Representation of Ice Geometry by Parametric Functions: Construction of Approximating NURBS Curves and Quantification of Ice Roughness—Year 1: Approximating NURBS Curves

Loren H. Dill
University of Akron, Akron, Ohio

April 2004
Since its founding, NASA has been dedicated to the advancement of aeronautics and space science. The NASA Scientific and Technical Information (STI) Program Office plays a key part in helping NASA maintain this important role.

The NASA STI Program Office is operated by Langley Research Center, the Lead Center for NASA's scientific and technical information. The NASA STI Program Office provides access to the NASA STI Database, the largest collection of aeronautical and space science STI in the world. The Program Office is also NASA's institutional mechanism for disseminating the results of its research and development activities. These results are published by NASA in the NASA STI Report Series, which includes the following report types:

- TECHNICAL PUBLICATION. Reports of completed research or a major significant phase of research that present the results of NASA programs and include extensive data or theoretical analysis. Includes compilations of significant scientific and technical data and information deemed to be of continuing reference value. NASA's counterpart of peer-reviewed formal professional papers but has less stringent limitations on manuscript length and extent of graphic presentations.

- TECHNICAL MEMORANDUM. Scientific and technical findings that are preliminary or of specialized interest, e.g., quick release reports, working papers, and bibliographies that contain minimal annotation. Does not contain extensive analysis.

- CONTRACTOR REPORT. Scientific and technical findings by NASA-sponsored contractors and grantees.

- CONFERENCE PUBLICATION. Collected papers from scientific and technical conferences, symposia, seminars, or other meetings sponsored or cosponsored by NASA.

- SPECIAL PUBLICATION. Scientific, technical, or historical information from NASA programs, projects, and missions, often concerned with subjects having substantial public interest.

- TECHNICAL TRANSLATION. English-language translations of foreign scientific and technical material pertinent to NASA's mission.

Specialized services that complement the STI Program Office’s diverse offerings include creating custom thesauri, building customized databases, organizing and publishing research results . . . even providing videos.

For more information about the NASA STI Program Office, see the following:

- Access the NASA STI Program Home Page at http://www.sti.nasa.gov
- E-mail your question via the Internet to help@sti.nasa.gov
- Fax your question to the NASA Access Help Desk at 301–621–0134
- Telephone the NASA Access Help Desk at 301–621–0390
- Write to: NASA Access Help Desk NASA Center for AeroSpace Information 7121 Standard Drive Hanover, MD 21076
Representation of Ice Geometry by Parametric Functions: Construction of Approximating NURBS Curves and Quantification of Ice Roughness—Year 1: Approximating NURBS Curves

Loren H. Dill
University of Akron, Akron, Ohio

Prepared under Grant NAG3–2848

National Aeronautics and Space Administration

Glenn Research Center

April 2004
This report contains preliminary findings, subject to revision as analysis proceeds.

Available from

NASA Center for Aerospace Information
7121 Standard Drive
Hanover, MD 21076

National Technical Information Service
5285 Port Royal Road
Springfield, VA 22100

Available electronically at http://gltrs.grc.nasa.gov
Representation of Ice Geometry by Parametric Functions: 
Construction of Approximating NURBS Curves and Quantification of 
Ice Roughness—Year 1: Approximating NURBS Curves

Loren H. Dill
The University of Akron
Akron, Ohio 44325

Abstract

Software was developed to construct NURBS curves that approximate the geometries of iced airfoils. Users specify a tolerance that determines the extent to which the approximating curve follows the rough ice. This ability to smooth the ice geometry in a controlled manner will assist the generation of grids suitable for numerical aerodynamic simulations, and ultimately aid studies of the effects of smoothing upon the aerodynamics of iced airfoils. The software was applied to several different types of iced airfoil data collected in the Icing Research Tunnel at NASA Glenn Research Center, and was found to efficiently generate suitable approximating NURBS curves for all geometries. This method is an improvement over the current “control point formulation” of SmaggIce (v.1.2). In this report, we present the relevant theory of approximating NURBS curves and discuss typical results of the software.

Introduction

The detrimental effect of ice upon airfoil performance has long been a safety concern, and is largely studied using experimental models in special wind tunnels such as the National Aeronautics and Space Administration’s Icing Research Tunnel at the Glenn Research Center. In recent years, the reduced aerodynamic performance of airfoils with moderate ice has been successfully simulated via computational fluid dynamics (CFD) [1-4]. There is hope that CFD can be used to simulate aerodynamic performance under more general icing conditions. Experimental and computational efforts would then serve more complementary roles in the development of safe aircraft, and safety margins could be enhanced at reduced costs.

In general, the presence of ice on an airfoil greatly increases the difficulty of a CFD analysis over that of a clean airfoil. The ice often has deep narrow crevices and exhibits varying degrees of roughness. The associated flow over iced airfoils is very complex. In particular, existing grid generation codes are generally ill equipped to accommodate the wide variety of shapes and sizes of ice that is typically found on an airfoil. So instead of being automated as it is for a clean airfoil, the generation of quality grids for iced airfoils is frequently a labor intensive, interactive process. To address this problem, NASA has been developing an interactive software toolkit [5] called SmaggIce 2D. This software enables the user in the tasks of geometry preparation, domain decomposition, block boundary discretization, gridding, and linking to the flow solver for a two-dimensional airfoil.

The current version (v1.2) of SmaggIce 2D provides an option to represent the ice geometry via a Non-Uniform Rational B-Spline (NURBS) curve [6-8]. NURBS have a number of attractive features, including fast and numerically stable algorithms, and easy-to-understand geometric interpretations. Designers of the current software found it convenient to place the control points of the NURBS curve on
the x-y data that specifies the ice geometry. Ordinarily, control points are not placed on the geometry to be represented, but in this case the error is small due to the large number and high density of the given data. Nodes are later placed along the representation for numerical simulation.

Smoothing of the ice geometry representation is often required to generate a quality grid by state-of-the-art grid generators. SmaggIce 2D permits the user to smooth the NURBS representation by reducing the number of control points [5]. In this method, the user can only coarsely control the smoothed NURBS representation of the ice geometry. This smoothing facilitates the subsequent gridding and numerical simulation, but the lack of control makes it difficult to access the effects of smoothing on the CFD results.

In our approach, the user requires the NURBS curve to satisfy a tolerance, i.e., a characteristic distance between the ice geometry and its representation. In this way, a user can more precisely control the extent to which the representation is smoothed. Not only will this ease the difficulty of generating quality grids over ice, but it will also aid investigation of the relation between ice roughness and aerodynamic characteristics of the iced airfoil.

We developed a software package that uses a NURBS curve to represent a two-dimensional cross-section of the ice surface as closely as desired by specification of a tolerance. Given a set of point data that defines the two-dimensional ice geometry, the software finds a NURBS representation to within the specified tolerance, which may be expressed either as a maximum distance or as a maximum root-mean-square (rms) distance between equivalent points of the curve and the prescribed data. The user is given various suitable options that further modify the NURBS representation, including the degree of the NURBS basis functions and whether or not to fix end-point derivatives. The latter option was found to permit the achievement of tighter tolerances. The code was applied to a wide range of experimental data from the Icing Research Tunnel, and was found both robust and efficient in all cases. In this report, we further document this software by providing requisite theory and typical results. The FORTRAN 77 code, which is included as an appendix, is available for inclusion in the next version of SmaggIce 2D (v1.8).

Theory

We here review the theory of Non-Uniform-Rational-B-Splines. Our emphasis is upon generating a curve $C(u) = (x(u), y(u))$ that approximates prescribed data points $Q_k (k = 0,1,\ldots,mdata)$ in two dimensions. Both the theory presented below and the software are limited to a large subclass of NURBS curves, the non-rational B-spline curves, to avoid nonlinearities associated with determining all parameters needed in the more general class. B-spline curves (we henceforth drop the word non-rational) cannot exactly represent certain curves (e.g., perfect circles) that a more general NURBS can. Nevertheless, a B-spline can represent any smooth curve to within a tolerance, which is all that is needed in the present application. A list of symbols is provided in Appendix A.

Preliminaries

We represent a B-spline curve as the finite sum,

$$C(u) = \sum_{i=0}^{n} N_{i,p}(u)P_i,$$  \hspace{1cm} (1)

with curve parameter $u$ in the interval $[0,1]$, $P_i$ the control points in a two-dimensional space, and $N_{i,p}(u)$ the $p$th degree B-spline basis functions. These basis functions are defined on a knot vector $U = \{u_0,\ldots,u_{m\text{\scriptsize knot}}\}$, where $u_i$ ($i = 0,\ldots,m\text{\scriptsize knot}$) are the knots. For present purposes, the knot vector has the form
in which the unknown knots increase in numerical order in the open interval (0,1). Knowledge of the
degree \( p \) (which the user provides), the knot vector \( U \), and the control points \( P_i \) is required to fully
specify a B-spline. In the procedure to be described below, a sequence of approximating B-splines is
generated. When the procedure is successful, each spline in the sequence more closely represents the
given data.

Piegl and Tiller [6] furnish several properties of the basis functions and B-spline curves. We here list
those that are most relevant to the present application, and refer the interested reader to Piegl and Tiller
for additional details and references.

**Non-zero basis functions:** According to the local support property, \( N_i,p(u) = 0 \) for \( u \) outside the interval
\( [u_i, u_{i+p+1}) \). It follows that for any given value of the curve parameter \( u \), at most only \( p+1 \) of the \( n+1 \)
basis functions that appear in Equation (1) are non-zero. Therefore, for any value of \( u \), only \( p+1 \)
consecutive terms in the equation actually need to be determined and summed.

**Efficient computation of basis functions:** Given a value \( u \) of the curve parameter and a knot vector \( U \),
the non-zero basis functions may be computed simultaneously and efficiently.

**Relation between the number of knots, the number of terms in the sum, and the degree:** These three
quantities are related by the simple equation

\[
m_{knot} = n + p + 1, \tag{3}
\]

after subtraction of one from both sides. In the iterative procedure discussed below, the degree \( p \) is fixed,
and \( m_{knot} \) is increased so that the B-spline may more closely follow the ice geometry. This relation
requires an identical increase in \( n \).

**Continuity and differentiability of \( C(u) \):** Given a knot vector of the assumed form, the associated B-
spline curve is continuous, infinitely differentiable in the interior of knot intervals, and at least \( p \) times
differentiable at each knot. The curvature of the B-spline is often required when determining the
placement of nodes along the curve for grid generation [9]. Calculation of curvature involves the second
derivative \( \frac{d^2}{du^2} C(u) \). Continuity of curvature is assured if \( p \geq 3 \).

**Endpoint interpolation of \( C(u) \):** At \( u = 0 \) all basis functions are identically zero except \( N_{0,p}(0) = 1 \);
similarly, at \( u = 1 \) all basis functions are identically zero except \( N_{n,p}(0) = 1 \). It follows from Equation (1)
that a B-spline must interpolate the first and last control points: \( C(0) = P_0 \) and \( C(1) = P_n \). All B-splines of
interest to us also interpolate the first and last points of the prescribed ice geometry. This immediately
leads to the conclusion that for our purposes first and last control points respectfully coincide with the
first and last data points:

\[
P_0 = Q_0, \quad P_n = Q_{n_{data}}. \tag{4}
\]

**Endpoint derivatives of \( C(u) \):** First-order derivatives \( C'(0) \) and \( C'(1) \) of a B-spline are related to the
first and last pairs of control points via the respective expressions [6]
\begin{align*}
C'(0) &= \frac{p}{u_{p+1}} (P_1 - P_0) \\
C'(1) &= \frac{p}{1-u_{\text{sub}^{p-1}}} (P_n - P_{n-1}).
\end{align*}
\hspace{1cm} (5)

These equations may be solved for control points \( P_1 \) and \( P_{n-1} \):

\begin{align*}
P_1 &= Q_0 + \frac{u_{p+1}}{p} C'(0) \\
P_{n-1} &= Q_{\text{data}} - \frac{1-u_2}{p} C'(1)
\end{align*}
\hspace{1cm} (6)

The above relations will be useful in the development of B-spline representations that satisfy prescribed endpoint derivative conditions.

**Global Approximation by a B-Spline**

Piegl and Tiller [6] offer a procedure for the global approximation of prescribed data by a B-spline curve provided the endpoint derivatives are free. We summarize their method below, and later extend it to the case for which endpoint derivatives are specified. In both cases, a suitable global approximation is obtained by iteration. The user provides certain information such as the point data to be approximated, a tolerance to be satisfied, and an initial value of \( n \). Based upon this information, an initial approximating B-spline is developed using the method of least-squares. This approximation is tested against the tolerance requirement. If the requirement is met, the approximation is accepted. If not, the number of knots and terms in Equation (1) is increased and a new B-spline is generated. This process continues until the specified tolerance is achieved or the process fails. Failure could occur, e.g., if the specified tolerance is too small.

**Free Endpoint Derivatives**

The data \( Q_k \), degree \( p \) of the approximating B-spline, an initial value of \( n \), and a tolerance requirement are presumed given. Because we seek an approximating B-spline that interpolates first and last data points, first and last control points are given by Equation (4). The remaining control points and the knot vector are obtained as follows. We first parameterize the given data; i.e., a parameter value is assigned to each data point. While there are several ways in which this can be done, the chord length method is generally satisfactory. Let \( d \) be the total chord length,

\begin{equation}
d = \sum_{k=0}^{m_{\text{data}}-1} |Q_{k+1} - Q_k|,
\end{equation}
\hspace{1cm} (7)

and require first and last points to correspond to the respective parameter values \( \bar{u}_0 = 0 \) and \( \bar{u}_{m_{\text{data}}} = 1 \). The remaining data parameters are then defined by the recursive equation

\begin{equation}
\bar{u}_k = \bar{u}_{k-1} + \frac{|Q_k - Q_{k-1}|}{d}
\end{equation}
\hspace{1cm} (8)

for \( k = 1, \ldots, m_{\text{data}} - 1 \). These data parameters remain unchanged throughout the rest of the analysis. Data points can be numbered from either end of the ice geometry. To be definite, we assume that points are numbered consecutively from the upper to lower airfoil surface.
An initial knot vector must be generated that in some sense corresponds to the distribution of data parameters. Recall from Equation (2) that the first and last \( p + 1 \) knots are identically zero and unity, respectively. To determine the remaining knots, we define a new temporary variable \( d \) via the expression

\[
d = \frac{m_{\text{data}} + 1}{n - p + 1},
\]

which represents the average number of data points per knot span of nonzero length. The remaining knots are found from the following equations [6]:

\[
i = \text{int}(j * d) \\
\alpha = j * d - i \\
u_{p+j} = (1 - \alpha) * \overline{u}_{i-1} + \alpha * \overline{u}_j \quad \text{for } j = 1, \ldots, n - p
\]

Here, the staircase function \( y = \text{int}(x) \) is the largest integer such that \( y \leq x \). The above definition is successful in placing at least one data parameter \( k_u \) in every knot span of nonzero length if \( d \geq 1 \).

Provided this is true, a key symmetric matrix (\( N^T N \), which is encountered below) in the approximation procedure is positive definite and well-conditioned [6-8], at least in theory.

Determination of the internal control points \( P_i \) \((i = 1, \ldots, n - 1)\) is the final major step in defining the initial B-spline. These control points are selected so as to minimize the sum of the squared distances of the curve from data points \( Q_1, \ldots, Q_{m_{\text{data}}-1} \):

\[
\sum_{k=1}^{m_{\text{data}}-1} |Q_k - C(\overline{u}_k)|^2
\]

Take the derivative of this expression with respect to control point \( P_i \) and set the result to zero. This operation yields \( n - 1 \) equations that can be collectively represented by the matrix equation

\[
(N^T N) P = R.
\]

Here, \( N \) is the \((m_{\text{data}} - 1) \times (n - 1)\) matrix

\[
N = \begin{pmatrix}
N_{1,p}(\overline{u}_1) & \cdots & N_{n-1,p}(\overline{u}_1) \\
\vdots & \ddots & \vdots \\
N_{1,p}(\overline{u}_{m_{\text{data}}-1}) & \cdots & N_{n-1,p}(\overline{u}_{m_{\text{data}}-1})
\end{pmatrix},
\]

whose elements consist of basis functions evaluated at certain values of the data parameters. Recall that no more than \( p + 1 \) of the basis functions are non-zero for any given value of the curve parameter \( u \).

Thus, each row of \( N \) contains at most \( p + 1 \) non-zero entries. Also appearing in Equation (11) is a \((n - 1) \times 2\) matrix \( P \) of unknown control points,

\[
P = \begin{pmatrix}
P_1 \\
\vdots \\
P_{n-1}
\end{pmatrix},
\]

and a \((n - 1) \times 2\) matrix \( R \):
Each \((1 \times 2)\) row vector \(\mathbf{R}_k\) that appears above is defined by the expression

\[
\mathbf{R}_k = \mathbf{Q}_k - N_{0,p} (\bar{u}_k) \mathbf{Q}_0 - N_{n,p} (\bar{u}_k) \mathbf{Q}_{\text{data}}
\]

with \(k = 1, \ldots, m_{\text{data}} - 1\). (Recall that \(\mathbf{Q}_k \equiv (x_k, y_k)\) is defined within a two-dimensional space, which explains why \(\mathbf{P}\) and \(\mathbf{R}\) each have two columns and why \(\mathbf{R}_k\) is a two-element vector.) Equation (11) thus represents two linear systems of equations having the same matrix coefficient but with different right-hand sides.

Given that each knot span of non-zero length contains at least one data parameter, the \((n-1) \times (n-1)\) matrix \(\mathbf{N}^T \mathbf{N}\) (with superscript \(T\) denoting the transpose operation) is symmetric, positive definite, and in principle well-conditioned. (However, in practice the matrix can become singular as \(n\) becomes large and \(d\) in Equation (9) approaches unity. This is presumably due to limitations of finite arithmetic on a digital computer.) Moreover, it has a semi-bandwidth of \(p+1\), and can be stored in a compact form. The Cholesky method efficiently factorizes the matrix. Control points \(\mathbf{P}_1, \ldots, \mathbf{P}_{n-1}\) are then easily found via back substitution.

Solution of Equation (11) leads to a complete B-spline representation of the ice data. The next major step is to determine whether or not this representation satisfies the specified tolerance requirement. Two such tolerance criteria are implemented. The user may either specify a maximum tolerance requirement,

\[
d_{\text{max}} = \max_{1 \leq k \leq m_{\text{data}} - 1} |\mathbf{C}(\bar{u}_k) - \mathbf{Q}_k| \leq \epsilon_{\text{max}},
\]

or a rms tolerance requirement:

\[
d_{\text{rms}} = \sqrt{\frac{\sum_{k=1}^{m_{\text{data}}-1} |\mathbf{Q}_k - \mathbf{C}(\bar{u}_k)|^2}{m_{\text{data}} - 1}} \leq \epsilon_{\text{rms}}.
\]

In the above, \(d_{\text{max}}\) is the maximum separation distance of all the distances \(|\mathbf{C}(\bar{u}_k) - \mathbf{Q}_k|\), \(\epsilon_{\text{max}}\) the maximum tolerance, \(d_{\text{rms}}\) the rms distance, and \(\epsilon_{\text{rms}}\) the rms tolerance. If the specified tolerance requirement is satisfied, the B-spline is accepted as a suitable approximation to the ice geometry. If not, a new knot vector having additional knots is created, and a new B-spline approximation is generated using the above procedure. This cycle is repeated until the desired tolerance requirement is satisfied. Details concerning how the new knot vector is generated and how the tolerance criteria are implemented are reserved for a later section.

Before leaving this section, we note that \(\mathbf{C}(\bar{u}_k)\) is generally not the closest point on the curve to data point \(\mathbf{Q}_k\). Nevertheless, the distance \(|\mathbf{C}(\bar{u}_k) - \mathbf{Q}_k|\), which appears in both tolerance criteria, is useful and convenient for the intended use. An alternative, more exact maximum tolerance criterion,

\[
\max_{1 \leq k \leq m_{\text{data}} - 1} |\mathbf{C}(u_{k})_{\text{min}} - \mathbf{Q}_k| \leq \epsilon_{\text{max}},
\]

could be used. Here \(u = (u_{k})_{\text{min}}\) is the parameter value that minimizes the distance \(|\mathbf{C}(u) - \mathbf{Q}_k|\) for the \(k\)th point. Determination of each \(u = (u_{k})_{\text{min}}\) is a nonlinear problem that can be solved, e.g., using a Newton
iteration procedure [6]. The advantage in using the latter tolerance requirement in the present application does not appear to offset the extra required computational effort: up to \( mdata - 1 \) values of \( u = (u_k)_{\text{min}} \) are required per loop of the iteration that ultimately results in an acceptable B-spline curve. In any case, because \( \left| C((u_k)_{\text{min}}) - Q_k \right| \leq \left| C(\overline{u}_k) - Q_k \right| \), we are assured that any curve that satisfies Equation (16) also satisfies the above exact requirement. Our decision to use \( u = \overline{u}_k \) rather than \( u = (u_k)_{\text{min}} \) may result in a larger final value of \( n \) than that needed to meet the above exact tolerance requirement.

**Fixed Endpoint Derivatives**

Fixing of the endpoint derivatives requires a straightforward modification of the above least-squares procedure. Recall from Equation (5) that the endpoint derivatives of a B-spline curve are related to the first and last pairs of control points. If we specify these derivatives, Equation (6) gives control points \( P_1 \) and \( P_{n-1} \) so that \( P_0, P_1, P_{n-1}, \) and \( P_n \) are all known prior minimizing the sum of squared distances. In this case, we minimize the sum with respect to the remaining control points: \( P_2, \ldots, P_{n-2} \). The least-squares procedure is altered only in that matrices \( N, P, R, \) and vector \( R \) must be redefined. If endpoint derivatives are fixed, \( N \) is the \((mdata - 1) \times (n - 3)\) matrix

\[
N = \begin{pmatrix}
    N_{2,p}(\overline{u}_1) & \cdots & N_{n-2,p}(\overline{u}_1) \\
    \vdots & \ddots & \vdots \\
    N_{2,p}(\overline{u}_{mdata-1}) & \cdots & N_{n-2,p}(\overline{u}_{mdata-1})
\end{pmatrix},
\]

and \((n - 3) \times 2\) matrices \( P \) and \( R \) have the respective new definitions

\[
P = \begin{pmatrix}
P_2 \\
\vdots \\
P_{n-2}
\end{pmatrix}
\]

and

\[
R = \begin{pmatrix}
    \sum_{k=1}^{mdata-1} N_{2,p}(\overline{u}_k)R_k \\
    \vdots \\
    \sum_{k=1}^{mdata-1} N_{n-2,p}(\overline{u}_k)R_k
\end{pmatrix}.
\]

Finally, each vector \( R_k \) is defined by

\[
R_k = Q_k - \left( N_{0,p}(\overline{u}_k)P_0 + N_{1,p}(\overline{u}_k)P_1 + N_{n-1,p}(\overline{u}_k)P_{n-1} + N_{n,p}(\overline{u}_k)P_n \right),
\]

with \( P_0, P_1, P_{n-1}, \) and \( P_n \) given by Equations (4) and (6). The unknown control points are found by solving matrix Equation (11) with these modified definitions.

**Software Implementation**

In this section, we discuss how the two tolerance requirements are implemented in the software, and provide formulas for calculation of default values of endpoint derivatives. This background will prove important in understanding sample results produced by the code as discussed in the next major section. Brief descriptions of major elements of the code may be found in Appendix B, and the code itself is given in Appendix C.
Maximum Tolerance

The maximum tolerance requirement, Equation (16), is implemented as follows: Beginning with \( k = 1 \),
the distance \( |C(\tilde{u}) - Q_k| \) is compared with the maximum tolerance. If the distance is less than or equal
to \( \epsilon_{\text{max}} \), \( k \) is incremented by one and the test is repeated. If the test fails for any \( k \), the knot span associated
with \( \tilde{u}_k \) (i.e., the span such that \( u_i \leq \tilde{u}_k < u_{i+1} \)) is labeled as non-converging. To avoid unnecessary
evaluations, the remaining distances within that knot span are skipped. The next distance compared with \( \epsilon_{\text{max}} \) is that corresponding to the first \( \tilde{u}_k \) located within the next knot span. The test continues until all
knot spans are labeled as converging or non-converging.

If the specified tolerance is not achieved by all knot spans, a new knot vector is generated as follows: A
new knot is inserted at the midpoint of all non-converging knot spans, and both \( n \) and \( \text{mkn} \) are increased
by the number of added knots. Next, the knot vector is inspected to insure that each knot span of nonzero
length contains at least one data parameter \( \tilde{u}_k \). If it does, the above least-squares procedure is repeated to
generate a new set of internal control points and a new B-spline curve. The maximum tolerance criterion
above is checked again. If the maximum distance \( d_{\text{max}} \) from the prescribed data is less than or equal to the
specified tolerance \( \epsilon_{\text{max}} \), the iterative procedure ends. If the tolerance is not achieved, a new knot vector
is generated by inserting a new knot at the midpoint of each new non-converging knot span, and the cycle
repeats.

A knot vector formed by repeated insertion of knots may have one or more knot spans of nonzero length
that are empty, i.e., ones that do not contain at least one data point parameter \( \tilde{u}_k \). In this case, the matrix
\( N^T N \) is no longer guaranteed to be positive definite and well-conditioned. Provided the new value of \( n \)
is sufficiently small (\( n \leq \text{md} + p - 1 \)), a completely new knot vector is generated using Equation (10)
that has at least one data point parameter located within each knot span of nonzero length. A new B-spline
is generated based upon this new knot vector, and the iteration procedure continues as usual. If the new
value of \( n \) is too large (\( n > \text{md} + p - 1 \)), the procedure will abort.

Root-Mean-Square Tolerance

The maximum tolerance requirement of Equation (16) simply requires that the maximum distance that
separates the ice geometry from equivalent points on the B-spline curve is less than a specified distance
called the maximum tolerance. This measure of closeness of the spline to the ice geometry does not
distinguish between a curve that is close almost everywhere with the exception of one or a few values of
\( \tilde{u}_k \) and one that is far (but less than or equal to the maximum tolerance) almost everywhere. The ability to
specify a rms tolerance, \( \epsilon_{\text{rms}} \), permits the user to require the curve to be close to the data in a rms sense.
We also point out that the rms distance \( d_{\text{rms}} \), which is defined in Equation (17), is closely related to the
actual quantity that is minimized in the least-squares procedure.

The iterative procedure described above to achieve the maximum tolerance criteria is modified to satisfy
the rms requirement. In this case, the user must supply an initial maximum tolerance \( \epsilon_{\text{max}} \), a rms
tolerance \( \epsilon_{\text{rms}} \), and a scalar \( \alpha \). The rms tolerance is restricted to the interval \((0, \epsilon_{\text{max}})\) while \( \alpha \) must lie in
the interval \((0,1)\); the default value given \( \alpha \) is 0.9. During iterations, the initial maximum tolerance
requirement is satisfied as described above. Prior to exiting the loop, the rms tolerance test is performed.
If \( d_{\text{rms}} \leq \epsilon_{\text{rms}} \), the loop exits normally. Otherwise, the maximum tolerance is reduced according to the
expression \( \epsilon_{\text{max}} = \alpha \cdot d_{\text{max}} \) and each knot span is re-examined to determine the non-converging knot spans
relative to the new maximum tolerance. Knots are inserted at the midpoint of each such span, and
iterations continue until the new maximum tolerance requirement is satisfied. The rms criterion is then
tested again. The loop exits normally when both the modified maximum tolerance and the rms tolerance
requirements are met.
**Endpoint Derivatives**

First-order endpoint derivatives are specified in polar coordinates. The angle must be measured from the positive x axis and specified in units of degrees. Because these derivatives are vectors, both direction and magnitude must be supplied for each endpoint. For convenience, default values for these derivatives are calculated based upon finite differences of the first and last pairs of data points:

\[
C'(0) = \frac{Q_1 - Q_0}{\bar{u}_1 - \bar{u}_0} = \frac{Q_1 - Q_0}{\bar{u}_1}
\]

\[
C'(1) = \frac{Q_{\text{mdata}} - Q_{\text{mdata}-1}}{\bar{u}_{\text{mdata}} - \bar{u}_{\text{mdata}-1}} = \frac{Q_{\text{mdata}} - Q_{\text{mdata}-1}}{1 - \bar{u}_{\text{mdata}-1}}
\]  

Before leaving this section, we note that these derivatives are taken with respect to the curve parameter \(u\), and whether this parameter is increasing or decreasing when moving along the B-spline ultimately depends upon the sense of direction of the original data. The user must take this into account when specifying non-default values for \(C'(0)\) and \(C'(1)\).

**Application: Sample Results for an Ice Geometry**

The FORTRAN 77 code was compiled and run on a 1.1 GHz personal computer (Intel Pentium III processor under Linux). Suitable B-spline approximations that corresponded to a wide range of tolerances were obtained for three ice geometries, which were provided by the Icing Research Tunnel. To illustrate typical behavior, we investigate several representations for one ice geometry.

Most of our runs – and all discussed here – used cubic B-splines. The few runs conducted with degree \(p = 4\) or \(5\) yielded B-splines that, on a graph at least, appeared equivalent to a cubic B-spline for the same geometry and parametric values. No novel features were discovered in these runs.

Figure 1 shows the front portion of a clean airfoil, the attached ice, and an approximating B-spline. In this case, we required the B-spline to satisfy a maximum tolerance of \(\epsilon_{\text{max}} = 1.0 \times 10^{-2}\). Endpoint derivatives were free, and \(n\) was set initially equal to the default value of three. The original data was normalized with respect to the chord length; all lengths, including tolerances, are therefore similarly normalized in this and all subsequent figures. A total of 525 points defines the full ice geometry. A significant fraction of these points are denoted in the figure by small plus signs. Though in this figure one cannot determine the particular point \(C(\bar{u}_k)\) on the B-spline curve that corresponds to the \(k\)th data point \(Q_k\), it appears that every point on the B-spline curve is located within the maximum distance \(1.0 \times 10^{-2}\) of some point on the ice geometry.
Figure 1: Global view of front portion of clean airfoil, ice data, and approximating B-spline ($p = 3$; $\epsilon_{\text{max}} = 1.0 \times 10^{-2}$; free endpoint derivatives; initial $n = 3$)

Figure 2: Close-up view of three B-spline representations of ice data at the indicated moderate values of the maximum tolerance ($p = 3$; free endpoint derivatives; initial $n = 3$)
Figure 2 shows a magnified section of the clean airfoil, the ice data, and three approximating splines near the upper icing limit. Maximum tolerances of the approximating curves are, in decreasing order, $\varepsilon_{\text{max}} = 1.0 \times 10^{-2}, 5.0 \times 10^{-3}, \text{and} 1.0 \times 10^{-3}$. Visual examination shows that the average distance between a B-spline and the ice data correlates with the specified maximum tolerance. At this scale one can see that all three B-spline curves interpolate the first point $Q_0$ of the ice data as required. The consequence of not specifying the endpoint derivatives is also seen: the slopes of the three splines differ enormously at the upper endpoint.

The beginning of the ice data and a B-spline (the one with the smallest tolerance in Figure 2) is shown at greater magnification in Figure 3. Circles on the B-spline curve represent $C(u_k)$ for $k = 0, \ldots, 10$. Observe that the curve passes somewhat closer to each point $Q_k$ (indicated by plus signs) than the nominal separation distance $|C(u_k) - Q_k|$. 

The two B-spline curves that appear in Figure 4 were generated with the maximum tolerance set at $\varepsilon_{\text{max}} = 1.0 \times 10^{-3}$. The curve with large wiggles near the endpoint was constructed with free endpoint derivatives; the other curve had its endpoint derivatives specified as the default values. Both curves were generated using the appropriate default initial values for $n$ ($n = 3$ for the free endpoint derivative curve; $n = 4$ for the fixed endpoint derivative curve). The success of the second curve as a suitable representation of the ice geometry is due to specification of the endpoint derivatives. (One might argue that the different initial values used for $n$ might also contribute to the observed difference in these B-splines because the initial knot vectors are different. However, for this geometry we found that the same B-spline is obtained in the free endpoint derivative case whether an initial value of $n = 3$ or $n = 4$ is used. This is because at
**Figure 4**: Effect of fixing endpoint derivatives in a B-spline 
\(( p = 3; \varepsilon_{\text{max}} = 0.0001; \text{ initial } n = 4 )\) at small tolerances

**Figure 5**: Close-up view of three B-spline representations of ice data at the indicated moderate values of the maximum tolerance \(( p = 3; \text{ fixed endpoint derivatives; initial } n = 3 )\)
some stage of the iterative process, both procedures use Equation (10) to form a new knot vector for the same value of \( n \). All subsequent steps in the two procedures to generate the final B-splines are then identical.) This same behavior was found in corresponding representations for both of the other ice geometries investigated. It is widely known that B-splines have a tendency to exhibit wiggles when approximating noisy data at small tolerances [6]. We note that, despite specification of the endpoint derivatives, unacceptably large wiggles in the B-spline representation may arise at smaller tolerance levels. These were observed for the present geometry with a maximum tolerance of \( \epsilon_{\text{max}} = 2.5 \times 10^{-3} \).

It may be advantageous to fix the endpoint derivatives at more moderate levels of tolerance. For example, the slopes of the iced airfoil near the icing limits are sometimes relevant in the merging of the ice representation with the clean airfoil to avoid the introduction of significant discontinuities [9]. In any case, specification of endpoint derivatives can have a dramatic effect on the entire B-spline, not just near the endpoints. Figure 5 shows three B-splines for which the endpoint derivatives were set to their default values; all curves in the figure appear to have the appropriate endpoint derivative. The maximum tolerances of the three splines in Figure 5 are the same as those in Figure 2; corresponding curves in the two figures should be compared. These two figures, along with Figure 4, suggest the following two general rules: a) the effect of specifying an endpoint derivative decreases with distance (along the curve) from the endpoint, and b) the distance over which the endpoint derivative has a significant effect is inversely related to the tolerance. If the tolerance is sufficiently large, as it is for the two \( \epsilon_{\text{max}} = 1.0 \times 10^{-2} \) curves, the curves can appear significantly different from each other for the whole domain. Furthermore, by controlling the values of the endpoint derivatives, one can obtain intermediate representations of the ice geometry. These observations appear consistent with the local support property of the basis functions and the (frequently) inverse relationship between tolerance and the number of knot spans in the resulting B-spline.

We note that in those runs reported above with acceptable representations of the ice geometry (i.e., no significant wiggles), the ratio \( d_{\text{max}} / \epsilon_{\text{max}} \) varied over the range (0.77, 0.99). Moreover, the variation of this ratio was not monotonic with respect to the tolerance. That the ratio is not constant should not be unexpected because the algorithm does not control this quantity beyond the requirement that it be located within the interval (0,1]. Similarly, the algorithm does not control the ratio \( d_{\text{rms}} / \epsilon_{\text{rms}} \) when the rms tolerance is specified; it too is limited to the interval (0,1]. The variation of these ratios has an interesting potential consequence when two B-spline approximations of the same ice geometry is generated with either the maximum or rms tolerances slightly different but all other parameters unchanged. The usual expectation is that the distance \( d_{\text{max}} \) or \( d_{\text{rms}} \), depending upon which tolerance is specified, will be smaller for the spline with smaller specified tolerance. While this pattern frequently will be true, there may be exceptions (e.g., \( d_{\text{max}} \) larger for the spline generated with smaller \( \epsilon_{\text{max}} \)) because the corresponding ratio is not constant. This behavior in no way detracts from the anticipated use of the software, which is to easily generate in a controlled manner B-spline representations of ice geometries that have a wide variety of roughness levels. Other software will quantify the roughness characteristics of the resulting B-spline representations, including the characteristic distance of the curve from the data.

**Summary**

A FORTRAN 77 program that generates a smoothed B-spline representation of a given ice geometry has been developed. The user specifies a maximum tolerance or a root-mean-square tolerance along with other necessary parameters. The program returns a B-spline curve that satisfies the given tolerance. The software permits the rapid generation of several B-splines that satisfy a wide range of tolerance requirements. This approach represents a significant improvement over the current technique of smoothing in SmaggIce 2D, in which selected control points are deleted. This program was developed for possible incorporation into the next version of SmaggIce 2D (v1.8).
Any B-spline curve that approximates the ice data to within a given tolerance is not unique. Several additional parameters are available to the user to produce a different B-spline representation of the ice data. Sometimes use of these additional parameters is necessary to obtain a useful representation of the ice geometry. For example, for each of three different ice geometries investigated, it was found that if the endpoint derivatives were free and the maximum tolerance sufficiently small, the resulting approximating B-spline curve possessed large wiggles near an endpoint. In each of these cases, also requiring satisfaction of suitable endpoint derivative constraints led to an appropriate representation.

This Year 1 report discusses the theory behind the program and illustrates its use with respect to a given ice geometry. The program itself appears in Appendix C.
References


Appendix A—List of Symbols

<table>
<thead>
<tr>
<th>Symbols</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>$d_{\text{max}}$</td>
<td>Maximum separation distance between B-spline and ice data; defined in Equation (16)</td>
</tr>
<tr>
<td>$d_{\text{rms}}$</td>
<td>Rms separation distance between B-spline and ice data; defined in Equation (17)</td>
</tr>
<tr>
<td>$m\text{data}$</td>
<td>Last index of ice data</td>
</tr>
<tr>
<td>$m\text{knot}$</td>
<td>Last index of knot vector</td>
</tr>
<tr>
<td>$n$</td>
<td>Last index of control points; last $i$-index of basis functions $N_{i,p}(u)$</td>
</tr>
<tr>
<td>$p$</td>
<td>Degree of B-spline curve</td>
</tr>
<tr>
<td>$u$</td>
<td>Curve parameter, $0 \leq u \leq 1$</td>
</tr>
<tr>
<td>$u_i$</td>
<td>$i$th knot; $i = 0, \ldots, m\text{knot}$</td>
</tr>
<tr>
<td>$\overline{u}_k$</td>
<td>$k$th data parameter; defined in Equation (8); $k = 0, \ldots, m\text{data}$</td>
</tr>
<tr>
<td>$C(u)$</td>
<td>Two-dimensional B-spline curve; functionally dependent upon curve parameter $u$</td>
</tr>
<tr>
<td>$C'(u)$</td>
<td>First derivative curve of B-spline; $C'(u) = dC(u)/du$</td>
</tr>
<tr>
<td>$N_{i,p}(u)$</td>
<td>$i$th B-spline basis function of degree $p$; $i = 0, \ldots, n$; functionally dependent upon curve parameter $u$</td>
</tr>
<tr>
<td>$N$</td>
<td>Matrix of basis functions; defined in Equation (12) or (18)</td>
</tr>
<tr>
<td>$P$</td>
<td>Matrix of control points; defined in Equation (13) or (19)</td>
</tr>
<tr>
<td>$P_i$</td>
<td>$i$th two-dimensional control point; $i = 0, \ldots, n$</td>
</tr>
<tr>
<td>$Q_k$</td>
<td>$k$th two-dimensional data point; $k = 0, \ldots, m\text{data}$</td>
</tr>
<tr>
<td>$R$</td>
<td>Matrix defined in Equation (14) or (20)</td>
</tr>
<tr>
<td>$R_k$</td>
<td>Vector defined in Equation (15) or (21)</td>
</tr>
<tr>
<td>$U$</td>
<td>Knot vector; form given in Equation (2)</td>
</tr>
<tr>
<td>$\alpha$</td>
<td>Factor used to generate new maximum tolerance from relation $\varepsilon_{\text{max}} = \alpha \cdot d_{\text{max}}$ during iteration to satisfy rms tolerance</td>
</tr>
<tr>
<td>$\varepsilon_{\text{max}}$</td>
<td>Maximum tolerance</td>
</tr>
<tr>
<td>$\varepsilon_{\text{rms}}$</td>
<td>Rms tolerance</td>
</tr>
</tbody>
</table>
Appendix B—Program Description

Main Program
This software is intended to be incorporated into SmaggIce 2D, which has its own GUI interface. A simple text-based interface with users was therefore adopted for development purposes. All input and output is controlled by the main program. The user sequentially enters the following data:

1. Run number (two digits)
2. Name of data file
3. Are endpoint derivatives specified? If yes, either accept default values or provide values.
4. Degree $p$ of B-spline (default value is $3$; must be $3, 4, \text{or} 5$.)
5. Initial value of $n$ (default value is the minimum value, which is $p$ or $p + 1$ depending on whether endpoint derivatives are free or fixed)
6. Maximum tolerance (default value is $\epsilon_{\max} = 1.0 \times 10^{-3}$)
7. Enter rms tolerance if desired. If so, factor $\alpha$ (default value: $\alpha = 0.9$) must also be entered.

This concludes the user’s input.

The data file is presumed to be an ASCII file structured as follows:

Line 1: Number of data sets (1 for just ice data or 2 for both clean and iced airfoil data)
Line 2: Number of points in first data set
Lines 3 to end of data set 1: x-y data of first data set
Next line: Number of point in second data set (if present)
Next line to end: x-y data of second data set

The first data set may represent just the ice geometry or the iced airfoil. The second data set, if present, represents the clean airfoil. The program reads in the data file, and if necessary separates the ice data from the airfoil data. Data is written to appropriate file(s), and ice data is retained in memory to calculate the approximating B-spline curve using the iterative procedure described previously.

The most important file exported by the main program is nurbs??.dat. It contains the data that defines the approximating B-spline curve. Here, ?? denotes a two-digit run number. The file has the following format:

Line 1: $p$ (degree)
Line 2: $n$ (final value)
Lines 3: $mknot$ (final value)
Next $mknot + 1$ lines: Knot vector
Next $n + 1$ lines: Control points (x-y format)
The table below summarizes all the files exported by the main program. Following the table are brief statements of purpose for each subroutine included in the program file.

<table>
<thead>
<tr>
<th>File Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>clean.dat</td>
<td>Clean airfoil data (x-y format)</td>
</tr>
<tr>
<td>cukb???.dat</td>
<td>C(\overline{\alpha}_k), k = 0, \ldots, mdata (x-y format)</td>
</tr>
<tr>
<td>ice.dat</td>
<td>Ice geometry (x-y format)</td>
</tr>
<tr>
<td>icecv???.dat</td>
<td>B-spline curve (x-y format)</td>
</tr>
<tr>
<td>log.txt</td>
<td>Log file of run</td>
</tr>
<tr>
<td>nurbs???.dat</td>
<td>NURBS data file</td>
</tr>
<tr>
<td>sum???.txt</td>
<td>Summary of run</td>
</tr>
</tbody>
</table>

**FindSpan**

The semi-closed interval \([u_i, u_{i+1})\) represents the \(i\)th knot span. FindSpan is a function that, given the curve parameter \(u\), returns the knot span in which \(u\) is located. The value \(u = 1\) is an exception to the above definition; it is assigned knot span \(n\).

**FindSpanA**

If \(u\) represents a 1D array of curve parameter values, subroutine FindSpanA returns a 1D array of the corresponding knot span indices.

**NBasis**

Given the scalar \(u\), its knot span index \(i\), degree \(p\), and knot vector \(U\), subroutine NBasis computes the set of nonzero basis functions \(N_{i-p,p}(u), \ldots, N_{i,p}(u)\) and returns their values in the 1D array \(N\).

**NBasisA**

Given the 1D array \(u\), a corresponding set of knot span indices, degree \(p\), and knot vector \(U\), subroutine NBasisA computes the full set of nonzero basis functions, returning them in the 2D array \(NA\). The \(j\)th column of \(NA\) corresponds to the \(j\)th element of \(u\).

**Cparam**

Given the point data \(Q_0, \ldots, Q_m\) for the ice, subroutine Cparam calculates the set of data parameters \(\overline{\alpha}_0, \ldots, \overline{\alpha}_m\) according to Equation (8), returning their values in the 1D array \(ukb\).

**Knotvec**

Subroutine Knotvec generates a knot vector \(U\) based upon Equation (10).

**NCurve**

Subroutine NCurve computes the point \((x, y) = C(u)\) on the B-spline curve for the scalar curve parameter \(u\).
NCurveA
Subroutine NCurveA computes the set of \((x,y)\) points on the B-spline curve that correspond to a 1D array of curve parameters.

Rkarray
Subroutine Rkarray computes \(R_1, \ldots, R_{\text{ndata}-1}\) corresponding to either Equation (15) or (21), depending upon whether the endpoint derivatives are free or fixed, respectively.

RightHandSide
Subroutine RightHandSide computes \(R\) according to either Equation (14) or (20), depending upon whether the endpoint derivatives are free or fixed, respectively.

ABMatrix
Subroutine ABMatrix computes the matrix \(N^T N\) and stores it in an upper band form that is suitable for the LAPACK library routine DPBSV [10]. (Subroutine DPBSV solves the linear system, Equation (11), based upon the Cholesky method.) The matrix \(N\) is given either by Equation (12) or (18), depending upon whether the endpoint derivatives are free or fixed, respectively.

SpanTest
Subroutine SpanTest returns in the 1D array nonspan the indices of all nonconforming knot spans; i.e., those spans that do not satisfy the tolerance requirement \(C(\bar{u}_k) - Q_k \leq \varepsilon_{\max}\).

Deviations
Subroutine Deviations returns the maximum separation distance \(d_{\max}\), the root-mean-square deviation \(d_{\text{rms}}\), and a 2D array containing the \((x,y)\) values corresponding to \(C(\bar{u}_0), \ldots, C(\bar{u}_{\text{ndata}})\). The quantities \(d_{\max}\) and \(d_{\text{rms}}\) are respectively defined by Equations (16) and (17), and in the program correspond to FORTRAN variables epmax and rms.

NewU
Subroutine NewU generates a new knot vector by inserting a new knot at the midpoint of every nonconforming knot span.

Verify
Subroutine Verify checks each knot span to insure that each contains at least one value of the set of data parameters \(\bar{u}_0, \ldots, \bar{u}_{\text{ndata}}\).
Appendix C—FORTRAN 77 Program

program globalapp

Program Summary

Given a set of Q=(x,y) data points of length mdata + 1, two integers n and p (such that 3 ≤ p ≤ n < mdata), and a maximum distance criterion ep (0 < ep ≤ 0.1, say), the program globalapp employs an iterative least-squares procedure to calculate a NURBS curve of degree p that approximates the data to within the nominal distance ep. Optionally, the user may also specify a root-mean square distance criterion (eprms). Endpoint derivatives may be free (default) or fixed. All NURBS curves generated by this program have endpoints that exactly coincide with the prescribed ice data. The first and last control points are equal to the respective endpoints.

Data is presumed read in a counter-clockwise direction about the airfoil. This is only important in referencing upper and lower endpoints of the ice region.

After entry of the requisite data, the program calculates a knot vector, and a do while loop is entered. A matrix and right-hand side vector (2 columns) is then generated with control points as unknowns. The matrix is banded, symmetric, and positive definite. A special LAPACK solver that uses the efficient Cholesky method is then used to obtain the unknown control points. The resulting NURBS curve is used to determine whether the distance criterion eptemp (initially, the same as ep) is satisfied. If not, n is appropriately increased by the insertion of knots at the midpoints of non-converging spans, thereby creating a new knot vector. The do while loop is then entered again from the top, and the cycle is repeated until the NURBS curve satisfies the tolerance requirement, or the resulting matrix is singular. If at any stage the newly created knot vector has a span that does not contain a value of the parameterized curve, the whole knot vector is recalculated by redistribution based upon the original algorithm. If the redistributed knot vector also contains one or more spans without a value of the parameterized curve, the do
while loop is exited, and the approximation fails.

If eprms is specified, the iteration procedure is slightly modified as follows. The specified maximum distance specification is first met as indicated above. If eprms is specified, the root mean squared distance is compared with eprms. If this specification is satisfied, the iteration loop is exited. If not, the maximum distance specification eptem is reduced by a factor alpha, $0 < \alpha < 1$, and iterations continue until this new maximum distance specification is met. The rms specification is then checked again. The cycle repeats until the rms specification is satisfied.

The procedure when endpoint derivatives are specified is nearly the same. An endpoint derivative is fully specified by a polar angle and a magnitude. By default, first-order finite difference calculations are used to prescribe these derivatives. After a knot vector is first defined as above, the derivative information is used to determine the 2nd and next to last control points. The do while loop is then entered as above. Though governing equations differ slightly in this case, a linear system of equations is solved to find the remaining unknown control points. A modified knot vector is then created in a manner similar to that above. If at any time the knot vector needs to be redistributed, the second and second to last control points are recalculated. The fixed derivative algorithm was specially developed for this program, and is not found in the reference below.

After a NURBS curve is found that satisfies specified tolerance(s), the requisite information of the NURBS curve is saved to a file for later reconstitution. A log file is also kept, as well as a file that summarizes the results of the analysis. Also output is an x-y data file that represents the generated NURBS curve for convenient graphing.

************************************************************************

Reference


************************************************************************
* NOTE: In Piegl and Tiller, the first index of most vectors begin
* with zero. To avoid confusion, we follow that convention as
* appropriate. We also use indices 1 and 2 to denote x and y
* components of certain arrays.
*
* ************************************************************************
* *
* Record of Revisions:
* *
* Date            Programmer                Description of Change
* ====          ============              =======================
* 05/20/03      Loren H. Dill             Original code
* 05/22/03      L.H. Dill                 Added rms tolerance
* 08/06/03      L.H. Dill                 Increased saved digits
*                                             in output (nurbs??.dat)
* *
* ************************************************************************
* *
* Parameters
* *
* mcleandatamax Largest index of x-y points output to data file
*                for clean airfoil (file: clean.dat)
* mdatamax      Largest anticipated index of input data vector,
*                clean plus ice
* m2datamax     Largest index of output x-y data vector for rough
*                ice (along NURBS curve) (file: icecv???.dat)
* nmax          Largest anticipated index of control points
* nredistmax    Largest index of nvalues vector
* pmax          Highest anticipated degree of NURBS curve
* *
* ************************************************************************
* *
* Major Input Variables
* *
* mdata Number of x-y data points (less 1 after initial index set
* to 0). mdata is initially read in, and depending upon
* dataset format, may be total number of ice+clean data
* points. Throughout the main part of the program, mdata
* represents the number of ice data less one.
* Q       x,y ice data points -- a 2D array
* p       Desired degree of NURBS curve
* n       Largest index of NURBS curve. Initially given and then
*         increased during the iteration procedure
phi0, phiM First and last polar angles corresponding to desired
slopes of NURBS curve at beginning and end. (in degrees).
D0L, DmL Magnitude of first derivative vectors of NURBS curve
at beginning and end
sD0L, sDmL Scale factors, relative to default values, of the
magnitude of first derivative vectors of NURBS curve
at beginning and end
ep Desired maximum tolerance between NURBS curve and prescribed
data
eprms Desired maximum rms tolerance between NURBS curve and
prescribed data
run A two-digit character array designated the run number. Used
to associate output file names with given runs.
infilename A character array of length 12 used to identify
the file containing the x-y ice data.

Major Working Variables

epmax Calculated value of maximum distance from curve to the
ice data. Actually a nominal value.
eptemp If only maximum tolerance is specified, eptemp = ep.
If eprms tolerance is specified, eptemp < ep if eprms
is not achieved. eptemp is periodically reduced until
eprms is achieved.
mknot Largest index of knot vector U, mknot = n+p+1
INFO Variable from LAPACK routine dpbsv that solution
of linear matrix equation was successful or not
in nonspan number of nonconverging knot spans
nonspan vector containing spans that have not converged
nvalues a 1-D array containing the n values for which
the knot vector was redistributed. Only last
nredistmax values are retained.
offset Derivative flag.
offset = 1 if endpoint derivatives are not set.
offset = 2 if they are
P Control points
ukb array containing values of parameterized curve; each
value of array corresponds to respective data point
spanA an array that contains the span indices of ukb relative
to a given knot vector.
u Curve parameter
U Knot vector
* Unew A new knot vector awaiting verification before acceptance
* NA Array containing the non-zero basis functions for each
  element of a parameter array
* Rk array containing the Rk values of Piegl & Tiller, p 411,
  if end derivatives are free, and a modified version
  if end derivatives are fixed.
* R array representing right-hand side of eq. 9.65 of Piegl
  & Tiller, p 411, if end derivatives are free, and a
  modified version if end derivatives are fixed.
* NTNB matrix (in banded form) for linear eq. 9.65 of P&T,
  if end derivatives are free, and a modified version
  if end derivatives are fixed.
* CA array containing the x-y data of the NURBS curve
  corresponding to a given parameter array
* D0,DM Vectors representing endpoint derivatives if prescribed
* rms the root mean square distance between the curve and the
  set of data points. Again a nominal value.
* success A logical variable indicating that a newly created knot
  vector has an element of ukb within each span, or not.
*
*************************************************************************
* OUTPUT FILE DESCRIPTIONS
*************************************************************************
* clean.dat The clean airfoil data, if given upon input
* cukb??.dat Data file containing x-y data of NURBS curve
  corresponding to parameter array ukb
* ice.dat The ice data
* icecv??.dat The NURBS curve x-y data
* log.txt A log file for the last execution of glapp
* nurbs??.dat A file containing NURBS data, such as the knot vector
* sum???.txt A file that summarizes a run
*
* In the above, ?? denotes the run number given upon input.
*
*************************************************************************
* Compile Command for g77 compiler
*g77 -fssource-case-preserve -Wunused -Wall -Wsurprising glapp.f -llapack -lblas -o glapp
* Use xmgrace or other plotting program to examine output *dat files
* implicit none

integer mdata,mdatamax,m2datamax,nmax, pmax,mcleandatamax
integer ndatasets,nredist,nredistmax,mcleandata,nmaxm

parameter (mdatamax=600,nmax=600,pmax=5,m2datamax=8192)
parameter (mcleandatamax=250,nredistmax=5)

integer i,j,mknot,offset,p,n,spanA(0:m2datamax)
integer nonspan(nmax-1),inonspan
integer nvalues(nredistmax),nfirst,nlast
integer INFO

double precision U(0:nmax+pmax+1),du,u(0:m2datamax)
double precision Q(2,0:mdatamax),Qclean(2,0:mcleandatamax)
double precision ukb(0:mdatamax),P(2,0:nmax),Unew(0:nmax+pmax+1)
double precision NA(0:pmax,0:mdatamax)
double precision CA(2,0:m2datamax)
double precision Rk(2,0:mdatamax),R(nmax-1,2)
double precision NTNB(pmax+1,nmax-1),ep,eprms,eptemp,alpha
double precision D0(2),Dm(2),sD0L,sDmL
double precision phi0, phim,D0L,DmL,pi,epmax, rms

character*12 infilename
character*1 ans,ansD,ansDC,ansDr,ansphi,ansdeg,ansn,ansrms
character*2 run

logical success

*  Open all output files and write a blank to clear data from
*  any previous run of same number. Open and close later as needed.
write(*,*)'Please enter a 2-digit run number.'
read(*,*)run
open(7,file='icecv'//run//''.dat')
open(8,file='log.txt')
open(9,file='clean.dat')
open(11,file='ice.dat')
open(12,file='nurbs//'run//''.dat')
open(13,file='sum//'run//''.txt')
open(14,file='cukb//'run//''.dat')
write(7,'(a)');write(8,'(a)');write(9,'(a)'
write(11,'(a)');write(12,'(a)');write(13,'(a)'
write(14,'(a)'
close(7);close(8);close(9)
close(11);close(12);close(13);close(14)
Read in airfoil data. First line of data file contains number of datasets (1 = only ice data; 2 = both clean and ice data).
Second line contains number of data points of first data set. Ask user for name of input file then read first two lines.

```fortran
write(*,*)'Enter name of data file as string.'
read(*,*)infilename

open(10,file=infilename)
read(10,*)ndatasets
read(10,*)mdata

mdata = mdata - 1 ! Adjust maximum because indices begin at 0
if (ndatasets .eq. 1) then
  if (mdata .gt. mdatamax) then
    write(*,*)'Number of data points ',mdata,' exceeds maximum ',mdatamax,'. Aborting . . .'
    open(13,file='sum//run//.txt')
    write(13,*)'Number of data points ',mdata,' exceeds maximum ',mdatamax,'. Aborting . . .'
    close(13)
    stop
  end if
  do i=0,mdata
    read(10,*)Q(1,i),Q(2,i)
  end do
  close(10)
else if(ndatasets .eq. 2) then
  mcleanedata = mdata
  if (mcleanedata .gt. mcleandatamax -1) then
    write(*,*)'Number of clean airfoil data points ',mcleanedata,' exceeds maximum ',mcleandatamax,'. Aborting . . .'
    open(13,file='sum//run//.txt')
    write(13,*)'Number of clean airfoil data points ',mcleanedata,' exceeds maximum ',mcleandatamax,'. Aborting . . .'
  end if
  do i=0,mcleanedata
    write(11,10)Q(1,i),Q(2,i)
  end do
  close(11)
else if(ndatasets .eq. 3) then
  mdatamax = mdatamax - 1
  mcleanedata = mdata
  if (mcleanedata .gt. mcleandatamax -1) then
    write(*,*)'Number of clean airfoil data points ',mcleanedata,' exceeds maximum ',mcleandatamax,'. Aborting . . .'
    open(13,file='sum//run//.txt')
    write(13,*)'Number of clean airfoil data points ',mcleanedata,' exceeds maximum ',mcleandatamax,'. Aborting . . .'
  end if
  do i=0,mcleanedata
    write(11,10)Q(1,i),Q(2,i)
  end do
  close(11)
end if
```

Write ice data to file
open(11,file='ice.dat')
do i=0,mdata
  write(11,10)Q(1,i),Q(2,i)
end do
close(11)
close(13)
stop
end if
do i=0,mcleandata
   read(10,*)Qclean(1,i),Qclean(2,i)
end do
open(9,file='clean.dat')
do i=0,mcleandata
   write(9,10)Qclean(1,i),Qclean(2,i)
end do
close(9)
read(10,*)mdata
if (mdata .gt. mdatamax-1) then
   write(*,*)'Number of data points ',mdata,' exceeds maximum '1
   ,'Increase mdatamax. Aborting . . .'
   open(13,file='sum'//run//'sum.dat')
   write(13,*)'Number of data points ',mdata,' exceeds maximum'
   ', Increase mdatamax. Aborting . . .'
   close(13)
stop
end if
mdata = mdata - 1
do i=0,mdata
   read(10,*)Q(1,i),Q(2,i)
end do
close(10)

* Check that the length of the ice data exceeds the length
* of the clean airfoil data by a sufficient amount. The 20
* is not a rigid requirement. This is only to separate ice and
* clean airfoil data
if(mdata .lt. mcleandata+20)then
   write(*,*)'Ice data length too short to proceed.'
   write(*,*)'Aborting'
   stop
end if

* Determine where ice begins in ice/airfoil data
i=0
do while( (abs(Qclean(1,i)-Q(1,i)) .lt. 1. e-6 ) .and.1
   (abs(Qclean(2,i)-Q(2,i) ) .lt. 1. e-6 ) )
i=i+1
end do
nfirst=i

* Determine where ice ends in ice/airfoil data
  i=0
  do while( (abs(Qclean(1,mcleandata-i)-Q(1,mdata-i)).lt.1.e-6 )
  .and.(abs(Qclean(2,mcleandata-i)-Q(2,mdata-i)).lt.1.e-6 ))
    i=i+1
  end do
  nlast=i
  mdata = mdata - nfirst - nlast

* Transfer ice data to beginning of arrays
  do i = 0,mdata
    Q(1,i)=Q(1,nfirst+i)
    Q(2,i)=Q(2,nfirst+i)
  end do

* Write ice data to file
  open(11,file='ice.dat')
  do i=0,mdata
    write(11,10)Q(1,i),Q(2,i)
  end do
  close(11)
end if

* Begin summary and log files. Get input file name and run number.
  open(13,file='sum//run//'.txt')
  write(13,*)'Input data filename = ',infilename
  write(13,*)'Run number = ',run
  open(8,file='log.txt')

* Parameterize data
  call Cparam (mdata, Q, ukb)

* Inquire about endpoint derivatives
  pi=4.0*atan(1.0)
  ans = '0'
  do while (ans .ne. '1')
    write(*,*)'Do you want endpoint derivatives specified (y/n)??'
    read(*,*)ansD
    if (ansD .eq. 'Y' .or. ansD .eq. 'y')then
      D0(1)=(Q(1,1)-Q(1,0))/ukb(1) ! Calculate default values
      D0(2)=(Q(2,1)-Q(2,0))/ukb(1)
      Dm(1)=(Q(1,mdata)-Q(1,mdata-1))/(1-ukb(mdata-1))
Dm(2)=((Q(2,mdata)-Q(2,mdata-1))/(1-ukb(mdata-1))
D0L = sqrt(D0(1)*D0(1)+D0(2)*D0(2))
DmL = sqrt(Dm(1)*Dm(1)+Dm(2)*Dm(2))
phi0=atan2(D0(2),D0(1))*180./pi
phim=atan2(Dm(2),Dm(1))*180./pi
write(13,*)
write(13,*)'Endpoint derivatives specified.'
write(*,*)'Do you want endpoint derivatives calculated '
write(*,*)'by first-order differencing of first and last '
write(*,*)'endpoint data pairs (y/n)?'
read(*,*)ansDC
if (ansDC .eq. 'Y' .or. ansDC .eq. 'y')then
    write(13,*)'Derivatives calculated by differencing'
    write(13,*)'of endpoint data pairs'
    ans = '1'
    offset=2
else if (ansDC .eq. 'N' .or. ansDC .eq. 'n') then
    write(*,*)
    write(*,*)'Both direction and magnitudes of endpoint'
    write(*,*)'vectors must be specified. Do'
    write(*,*)'you want the default directions determined'
    write(*,*)'from finite differencing of the first and'
    write(*,*)'last pairs of prescribed data? (y/n)'
    read(*,*)ansphi
if (ansphi .eq. 'N' .or. ansphi .eq. 'n') then
    write(*,*)'Default values for upper and lower'
    write(*,*)'angles in degrees:'
    write(*,50)phi0,phim
    write(*,*)
    write(*,*)'Enter upper and lower endpoint'
    write(*,*)'directions as polar angles in degrees:'
    read(*,*)phi0,phim
end if
write(*,*)
write(*,*)'Do you want finite differences of endpoint'
write(*,*)'pairs to determine the magnitudes of '
write(*,*)'these derivatives (y/n),'
read(*,*)ansDr
if (ansDr .eq. 'N' .or. ansDr .eq. 'n') then
    write(*,*)
    write(*,*)'Specify the upper and lower '
    write(*,*)'magnitudes of these vectors as'
    write(*,*)'scale factors of the default values:'
read(*,*) sD0L, sDmL
D0L=sD0L*D0L
DmL=sDmL*DmL
end if
D0(1)=D0L*cos(phi0*pi/180.)
D0(2)=D0L*sin(phi0*pi/180.)
Dm(1)=DmL*cos(phim*pi/180.)
Dm(2)=DmL*sin(phim*pi/180.)
write(13,*)'Derivatives specified by user:  '
ans = '1'
offset=2
end if
write(13,*)'Polar angle (degrees):'
write(13,*)'  upper:  ',phi0
write(13,*)'  lower:  ',phim
write(13,*)'Lengths:  '
write(13,*)'  upper:  ',D0L
write(13,*)'  lower:  ',DmL
write(13,*)
else if (ansD .eq. 'N' .or. ansD .eq. 'n') then
write(13,*)'Endpoint derivatives not specified by user.'
write(13,*)
ans = '1'
offset = 1
end if
end do

* Set degree p and initial number of terms via n
p=3
write(*,*)'The default degree of the NURBS curve is 3.'
write(*,*)'Do you want the default degree (y/n)'
read(*,*)ansdeg
if (ansdeg .eq. 'N' .or. ansdeg .eq. 'n') then
write(*,*)'Degree must satisfy 3 <= p <= pmax = ',pmax
write(*,*)'Enter degree:'
read(*,*)p
if( p .gt. pmax .or. p .lt. 3)then
write(*,*)'NURBS degree p = ',p,' not in range'
write(*,*)' Aborting run...' 
write(13,*)'NURBS degree p = ',p,' not in range'
write(13,*)' Aborting run...'
close(13)
stop
end if
end if

nmaxm = min(nmax,mdata - p - 2)
if ( (offset .eq. 1 .and. nmaxm .lt. p) .or.
  (offset .eq. 2 .and. nmaxm .lt. p+1) )then
  write(*,*)'Your dataset is too small to proceed. Aborting..'
  write(8,*)'Your dataset is too small to proceed. Aborting..'
  write(13,*)'Your dataset is too small to proceed. Aborting..'  
  stop
end if
write(*,*)
write(*,*)'For the number of terms in the initial NURBS'
write(*,*)'curve, do you want the minimum acceptable value (y/n)?'
read(*,*)ansn
if (ansn .eq. 'Y' .or. ansn .eq. 'y')then
  if (offset .eq. 1)n=p
  if (offset .eq. 2)n=p+1
else
  write(*,*)'Enter n to specify number of terms in '
    ',initial NURBS curve'  ! n actually specifies number
  if (offset .eq. 1) then ! terms less one
    write(*,*)'Initial n must satisfy ',p,' < = n < ',nmaxm
  else
    write(*,*)'Initial n must satisfy ',p+1,' < = n < ',nmaxm
  end if
write(*,*)
write(*,*)'(This maximum is an upper limit. Typically want'
write(*,*)' to set n near lower limit and n << ',nmaxm,'.'
write(*,*)'Matrix may be singular for n < ',nmaxm,' depending '
  ',upon data set.)'
write(*,*)
write(*,*)'What value of n do you want?'
read(*,*)n
if( (offset .eq. 1 .and. n .lt. p) .or.
  (offset .eq. 2 .and. n .lt. p+1) .or.
  n .gt. nmaxm )then
  write(*,*)'n is outside specified range.'
  write(*,*)'Aborting run...'
  write(13,*)'Specified n = ',n,' is outside range.'
  write(13,*)'Aborting run...'  
  close(13)
stop
end if
end if
write(13,*)'Initial n: ',n

Inquire about desired maximum tolerance
ans = '0'
do while (ans .ne. '1')
write(*,*)'Do you want the default maximum tolerance ','
1     '(ep = 0.001)(y/n)?'
read(*,*)ans
if (ans .eq. 'Y' .or. ans .eq. 'y')then
    ep= 1.0d-3
    ans = '1'
else if (ans .eq. 'N' .or. ans .eq. 'n') then
    write(*,*)'Enter maximum tolerance (0 < ep <= 0.1)'
    read(*,*)ep
    ans = '1'
end if
end do
write(*,*)
if ( (ep .gt. 0.1 ) .or. (ep .le. 0.) ) then
    write(*,*)'Distance criterion ep = ',ep,' should be '
1     ,'in range 0 < ep <= 0.1. Aborting ...'
write(13,*)'Distance criterion ep = ',ep,' should be '
1     ,'in range 0 < ep <= 0.1. Aborting ...'
close(13)
stop
end if

Set eptemp equal to ep. eptemp will only differ from ep
* if root-mean square tolerance is set.
eptemp = ep
* Inquire about root-mean square tolerance
ans = '0'
do while (ans .ne. '1')
write(*,*)'Do you want to specify a root mean square tolerance ','
1     '(y/n)?'
read(*,*)ansrms
if (ansrms .eq. 'Y' .or. ansrms .eq. 'y')then
    write(*,*)'Enter the root mean square tolerance. Your value ','
1     'should lie between 0 and ',ep
    read(*,*)eprms
    if (eprms .gt. 0. .and. eprms .lt. ep)then
alpha = 0.9
write(*,*)'The default factor by which to reduce the'
write(*,*)'maximum deviation when attempting to satisfy'
write(*,*)'your root mean square tolerance is 0.9.'
write(*,*)'Do you accept the default factor?',
1           ' (y/n)?'
read(*,*)ans
if (ans .eq. 'N' .or. ans .eq. 'n')then
   write(*,*)'Enter the desired factor.'
   read(*,*)alpha
   if (alpha .gt. 0. .and. alpha .lt. 1)then
      ans = '1'
   else
      ans = '1'
   end if
else
   ans = '1'
end if
write(8,*)'Reduction factor to achieve eprms:',alpha
else
   ansrms = 'n'
   ans = '1'
end if
end do
write(8,*)'p , n, ep = ',p,' ',n,' ',ep
if (ansrms .eq. 'Y' .or. ansrms .eq. 'y')then
   write(8,*)'eprms = ',eprms
end if
write(8,*)'x,y, ukb data'
do i = 0,mdata
   write(8,10)Q(1,i),Q(2,i),ukb(i)
end do
write(13,*)'Specified NURBS degree: ',p
write(13,*)'Specified nominal maximum tolerance: ',ep
if (ansrms .eq. 'Y' .or. ansrms .eq. 'y')then
   write(13,*)'Specified rms tolerance: ',eprms
end if

* Create knotvector based upon Piegl & Tiller, p. 412, eqn. (9.69)
* and calculate mknot (number of knots less one). Keep track of
* value of n when Knotvec is called via vector nvalues.
nredist=1
nvalues(1)=n
call Knotvec (mdata, n, p, ukb, U)
mknot = n + p + 1
write(8,*)
write(8,*)'Knot Vector U(i)'
write(8,20)(i,U(i),i=0,mknot)
write(8,*)

* Verify each knot span contains at least one value of ukb. Output
* argument success returns .true. if all is OK. If unsuccessful,
* execution is stopped.
call Verify(mdata,n,p,ukb,U,success)

if (success .eqv. .false.) then
  write(8,*)'Before entering main loop, not all knot spans',
  write(8,*)'contain at least one value of ukb. Aborting ...'
  write(*,*)'Before entering main loop, not all knot spans',
  write(*,*)'contain at least one value of ukb. Need to reduce',
  write(13,*)'Before entering main loop, not all knot spans',
  write(13,*)'contain at least one value of ukb. Need to reduce',
  close(13)
  stop
end if

************************************************************************

* Enter iteration loop to calculate global approximating NURBS
* curve. Require at least one iteration by setting number of non-
* converging spans greater than zero, e.g., inonspan = 1. Loop
* repeats until the number of non-converging spans drops to zero
* (inonspan = 0), at least one knot span in new knot vector does not
* contain a parameter value, or the number of terms in the next
* NURBS representation exceeds nmax + 1 (i.e., n > nmax ).

inonspan = 1
write(8,*)'Entering do while' ! Main iteration loop.
do while ( inonspan .gt. 0 .and. success .and. n .le. nmax )
  write(8,*)
  write(8,*)'Top of do while '
  write(8,*)'n = ',n,' mknot = ',mknot
First, find span indices of all elements of ukb vector.
Next, calculate all non-zero basis functions associated with elements of ukb.

\[
\text{call FindSpanA(mdata, n, p, ukb, U, spanA)}
\]
\[
\text{call NBasisA(mdata, mknot, p, pmax, spanA, ukb, U, NA)}
\]

The first and last control points are simply the specified endpoint data. If end derivatives are specified, the second and next to last control points are determined by the derivative information.

\[
P(1,0) = Q(1,0) \\
P(2,0) = Q(2,0)
\]

if (offset .eq. 2) then
  \[
P(1,1) = Q(1,0) + \frac{U(p+1)}{p} \times D0(1) \\
P(2,1) = Q(2,0) + \frac{U(p+1)}{p} \times D0(2) \\
P(1,n-1) = Q(1,mdata) - \frac{(1.0 - U(n))}{p} \times Dm(1) \\
P(2,n-1) = Q(2,mdata) - \frac{(1.0 - U(n))}{p} \times Dm(2)
\]
end if

\[
P(1,n) = Q(1,mdata) \\
P(2,n) = Q(2,mdata)
\]

Next, calculate the set of Rk arrays, Piegl & Tiller, eqn. (9.63) on page 411, and then the right-hand side of (9.65):

\[
\text{call Rkarray(mdata, n, offset, p, pmax, spanA, Q, ukb, P, U, NA, Rk)}
\]
\[
\text{call RightHandSide(mdata, n, nmax, offset, p, pmax, spanA, NA, Rk, R )}
\]

Calculate the NTNB matrix of eqn. (9.65) in banded form.

\[
\text{call ABMatrix(mdata, n, offset, p, pmax, spanA, NA, NTNB)}
\]

Call the LAPACK banded matrix solver dpbsv to find the locations of the internal control points. (The first and last control points are simply the first and last data points. If endpoint derivatives are specified, the next inner control points are determined from derivative information.). If the solver fails (INFO .ne. 0), an error message is generated.

\[
\text{call dpbsv('u', n-2*offset+1, p, NTNB, pmax+1, R, nmax-1, INFO )}
\]

if (INFO .eq. 0) then
do j=offset,n-offset
  P(1,j)=R(j+1-offset,1)
  P(2,j)=R(j+1-offset,2)
end do
else if (INFO .lt. 0)then
  write(8,*)'The ',-INFO,' argument had an illegal value', 1
  ' in dpvsv. Aborting ...
  write(*,*)'The ',-INFO,' argument had an illegal value', 1
  ' in dpvsv. Aborting ...
  write(13,*)'The ',-INFO,' argument had an illegal value', 1
  ' in dpvsv. Aborting ...
close(13)
stop
else
  write(8,*)'Matrix in dpbsv is singular (U(',INFO,INFO, 1
  ') = 0. Aborting ...
  write(*,*)'Matrix in dpbsv is singular (U(',INFO,INFO, 1
  ') = 0.'
  write(*,*)'May need to increase tolerance ep or '
  write(*,*)'decrease initial n. Aborting ...
write(13,*)'Matrix in dpbsv is singular (U(',INFO,INFO, 1
  ') = 0.'
write(13,*)'May need to increase tolerance ep or '
write(13,*)'decrease initial n. Aborting ...
close(13)
stop
end if
write(8,*)
write(8,*)'Control points P(1,j) & P(2,j) for n = ',n
write(8,30)(j,P(1,j),P(2,j),j=0,n)

* The distance  \|C( ukb(k) ) - Q(k) \|  for each k = 1, mdata -1 is
* compared with distance specification ep (or eptemp if attempting
* to satisfy rms requirement). Here, Q(k) represents
* the kth data point of the airfoil. If this distance exceeds the
* specification for any data point within a given span, the entire
* span is declared to be non-converging and is tagged for
* subdivision for the next iteration.
call SpanTest(eptemp,mdata,n,p,Q,ukb,P,U,nonspan,
If all spans have converged (inonspan = 0), we calculate maximum (epmax) and root mean square (rms) deviations between curve and data. If eprms tolerance is not specified, the do while loop is exited. Otherwise, calculated and specified values for the root mean square error is compared. If rms > eprms, eptemp is set equal to alpha * epmax and SpanTest is called again. Here, alpha is a give parameter that lies between 0 and 1. Thus, we are guaranteed that at least one span will be non-compliant; i.e., inonspan will be greater than 0 as determined by SpanTest.

Note: If alpha is set too low, the tolerance eptemp may become unnecessarily small, and the program could fail. On the other hand, if alpha is too close to unity, eptemp will be reduced by only a small amount, and most likely only one span will be non-compliant. Hence, lots of iterations of the do while loop might be necessary for convergence to eprms. The default value of alpha = 0.9 should insure rapid convergence and in most cases eptemp should not become so small as to cause a singular matrix. If the procedure does bomb out, either eprms or alpha needs to be increased appropriately.

If inonspan > 0, we determine whether n will exceed nmax if another iteration is performed. If it will, execution is stopped. If the new n will be less than nmax, a new knot vector (Unew) is calculated. The new knot vector is formed by adding a new knot to the midpoint of each non-converging span. This new vector is tested to insure each span contains at least one value of ukb. If it does, the do while loop is executed again. Otherwise, a new knot vector is created for which all knots are redistributed. If this new knot vector contains any spans lacking a value of ukb, execution is stopped because the matrix NTNB is no longer guaranteed to be positive definite and well conditioned.

if (inonspan .eq. 0) then
  if (eptemp .eq. ep) then
    write(8,*)
    write(8,*)'Maximum tolerance achieved.'
    write(13,*)
    write(13,*)'Maximum tolerance achieved.'
  end if
call Deviations(mdata,n,p,pmax,Q,ukb,P,U,epmax,rms,CA)
if ( (ansrms .eq. 'Y' .or. ansrms .eq. 'y') .and. 
    rms .gt. eprms) then
  write(8,*)
  write(8,'(a)') 'epmax, rms = ',epmax,' ',rms
  eptemp = alpha * epmax
  write(8,'(a)') 'eptemp = ',eptemp
  call SpanTest(eptemp,mdata,n,p,Q,ukb,P,U,nonspan,
                1 inonspan)
else if ( (ansrms .eq. 'Y' .or. ansrms .eq. 'y') .and. 
    rms .le. eprms) then
  write(8,*)'rms tolerance achieved'
  write(8,*)'epmax, rms = ',epmax,' ',rms
end if
end if
* if there are non-converging spans, and new n > nmax:
if ( inonspan .gt. 0 .and. n + inonspan .gt. nmax ) then
  write(8,*)'Global approximation not converged, and next '
  1 'iteration will exceed maximum n. Aborting...'
  write(13,*)'Global approximation not converged, and next '
  1 'iteration will exceed maximum n. Aborting...'
  close(13);close(8)
  stop
* if there are non-converging spans, and new n <= nmax:
elseif ( inonspan .gt. 0 .and. n + inonspan .le. nmax )then
call NewU(mdata,n,nonspan,thononspan,p,ukb,U,Unew)
mknot = n+p+1
write(8,'(a)')
write(8,'(a)') 'Potential new U for n = ',n
write(8,'(a)') 'New mknot = ',mknot
write(8,20)(i,Unew(i),i=0,mknot)
write(8,*)
call Verify(mdata,n,p,ukb,Unew,success)
if (success) then ! The new knot vector created by inserting
  do i=0,mknot ! new knots at midpoints of non-converging
    U(i)=Unew(i) ! spans is OK. Save new knot vector
  end do ! to log file.
  write(8,*)
  write(8,'(a)') 'New Knot Vector OK for n = ',n
else
  write(8,'(a)') 'Modified knot vector had span with no ',
  1 'parameter value. Redistributing knots to ',

'create all new knot vector.'

nredist = nredist + 1

if (nredist .le. nredistmax) then
  nvalues(nredist) = n
else
  do i = 1, nredistmax - 1
    nvalues(i) = nvalues(i + 1)
  end do
  nvalues(nredistmax) = n
end if

call Knotvec(mdata, n, p, ukb, Unew)

write(8,*)

write(8,*)'All new knot vector Unew(i) for n = ', n
write(8,*)'Before Verify'
write(8,20)(i, Unew(i), i=0, mknot)
write(8,*)
call Verify(mdata, n, p, ukb, Unew, success)

if (success) then  ! Redistributed knot vector is OK.
  do i=0, mknot
    U(i) = Unew(i)
  end do
write(8,*)'New Knot Vector OK for n = ', n
end if

* If endpoint derivatives are specified, define new control
* pts P(1) and P(n-1)

if (offset .eq. 2) then
  P(1,1) = Q(1,0) + U(p+1)/p * D0(1)
  P(2,1) = Q(2,0) + U(p+1)/p * D0(2)
  P(1, n-1) = Q(1, mdata) - (1.0 - U(n))/p * Dm(1)
  P(2, n-1) = Q(2, mdata) - (1.0 - U(n))/p * Dm(2)
end if
else

write(8,*)'Totally new knot vector still has',
1 ' span with no ukb value.'
write(*,*)'Totally new knot vector still has',
1 ' span with no ukb value.'
write(13,*)'Totally new knot vector still has',
1 ' span with no ukb value.'
if (eptemp .eq. ep) then
  write(8,*)'Maximum tolerance specification ',
1 ' ep,' is too small'
write(13,*)'Maximum tolerance specification ',
1 ' ep,' is too small'
else

write(8,*)'RMS error spec is too small. Actually',
1                ' achieved rms of ',rms
write(13,*)'RMS error spec is too small. Actually',
1                ' achieved rms of ',rms
end if
close(13)
stop
end if
end do
write(8,*)'n, mknot ep = ',n,' ',mknot,' ',ep
write(8,*)'Control points P(1,j) & P(2,j) for n = ',n
write(8,30)(j,P(1,j),P(2,j),j=0,n)
write(8,*)
write(8,*)'Knot Vector U(i)'
write(8,20)(i,U(i),i=0,mknot)
close(8)

open(12,file='nurbs//run//'.dat')
write(12,30)p
write(12,30)n
write(12,30)mknot
do i = 0,mknot
   write(12,10)U(i)
end do
do i = 0, n
   write(12,10)P(1,i),P(2,i)
end do
close(12)
if (ansrms .eq. 'Y' .or. ansrms .eq.'y') then
   write(13,*)'RMS tolerance spec achieved.'
end if
write(13,*)'Knot vector redistributed for '
write(13,*)'following values of n. If redistribution'
write(13,*)'occured more than ',nredistmax, ' times,'
write(13,*)'only last ',nredistmax, ' times are reported.'
write(13,40)(nvalues(i),i=1,min(nredist,nredistmax))
write(13,*)'Knot vector was redistributed a total of ',nredist,
1     ' times.'
write(13,*)
write(13,*)'Final value of n: ',n
write(13,*)
if (offset .eq. 1) then
    D0(1)=p*(P(1,1)-P(1,0))/U(p+1)
    D0(2)=p*(P(2,1)-P(2,0))/U(p+1)
    Dm(1)=p*(P(1,n)-P(1,n-1))/(1.0-U(n))
    Dm(2)=p*(P(2,n)-P(2,n-1))/(1.0-U(n))
    D0L = sqrt(D0(1)*D0(1)+D0(2)*D0(2))
    DmL = sqrt(Dm(1)*Dm(1)+Dm(2)*Dm(2))
    phi0=atan2(D0(2),D0(1))*180./pi
    phim=atan2(Dm(2),Dm(1))*180./pi
    write(13,*)'Endpoint Derivative Information:'
    write(13,*)'Polar angle (degrees):'
    write(13,*)'  upper:  ',phi0
    write(13,*)'  lower:  ',phim
    write(13,*)'Lengths:  '
    write(13,*)'  upper:  ',D0L
    write(13,*)'  lower:  ',DmL
end if
write(13,*)'Maximum nominal deviation between NURBS curve'
write(13,*)'and ice data:  ',epmax
write(13,*)'Nominal rms deviation between NURBS curve'
write(13,*)'and ice data:  ',rms
close(13)
open(14,file='cukb'/run//'.dat')
write(14,60)(CA(1,i),CA(2,i),i=0,mdata)
mdata=m2datamax
du=1.0/dble(mdata)
u(0)=0.0
do i=1,mdata
    u(i)=u(i-1)+du
end do
call FindSpanA(mdata,n,p,u ,U,spanA)
call NCurveA( mdata,n,p,pmax,spanA,u,  
1     P, U,CA )
open(7,file='icecv'/run//'.dat')
do i = 0,mdata
    write(7,10)CA(1,i),CA(2,i)
end do
close(7)

stop
10 format(1x,3(e22.16,2x))
20 format(1x,i3,3x,f14.10)
integer function FindSpan(n,p,u,U)

* FindSpan calculates the span index of a curve parameter u via a
  * bisection routine

* Record of Revisions:
  *
  * Date            Programmer            Description of Change
  * ====            ============              =======================
  * 05/20/03        Loren H. Dill         Original code
  *

* Main Variables
  *
  * n     Largest index of control pnts vector
  * p     Degree of basis functions
  * u     curve parameter
  * U     knot vector
  *
* Reference:  Piegl & Tiller, p.68

implicit none

integer n,p,low,high,mid
double precision u, U(0:n+p+1)

if(u .eq. U(n+1)) then
    FindSpan = n
    return
end if
low=p;high=n+1;mid=(low+high)/2
do while((u .lt. U(mid)) .or. (u .ge. U(mid+1)))
if( u .lt. U(mid)) then
  high = mid
else
  low = mid
endif
mid = (low+high)/2
end do

FindSpan = mid

return
end

************************************************************************
subroutine FindSpanA(mdata,n,p,ukb,U,spanA)
************************************************************************
* FindSpanA calculates the span indices corresponding to an array
* of curve parameters ukb. Uses same algorithm as FindSpan, but
* modified for an array of curve parameter values.
************************************************************************
* Record of Revisions:
* *
* Date            Programer                Description of Change
* ====          ============              =======================
* 05/20/03       Loren H. Dill             Original code
* *
************************************************************************
* Main Variables
* *
* INPUT
* mdata     Largest index of data array ukb
* n          Largest index of control pnts vector
* p          Degree of basis functions
* ukb       1-D array of curve parameter values
* U          knot vector
* *
* OUTPUT
* spanA      1-D array of indices for ukb data
* *
************************************************************************
* Reference: Piegl & Tiller, p.68
implicit none

integer i,mdata,n,p,low,high,mid
integer spanA(0:mdata)
double precision ukb(0:mdata), U(0:n+p+1)

do i=0,mdata
  if(ukb(i) .eq. U(n+1)) then
    spanA(i) = n
  else
    low=p;high=n+1;mid=(low+high)/2
    do while((ukb(i) .lt. U(mid)) .or. (ukb(i) .ge. U(mid+1)))
      if( ukb(i) .lt. U(mid)) then
        high = mid
      else
        low = mid
      end if
      mid = (low+high)/2
    end do
    spanA(i) = mid
  end if
end do

return
end

************************************************************************
*                                                                      *
subroutine NBasis( i,mknot,p,u,U,N )
************************************************************************
*     NBasis calculates the p+1 non-zero basis functions within knot   *
*     span index i                                                     *
************************************************************************
*     Record of Revisions:
*
*     Date            Programer                Description of Change
*     ====          ============              =======================
*     05/20/03       Loren H. Dill             Original code
************************************************************************
* * INPUT Variables *
* i  knot span index of u *
* mknot largest index of U *
* p  degree of NURBS curve *
* u  value of curve parameter *
* U  knot vector of length m+1 *
* *
* OUTPUT Variable *
* N  Basis function array:  (N(i-p,p), ..., N(i,p)) *
* *
************************************************************************
* Reference:  Piegl & Tiller, p.70 *
* implicit none *

integer i,j,mknot,p,r
double precision u,U(0:mknot),N(0:p),left(p),right(p),saved,temp

N(0)=1.0
do j=1,p
  left(j) = u-U(i+1-j)
  right(j) = U(i+j)-u
  saved = 0.0
  do r=0,j-1
    temp = N(r)/(right(r+1)+left(j-r))
    N(r) = saved + right(r+1) * temp
    saved = left(j-r) * temp
  end do
  N(j) = saved
end do
return
end

************************************************************************
* subroutine NBasisA( mdata,mknot, p,pmax,spanA,ukb,U,NA ) *
************************************************************************
* NBasisA calculates the p+1 non-zero basis functions corresponding* *
* to each element in the 1-D array of parameter values ukb         *
* *
************************************************************************
* Record of Revisions:
Date            Programer                Description of Change
====          ============              =======================
05/2003       Loren H. Dill             Original code

************************************************************************

INPUT Variables                                                  *
*  mdata    largest index of parameter array ukb                    *
*  mknot    largest index of U                                      *
*  p        degree of NURBS curve                                   *
*  pmax     highest possible degree of NURBS curve                  *
*  spanA    1-D array of span indices corresponding to ukb          *
*  ukb      1-D array of curve parameter values                     *
*  U        knot vector of length mknot+1                           *
*                                                                         *
OUTPUT Variable                                                  *
*  NA     Basis function 2-D array. Column i contains to the         *
*          p+1 non-zero basis functions for given ukb(i)             *
*                                                                         *
************************************************************************

Reference:  Piegl & Tiller, p.70

implicit none

integer i,j,mdata,mknot!,offset
integer p,pmax,r, spanA(0:mdata)
double precision ukb(0:mdata),U(0:mknot),NA(0:pmax,0:mdata)
double precision left(p),right(p),saved,temp

do i=0, mdata
   NA(0,i)=1.0
   do j=1,p
      left(j) = ukb(i)-U(spanA(i)+1-j)
      right(j) = U(spanA(i)+j)-ukb(i)
      saved = 0.0
      do r=0, j-1
         temp = NA(r,i)/(right(r+1)+left(j-r))
         NA(r,i) = saved + right(r+1) * temp
         saved = left(j-r) * temp
      end do
      NA(j,i) = saved
   end do
end do
end do
return
end

************************************************************************
subroutine Cparam ( mdata,Q, ukb )
************************************************************************
*
* Computes a normalized parametric variable based upon arc length *
* for a 2D Cartesian curve. *
*
************************************************************************
* Record of Revisions:
*
* Date            Programer                Description of Change
* ====          ===========              =======================
* 4/14/03       Loren H. Dill             Original code
*
************************************************************************
* INPUT VARIABLES
* mdata  mdata + 1 = Number of points defining curve
* Q    (x,y) Cartesian coordinates for curve (2-D array)
*
* OUTPUT VARIABLE
* ukb  Parametric variable for curve   (1-D array)
*
************************************************************************

implicit none

integer mdata, k

double precision ukb(0:mdata),Q(2,0:mdata)

* Compute the parametric variable ukb based on arc length and
* normalize to unity

ukb(0) = 0.0
do k = 1, mdata
    ukb(k) = ukb(k-1) +
    1        sqrt( ( Q(1,k) - Q(1,k-1) )*( Q(1,k) - Q(1,k-1) )
    + ( Q(2,k) - Q(2,k-1) )*( Q(2,k) - Q(2,k-1) ) )
end do
do k = 1, mdata - 1
    ukb(k) = ukb(k)/ukb(mdata)
end do

ukb(mdata) = 1.0

return
end

************************************************************************
subroutine Knotvec( mdata, n, p, ukb, U )
************************************************************************
*     Knotvec calculates a knot vector for global approximation based   *
*     upon equations (9.68) and (9.69) of Piegl and Tiller (1997).    *
*     The routine is said to guarantee that every knot span contains  *
*     at least one ukb point, resulting in a matrix NTNB that is       *
*     positive definite and well-conditioned.                         *
************************************************************************
*     Record of Revisions:
*
*     Date           Programer                Description of Change
*     ====          ============              =======================
*     05/20/03       Loren H. Dill             Original code
*
************************************************************************
*     Main variables                                                  *
*     INPUT                                                           *
*     mdata    m+1 is number of data points                            *
*     n        n+1 is number of control points in approximation        *
*     p        degree of urbs curve approximation                      *
*     ukb      array of parameter values corresponding to data points *
*     OUTPUT                                                          *
*     U        knot vector of length n + p + 2                          *
************************************************************************

implicit none
integer i,j,mdata,n,p
double precision d, alpha, ukb(0:mdata),U(0:n+p+1)
do i = 0,p
    U(i)= 0.0
    U(n+p+1-i)= 1.0
end do

d = dble( mdata+1 )/dble( n-p+1 )
do j = 1, n-p
    i = int( j * d )
    alpha = dble( j )* d - dble( i )
    U( p + j ) = ( 1.0 - alpha )* ukb( i - 1 ) +
    1        alpha * ukb( i )
end doeturn
end

************************************************************************

subroutine NCurve( n,p,u, P, U,C )
************************************************************************

* Subroutine NCurve calculates the (x,y) point corresponding to a scalar parameter u of a nonrational NURBS 2-D curve.

************************************************************************

* Record of Revisions:

* Date            Programer                Description of Change
* ====          ============              =======================
* 05/20/03       Loren H. Dill             Original code

************************************************************************

* INPUT VARIABLES

* n        n+1 is number of control points
* p        degree of B-spline
* u        curve parameter
* P        control points, 2-D array in column format
* U        knot vector

************************************************************************

* OUTPUT

*
* C calculated (x,y) point on NURBS curve, 1-D array * *
* *****************************************************************

implicit none

integer i,j,k, mknot, n, p
integer FindSpan

double precision u, P(2,0:n ), U( 0:n+p+1 )
double precision N(0:p)
double precision C(2)

mknot=n+p+1
i = FindSpan( n,p, u, U )
call NBasis( i, mknot, p, u, U, N)
C(1) = 0.0
C(2) = 0.0
j = i - p

* Sum the nonzero terms only
do k = 0, p
   C(1) = C(1) + N( k ) * P( 1, j + k )
   C(2) = C(2) + N( k ) * P( 2, j + k )
end do

return
end

********************************************************************
* subroutine NCurveA( mdata,n,p,pmax,spanA,u, P, U,CA ) *
* ********************************************************************
* Subroutine NCurveA calculates all (x,y) points corresponding to *
* the 1D array u for a nonrational NURBS 2-D curve. *
* ********************************************************************
* Record of Revisions:
* *
* Date Programer Description of Change
**INPUT VARIABLES**

* mdata  largest index of parameter vector u
* n      largest index of control points
* p      degree of NURBS curve
* pmax   maximum degree of NURBS curve
* spanA  1-D array of span indices corresponding to ukb
* u      curve parameter, 1-D array
* P      control points, 2-D array in column format
* U      knot vector

**OUTPUT**

* CA     calculated array of (x,y) points on NURBS curve,
         2-D array

implicit none

integer i,j,k, mdata, mknot, n, p, pmax
integer spanA(0:mdata)
double precision u(0:mdata), P(2,0:n ), U( 0:n+p+1 )
double precision NA(0:pmax,0:mdata)
double precision CA(2,0:mdata)
call NBasisA( mdata, mknot,p, pmax, spanA, u, U, NA)
do i=0,mdata
   CA(1,i) = 0.0
   CA(2,i) = 0.0
   j = spanA(i) - p
   do k = 0, p
      CA(1,i) = CA(1,i) + NA(k, i ) * P( 1, j + k )
      CA(2,i) = CA(2,i) + NA(k, i ) * P( 2, j + k )
   end do
end do

return
end

************************************************************************
*
subroutine Rkarray(mdata,n,offset,p,pmax,spanA,Q,ukb,P,U,NA,Rk)
*
************************************************************************
*
Rkarray calculates a 2-D array that contains the m-1 values for
* the x- and y- components for eqn 9.63 in Piegl & Tiller if end
* derivatives are free (offset = 1). If end derivatives are fixed
* (offset = 2), the algorithm is appropriately modified.
*
************************************************************************
*
Record of Revisions:
*
* Date            Programer                Description of Change
* ====          ============              =======================
* 05/20/03       Loren H. Dill             Original code
*
************************************************************************
*
INPUT VARIABLES
*
*
mdata    largest index of parameter vector u
*n        largest index of control points
*offset   indicates whether endpoint derivatives are free
*        (1) or fixed (2)
*p        degree of NURBS curve
*pmax     maximum degree of NURBS curve
*spanA    1-D array of span indices corresponding to ukb
*Q        (x,y) prescribed data, 2-D array
*ukb      curve parameter, 1-D array
*P        control points, 2-D array in column format
*U        knot vector
*NA       array of nonzero basis functions corresponding to ukb
*
OUTPUT
*
Rk 2-D array containing m-1 values. Required for calculation of right-hand side of matrix equation.

 implicit none

 integer k,mdata, n, offset, p, pmax, spanA(0:mdata)

double precision Q(2,0:mdata),ukb(0:mdata),U(0:n+p+1)
double precision NA(0:pmax,0:mdata),Rk(2,0:mdata)
double precision P(2,0:n)

* Calculate the Rk vectors.
do k=1, mdata-1
   Rk(1,k)=Q(1,k)
   Rk(2,k)=Q(2,k)
   if (spanA(k) .eq. p) then
     Rk(1,k)= Rk(1,k) - NA(0,k) * P(1,0)
     Rk(2,k)= Rk(2,k) - NA(0,k) * P(2,0)
   end if
   if (offset .eq. 2) then
     Rk(1,k)=Rk(1,k)-NA(1,k)*P(1,1)
     Rk(2,k)=Rk(2,k)-NA(1,k)*P(2,1)
   end if
   if (offset .eq. 2 .and. spanA(k) .eq. p+1) then
     Rk(1,k)=Rk(1,k)-NA(0,k)*P(1,1)
     Rk(2,k)=Rk(2,k)-NA(0,k)*P(2,1)
   end if
   if( offset .eq. 2 .and. spanA(k) .eq. n-1) then
     Rk(1,k)=Rk(1,k)-NA(p,k)*P(1,n-1)
     Rk(2,k)=Rk(2,k)-NA(p,k)*P(2,n-1)
   end if
   if( spanA(k) .eq. n )then
     if (offset .eq. 2) then
       Rk(1,k)=Rk(1,k)-NA(p-1,k)*P(1,n-1)
       Rk(2,k)=Rk(2,k)-NA(p-1,k)*P(2,n-1)
     end if
     Rk(1,k)= Rk(1,k) - NA(p,k)*P(1,n)
     Rk(2,k)= Rk(2,k) - NA(p,k)*P(2,n)
   end if
end do
return
end

************************************************************************
subroutine RightHandSide(mdata,n,nmax,offset,p,pmax,spanA,NA,Rk,R)
************************************************************************

* RightHandSide calculates the R array, a n-1 X 2 array, which
* corresponds to the right side of eqn. 9.65 of Piegl & Tiller if
* end derivatives are free (offset = 1). If end derivatives are
* fixed (offset = 2), the algorithm is appropriately modified.
*
************************************************************************

* Record of Revisions:
*
* Date            Programer                Description of Change
* ====          ============              =======================
* 05/20/03       Loren H. Dill             Original code
*
************************************************************************

* INPUT VARIABLES
*
* mdata    largest index of parameter vector u
* n        largest index of control points
* nmax     largest value of n permitted
* offset   flag indicating whether endpoint derivatives are free
*           (1) or fixed (2)
* p        degree of NURBS curve
* pmax     maximum degree of NURBS curve
* spanA    1-D array of span indices corresponding to ukb
* NA       array of nonzero basis functions corresponding to ukb
* Rk       array corresponding to eqn. 9.65 of Piegl & Tiller
*
************************************************************************

* OUTPUT
*
* R        Right-hand side of linear matrix equation
*
************************************************************************

implicit none
integer j,k,mdata, n,nmax,offset,p,pmax,row, spanA(0:mdata)
double precision NA(0:pmax,0:mdata),Rk(2,0:mdata),R(nmax-1,2)
do j=offset,n-offset
   R(j+1-offset,1)=0.0
   R(j+1-offset,2)=0.0
   do k=1,mdata-1
      row = j-(spanA(k)-p)
      if ( (row .ge. 0) .and. (row .le. p) ) then
         R(j+1-offset,1) = R(j+1-offset,1) +
         1              NA(row,k) * Rk (1, k)
         R(j+1-offset,2) =  R(j+1-offset,2) +
         1              NA(row,k) * Rk (2, k)
      end if
   end do
   return
end

************************************************************************
subroutine ABMatrix (mdata,n,offset,p,pmax,spanA,NA,NTNB)
************************************************************************
*                                                                      *
*     AMatrix computes the NTN matrix of Piegl and Tiller, p. 411, and *
*     stores the matrix in upper band form suitable for Lapack         *
*     subroutine dpbsv. Matrox NTN has p superdiagonals.              *
*                                                                      *
************************************************************************
*     Record of Revisions:                                            *
*
*     Date            Programer                Description of Change    *
*     ====          ============              ======================= *
*     05/20/03       Loren H. Dill             Original code            *
*
************************************************************************
*                                                                      *
*     INPUT VARIABLES                                                  *
*                                                                      *
*     mdata       largest index of input data                          *
*     n           largest index of control vectors                      *
*     offset      flag for derivatives specification: 1 if not, 2 if    *
*                 specified                                            *
*     p           current degree of NURBS curve                         *
*     pmax        parameter, max degree of NURBS curve                  *
* spanA  1-array of span indices corresponding input data  *
* NA    a 2-D array containing the p+1 non-zero basis   *
*        functions corresponding to each data point.     *
*        *
************************************************************************
* OUTPUT  *
*        *
*      NTNB      The NTN (Transpose(N) * N) array using band storage *
*        *
************************************************************************

implicit none

integer i,j,k,mdata,n,offset,p,pmax
integer kd, row,rowi,rowj,spanA(0:mdata)
double precision NA(0:pmax,0:mdata)
double precision NTNB(pmax+1,n-1)

kd=p
do i=offset,n-offset
   do j=i,n-offset
      if( max(1,j+1-offset-kd) .le. i+1-offset) then
         row=kd+1+i-j
         NTNB(row,j+1-offset)=0.0
         do k=1,mdata-1
            rowi = i-(spanA(k)-p)
            rowj = j-(spanA(k)-p)
            if ( (rowi .ge. 0 ) .and. (rowi .le. p) .and. (rowj .ge. 0) .and. (rowj .le. p) ) then
               NTNB(row,j+1-offset) = NTNB(row,j+1-offset) +
               NA(rowj,k) * NA(rowi,k)
            end if
         end do
      end if
   end do
end do
return
end

************************************************************************
subroutine SpanTest(ep,mdata,n,p,Q,ukb,P,U,nonspan,i)
************************************************************************
SpanTest returns in variable nonspan the indices of spans in U that have not converged. In order for a span j to converge, the distances $|C(ukb(k)) - Q(k)|$ must be less than or equal to $ep$, the distance criterion, for all $ukb(k)$ located in span j. Variable i gives the number of non-converging spans in knot vector U. Here, $Q(k) = (x(k), y(k))$ is one of the prescribed iced-airfoil data points.

INPUT VARIABLES

$ep$ nominal tolerance between NURBS curve and each data point

$mdata$ largest index of input data

$n$ largest (current) index of control vectors

$p$ current degree of NURBS curve

$Q$ $(x,y)$ the prescribed data, 2D array

$ukb$ parameter array corresponding to data

$P$ control points of NURBS curve

$U$ knot vector of NURBS curve

OUTPUT

nonspan 1-D array containing indices of non-converging spans

i number of non-converging spans

implicit none

integer p,n,mdata,mknot, nonspan(n+1-p)
integer i,j,k,spanA(0:mdata),offset
double precision P(2,0:n),U(0:n+p+1),ukb(0:mdata)
double precision Q(2,0:mdata)
double precision ep, C(2), epsq,distancesq,diff(2)

mknot = n+p+1
offset = 1
epsq=ep*ep  ! Actually compare the squared distances

call FindSpanA(mdata,n,p,ukb,U,spanA)
write(8,*)
k=1
i=0
do while (k .le. mdata-1)
call NCurve( n,p,ukb(k), P, U,C )
diff(1)= C(1)-Q(1,k)
diff(2)= C(2)-Q(2,k)
distancesq=diff(1)*diff(1)+diff(2)*diff(2)
if (distancesq .gt. epsq) then
  write(8,*)'non-converging span, spanA(k) = ',spanA(k)
i = i+1
nonspan(i)= spanA(k)
if(spanA(k) .eq. n) then
  k= mdata
else
  j=k+1
  do while (spanA(k) .eq. spanA(j) )
    j = j + 1
  end do
  k = j-1
end if
end if
k=k+1
end do
return
end

************************************************************************
subroutine Deviations(mdata,n,p,pmax,Q,ukb,P,U,epmax,rms,CA)
************************************************************************

* Deviations returns in variable epmax the maximum nominal distance
* between the nurbs curve and the discrete ice data by evaluating
* all distances |C(ukb(k))-Q(k)| Here, Q(k)= ( x(k),y(k) ) is one
* of the prescribed iced-airfoil data points. The routine also

NASA/CR—2004-213071 58
* returns curve values \( CA = C(\text{ukb}) \) for each parameter value \( \text{ukb} \).
* Deviations also calculates and returns the root mean squared
* distance (rms) from the data to the curve. This is again a
* nominal value in the sense that the curve passes somewhat closer
* to the each data point than is represented by \(|C(\text{ukb}(k))-Q(k)|\).
*
************************************************************************************
* Record of Revisions:
* *
* Date            Programer                Description of Change
* ====          ============              =======================
* 5/20/03       Loren H. Dill             Original code
* 5/22/03       L.H. Dill                 Added rms capability
************************************************************************************
* INPUT VARIABLES                                                  *
*                                                                      *
* mdata       largest index of input data                          *
* n           largest (current) index of control vectors           *
* p           current degree of NURBS curve                        *
* pmax        parameter, max degree of NURBS curve                 *
* Q           (x,y), the prescribed data, 2D array                *
* ukb         parameter array corresponding to data                *
* P           control points of NURBS curve                        *
* U           knot vector of NURBS curve                           *
************************************************************************************
* OUTPUT                                                           *
*                                                                      *
* epmax       maximum nominal deviation between NURBS curve and     *
*             prescribed data                                        *
* rms         root mean square deviation between the curve and      *
*             prescribed data                                        *
* CA          array containing all the x-y values along the NURBS    *
*             curve corresponding to curve parameter values \( \text{ukb} \)  *
************************************************************************************

implicit none

integer p,n,mdata,mknot,pmax
integer k,spanA(0:mdata)
double precision P(2,0:n),U(0:n+p+1),ukb(0:mdata)
double precision Q(2,0:mdata)
double precision epmax, epmax2,rms,sum,CA(2,0:mdata)
double precision dist2,diff(2)

mknot = n+p+1
epmax2=0.0
sum=0.0

call FindSpanA(mdata,n,p,ukb,U,spanA)
call NCurveA( mdata,n,p,pmax,spanA,ukb, P, U,CA )
do k=1,mdata-1
   diff(1)= CA(1,k)-Q(1,k)
diff(2)= CA(2,k)-Q(2,k)
dist2=diff(1)*diff(1)+diff(2)*diff(2)
   sum=sum+dist2
   epmax2=dmax1(epmax2,dist2)
end do
epmax = sqrt(epmax2)
rms = sqrt( sum/dble(mdata-1) )
return
end

************************************************************************
subroutine NewU(mdata,n,nonspan,inonspan,p,ukb,U,Unew)
************************************************************************
*
*     NewU adds a knot at the midpoint of every span that contains
*     data points that have not met the distance criterion.
*
************************************************************************
*
************************************************************************
*
*     Record of Revisions:
*
*     Date            Programer                Description of Change
*     ====          ============              =======================
*     05/20/03       Loren H. Dill             Original code
*
************************************************************************
*
************************************************************************
*
*     INPUT VARIABLES
*
*     mdata    Largest index of ukb, the parameter array
*     n        On input, largest index of sum in current NURBS curve
*     nonspan  1-D array containing the span indexes of current knot
*     vector that have not converged.
* inonspan number of elements in nonspan
* p degree of NURBS curve
* ukb the parameter array, of length mdata + 1
* U Original knot vector
************************************************************************
* OUTPUT VARIABLE
*
* n Largest index of sum in new NURBS curve
* Unew New knot vector
************************************************************************

implicit none
integer i, inonspan, j, mdata, mknot, n, nonspan(inonspan+1), p

double precision ukb(0:mdata), U(0:n+p+1)
double precision Unew(0:n+p+1+inonspan), unew(inonspan)

* Set nonspan(inonspan+1) to mknot for termination
* condition

mknot=n+p+1
nonspan(inonspan+1)=mknot

do i=0, p
   Unew(i)=0.0
   Unew(mknot+inonspan-i)=1.0
end do

do i=1, inonspan
   unew(i) = 0.5*( U( nonspan(i) ) + U( nonspan(i) + 1 ) )
end do

j = 1

do i = p+1, mknot+inonspan-p-1
   if(i .le. nonspan(j)+ j -1) then
      Unew(i)= U(i - j + 1 )
   else
      Unew(i)= unew(j)
      j=j+1
   end if
end do

n = n + inonspan
**subroutine Verify(mdata,n,p,ukb,U,success)**

* Verify checks knot vector U to insure at least one parameter value is located within each knot span. If it does, output variable success reports True. If not, success reports False

**Record of Revisions:**

* Date            Programer                Description of Change
* ====          ============              =======================
* 05/20/03       Loren H. Dill             Original code

**INPUT VARIABLES**

* mdata    Largest index of ukb, the parameter array
* n        Largest index of control points
* p        degree of NURBS curve
* ukb      the parameter array, of length mdata + 1
* U        Original knot vector

**OUTPUT VARIABLE**

* success   Logical variable indicating if .true. that a element of ukb is located within each span of new knot vector

**implicit none**

type (integer, mdata, n, p, spanA(0:mdata))

type (double precision, ukb(0:mdata), U(0:n+p+1))

type (logical, success)

success = .true.

call FindSpanA(mdata, n, p, ukb, U, spanA)

do i=1, mdata-1
if (spanA(i+1)-spanA(i) .gt. 1 ) then
  success = .false.
  write(8,*)'No ukb element in span ',spanA(i)+1
  return
end if
end do

return
end
**Title and Subtitle:**
Representation of Ice Geometry by Parametric Functions: Construction of Approximating NURBS Curves and Quantification of Ice Roughness—Year 1: Approximating NURBS Curves

**Author(s):**
Loren H. Dill

**Performing Organization Name(s) and Address(es):**
University of Akron
302 Buchtel Mall
Akron, Ohio 44325

**Performing Organization Report Number:**
NASA CR—2004-213071

**Abstract:**
Software was developed to construct approximating NURBS curves for iced airfoil geometries. Users specify a tolerance that determines the extent to which the approximating curve follows the rough ice. The user can therefore smooth the ice geometry in a controlled manner, thereby enabling the generation of grids suitable for numerical aerodynamic simulations. Ultimately, this ability to smooth the ice geometry will permit studies of the effects of smoothing upon the aerodynamics of iced airfoils. The software was applied to several different types of iced airfoil data collected in the Icing Research Tunnel at NASA Glenn Research Center, and in all cases was found to efficiently generate suitable approximating NURBS curves. This method is an improvement over the current "control point formulation" of Smaggice (v.1.2). In this report, we present the relevant theory of approximating NURBS curves and discuss typical results of the software.

**Subject Terms:**
Aircraft icing; Computer software

**Security Classification of Report:**
Unclassified

**Number of Pages:**
69

**Price Code:**
E-14547

**DISTRIBUTION/AVAILABILITY STATEMENT:**
Unclassified - Unlimited
Subject Categories: 02 and 61

**Distribution:** Nonstandard

Available electronically at [http://gltrs.grc.nasa.gov](http://gltrs.grc.nasa.gov)

This publication is available from the NASA Center for AeroSpace Information, 301–621–0390.