

# Surface Description Using Bicubic B-splines

## Curvature Based Smoothing of Plane Cubic B-spline Curves

**Hvid, Søren Lovmand; Larsen, Poul Scheel; Sørensen, Jens Nørkær**

*Publication date:*
1993

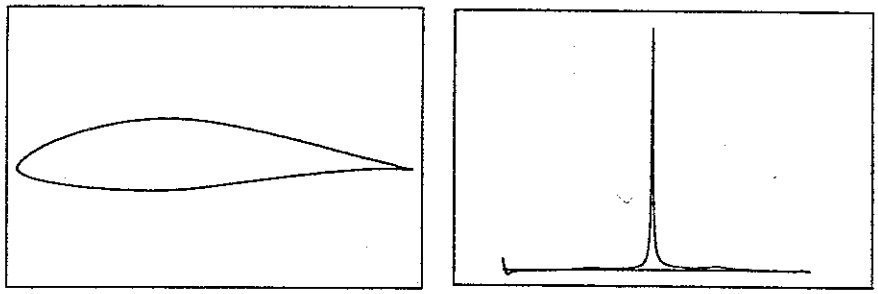
*Document Version*
Publisher's PDF, also known as Version of record

**AFM - 93-08**
ISSN 0590-8809

August 1993

# Curvature Based Smoothing of Plane Cubic B-spline Curves

Søren Lovmand Hvid



**DEPARTMENT OF FLUID MECHANICS**

**TECHNICAL UNIVERSITY OF DENMARK**

Printed 1993
Department of Fluid Mechanics,
Building 404,
Technical University of Denmark
DK–2800 Lyngby

# Curvature Based Smoothing of Plane Cubic B-spline Curves

Søren Lovmand Hvid

August 3, 1993

# Contents

# Preface

This dissertation is submitted in partial fulfilment of the requirements for the Danish Ph.D. degree from the Technical University of Denmark. The dissertation is based on theoretical work carried out at the Department of Fluid Mechanics during the period September 1990 to May 1993. The work has been carried out under the supervision of Professor, Ph.D. P. Scheel Larsen and Associate Professor, Ph.D. J. Nørkær Sørensen.

I would like to express my gratitude to my supervisors for their guidance and support throughout the period of my research, and during the preparation of this dissertation.

I would also like to express my gratitude to my colleagues at the department for their assistance and moral support.

Finally, I would like to thank my wife for her everlasting support, in spite of my intolerably late office hours.

## Abstract

Curve smoothing has become an important application for numerical manipulations of digitally described geometries. These manipulations include flow computations on e.g. wing sections defined by a tabulated set of coordinates.

In flow computations on wing sections, it is considered important not only to have a smooth geometry description, but also to be able to distribute points at will on the geometry. This calls for an interpolation scheme between the definition points, and cubic splines have long been in extensive use to this end.

It is also considered important that the shape of the profile, in particular in the leading edge region, not be altered by a smoothing algorithm. In order to fulfil this condition, a smoothing algorithm which ensures that the curvature at the leading edge is preserved, has been produced. The algorithm is based on the use of cubic B-splines for the curve representation, and the *local support* property of B-splines is of crucial importance to the algorithm.

Two versions of the curve smoothing algorithm are supplied and documented in this report:

**Smooth2D:** Smoothing of 2D curves in general.

**Smoothfoil:** Smoothing of airfoils.

# Chapter 1

# Introduction

In a number of different applications involving digital representation of geometries, there is a need to obtain smooth geometries to some extent. This may be for aesthetic reasons as in car design, or it may be for performance reasons as in wing design. Cubic B-splines have become the standard choice for computer aided design applications, and much effort has been invested in providing algorithms which produce smooth interpolations of discrete geometries. In this context, *smooth* refers to the variation of the curvature of the geometry, where the curvature $\kappa$ is the inverse of the radius of curvature $\rho$ at all points.

A number of different approaches to the curve smoothing (or fairing) problem have been investigated in the past. Knot removal and insertion algorithms have been investigated extensively [5, 12], Farin has used a *degree reduction fairing* algorithm succesfully [4], while Kjellander repositions points on cubic Hermitian interpolants to curve segments [9].

In this report, a method for smoothing plane cubic B-spline curves will be presented. The fundamental principle of the method is to prescribe a smooth curvature variation for which the corresponding curve is then computed.

# Chapter 2

# Cubic B-spline curve basics

B-spline curves are piecewise polynomial curves defined over a set of knots or parameter values. Given a knot set $u_i$ and a control polygon $d_i$, the cubic B-spline curve $s(u)$ is defined:

$$s(u) = \sum_{i=-1}^{n+1} d_i N_i(u) \quad , \quad u_0 \leq u_n \tag{2.1}$$

where $N_i, i = -1, ..., n+1$, are the *normalized cubic B-splines* defined as

$$N_i(u) = (u_{i+2} - u_{i-2}) \sum_{j=i-2}^{i+2} \frac{(u-u_j)_+^3}{w_j'(u_j)} \quad , \quad u_0 \leq u_n. \tag{2.2}$$

Here $(u-u_j)_+^3$ is defined by

$$(u-u_j)_+^3 = \begin{cases} 0 & \text{if } u < u_j \\ (u-u_j)^3 & \text{if } u \geq u_j \end{cases} \tag{2.3}$$

and $w_j$ is the 5th degree polynomium

$$w_j(u) = \prod_{k=j-2}^{j+2} (u - u_k). \tag{2.4}$$

The normalized cubic B-splines $N_i$ posess the following properties:

$$N_i(u) = 0 \text{ for } u_0 \leq u < u_{i-2} \text{ and } u_{i+2} < u \leq u_n \tag{2.5}$$

$$N_i^{(p)}(u_{i-2}) = N_i^{(p)}(u_{i+2}) = 0 \text{ for } p = 0, 1, 2 \tag{2.6}$$

$$N_i(u) > 0 \text{ for } u_{i-2} < u < u_{i+2} \tag{2.7}$$

$$\sum_{i=-1}^{n+1} N_i(u) = 1 \text{ for } u_0 \leq u \leq u_n \tag{2.8}$$

As to the computation of B-splines, the definition (2.2) is never used. Instead it is possible to employ a recursive formula due to de Boor. For a detailed description of B-spline curves including algorithms for practical computations, consult [3] or [11].

Some important properties of B-spline curves are:

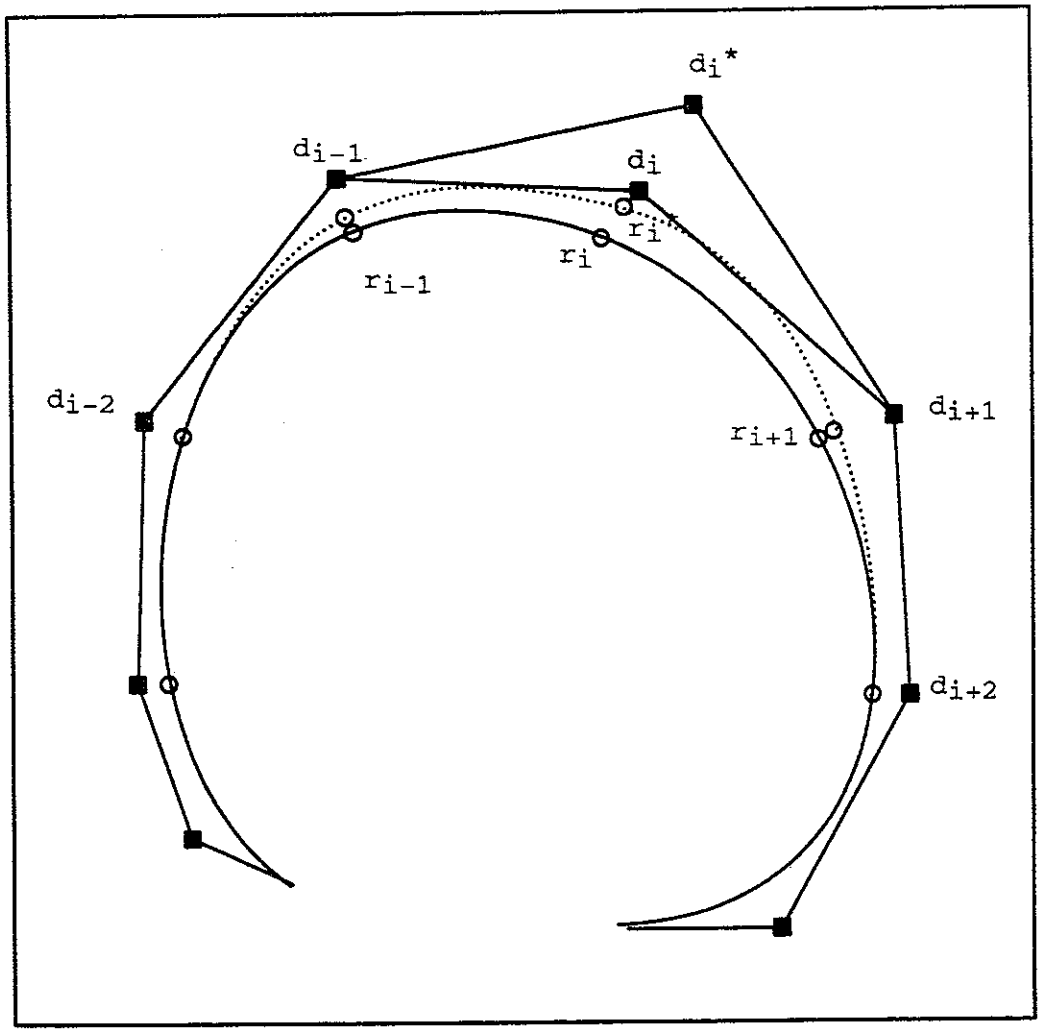



Figure 2.1: Effect of moving a cubic B-spline control point

**Linear precision** A straight line is reproduced as a straight line.

**Convex hull property** Each point on the curve lies in the convex hull of no more than four control points.

**Local control** Each segment of the curve is influenced by no more than four control points.

The property of local control is very useful when manipulating with spline curves, since the sphere of influence is known a priori. If one B-spline polygon vertex is moved, the corresponding curve only changes in the immediate vicinity of the displaced point. In the case of cubic B-splines, moving one B-spline control point affects only the two neighbouring segments on the curve. to each side of the displaced point. This is illustrated in figure 2.1.

The curvature $\kappa$ of a parametric curve is given by

$$\kappa = \frac{|\dot{\vec{r}} \times \ddot{\vec{r}}|}{|\dot{\vec{r}}|^3}, \tag{2.9}$$

where $\dot{()} = \frac{\partial()}{\partial u}$. In the case of a plane curve ($\vec{r}(u) = \{x, y\}^T(u)$, and equation (2.9) reduces to

$$\kappa = \frac{\ddot{x}\dot{y} - \dot{x}\ddot{y}}{(\dot{x}^2 + \dot{y}^2)^{3/2}} \tag{2.10}$$

Since the cubic spline is $C^2$ everywhere, we may note that $\vec{r}^{*(p)}_{i-2} = \vec{r}^{(p)}_{i-2}$ , $p = 0, 1, 2$ and $\vec{r}^{*(p)}_{i+2} = \vec{r}^{(p)}_{i+2}$ , $p = 0, 1, 2$. Consequently, the curvature only changes at $\vec{r}_{i-1}$, $\vec{r}_i$ and $\vec{r}_{i+1}$ as a result of the move.

More information on the subject of B-splines and their usage may be found in the textbooks listed in the reference section.

# Chapter 3

# Curvature smoothing

## 3.1 Curvature and smoothness

How is smoothness of a curve defined? One may think of smoothness in terms of being $C^n$ where a larger $n$ means a smoother curve, since it has continuous derivatives up to the $n$th derivative. However, this does not mean that the curve will appear to be smooth in the aesthetic sense, or in the sense of not changing rapidly within a short distance. To a stylist or a designer a *fair* or *smooth* curve is, in popular terms, one which can be drawn using only a small number of french curves. These curves are of simple form, e.g. spirals, and have curvatures that vary smoothly (e.g. linearly) with arc length. It therefore appears that curvature is an important factor when judging the smoothness of a curve in the aesthetic sense. Consider *smoothness* in the context of the resulting flow over a surface. It would be natural to expect that a *smooth* surface does not introduce disturbances in the flow. This implies that the variation in curvature should be slow and (at the very least) continuous. This has been known for a long time in the field of aerodynamics, where good characteristics of a wing or an airfoil is only obtained if it is free of 'bumps'. This is the equivalent of not having a rapidly varying curvature, and in particularly not having inflection points, i.e. points where the sign of curvature[1] changes. In order to produce a smooth curve, we therefore have to produce a curve with a smooth variation of curvature.

At this point it is worth noting, that there is a unique connection between a plane curve and its curvature, the curve being completely defined by prescribing its curvature but for an arbitrary rigid body displacement.

[1]Note that equation (2.10) allows a definition of signed curvature. In general the curvature of a curve is unsigned, since it is just the inverse of the *radius of curvature* which is always positive

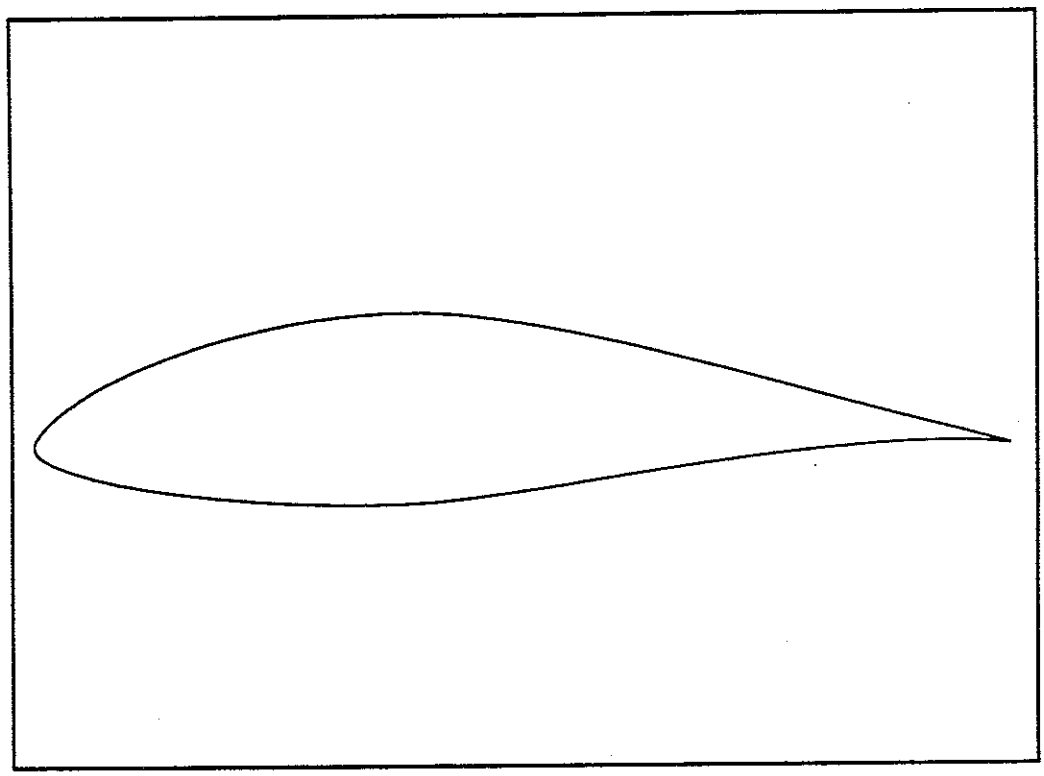


Figure 3.1: Cubic B-spline approximation to an airfoil

### 3.1.1 Curvature plots

A very useful tool in the generation of smooth curves is that of the *curvature plot*. This is a plot that shows how the curvature changes as one moves along the curve. It is very effective when one has to determine the smoothness of a curve, and perhaps in which areas the curve is not good.

In figure 3.1 a curve is shown, which is a cubic B-spline approximation to a given data set (in this case an airfoil), and in figure 3.2 the curvature plot corresponding to this curve is shown.

It is obvious from the curvature plot that the airfoil is not smooth. Many oscillations are present on the curvature plot, and even if these are not visible to the eye on the plot of the airfoil, they may quite well be of some significance, e.g. in a flow computation of the airfoil. It is therefore desirable to eliminate oscillations such as these.

In design applications, the curve may usually be manipulated quite freely, as long as the designer is still satisfied with it. Therefore, the restraints enforced on a curve smoothing algorithm are only such as to improve smoothness without altering the shape of the curve 'too much'. In the case of the airfoil, this notion of 'too much' takes an altogether more restricted meaning, for if the original curve is altered noticeably, one has in effect created a new airfoil with - perhaps - different characteristics than those of the original. In other words, a limit on the displacement of points is desired. This may either be included in the smoothing algorithm, or it

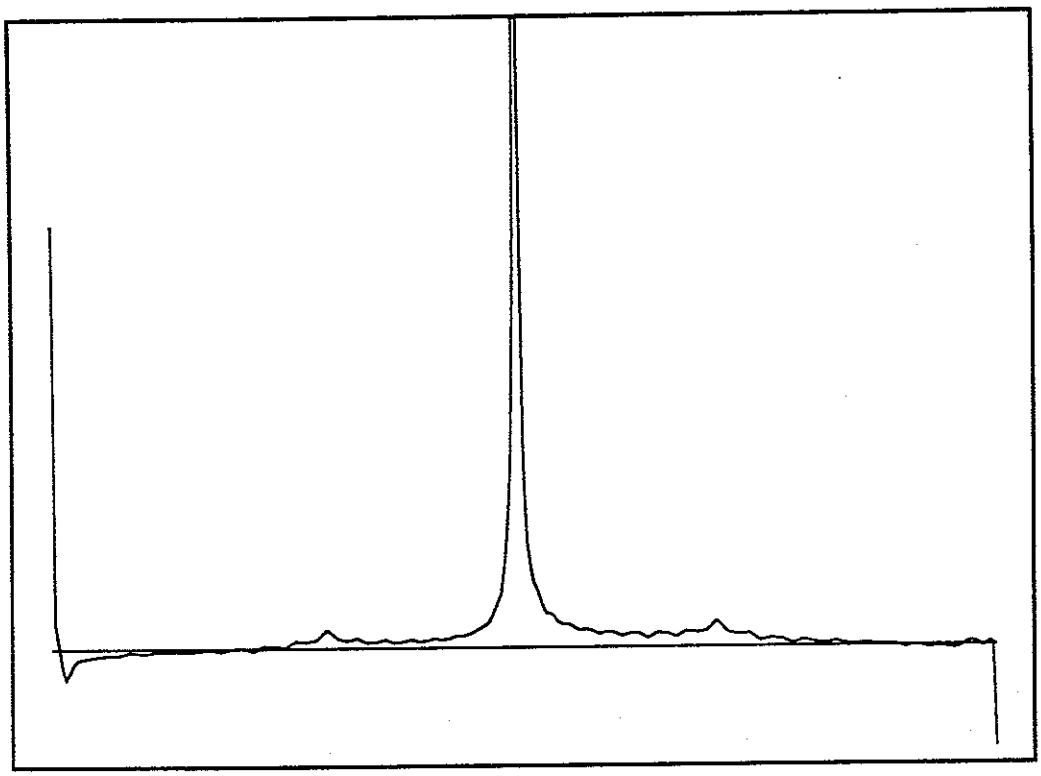


Figure 3.2: Curvature plot of the airfoil shown in figure 3.1

may be verified after smoothing has been performed. The latter strategy has been followed in the present work, where it is then left to the user to accept what he considers an acceptable result.

## 3.2 A smoothing algorithm

Bearing in mind that there is a unique relationship between a plane curve and its curvature, and that cubic B-splines are of local control, a smoothing algorithm may be formulated, which computes a curve from a given (smooth) curvature variation, rather than trying to adjust the curve in some way, and then checking whether or not the curve has become smoother. The algorithm may loosely be formulated in the following way:

*Given a B-spline polygon with vertices* $\vec{d}_{-1}, ...., \vec{d}_{n+1}$, *which interpolates to the* $n+1$ *data points* $\vec{r}_0, ...., \vec{r}_n$ *at the parameter values* $u_0, ...., u_n$. *Smooth the interpolating curve by adjusting the polygon vertices* $\vec{d}_i$, *according to the following steps:*

1. *Compute the curvature* $\kappa_i$ *at each* $u_i$.

2. *Compute an improved curvature variation* $\kappa^*(u)$ *by smoothing the original curvature variation* $\kappa(u)$. *Compute offset curvatures* $\Delta\kappa_i$ *at each* $u_i$.

3. *Compute the Jacobian* $\boldsymbol{J} = \frac{\partial \kappa_i}{\partial \varepsilon \vec{n}_i}$, *where* $\varepsilon \vec{n}_i$ *denotes the displacement of the polygon vertex* $\vec{d}_i$ *in the direction of the normal* $\vec{n}_i$ *to the curve at* $\vec{r}_i$.

4. *Solve the system* $\boldsymbol{J}\vec{\delta} = \vec{\Delta\kappa}$ , *where* $\vec{\Delta\kappa} = \vec{\kappa} - \vec{\kappa}^*$.

5. *The solution* $\vec{d}_i^*$ *is then:* $\vec{d}_i^* = \vec{d}_i + \delta_i \vec{n}_i$.

The curvatures $\kappa_i$ to be computed in step 1 are easily found, since both first and second derivatives are implicitly given in the B-spline interpolation algorithm, see [11]. In other words, no subsequent calculation of these derivatives using finite differences is needed, improving both speed and accuracy of the algorithm.

The smoothing of the curvature variation in step 2 may be performed in a multitude of ways. Of interest is only the result of the smoothing, which should preserve all significant features of the curvature variation. This poses a severe problem for most smoothers, because it is not known a priori which variations (which scales) are significant and which not. Indeed, both significant and insignificant variations may be expected to be present on all scales. The solution to this problem is in most cases the interaction with an intelligent user. In the present code the user may choose to use either a linear least squares algorithm, or an *ad hoc* smoother: $\kappa_i^* = \frac{1}{3}(\kappa_{i-1} + \kappa_i + \kappa_{i+1})$.

Computation of the Jacobian $\boldsymbol{J} = \frac{\partial \kappa_i}{\partial \varepsilon \vec{n}_i}$ is performed by using a small displacement $\varepsilon$ and finding the corresponding change in curvature. In other words $\frac{\partial \kappa_i}{\partial \varepsilon \vec{n}_i}$ is approximated by $\frac{\Delta \kappa_i}{\Delta \varepsilon \vec{n}_i}$.

The Jacobian $\boldsymbol{J}$ is tridiagonal, following earlier remarks, so the system in step 4 may be solved very efficiently. Steps 3,4 and 5 comprise a secant method (a Newton method if $\boldsymbol{J}$ had not been approximated), and are repeated until a predefined accuracy is obtained. This is usually reached in very few iterations.

### Conditioning the algorithm

The algorithm works well as described above, but for practical problems it is usually necessary to redistribute the data points on the curve. This becomes necessary when the distribution of points in e.g. a data set is very irregular, or if it is too coarse. If the smoothing algorithm is applied to such an irregular set of points, the displacements will usually not be within acceptable bounds. Apart from the need to have enough points to allow for a flexible adjustment, it is also of importance to distribute these in a suitable manner, i.e. more points in highly curved regions than elsewhere. It is not often that these conditions are met in a tabulated data set. As part of the algorithm, it is therefore made possible to prescribe a distribution of points, which is able to employ the curvature as a weighting factor. The smoothing algorithm is then applied to the curve described by the redistributed point set.

In the case of the airfoil, it is also important that the chord length, which is used as a reference, is preserved. To this end, a scaling of the smoothed airfoil has been implemented, so that the chord length will be the same as the one of the original airfoil.

### 3.2.1 Smoothing an airfoil

Many airfoils or wing sections are given as a tabulated set of coordinates, which originate from a digitizing of the actual airfoil. In order to give a continuous description of the airfoil one must then interpolate in some way between the given points. The cubic spline is extensively used for this task. However, digitizing errors from the actual measurement of the airfoil will be present in the description, and so will rounding errors as the coordinates in the table are rounded to typically two or three significant digits. The errors thus induced will not be apparent in a plot of the airfoil itself, but they will have a clear effect on the curvature of the airfoil. This has been illustrated earlier in figures 3.1 and 3.2, which show an airfoil and its corresponding curvature plot. The wing section here is given as a table of 86 points, and has been plotted using 150 points. The oscillations present in the curvature plot originate from digitizing or rounding errors. The smoothing algorithm will now be applied to this airfoil.

First an improved curvature variation is found by smoothing the original one. Four passes through the *ad hoc* smoother mentioned earlier produces the result shown in figure 3.3.

Offset curvatures $\Delta\kappa_i$ are then computed at each $i$, and the iterative solution method (steps 3,4 and 5) is commenced. After a few iterations the convergence criterion has been fulfilled. The deviation between the smoothed and the original airfoil measured in percent of the chord is shown in figure 3.4, as a function of normalized arc length. The deviation $ds$ has been computed as the distance between a point on the original airfoil and the "same" point on the smoothed airfoil, i.e. points at the same parameter value $u_i$ using identical distribution functions. The reason for this is purely ease of implementation, but it means that $ds$ is in effect an *upper bound* for the deviation between the airfoils.

It is obvious that the smoothed result is very close to the original airfoil, the largest deviation being roughly $3.0 \times 10^{-4}$ or 0.03% of the chord. The two airfoils are practically indistinguishable.

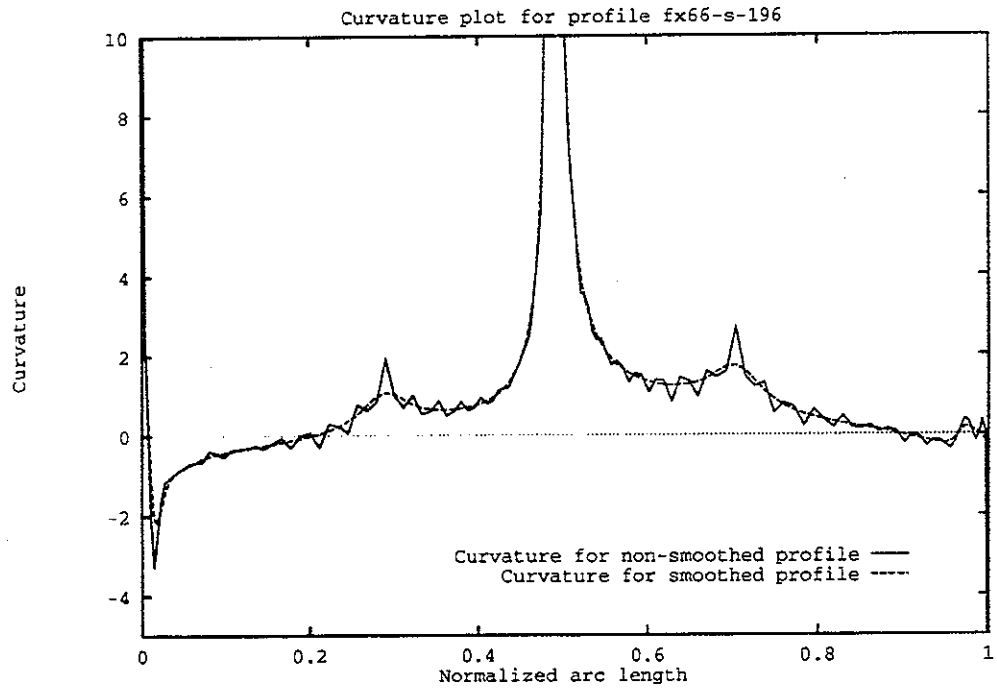



Figure 3.3: Smoothed and non-smoothed curvature variation for airfoil fx66-s-196

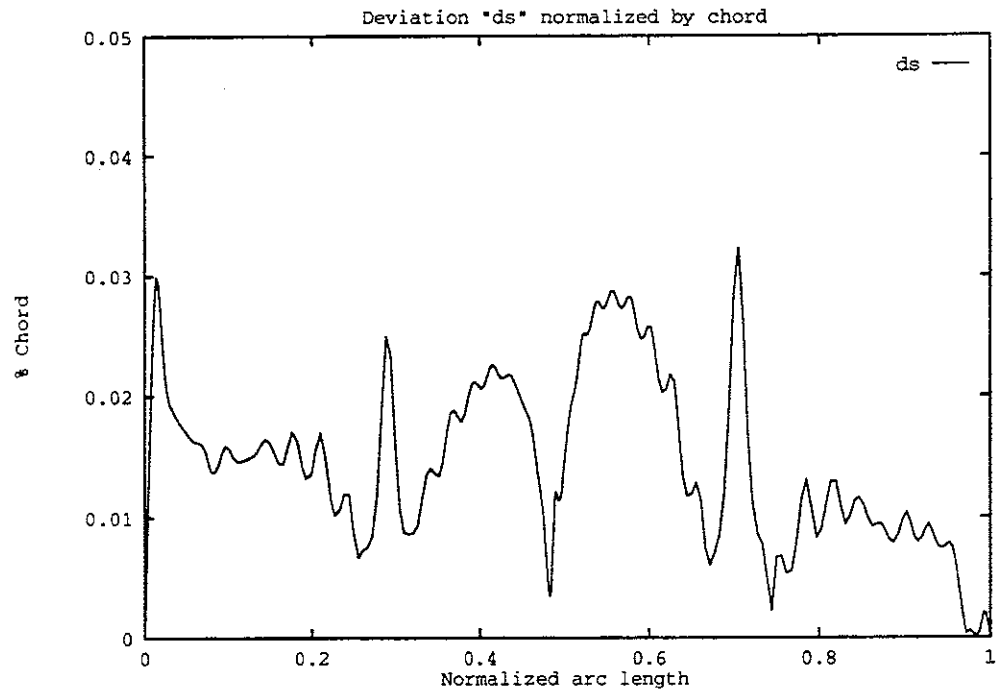



Figure 3.4: Deviation between smoothed and non-smoothed airfoil fx66-s-196. The difference is extremely slight

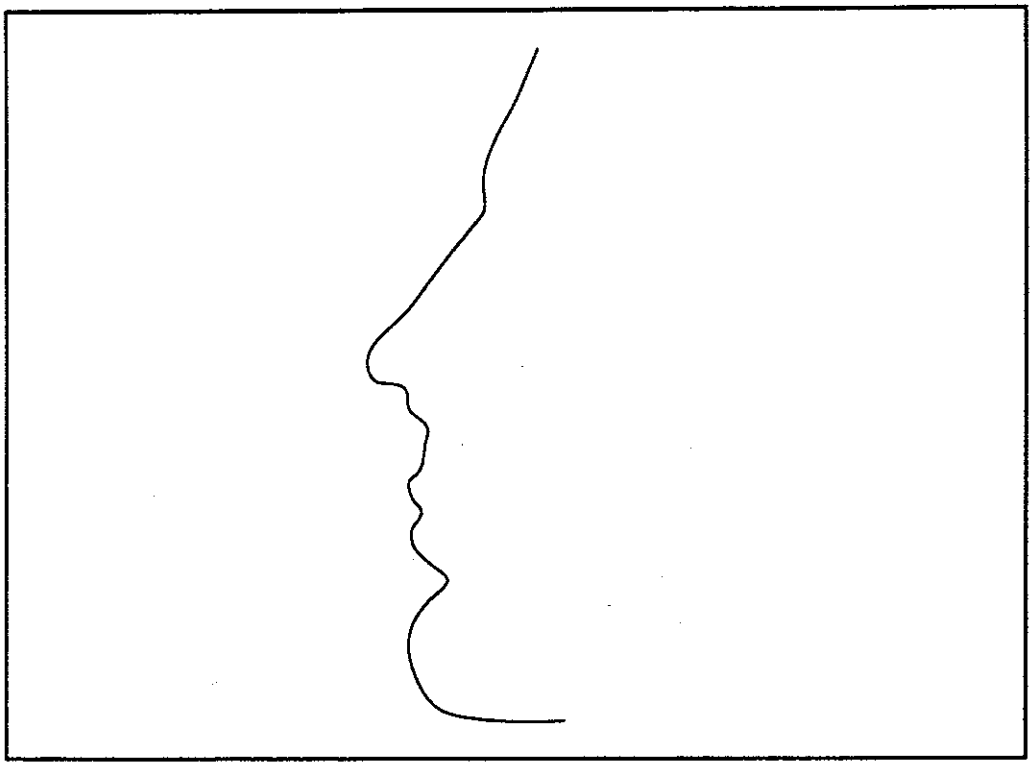


Figure 3.5: A face profile

### 3.2.2 Smoothing a face profile

As for the wing profile the face profile is given by a set of data points $\vec{r}_i$, measured on a face. The profile is shown in figure 3.5.

It is obvious that a curve such as this has a very complicated curvature variation, and this is shown in figure 3.6.

It is not immediately obvious which variations are significant to the form of the face, and which variations one may contribute to errors. One may be sure though that errors are present, and probably to a considerable extent. Smoothing the face with the developed smoother yields the result shown in figure 3.7, where both smoothed and non-smoothed face profiles are present. The corresponding curvature variations are shown in figure 3.8. Notice, that it has not been possible to smooth out completely the oscillations deemed insignificant. If one tries to smooth more, using the present form of the algorithm, the solution will not converge. This problem could probably be solved by inventing a better type of smoother for the curvature variation than the *ad hoc* scheme applied in the algorithm. This better smoother should be able to decide which oscillations to remove, and which not.

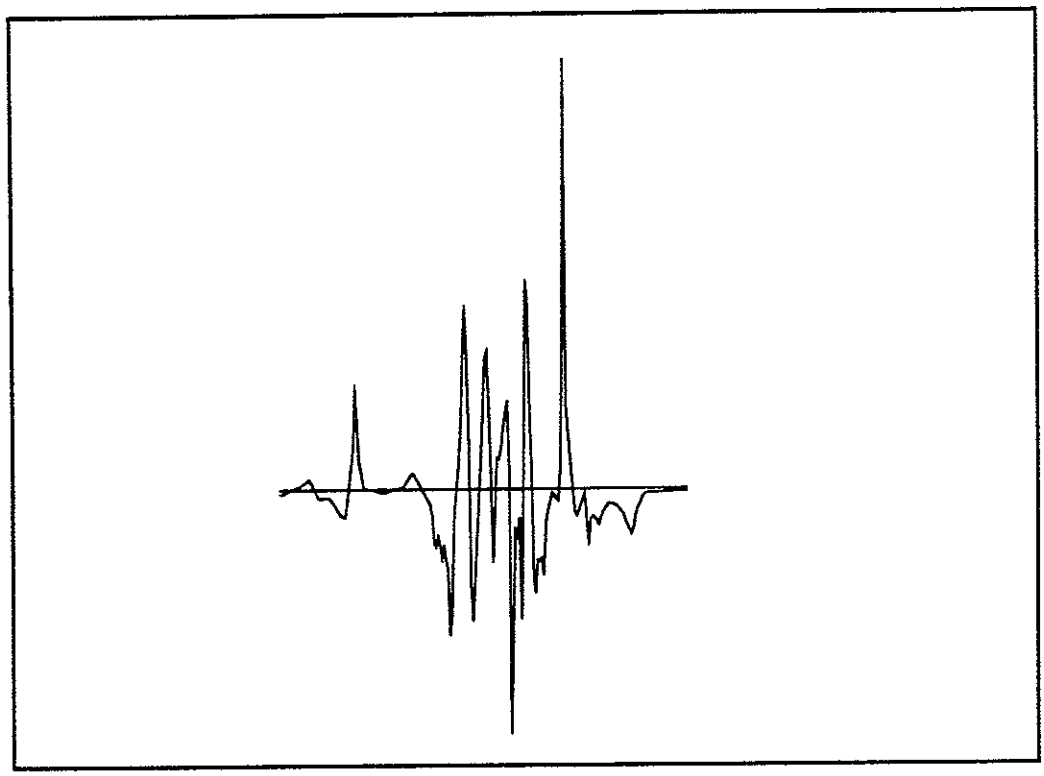


Figure 3.6: Curvature variation of the face in figure 3.5

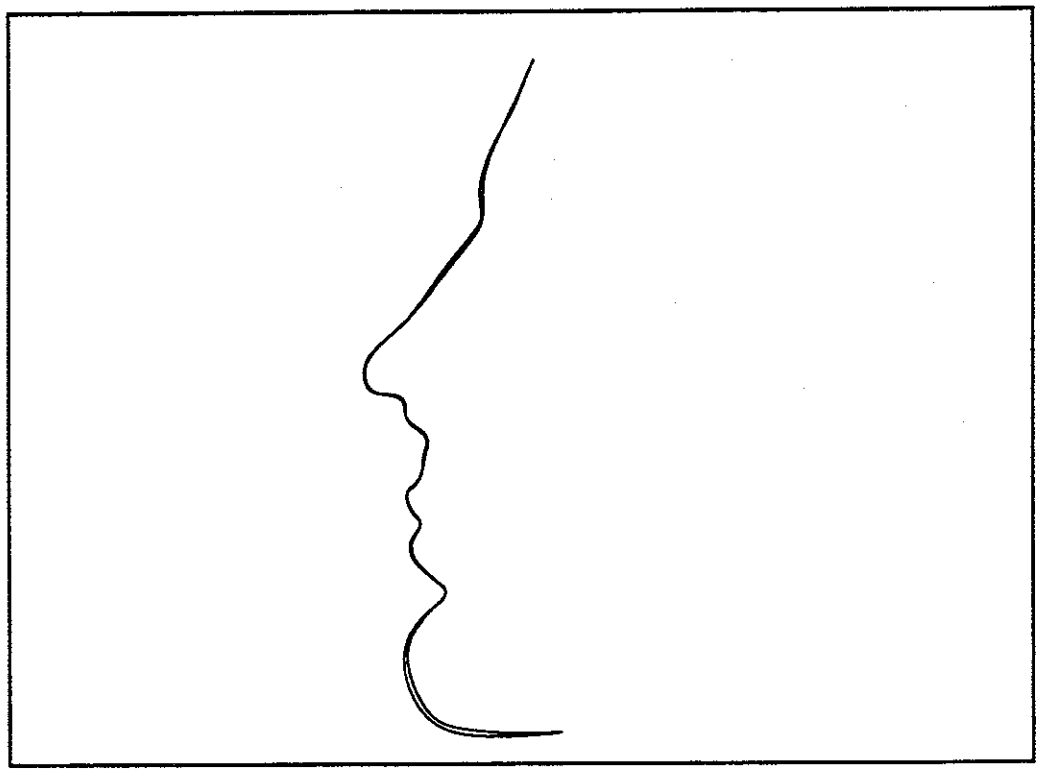


Figure 3.7: The smoothed face profile and the original

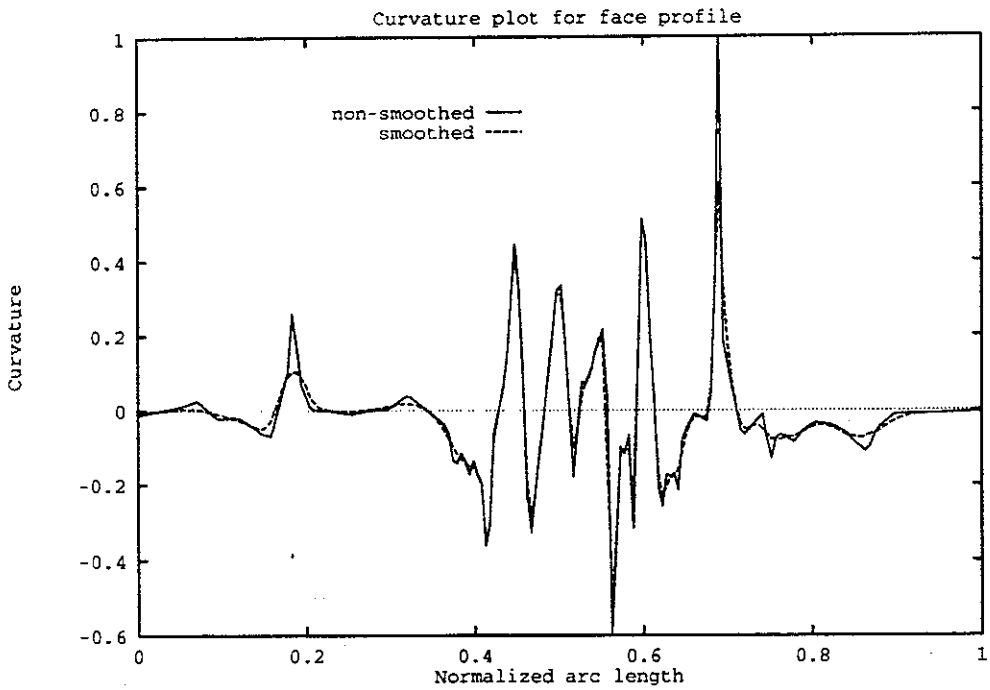



Figure 3.8: Curvature variation after smoothing

## 3.3 Using the smoothing algorithm

Two versions of the smoothing algorithm are available, one being tailored for the smoothing of airfoil sections since these require some special treatment. The airfoil version is called *smoothfoil* and the version for use with any general 2D curve is called *smooth2D*.

It must be emphasized, that these smoothing algorithms are aimed at removing disturbances which are on very small scales. They are not well suited for curve fitting as such.

Prior to invoking either smoother, the user is required to:

- Edit the jobfile "jobcard.dat", which should be present in the current directory. This file contains basic parameters for the smoother. A description of these is given in section 3.4. An example of the jobfile is shown in appendix A.
- Produce the geometryfile holding the data set (see section 3.4).
- Edit the subroutine "output.f" if special output is required (and the call to "output" from main if the argument list is changed).

### User input

During the session, the user is requested to enter parameters for the determination of the distribution function $f(u)$, where the parameter $u$ is normalized arc length along the curve, i.e. $u \in [0, 1]$.

**Smooth2D** In the general case (smooth2D), only one parameter $\alpha$ is requested. This parameter is used to dampen the effects of curvature dependent distribution, which is used to distribute points intelligently on the curve.[2] $\alpha$ may vary between 0 and $\infty$. Large values of $\alpha$ will yield a more uniform distribution, while small values of $\alpha$ will result in a more curvature dependent distribution. The value of $\alpha$ may be changed iteratively until a satisfactory result is obtained.

**Smoothfoil** In the airfoil case, the distribution function is a 7th order polynomial, which the user is able to control to some extent. Four parameters are requested:

- The end tangents $f'(0)$ and $f'(1)$.
- The value and tangent at $u = \frac{1}{2}$: $f(\frac{1}{2})$ and $f'(\frac{1}{2})$.

[2]This distribution function is just integrated and normalized curvature.

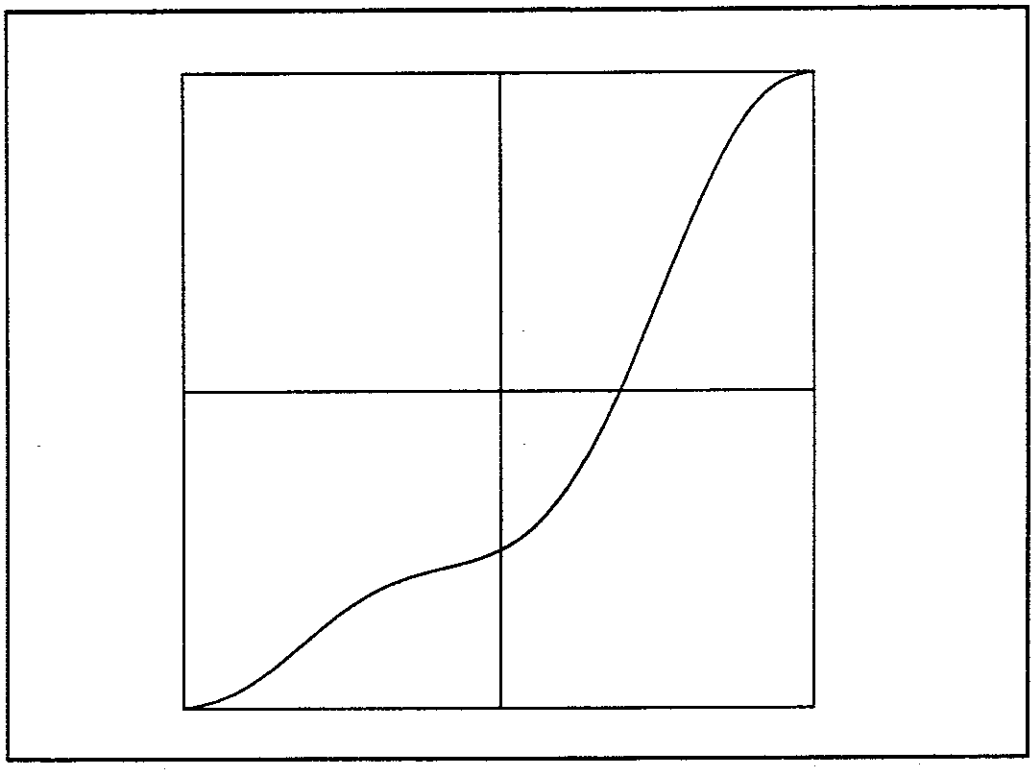


Figure 3.9: A 7th order polynomial distribution function

The remaining four conditions are preelected:

$$f(0) = 0 \quad f(1) = 1 \quad f''(0) = f''(1) = 0.$$

This polynomial provides resonable flexibility in prescribing a suitable distribution function for an airfoil. An example using the following input

$$f'(0) = f'(1) = 0.15 \quad f(\tfrac{1}{2}) = 0.25 \quad f'(\frac{1}{2}) = 0.5,$$

is shown in figure 3.9. This choice of parameters would result in a distribution of points, which would not be suitable for an airfoil. For the fx66 airfoil used here, the result is seen in figure 3.10. A high concentration of points is obtained when the slope of the distribution function is small and vice versa.
The distribution function *must* be monotone, and the user is alone responsible for ensuring this. The parameters may therefore be changed iteratively until the user is satisfied. For each new choice of parameters, the following is computed:

- The largest ratio between two adjacent curve segments.
- The length (normalized by chord) of the trailing edge segments

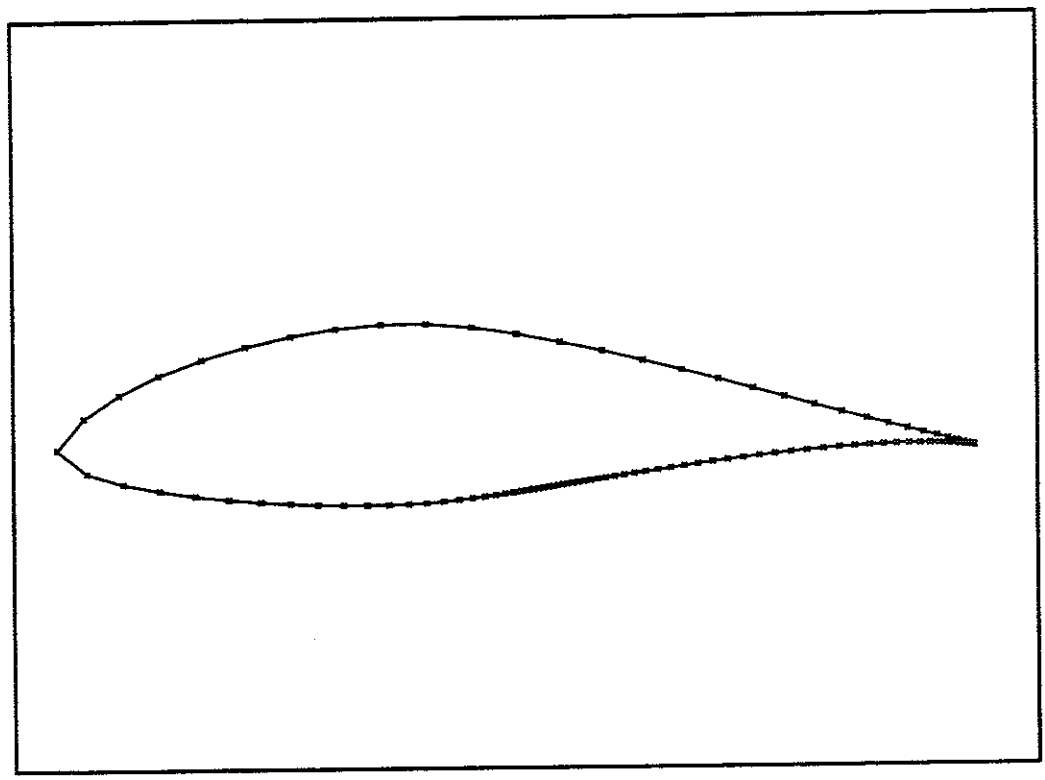


Figure 3.10: An fx66 airfoil based on the example distribution function

### Output

Default output is $np$ $(x, y)$-coordinate pairs of the smoothed curve, where $np$ is specified by the user in "jobcard.dat", written to the formatted file "smooth.dat". Formatted writing will destroy some of the work done by the smoother due to truncation. One method of avoiding this is to use unformatted writing, another is to incorporate the smoother into whichever code needs the output.

### Airfoil preparation

The *smoothfoil* algorithm assumes that the chord length of the airfoil is 1, and that the trailing edge is sharp. These requirements must be fulfilled for the smoother to function properly. Adjusting the chord length is a trivial matter of scaling equally parallel to and at right angles to the chord.

In order to smooth an airfoil with a finite trailing edge thickness, the following steps are suggested:

1. Extend the airfoil at the trailing edge using the end tangents there.
2. Scale the airfoil to a chord length of 1.
3. Smooth the airfoil.
4. Rescale to the same chord length as after step 1 and cut off the trailing edge at the appropriate position to restore the original geometry.

## 3.4 Example session

This section is an example of a runtime session with the airfoil smoother. Input is given partly through interactive communication with the user, and partly by parameters specified in a jobfile: *jobcard.dat*.

This session was executed with *jobcard.dat* as it is reproduced in the appendix. The geometryfile "fx66.dat" specified in *jobcard.dat* is also shown (in part) in the appendix.

### Explanation of *jobcard.dat* parameters

The inputfile "jobcard.dat" is to be edited by the user. It is used to set the following parameters:

**filename** This is the name of the geometryfile, holding a 2D tabulated set of data for the curve to be smoothed. The geometryfile contains the number of data points *num* in the first row, followed by *num* rows of data.

**np** This is the number of points wanted to represent the smoothed curve.

**iorder** Order of approximation for the first derivatives at curve ends. Iorder may be set to 2 or 3.

**scale** Pure scaling of curvature. This is only for presentation purposes of the curvature plots.

**smooth** If curve smoothing is desired, this should be a "t".

**ntrial** Maximum no. of iterations performed.

**tolx** Tolerance on function values, i.e. the difference "$\Delta f_i$" between two consecutive iteration steps.

**tolf** Tolerance on integrated difference, i.e. the sum of all "$\Delta f_i$".

**smtype** Type of smoothing for the curvature. When smoothing airfoils, setting smtype to 2 will ensure that the leading edge curvature is preserved (*if* the airfoil does not have areas of higher curvature than here !).

**plotpnts** For presentation purposes. Plots each point with a cross if set to true ("t").

The following is a transcript of a session.
Text written in the `typewriter type style` is output by the program.
Text written in **bold** is user typein.
Text written in *italics* is supplementary, and does not appear during runtime.

**smoothfoil**

```
fx66.dat filename of definition file
```

*A plot of the geometry as read in is displayed, see figure 3.11.*

```
main:  Determine distribution function:
enter end0, end1, cen, cengrad:
```

**.5 .5 .491 .1**

*A plot of the distribution function is displayed, see figure 3.11.*

```
Trailing edge:  length of bottom segment:   .006977224
length of top segment :   .006978219
Max.  ratio between panels:  1.141965713
```

*A plot of the geometry using this distribution is displayed.*

```
Do you want to change your choice?  (y or n):
```

**n**

```
main:  initial curve and curvature:
```

*A plot of the geometry using the chosen distribution is displayed, see fig. 3.12.*

```
Minimum curvature:  -10.693326
Maximum curvature:  111.926973
```

*A plot of the curvature using the chosen distribution is displayed, see fig. 3.12.*

```
Smoothing:
```

*A plot of the curvature after this smoothing pass is displayed, see fig. 3.13.*

```
smooth more ?  (y or n)
```

**n**

```
iteration no.  1 :  35.4787629427 .0096538271
iteration no.  2 :  .5098149497 .0004349605
iteration no.  3 :  .0037897220 .0000047014
iteration no.  4 :  .0000674969 .0000000867
iteration no.  5 :  .0000011546 .0000000015
tolx criteria!  .0000000198 .0000000000
main:  smoothed curve and curvature:
```

*A plot of the geometry after smoothing is displayed.*

```
Minimum curvature:  -2.830055
Maximum curvature:  109.367063
```

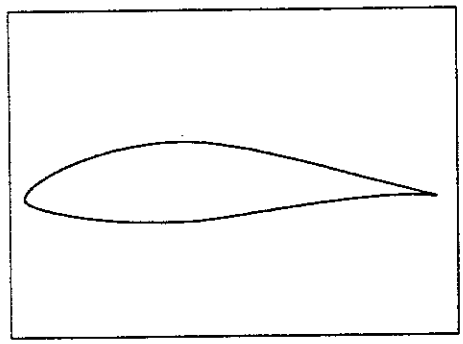
*A plot of the curvature after smoothing is displayed.*

```
Compare smoothed and unsmoothed curve/curvature:
```

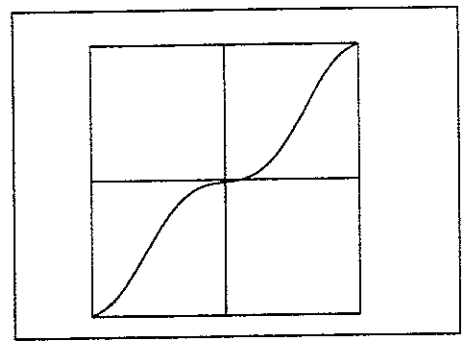
*Comparison plots of the geometries and curvatures before and after smoothing are displayed, see figures 3.14 and 3.15.*

plots displayed during the session:

1. geometry definition (cur.plt)
2. distribution function (distprof.plt)
3. redefined geometry (cur.plt)
4. final redefined geometry (cur.plt)
5. initial curvature distribution (sec.plt)
6. smoothed curvature compared with original (sec2.plt)
7. solution airfoil (cur.plt)
8. solution curvature (sec.plt)
9. curve comparison (cur2.plt)
10. curvature comparison (sec2.plt)

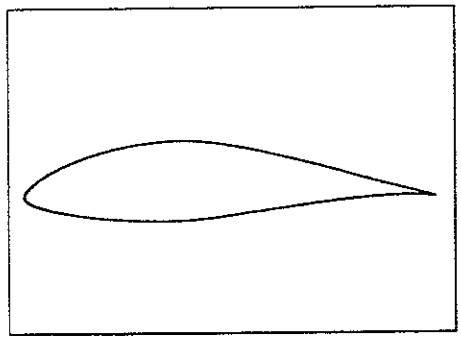


The original geometry as read from file.

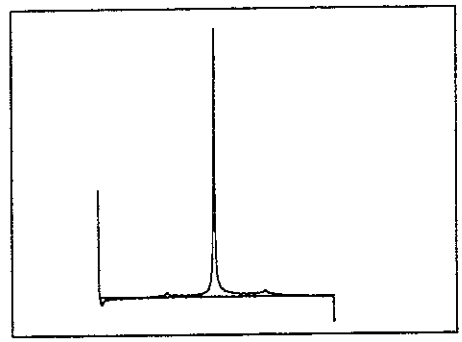


Distribution function chosen to describe the airfoil.

Figure 3.11:

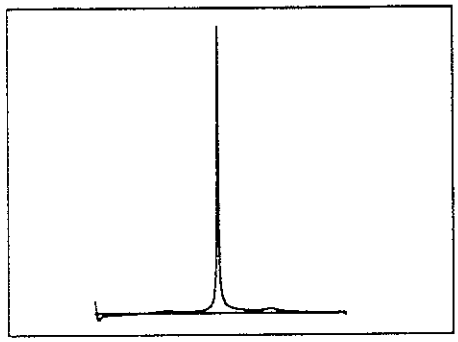


The geometry as described by the chosen distribution function.



Curvature of the airfoil using the chosen distribution function.

Figure 3.12:



Curvature of the airfoil after one smoothing pass.

Figure 3.13:

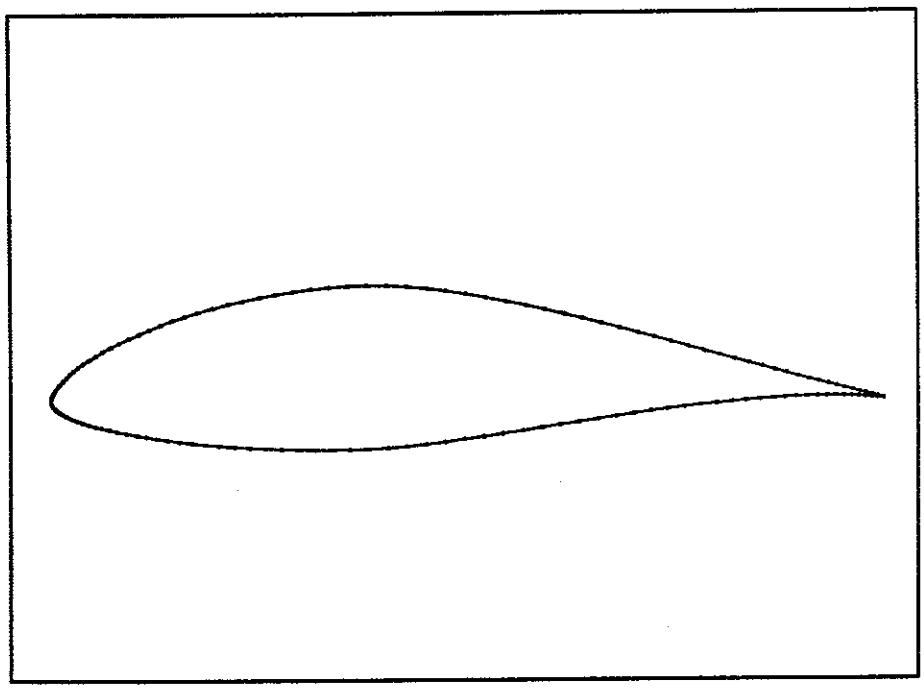


Figure 3.14: Comparison between smoothed and unsmoothed airfoils. The differences are invisible to the eye

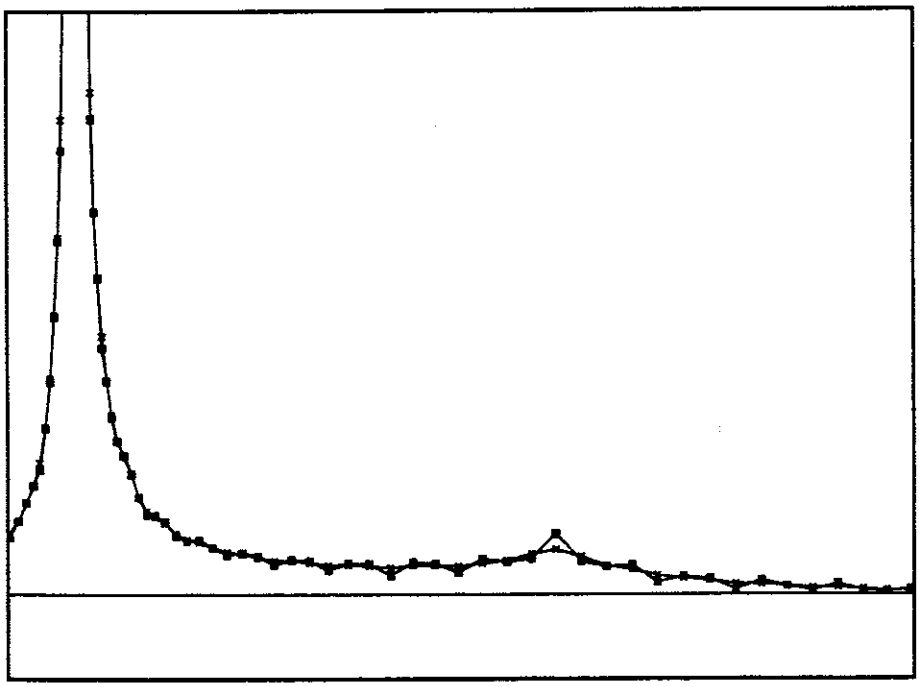


Figure 3.15: Comparison between curvature variations of smoothed and unsmoothed airfoil. Here, the differences are unmistakeable

# Chapter 4

# Inviscid flow computation

The smoothed and unsmoothed versions of the airfoil fx66-s-196 have been subjected to inviscid flow computations using the potential flow solvers developed at the Department of Fluid Mechanics by J. Tejlgaard. Calculations using 0th and 1st order geometric approximations, coupled with 0'th and 1'st order vortex distributions have been performed.

**Geometry** The order of approximation is:

1. In the 0'th order approximation the panels are assumed to be linear, i.e. panel midpoints lie on a straight line between panel endpoints.
2. In the 1'st order approximation the panels are "true", i.e. panel midpoints are computed using the same cubic spline as for the panel endpoints.

**Vortex distribution** The order of approximation is:

1. The 0'th order vortex distribution uses a constant vortex strength on the panels.
2. The 1'st order vortex distribution uses a linearly varying vortex strength on the panels.

Computations have been performed for respectively 1 and 8 degrees angle of attack, and all calculations use the same panel distribution, with 149 panels (150 points).

The results for some of these computations are shown graphically in figures 4.1 to 4.4. The illustrated results have all been computed using 1'st order geometry approximation and 1'st order vortex distribution. It is clearly seen from figures 4.1 and 4.2 that the effect of smoothing does not have a very significant influence on the result as a whole. There is a slight difference close to the leading edge (shown in blowup figures 4.3 and 4.4), but even here the differences are only slight. The only way in which an effect of the smoother geometry could be registered, was by integration of the axial pressure coefficient (computed from the calculated velocity distribution).

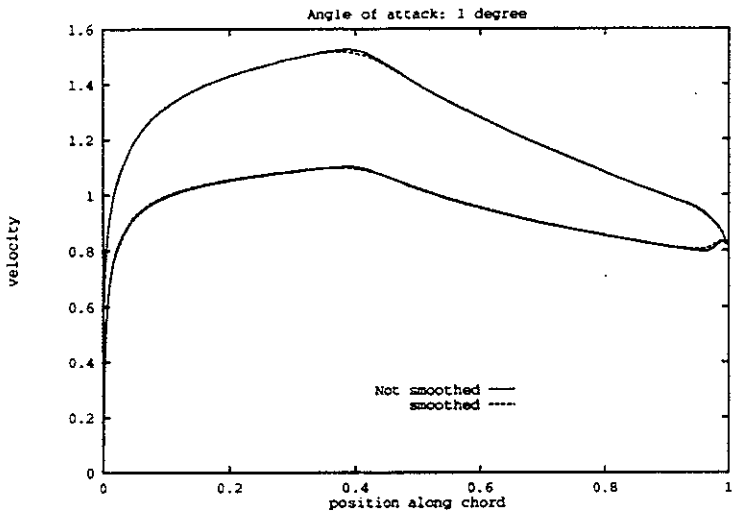



Figure 4.1: Potential flow calculation using 1'st order approximations for both geometry and vortex distribution. $\alpha = 1°$

| Vortex distr. | Geometry | cd | $\epsilon\sigma$ | Smooth |
|---|---|---|---|---|
| 0'th order | 0'th order | 0.000247 | 0.000062 | No |
| | | 0.000227 | 0.000061 | Yes |
| | 1'st order | 0.001170 | 0.000013 | No |
| | | 0.000187 | 0.000063 | Yes |
| 1'st order | 0'th order | 0.001014 | 0.001251 | No |
| | | 0.000492 | 0.001256 | Yes |
| | 1'st order | 0.001619 | 0.001314 | No |
| | | 0.000321 | -0.001251 | Yes |

Table 4.1: Integrated axial pressure vs. $\epsilon\sigma$

This measure of the accuracy of the solution should be zero. A comparison of this integrated axial pressure (labelled "cd" and the residual $\epsilon\sigma$ , which is the integrated value of source strengths on the airfoil, is shown in table 4.1. It seems that only when using a smooth geometry are the calculations able to solve the system so accurately that the integrated axial pressure drops below the residual $\epsilon\sigma$. However, there is a much more significant effect due to the use of 0'th or 1'st order approximations for both geometry and vortex distribution, so the conclusions are everything but straightforward. An object for future investigation would be to test the airfoils in a viscous - inviscous interaction code, to see whether the effects of smoothing are more pronounced here.

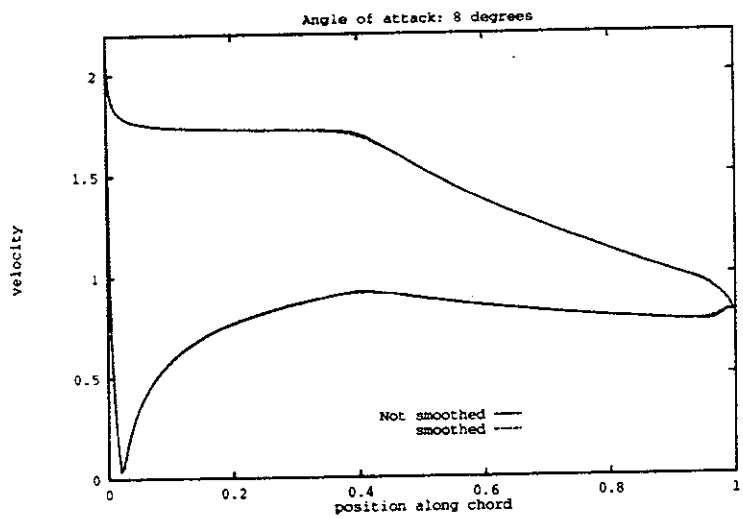



Figure 4.2: Potential flow calculation using 1'st order approximations for both geometry and vortex distribution. $\alpha = 8°$

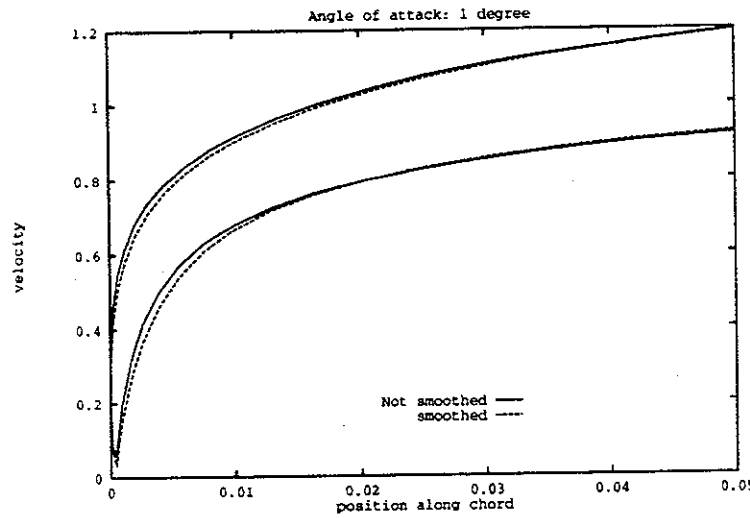



Figure 4.3: Velocity distribution close to leading edge. $\alpha = 1°$

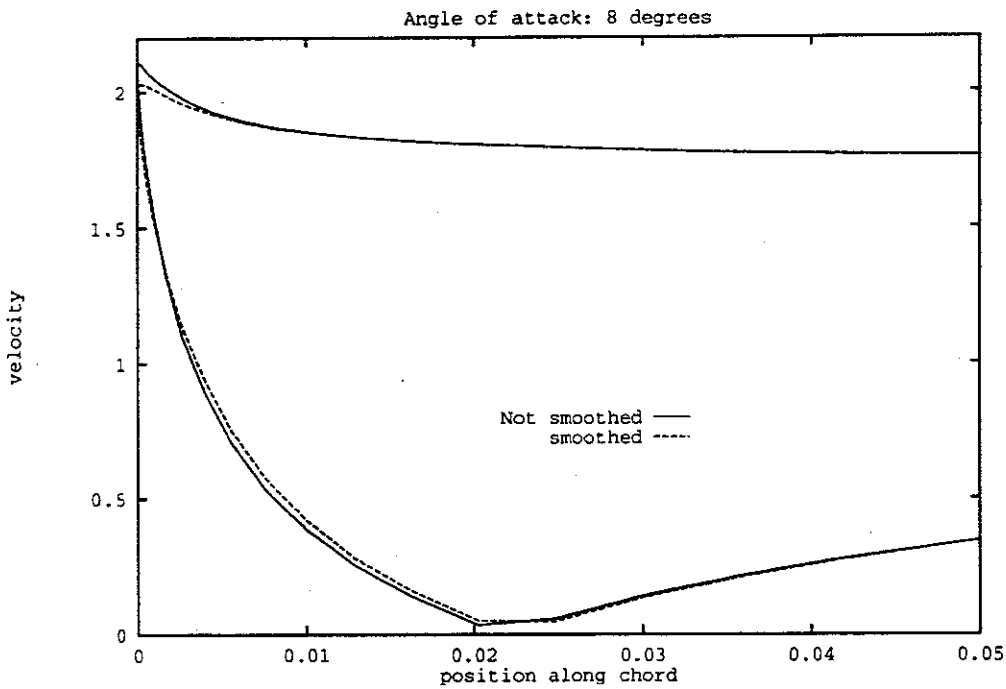



Figure 4.4: Velocity distribution close to leading edge. $\alpha = 8°$

# Chapter 5

# Conclusion and discussion

A smoothing algorithm for plane cubic B-spline curves has been produced. The algorithm is supplied in two versions:

**smooth2D** Smoothing of general plane curves.

**smoothfoil** Smoothing of airfoils.

The "smoothfoil" algorithm is tailored for the smoothing of airfoil sections, and ensures that the leading edge radius of curvature (which is important for airfoil characteristics) is preserved. It also ensures that the chord length of the smoothed airfoil is 1. The "smoothfoil" algorithm has been succesfully applied to a tabulated airfoil ((fx66-s-196), and curvature plots of the airfoil before and after smoothing show a much improved curvature variation of the smoothed airfoil, while the geometry of the airfoil in itself is left practically untouched.

It is believed that the smoothness of the airfoil influences flow computations, and this has been preliminarily investigated. Inviscid flow computations carried out on non-smoothed and smoothed versions of the same airfoil (fx66-s-196), do not display significant changes due to smoothing, although some (positive) effects are registered. In order to determine how much smoothness affects flow computations, it is thought to be necessary to subject smoothed and unsmoothed airfoils to viscous code computations.

Another interesting field of investigation is to compare airfoils smoothed with the present smoother, and airfoils smoothed using other curve smoothing algorithms, to see how the geometry of the leading edge affects the flow. It is expected that significant differences will be noted if the leading edge radius of curvature is altered by the smoother.

# Bibliography


[1] Boehm, W. *On Cubics: A Survey*, Computer Graphics and Image Processing **19**, 201-226, 1982.

[2] Boor, C. de *A Practical guide to splines*, Springer, 1978.

[3] Farin, G. *Curves and Surfaces for Computer Aided Geometric Design*, Academic Press, 1988.

[4] Farin, G. *NURBS and Grids*, paper presented at the *Third International Conference on Numerical Grid Generation in Computational Fluid Dynamics and Related Fields*, Barcelona, Spain, 1991.

[5] Farin, G., G. Rein, N. Sapidis and A. J. Worsey *Fairing cubic B-spline curves*, Computer Aided Geometric Design **4**, 91-103, 1987.

[6] Faux, I. D. and M.J. Pratt *Computational Geometry for Design and Manufacture*, Ellis Horwood Ltd., 1979.

[7] Hagen, H. *Geometric Modeling of Smooth Surfaces*, in *Computational Geometry and its Applications*, Lecture Notes in computer science (H. Noltemeier ed.), 333, 1988.

[8] Hoschek, J. and F.-J. Schneider *Spline conversion for trimmed rational Bé zier- and B-spline surfaces*, Computer Aided Design **22**, 9, 1990.

[9] Kjellander, J. A. *Smoothing of cubic parametric splines*, Computer Aided Design **15**, 3, 1983.

[10] Lott, N. J. and D. I. Pullin *Method for fairing B-spline surfaces*, Computer Aided Design **20**, 10, 1988.

[11] Nielsen, H. B. *Kubiske Splines*, Lecture notes: Hæfte 45, 2nd ed., Numerical Institute, Tech. Univ. of Denmark, 1991.

[12] Sapidis, N. and G. Farin *Automatic fairing algorithm for B-spline curves*, Computer Aided Design **22**, 2, 1990.

[13] Tejlgaard, J. *Aeroplanlære*, Lecture notes AFM 90-09, Dept. of Fluid Mech., Tech. Univ. of Denmark, 1990.

[14] Tejlgaard, J. *Viskos-inviskose interaktionsmetoder*, Lecture notes AFM 90-05, Dept. of Fluid Mech., Tech. Univ. of Denmark, 1990.

# Appendix

This appendix contains:

1. A list of the source files needed by the smoothing algorithm with a short description of each file.
2. An example of the jobfile "jobcard.dat".
3. An example of a geometryfile.

Note that most of the FORTRAN code has been developed with the use of spatial curves in mind, i.e. all geometry-holding arrays have a second index of "3" as a result hereof. The files listed below are for the *smoothfoil* algorithm.

List of files:

**jobcard.dat** User edited file containing runtime parameters

**fx66.dat** Example of a geometryfile

**params.h** Include file containing global declarations

**adjcur.f** Adjusts the chord length of a profile to one

**bisec.f** Finds a position in a table using bisection

**bnor.f** Calculates normalized B-splines

**bpol1d.f** Computes 3D B-spline points

**curinf.f** Interpolation routine. Computes curve and curvature

**deboor.f** deBoor algorithm

**distcheck.f** Checks distribution on profile

**distfoil.f** Creates distribution function

**drawcurve.f** Draws a 2D curve

**drawcurve2.f** Draws two 2D curves for comparison

**drawkappa.f** Draws curvature plot of 2D curve

**drawkappa2.f** Draws two curvature plots for comparison

**geometry.f** Reads from geometryfile

**inputpar.f** Reads parameters from file "jobcard.dat"

**jacobis.f** Computes jacobian for smoothing algorithm

**least.f** Smoothes curve using *ad hoc* scheme

**mnewt.f** Newton method for solving algebraic system of equations

**output.f** Output of data (user supplied or edited)

**polyinf.f** Computes curvature at knot values

**showdist.f** Plots distribution function

**smcurv.f** Main control of smoothing algorithm

**smoothfoil.f** Main

**tri.f** Solves a tridiagonal system of algebraic equations

For the *smooth2D* algorithm, there are some minor differences in the code. The most noticeable differences are:

- There are no calls to *adjcur*, *distcheck* or *distfoil*
- The distribution function is found by calls to *disfun* and *distrib*

The additional files for this code are:

**disfun.f** Creates curvature dependent distribution function

**distrib.f** Interpolates to the distribution function found by disfun.f

## Code access

The source code to the smoothing algorithm is available for students and researchers working at (or in colloboration with) the Department of Fluid Mechanics at the Technical University of Denmark. A copy of the code may be obtained at the department, or by contacting the author.

```
*********************************************************************
*     ------------------------------------------------              *
*     | jobcard.dat                                  |              *
*     | parameter settings for smoothing algorithm.  |              *
*     ------------------------------------------------              *
*********************************************************************
value     name        description
-----     ----        -----------------------------------------------
fx66.dat                filename of definition file
150       np            no. of points for curve in final layout
2         iorder        order of appprox. for end derivatives (2 or 3)
0.01      scale         scaling factor for curvature
t         smooth        smooth curve (t=yes,f=no)
180       nsmooth       no. of points for smoothing algorithm
10        ntrial        max. no. of iterations in smoothing algorithm
1.0d-10   tolx          tolerance on function value
1.0d-9    tolf          tolerance on integrated function
2         smtype        type of smoother (1:smooth all, 2: keep max value)
t         plotpnts      draw points on plots if .TRUE.
```

```
   87   (Num)       Geometryfile 'fx66.dat'
 1.00000  0.00000
 0.99893  0.00005
 0.99039  0.00066
 0.97347  0.00206
 0.94844  0.00288
 0.91573  0.00254
 0.87592  0.00080
 0.85355 -0.00068
 0.82967 -0.00260
 0.80438 -0.00489
 0.77779 -0.00764
 0.75000 -0.01080
 0.72114 -0.01435
 0.69134 -0.01827
 0.66072 -0.02256


........


 0.05156 -0.02903
 0.03806 -0.02489
 0.02653 -0.02065
 0.01704 -0.01639
 0.00961 -0.01211
 0.00428 -0.00784
 0.00107 -0.00354
 0.00000  0.00000
 0.00107  0.00621
 0.00428  0.01223
 0.00961  0.01918
 0.01704  0.02691
 0.02653  0.03520
 0.03806  0.04383
 0.05156  0.05273
 0.06699  0.06170
 0.08427  0.07067
 0.10332  0.07946
 0.12408  0.08803
 0.14645  0.09621
 0.17033  0.10398
 0.19562  0.11114
 0.22221  0.11772
 0.25000  0.12348
 0.27866  0.12848
 0.30866  0.13243
 0.33928  0.13537
 0.37059  0.13690
 0.40245  0.13691


........


 0.66072  0.09067
 0.69134  0.08272
 0.72114  0.07482
 0.75000  0.06699
 0.77779  0.05936
 0.80438  0.05197
 0.82967  0.04501
 0.85355  0.03845
 0.87592  0.03242
 0.91573  0.02193
 0.94844  0.01357
 0.97347  0.00700
 0.99039  0.00255
 0.99893  0.00028
 1.00000  0.00000
```