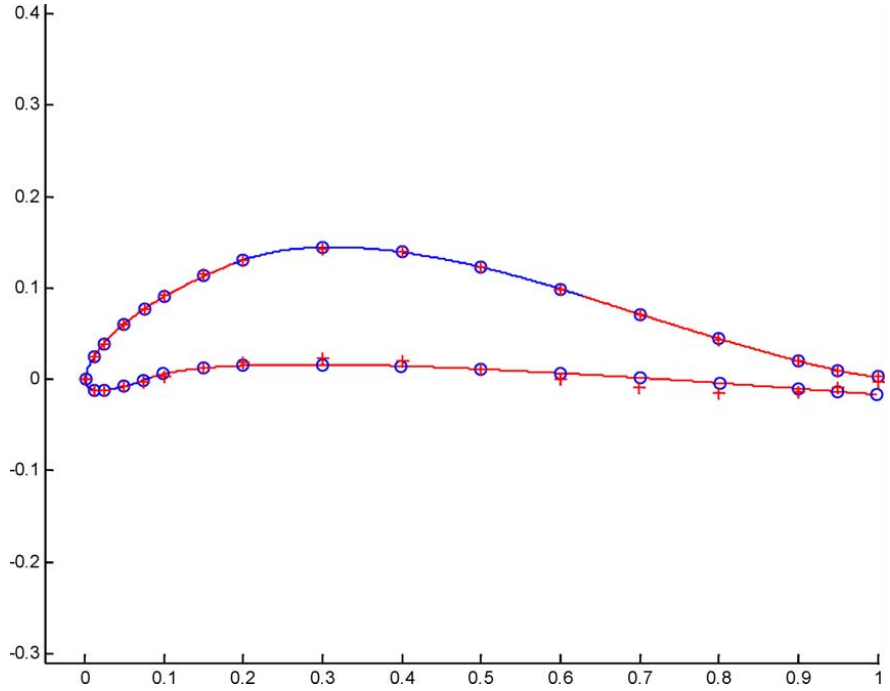

Fig. 13. NACA/Munk M-27 airfoil fitted with a third order B-spline curve with knot sequence 0, 1, 2, 3, 4, 5, 6, 7, 8, 9. The final squared residual was 0.649×10^{-3} which took 22 iterations.

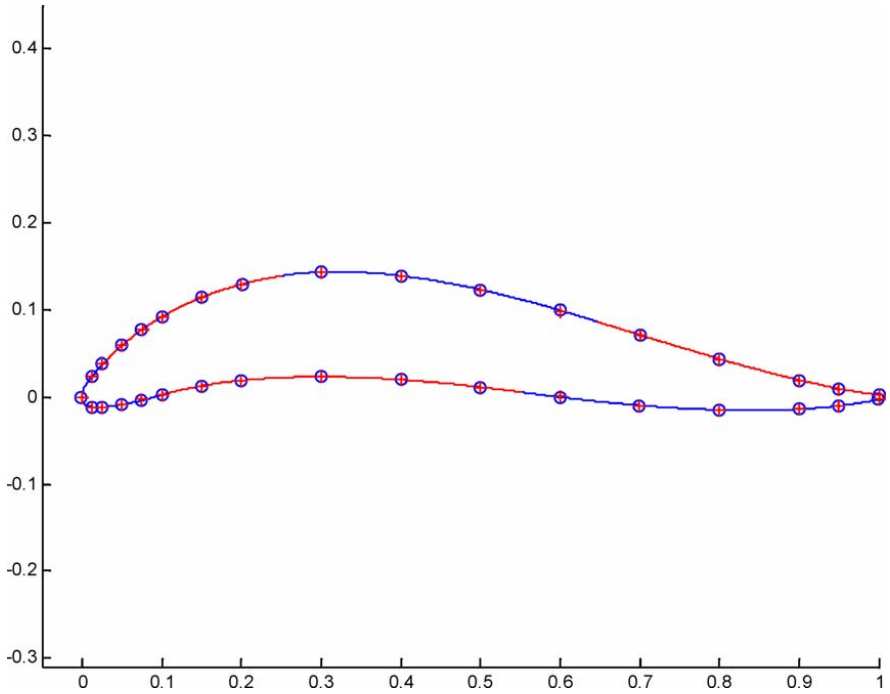

Fig. 14. NACA/Munk M-27 airfoil fitted with a third order B-spline curve with knot sequence 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 9. The final squared residual was 0.719×10^{-5} which took 43 iterations (the same result is achieved in 31 iterations using a warm restart with the nodes from the fitting in Fig. 13).

$N_i^n(t)$ for $i = 0, 1, 2, \ldots, L-n$ be the associated nth order B-spline basis functions (see (Farin, 1996)). The objective function becomes

$$\sum_{i=1}^{m}\left\{\left(u_i - \sum_{j=0}^{L-n} N_j^n(t_i)x_j\right)^2 + \left(v_i - \sum_{j=0}^{L-n} N_j^n(t_i)y_j\right)^2\right\}. \tag{11}$$

Another matrix function will simplify the notation. So

Definition 2. Given a vector $\mathbf{t} \in \mathbb{R}^m$, a positive integer n, and a non-decreasing knot sequence $k_0, k_1, \ldots, k_L$, where $L \geqslant 2n-1$ we define the *B-spline matrix*, which we denote by $\mathcal{N}(\mathbf{t})$, to be the real $m \times (L-n+1)$ matrix whose i, j element is $N_j^n(t_i)$. In particular

$$\mathcal{N}(\mathbf{t}) = \begin{bmatrix} N_0^n(t_1) & N_1^n(t_1) & \ldots & N_{L-n+1}^n(t_1) \\ N_0^n(t_2) & N_1^n(t_2) & \ldots & N_{L-n+1}^n(t_2) \\ \vdots & \vdots & & \vdots \\ N_0^n(t_m) & N_1^n(t_m) & \ldots & N_{L-n+1}^n(t_m) \end{bmatrix}.$$

The remainder of the derivation is the same as that for the Bézier curve segment (indeed the Bézier curve segment is a special case). One need only replace all occurences of the Bernstein matrix with the B-spline matrix. The greater generality of the B-spline curve is quite useful in fitting as one may use multi-segment curves. For our experiments we have used the same starting guesses as were used with the Bézier curve algorithm except that the values are affinely mapped to the domain of the given B-spline curve. The extremal data points are tied to the endpoints of the curve for the same reasons as before. We note that if a fitting is not acceptable it is reasonable to simply modify the knot sequence and try again. This brings up the possibility of using a *warm restart*, that is, using the nodal values from the previous fit as starting values for the new fit. Experience shows mixed results when doing this (see the example in Fig. 14 for an instance of this approach). Although the warm restart does sometimes lead to faster convergence, other times it actually leads to slower convergence.

References

Farin, G., 1996. Curves and Surfaces for Computer Aided Geometric Design: A Practical Guide, 4th ed. Academic Press, San Diego.

Foley, T., Nielson, G., 1989. A knot selection method for parametric splines, in: Lyche, T., Schumaker, L. (Eds.), Mathematical Methods in Computer Aided Geometric Design. Academic Press, New York, pp. 261–271.

Golub, G., Pereyra, V., 1973. The differentiation of pseudo-inverses and nonlinear least-squares problems whose variables separate. SIAM J. Numer. Anal. 10, 413–432.

Golub, G., Van Loan, C., 1996. Matrix Computations, 3rd ed. The Johns Hopkins University Press, Baltimore.

Hanson, R., Lawson, C., 1969. Extensions and applications of the Householder algorithm for solving linear least squares problems. Math. Comput. 23, 787–812.

Marin, S., Smith, P., 1994. Parametric approximation of data using odr splines. Computer Aided Geometric Design 11, 247–267.

Nielson, G., 1987. Coordinate free scattered data interpolation, in: Schumaker, L., Chui, C., Utreras, F. (Eds.), Topics in Multivariate Approximation. Academic Press, New York, pp. 175–184.

Rao, C., Mitra, S., 1971. Generalized Inverse of Matrices and it Applications. John Wiley, New York.