THE CITY UNIVERSITY

Department of Mathematics

NUMERICAL METHODS FOR THE INTERPOLATION AND APPROXIMATION OF DATA BY SPLINE FUNCTIONS

by

M G COX, BSc, AFIMA

Thesis submitted for the Degree of Doctor of Philosophy to the City University, St John Street, London

July 1975

To my wife Rosalie who suffered
my moods and had only a part-time
husband during the preparation
of this work.

ABSTRACT

NUMERICAL METHODS FOR THE INTERPOLATION AND APPROXIMATION OF DATA BY SPLINE FUNCTIONS

It is often important in practice to obtain approximate representations of physical data by relatively simple mathematical functions. The approximating functions are usually required to meet certain criteria relating to accuracy and smoothness. In the past, polynomials have frequently been used for this task, but it has long been recognised that there are many types of data set for which polynomial approximations are unsatisfactory in that a very high degree may be required to achieve the required accuracy. Moreover, even if such a polynomial can be computed, it frequently tends to exhibit spurious oscillations not present in the data itself.

In an attempt to overcome these difficulties attention has turned in recent years to the use of piecewise polynomials or spline functions. A spline function, or simply a spline, is composed of a set of polynomial arcs, usually of low degree, joined end to end in such a way as to form a smooth function. Splines tend to have greater flexibility than polynomials in the approximation of physical data and much attention has been devoted in the last decade to the theory of splines. The development of robust numerical methods for computing with splines has, however, lagged somewhat behind the theory. The main objective of this work is the construction and analysis of such methods. In order to obtain efficient and stable methods a representation of splines that is well-conditioned and that results in fast computational schemes is required. Representations in terms of B-splines prove to be eminently suitable and

accordingly we study B-splines in some detail and give various algorithms for calculations in which they are involved.

When B-splines are used as a basis for interpolation or least-squares data fitting the resulting linear algebraic systems to be solved for the spline coefficients have a special structure. Stable numerical methods that exploit this structure to the full are presented.

Our algorithms are used to obtain spline approximations to a variety of data sets drawn from practical applications. Their performance on these problems illustrates the power of splines over more conventional approximating functions.

ACKNOWLEDGEMENTS

This thesis is based on work carried out between 1969 and 1975 while the author was employed at the National Physical Laboratory and registered at the City University as a part-time student for the Degree of Doctor of Philosophy.

I am indebted to my internal supervisor Professor V E Price and my external supervisor Mr J G Hayes whose guidance and encouragement enabled me to complete this work.

I also wish to acknowledge many fruitful discussions with Mr E L Albasiny, Mr G T Anthony, Professor C W Clenshaw, Professor W M Gentleman, Dr J H Wilkinson and my supervisors on various aspects of linear algebra, error analysis, spline functions, data approximation and numerical methods in general.

Finally I thank Mrs Joan Mann for her accurate typing of the manuscript.

CONTENTS

INTRODUCTION

Many computations with polynomials have been systematized in the last two decades by the use of Chebyshev series. Expressing the approximate solution to a wide variety of problems as a polynomial in its Chebyshev-series form has often proved extremely beneficial. One of the main benefits of this approach stems from the fact that in many applications Chebyshev polynomials form an extremely well-conditioned basis for the class of polynomial functions. Examples of the application of Chebyshev series abound: in the fields of function and data approximation, interpolation, quadrature, differential equations and integral equations are to be found many interesting and practical results.

Polynomial splines are a generalization of polynomials in that a spline of order n includes, as special cases, all polynomials of degree less than n. We treat in some detail in this work what we consider to be a spline counterpart to the Chebyshev polynomials, viz the B-splines. The B-splines of a given order defined upon a prescribed set of knots form for many purposes a well-conditioned basis for the class of splines of that order with the same knots. Moreover, the B-splines too have application to many problems in numerical analysis, including those referred to above. Considered here are some of the properties of B-splines, many of which are new, and ways in which these properties can be utilized to advantage in problems of interpolation and approximation of discrete data.

The theory of splines has made significant advances, particularly in the last decade (see the bibliography by van Rooij and Schurer, 1973), after a relatively quiet period following the pioneering work of Schoenberg (1946). However, the development of reliable and efficient algorithms for spline computations has lagged significantly behind the theoretical development. Accordingly, in order to swing the balance a fraction in favour of the practical side, our approach is predominantly algorithmic. We concentrate upon the development of what we believe are fundamental and useful algorithms for computing with splines expressed in their B-spline form. Many of these algorithms are supported by practical results as well as by rigorous error analyses, the latter often indicating the degree of stability of the algorithms.

Of the ten chapters in this work the first five constitute "backbone" chapters upon which the remaining five depend.

Chapter 1 is primarily expository and discusses floating-point arithmetic and basic concepts relating to the error analysis of computational processes. Our approach is essentially that propounded by Wilkinson (see, in particular, Wilkinson, 1965; Peters and Wilkinson, 1971). We also describe the step-by-step manner in which our algorithms are presented and what we understand by the numerical stability of a computational process.

The first part of Chapter 2 is also mainly expository in that methods for the numerical solution of linear algebraic systems in both the determined and over-determined cases are surveyed. The work of Wilkinson (particularly Wilkinson, 1965; Peters and Wilkinson, 1970) has again strongly influenced our treatment. We then discuss the use of both classical and modern forms of plane (Givens) rotations (Gentleman, 1973; Hammarling, 1974) for solving over-determined (least-squares) systems and give reasons why we believe that plane rotations have advantages over other methods such as Householder transformations and modified Gram-Schmidt. These reasons are reinforced by a comparison, based on the timing analysis of Wichmann (1973), of the relative efficiencies of methods for least-squares problems. The second part of Chapter 2 contains detailed descriptions of some new algorithms for the solution of the structured (stepped-banded) linear systems that arise in spline interpolation and approximation problems. For the fully-determined square case (interpolation) we give algorithms based upon Gaussian elimination (GE) and elementary transformations, and for the rectangular case algorithms based upon classical and modern forms of plane rotation (PR). The GE algorithm can be considered as a generalization of the algorithm of Martin and Wilkinson (1967) for banded systems, and the PR algorithm as a specialization of the Givens algorithm of Gentleman (1973). Our algorithms prove to have advantages in terms of simplicity, speed and storage over those based on Householder transformations for stepped-banded linear systems given by Reid (1967) and Lawson and Hanson (1974). Finally, it is shown that the powerful singular value decomposition may be adapted to analyse stepped-banded systems efficiently.

In Chapter 3 polynomial splines and their properties are discussed and a particular form of fundamental spline, the B-spline, is introduced. A new identity (Cox, 1972) relating B-splines of consecutive degrees is then established. This identity, which expresses the value of a B-spline of order n as a convex combination of two B-splines of order n-1, and which proves fundamental to our work, was discovered simultaneously in the United States by de Boor (1972). We give algorithms based upon the conventional method employing divided differences and upon convex combinations for evaluating B-splines. Detailed error analyses and test computations are used to demonstrate conclusively that algorithms based upon the use of convex combinations are unconditionally stable for arbitrary (even multiple) knots, whereas algorithms employing divided differences may give extremely poor results.

In Chapter 4 a recurrance relation due to de Boor (1972) for the derivatives of B-splines is established. A new relation of this type is then obtained that proves to be an extension of the fundamental identity discovered in Chapter 3. Two results that prove to be of considerable use in subsequent chapters are then established: the values of all B-spline derivatives at the ends of the range, as well as certain derivatives at the knots, can all be computed in an unconditionally stable manner. A class of algorithms due to Butterfield (1975) for B-spline derivatives in the general case is then outlined. Finally, some results relating to the definite and indefinite integration of B-splines are given; these results all appear apparently for the first time, with the exception of one due to Butterfield (1975), which is a further generalization of the identity of Chapter 3, and one discovered independently by Gaffney (1974).

Chapter 5 is concerned with various computations arising from the representation of splines and polynomials in terms of B-splines. We present a particularly useful result due to de Boor (1972) which expresses a linear combination of B-splines in terms of B-splines of lower order with certain polynomial coefficients. This result is then used to establish a new proof that the B-splines form a linearly independent set of basis functions in terms of which an arbitrary spline $s(x)$ can be expressed, and to establish local lower and upper bounds for $s(x)$ in terms of its B-spline coefficients. Two schemes proposed by de Boor (1972) for the evaluation of $s(x)$ are described and, for the first time, error analyses of these schemes, which demonstrate their unconditional stability, already observed empirically by de Boor, are given. The problem of representing powers of x in terms of B-splines is then addressed and new algorithms for this problem are presented and detailed error analyses carried out. Methods for representing in their B-spline form the derivatives and integrals of $s(x)$ are then considered.

Chapter 6 is the first of three "applications" chapters and discusses the interpolation of a data set by splines of arbitrary order with arbitrary knot positions. A new algorithm, together with a detailed error analysis, is presented for this problem. Schumaker (1969) has spoken of the need for such an algorithm. In particular, it is shown that if B-splines are evaluated as recommended and if one of the algorithms proposed for solving stepped-banded systems is employed, the computed spline is the exact interpolant of a neighbouring data set. Choices for the exterior knots (required in order to define a full set of B-spline basis functions) and the interior knots are discussed; in particular the dependence of a certain condition number upon the positions of these knots is investigated using the singular value decomposition (SVD). Some informative numerical tests are carried out and a practical problem is solved.

Chapter 7 is the counterpart of Chapter 6 in the case where a least-squares approximation rather than an interpolating function is required. A new algorithm for testing whether a unique spline approximant exists in any given case is presented. For the least-squares spline-fitting problem itself an algorithm for splines of arbitrary order with arbitrary knot positions is proposed. This algorithm again utilizes the convex-combinations scheme and the methods for stepped-banded systems and is a generalisation of that given by Cox and Hayes (1973) for cubic splines. An error analysis of this algorithm is given and, with the aid of the SVD, an extremely encouraging conclusion is made relating to its stability. The important case of cubic splines is discussed and the question of knot placement is addressed. As well as a test example, a number of spline fits to real data sets are presented.

Chapter 8 concentrates on the type of problem where more information than that contained solely within the data set itself is prescribed. It is shown that some important types of continuous constraints upon the approximating spline may be enforced by imposing upon the spline a finite number of point constraints. A new representation of cubic splines is then used, in conjunction with an extension to algorithms due to Barrodale and Young (1966) for L_1- and L_∞ -approximation, for spline fitting subject to convexity and concavity constraints. Practical examples are given to demonstrate the usefulness of the approach.

In Chapter 9 the incorporation of linear equality constraints in spline approximation problems is discussed. In particular, it is shown that boundary conditions may be incorporated readily by a simple modification to the basis. For more general constraints, algorithms for linear least-squares problems with linear equality constraints are discussed.

Finally, Chapter 10 discusses briefly the extension of some of the methods of the earlier chapters to more than one independent variable. The interpolation and least-squares approximation to data given at all vertices of a rectangular mesh by a tensor product of univariate functions is first discussed. The case where the univariate functions are B-splines is then treated. The general problem of the least-squares spline approximation of arbitrary multivariate data, for which an algorithm has been given in the cubic case by Hayes and Halliday (1974), is then examined.

CHAPTER 1

FLOATING-POINT ARITHMETIC AND ERROR ANALYSIS

This chapter is one of three "backbone" chapters to this work; it serves as an introduction to floating-point arithmetic, error analysis, algorithms and numerical stability. In Section 1.1 we summarize the rudiments of floating-point arithmetic, adhering closely to the concepts developed by Wilkinson. In particular, we detail those aspects of floating-point arithmetic of which we shall make considerable usein subsequent chapters, where we analyze a number of computational processes relevant to spline approximation. In Section 1.2 we illustrate the type of error analysis we shall be carrying out by examining some simple formulae for linear transformations and, from the results of our analyses, make a conjecture relating to the analysis of more general processes. We also discuss running error analysis and the derivation of a posteriori and a priori error bounds. In Section 1.3 we give a brief discussion of algorithms and what we understand by numerical stability. We also outline the way in which we shall present algorithmic descriptions of our computational processes.

1.1 Floating-point arithmetic

Many of the numerical methods described in the following chapters will be analyzed in terms of their implementation in standard binary floating-point arithmetic. In this respect we shall follow closely the approach of Wilkinson (1963, 1965).

A number x is termed a standard binary floating-point number if it can be represented by an ordered pair (a,b) such that $x = a2^b$. Here b, the exponent, is an integer, positive or negative, usually restricted to the range $-2^c \leq b < 2^c$, where c is an integer, typically in the range 7 to 10;

a, the <u>mantissa</u>, is a binary number, usually satisfying $\frac{1}{2} \leq |a| < 1$, with no more than t binary digits. Typical values of t lie in the range 16 to 48. The value of 2^{-t} is termed the <u>relative machine precision</u>. The number zero is represented in the non-standard form a = b = 0.

A relation of the form

$$y = fl(x_1 * x_2 * x_3 * \ldots * x_n), \tag{1.1.1}$$

where each * denotes any one of the arithmetic operations +, −, × or ÷, implies that $x_1, x_2, \ldots, x_n$ and y are standard binary floating-point numbers (or zero), and that y is the result of performing the appropriate floating-point operations. The multiplication sign will frequently be omitted; thus x_1x_2 implies $x_1 \times x_2$. The division sign (÷) will frequently be replaced by slash (/) or a horizontal line, in the usual way. Parentheses on the right-hand side of (1.1.1) are often necessary to remove ambiguity or to emphasise the order of the computation. Otherwise the sequence of floating-point operations is assumed to take place from left to right, with the usual rules of precedence of × and ÷ over + and −. Thus, for example, $y = fl(x_1 \times x_2 \div x_3)$ implies (i) $y_1 = fl(x_1 \times x_2)$, (ii) $y = fl(y_1 \div x_3)$; $y = fl\left(\frac{x_1x_2 + x_3x_4}{x_5 - x_6}\right)$ implies (i) $y_1 = fl(x_1x_2)$, (ii) $y_2 = fl(x_3x_4)$, (iii) $y_3 = fl(y_1+y_2)$, (iv) $y_4 = fl(x_5-x_6)$, (v) $y = fl(y_3/y_4)$. Evidently, any rational arithmetic expression can be represented in floating-point arithmetic terms by compounding basic operations of the form $y = fl(x_1 * x_2)$.

We assume that the rounding errors in the operations are such that

$$fl(x_1 * x_2) = (x_1 * x_2)(1+\varepsilon), \tag{1.1.2}$$

where

$$|\varepsilon| \leq 2^{-t}. \tag{1.1.3}$$

For multiplication and division the value of ε will be taken as zero if either x_1 or x_2 is an integral power of 2. We assume further that relations of the type

$$fl(x_1 \pm x_2) = (x_1 \pm x_2)/(1+\varepsilon), \tag{1.1.4}$$

where ε satisfies (1.1.3), also hold. Relations (1.1.4) are due to Kahan (see Peters and Wilkinson, 1971) and are sometimes more convenient than (1.1.2). In any particular situation we shall use either (1.1.2) or (1.1.4) as appropriate.

Wilkinson (1963) states that some computers have less accurate rounding procedures than those which give the above results, but we assume (as do Peters and Wilkinson (1971) in a different context) that the differences are not of great consequence.

We shall also make use of the relations

$$(1+2^{-t})^{s} < 1 + 1.06s2^{-t}, \tag{1.1.5}$$

$$(1-2^{-t})^{-s} < 1 + 1.12s2^{-t}, \tag{1.1.6}$$

where s is a positive number (often integral). Relations (1.1.5) and (1.1.6) hold as long as s and t satisfy the mild restriction

$$s2^{-t} < 0.1. \tag{1.1.7}$$

We assume throughout this work that the inequality (1.1.7) is satisfied for all (reasonable) values of s that arise. (On the English Electric KDF9 computer, for which t=39, this means that s can be as large as $(0.1)2^{39} \doteq 5.5 \times 10^{10}$). Relation (1.1.5) is given by Wilkinson (1965: p113) and (1.1.6) by Cox (1972). Following Wilkinson (1965:p114) we shall sometimes use relation (1.1.5) in the form

$$(1+2^{-t})^s < 1 + s2^{-t_1}, \tag{1.1.8}$$

where

$$2^{-t_1} = (1.06)2^{-t}. \tag{1.1.9}$$

We observe that relation (1.1.7) is therefore equivalent to the inequality

$$s2^{-t_1} < 0.106 . \tag{1.1.10}$$

Moreover, (1.1.5), (1.1.6) and (1.1.7) yield

$$(1+2^{-t})^s < 1.106 \tag{1.1.11}$$

and

$$(1-2^{-t})^{-s} < 1.112 . \tag{1.1.12}$$

Throughout this work, unless otherwise stated, ε (with or without subscripts or superscripts) denotes a number satisfying

$$|\varepsilon| \leqslant 2^{-t} \tag{1.1.13}$$

and e (again with or without subscripts or superscripts) a number satisfying

$$|e| < 2^{-t_1}. \tag{1.1.14}$$

We shall often estimate the arithmetical work required by various computational processes by counting the number of <u>long operations</u> required. A long operation is one floating-point multiplication or one floating-point division.

1.2 Floating-point error analysis

As an illustration of the type of floating-point error analysis we shall be carrying out in subsequent chapters, we examine various formulae for linear transformations. Linear transformations are required in Chapters 5 and 6, where it is important that they are carried out in a numerically stable manner. We will see that the error analyses indicate very clearly whether a particular way of computing the transformation is stable or potentially unstable and, in the latter case, the reasons for the instability.

Consider the linear transformation

$$X = (2x - a - b)/(b - a), \qquad (1.2.1)$$

which maps the interval $[a,b]$ into $[-1, +1]$. When implemented in floating-point arithmetic the computed value $\bar{X}$ of X will be contaminated by rounding errors. Our aim is to produce a bound for $|\delta X|$, where

$$\delta X = \bar{X} - X, \qquad (1.2.2)$$

which holds for all $x \in [a,b]$. We seek a function $K(a,b)$ such that

$$|\delta X| \leqslant K(a,b)2^{-t}. \qquad (1.2.3)$$

It may seem somewhat surprising that we employ this formal approach to such an apparently innocuous computation as (1.2.1). The point we wish to stress, which we hope is brought out by our analyses, is that attention to detail is of vital importance in this and in many other computational processes. For instance, the nature of the error introduced in forming $\bar{X}$ is dependent upon the precise ordering of the basic arithmetic operations in (1.2.1) and, moreover, is influenced even more if (1.2.1) is re-expressed in certain other mathematically equivalent but computationally distinct forms.

Three possible ways of carrying out the transformation are given by

$$X = \frac{(2x-a)-b}{b-a}, \qquad (1.2.4)$$

$$X = \frac{2x-(a+b)}{b-a} \qquad (1.2.5)$$

and

$$X = cx - d, \tag{1.2.6}$$

where

$$c = 2/(b-a), \tag{1.2.7}$$

$$d = (a+b)/(b-a). \tag{1.2.8}$$

A floating-point error analysis of (1.2.4) yields

$$\bar{X} = \left\{(2x-a)(1+\varepsilon_1)-b\right\}(1+\varepsilon_2)(1+\varepsilon_3)(1+\varepsilon_4)/(b-a), \tag{1.2.9}$$

where

$$|\varepsilon_i| \leqslant 2^{-t} \quad (i=1,2,3,4), \tag{1.2.10}$$

from which

$$\delta X = \bar{X}-X = \left\{e_1(2x-a)+3e_2(2x-a-b)\right\}/(b-a), \tag{1.2.11}$$

where

$$|e_1|, |e_2| < 2^{-t_1}. \tag{1.2.12}$$

Thus

$$\delta X = e_1\left\{b/(b-a)+X\right\}+3e_2X \tag{1.2.13}$$

and hence

$$|\delta X| < \left\{|b|/(b-a)+4\right\} 2^{-t_1}. \tag{1.2.14}$$

We see immediately from (1.2.14) that the error in the computed value of X may be appreciable if the length b-a of the original interval is small compared with the magnitude of b.

Analysis of (1.2.5) and (1.2.6) result in bounds for δX similar in form to (1.2.14). This state of affairs is particularly unfortunate in the case of the third form of the transformation equation because the use of (1.2.6) appears to be eminently sensible if the transformation is to be used for a large number of x-values, since the constants c and d can be pre-computed from (1.2.7) and (1.2.8) with a consequent saving in arithmetic.

A fourth form of the transformation, which we now study, is unconditionally stable. Consider the use of the expression

$$X = \{(x-a)-(b-x)\}/(b-a) \tag{1.2.15}$$

to compute the value of X. An error analysis of this "somewhat unnatural" form gives

$$\bar{X} = \frac{\{(x-a)(1+\varepsilon_1)-(b-x)(1+\varepsilon_2)\}(1+\varepsilon_3)(1+\varepsilon_4)(1+\varepsilon_5)}{b-a}, \tag{1.2.16}$$

where

$$|\varepsilon_i| \leqslant 2^{-t} \quad (i=1,2,3,4,5), \tag{1.2.17}$$

from which

$$\delta X = \frac{e_1(x-a)-e_2(b-x)+3e_3(2x-a-b)}{b-a}, \tag{1.2.18}$$

where

$$|e_1|,\ |e_2|,\ |e_3| < 2^{-t_1}. \tag{1.2.19}$$

Thus, since $a \leqslant x \leqslant b$, it follows from (1.2.18) and (1.2.19) that

$$|\delta X| < (4)2^{-t_1}. \tag{1.2.20}$$

Note that the form (1.2.15) is computationally no more expensive than (1.2.4) or (1.2.5), but unlike them yields at worst a very small error.

We now consider briefly a second stable form, having an error bound only slightly inferior to (1.2.20). The approach is based upon carrying out the linear transformation (1.2.1) in two stages, viz. transformation to the interval $[0,1]$, followed by transformation to $[-1,1]$. Error analyses of the "obvious" transformations

$$X' = \frac{x-a}{b-a} \tag{1.2.21}$$

and

$$X = 2X'-1, \tag{1.2.22}$$

which carry out this two-stage process, yield

$$\bar{X}' = \frac{x-a}{b-a}(1+\varepsilon_1)(1+\varepsilon_2)(1+\varepsilon_3) \tag{1.2.23}$$

and

$$\bar{X} = (2X'-a)(1+\varepsilon_4) = \left\{\frac{2(x-a)}{b-a}(1+\varepsilon_1)(1+\varepsilon_2)(1+\varepsilon_3)-1\right\}(1+\varepsilon_4), \tag{1.2.24}$$

where X' is the value of the intermediate variable, computed values are denoted by "bars" as usual, and

$$|\varepsilon_i| \leqslant 2^{-t} \quad (i=1,2,3,4). \tag{1.2.25}$$

From (1.2.24),

$$\delta X = \bar{X}-X = \frac{6e_1(x-a)}{b-a} + e_2\left\{\frac{2(x-a)}{b-a} -1\right\}, \tag{1.2.26}$$

where

$$|e_1|,\ |e_2| < 2^{-t_1}, \tag{1.2.27}$$

from which

$$|\delta X| < (7)2^{-t_1}. \tag{1.2.28}$$

The transformations (1.2.21) and (1.2.22) can of course be combined to form the single transformation

$$X = \frac{2(x-a)}{b-a} -1 \tag{1.2.29}$$

or, expressed slightly differently, as

$$X = \frac{2(x-a)-(b-a)}{b-a}\,. \tag{1.2.30}$$

It is readily established that the use of (1.2.29) also gives an error satisfying (1.2.28) and that the bound for (1.2.30) satisfies

$$|\delta X| < (6)2^{-t_1}. \tag{1.2.31}$$

A much more detailed analysis, which takes into account the precise nature of the bit patterns in the mantissae of the floating-point representations of a, b and x, reveals that for nearly all values of these numbers the bound (1.2.14) is unduly pessimistic. In particular, the analysis shows that in these cases the value of ε_1 in (1.2.9) is zero, with the consequence that e_1 in (1.2.11) is zero and hence

$$|\delta X| < (3)2^{-t_1}. \qquad (1.2.32)$$

However, the detailed analysis also shows that there are values of the numbers a, b and x which result in ε_1 being exactly equal in modulus to 2^{-t}. In these cases the bound (1.2.14) proves to be realistic and predicts accurately the magnitude of the actual error in the computed value of X.

Detailed analyses of (1.2.5) and (1.2.6) reveal that the corresponding bounds are in fact realistic for most, rather than a few, values of a, b and x. I am indebted to Dr J H Wilkinson who suggested the method of approach to these detailed analyses.

The main conclusion to be drawn from the above relatively simple analyses is that for stability the transformation should be expressed in a form that ensures that the magnitude of each intermediate computed quantity is related as appropriate to the length of the original or of the transformed interval. We see that the unstable formulae (1.2.4), (1.2.5) and (1.2.6) all produce as intermediate quantities numbers related to the absolute value of the untransformed variable, a number having no relation to the length of the original interval. On the other hand, the intermediate quantities produced by the stable formulae (1.2.15), (1.2.21) and (1.2.22), (1.2.29), and (1.2.30) are all related to the lengths of the original or transformed range.

Extrapolating this conclusion we conjecture that numerical processes in general are more likely to be stable if, wherever possible, the intermediate computed quantities are not allowed to grow too large (or, in some rather special instances, too small). The principle certainly holds for Gaussian elimination, for it is known (Reid, 1971) that whatever strategy (whether it be partial pivoting, complete pivoting, pivoting down the main diagonal, etc) is employed, a bound for the departure of the linear system actually solved from that required to be solved is related directly to the largest matrix element at any stage of the reduction. If a linear system (square or rectangular) is solved using orthogonalization methods then no growth can occur (Peters and Wilkinson, 1970), with the result that the process is stable.

In the numerical methods we discuss we adhere to this general principle wherever possible. Particular instances are the use of plane rotations (Chapters 2 and 7), elementary stabilized transformations (Chapters 2 and 6) and the taking of convex combinations. The latter process is basic to many of our computations (Chapters 4, 5, 6 and 7 in particular).

We do not reproduce error analyses of well-accepted numerically stable methods such as the modified Gram-Schmidt process, Householder transformations and classical Givens rotations for solving linear systems, since such analyses abound in the literature, the key reference being Wilkinson (1965). However, wherever appropriate, we analyze methods that have appeared recently or have been developed during the course of this work.

We shall carry out, in later chapters, floating-point error analyses of various recurrence relations which arise in the solution of linear systems and in certain computations with splines. In particular we shall sometimes (i) employ a "running" error analysis (Peters and Wilkinson, 1971) to enable the computer itself to determine rigorous bounds on the errors it is making, (ii) obtain *a posteriori* absolute or relative error bounds and, occasionally, (iii) obtain <u>a priori</u> absolute or relative error bounds. To give the flavour of the types of results we obtain we analyze a simple example.

Consider the following recurrence relation which defines and generates the Fibonacci numbers:

$$\left.\begin{aligned} f_o &= f_1 = 1, \\ f_r &= f_{r-1} + f_{r-2} \quad (r=2,3,\ldots) \end{aligned}\right\} \tag{1.2.33}$$

Suppose this computation is carried out in floating-point arithmetic. Let $\bar{f}_r$ denote the computed value of f_r and $\delta f_r = \bar{f}_r - f_r$. Then

$$\left.\begin{aligned} \bar{f}_o &= f_o, \ \delta f_o = 0 \\ \bar{f}_1 &= f_1, \ \delta f_1 = 0 \end{aligned}\right\} \tag{1.2.34}$$

and

$$\bar{f}_r = fl(\bar{f}_{r-1}+\bar{f}_{r-2}) = (\bar{f}_{r-1}+\bar{f}_{r-2})/(1+\varepsilon_r) \qquad (r=2,3,\ldots). \tag{1.2.35}$$

Thus for $r \geqslant 2$,

$$(1+\varepsilon_r)\bar{f}_r = \bar{f}_{r-1}+\bar{f}_{r-2} \tag{1.2.36}$$

and therefore

$$f_r+\delta f_r+\varepsilon_r\bar{f}_r = f_{r-1}+\delta f_{r-1}+f_{r-2}+\delta f_{r-2} . \tag{1.2.37}$$

The use of (1.2.33) reduces (1.2.37) to

$$\delta f_r = \delta f_{r-1}+\delta f_{r-2}-\varepsilon_r\bar{f}_r. \tag{1.2.38}$$

Thus

$$|\delta f_r| \leqslant 2^{-t}F_r, \tag{1.2.39}$$

where

$$\left.\begin{aligned} F_0 &= F_1 = 0, \\ F_r &= F_{r-1}+F_{r-2}+\bar{f}_r \end{aligned}\right\} . \tag{1.2.40}$$

So, at the same time as it forms the $\bar{f}_r$, the computer can form the values F_r. Such a process is called a running error analysis. However, like the f_r, the values of F_r cannot be formed exactly, since rounding errors are made in computing the error relation (1.2.40)! This apparent difficulty is easily overcome as follows. Let $\bar{F}_r$ be the computed value of F_r. Then the computational equivalent of (1.2.40) is

$$\begin{aligned} \bar{F}_r &= fl(\bar{F}_{r-1}+\bar{F}_{r-2}+\bar{f}_r) \\ &= \left\{(\bar{F}_{r-1}+\bar{F}_{r-2})(1+\varepsilon_{1,r})+\bar{f}_r\right\}(1+\varepsilon_{2,r}). \end{aligned} \tag{1.2.41}$$

Thus, since the $\bar{F}_r$ and the $\bar{f}_r$ are non-negative, the contribution to the error incurred in computing F_r from (1.2.40) is at most a multiplicative factor $(1-2^{-t})^{-2}$. Hence, since $\delta f_0=\delta f_1=0$,

$$\left|\delta f_r\right| \leqslant 2^{-t}(1-2^{-t})^{2-2r}\bar{F}_r. \tag{1.2.42}$$

Now, by virtue of (1.1.12),

$$(1-2^{-t})^{2-2r} < 1.112. \tag{1.2.43}$$

Hence, since $\bar{F}_r > 0$ for $r \geqslant 2$,

$$\left|\delta f_r\right| < (1.112)2^{-t}\,\bar{F}_r. \quad (r \geqslant 2). \tag{1.2.44}$$

This result is an a posteriori absolute error bound. Although such a result is extremely useful in practice in that it enables a rigorous bound on the absolute error in the computed value to be obtained, it tells us nothing about the qualitative nature of the error growth in the computation. In other words it does not tell us whether the bound grows, for example, linearly, quadratically or exponentially etc, with r. In certain favourable cases the running error analysis approach can give rise to a posteriori bounds which not only display the qualitative nature of the growth but also obviate the need actually to use a running error relationship like (1.2.40) (which, incidentally, requires even more computational effort than the basic recurrence!). For instance, for the above example we shall show that, for $r \geqslant 2$, F_r satisfies the inequality

$$F_r \leqslant (1+2^{-t})^{r-2}(r-1)\bar{f}_r, \tag{1.2.45}$$

and hence that

$$|\delta f_r| \leqslant (1+2^{-t})^{r-2}(r-1)2^{-t}\bar{f}_r \,. \tag{1.2.46}$$

In order to establish this result we first assume it to be true for $F_2, F_3, \ldots, F_{r-1}$. Then the substitution of (1.2.45) (with r-1 and then r-2 replacing r) into the right-hand side of (1.2.40) and the use of (1.2.36) gives

$$\begin{aligned} F_r &\leqslant (1+2^{-t})^{r-3}(r-2)\bar{f}_{r-1}+(1+2^{-t})^{r-4}(r-3)\bar{f}_{r-2}+\bar{f}_r \\ &< (1+2^{-t})^{r-3}\left\{(r-2)(\bar{f}_{r-1}+\bar{f}_{r-2})+\bar{f}_r\right\} \\ &\leqslant (1+2^{-t})^{r-3}\left\{(r-2)(1+2^{-t})\bar{f}_r+\bar{f}_r\right\} \\ &< (1+2^{-t})^{r-2}(r-1)\bar{f}_r \,. \end{aligned} \tag{1.2.47}$$

But from (1.2.40), $F_2 = \bar{f}_2$. Hence (1.2.45) is true for r=2 and by induction therefore for all $r \geqslant 2$.

Having established a result of the form (1.2.45), it may then be possible to obtain an a priori relative error bound. Firstly (1.1.11) is used to simplify (1.2.45) slightly to give

$$F_r \leqslant 1.106(r-1)\bar{f}_r \tag{1.2.48}$$

and hence

$$\left|\delta f_r\right| \leqslant 1.106(r-1)2^{-t}\bar{f}_r. \tag{1.2.49}$$

But the relative error in $\bar{f}_r$ is simply

$$\left|\frac{\bar{f}_r - f_r}{f_r}\right| = \left|\frac{\delta f_r}{\bar{f}_r - \delta f_r}\right| = \left|\frac{\delta f_r/\bar{f}_r}{1-\delta f_r/\bar{f}_r}\right| \tag{1.2.50}$$

$$\leqslant \frac{1.106(r-1)2^{-t}}{1-1.106(r-1)2^{-t}}$$

$$< \frac{1.106(r-1)2^{-t}}{1-0.1106}$$

$$< 1.244(r-1)2^{-t}, \tag{1.2.51}$$

using (1.1.7). We can therefore state, before the computation is started, that the relative error in the computed value of f_r cannot exceed $1.244(r-1)2^{-t}$. This result is absolutely rigorous; in practice the statistical effects of rounding errors are more likely to give an actual error of the order of $(r-1)^{\frac{1}{2}}2^{-t}$. However, the importance of a result of the type obtained here is not only that the precise nature of the error bound has been obtained, but also that an a priori error bound can be obtained at all and, as we will see in Section 1.3, that the computation has been shown to be unconditionally numerically stable.

1.3 Algorithms and numerical stability

An algorithm is a procedure (set of rules, recipe) for obtaining a solution to a specific mathematical problem. An algorithm describes in an unambiguous manner the way in which a required set of numbers, the solution, may be computed from a given set of numbers, the data. For instance, the recurrence relation (1.2.33) constitutes an algorithm for computing the Fibonacci numbers $f_2, f_3, \ldots$ from the data (initial conditions) $f_0 = f_1 = 1$.

Let the m-vector $\underset{\sim}{x}$ denote a set of data values supplied to an algorithm A. Let the n-vector $\underset{\sim}{f}$ denote the solution obtained by A using exact arithmetic and the n-vector $\bar{\underset{\sim}{f}}$ the solution obtained by A using standard floating-point arithmetic.

Every algorithm has a domain of applicability X (Rice, 1971; Cox, 1974), defined by the set of data $\underset{\sim}{x}$ for which the algorithm can provide the desired solution $\underset{\sim}{f}$. For instance, $X = \left\{ x \mid x \geqslant 0 \right\}$ for an algorithm which computes the positive square root of a real number x; in practice there will be an upper bound M for the values of x for which the algorithm is designed, in which case $X = \left\{ x \mid 0 \leqslant x \leqslant M \right\}$.

A will be termed unconditionally numerically stable if, for all $\underset{\sim}{x} \in X$, the implementation of A in standard floating-point arithmetic provides a solution $\bar{\underset{\sim}{f}}$ which in some sense bears a close resemblance to $\underset{\sim}{f}$. Probably the most desirable form of closeness is

$$\|\bar{\underset{\sim}{f}} - \underset{\sim}{f}\| \leqslant K_1 2^{-t} \|\underset{\sim}{f}\|, \qquad (1.3.1)$$

where 2^{-t} is the relative machine precision, as before, and K_1 is related to the particular process employed in A. $\|\cdot\|$ denotes any convenient vector norm. If the computed solution is a single value then $\|\cdot\|$ may be replaced by $|\cdot|$ in the usual way. Often, for a particular process, K_1 is either a constant or depends upon a small number of parameters relating to that process. Sometimes an expression for K_1 can be determined a priori; in other cases K_1 may be the result of a running error analysis or an a posteriori analysis.

If $K_1 2^{-t} \ll 1$ then (1.3.1) may be considered an excellent bound in that the relative error in the computed solution will be small.

Sometimes it may be difficult or impossible to obtain a bound of the form (1.3.1). However, it may be possible to derive a bound of the form

$$\|\tilde{f}-f\| \leqslant K_2 2^{-t} M, \tag{1.3.2}$$

where, as before, K_2 is a constant or is related to the particular process, but

$$M = \max_{x \in X} \|f\| . \tag{1.3.3}$$

If $K_2 2^{-t} \ll 1$ then (1.3.2) may also indicate a stable algorithm. Of course, (1.3.2) is a somewhat weaker result than (1.3.1) in that whereas (1.3.1) gives a bound on the relative error and, consequently, on the absolute error, (1.3.2) merely gives a bound on the absolute error, which may or may not imply a satisfactory relative error bound.

An algorithm will be termed conditionally numerically stable if a result of the form (1.3.1) or (1.3.2) holds for an identifiable subset X' of X.

For some algorithms it is not easy to quote a result as straightforward as (1.3.1) or (1.3.2), even if such a result can be obtained at all. However, we can sometimes say that a particular algorithm is "good" because it exhibits stable behaviour in practice for most $x \in X$, although no theoretical statement of behaviour is easily obtained. The values of $x \in X$ for which the algorithm fails to produce good results may correspond to pathological or extreme situations, eg to data sets unlikely to arise in practical applications.

For some algorithms rigorous error bounds can be determined, but the bounds are most unlikely to be attained or even approached at all closely. A good example is the bound associated with Gaussian elimination with partial pivoting for solving linear algebraic systems (Wilkinson, 1965:p97), which contains a factor of 2^{n-1}, where n is the order of the system. It might be thought therefore that for systems of quite modest size the solutions obtained would have errors so large that the results were meaningless.

However, nothing could be further from the truth since, apart from artificially-constructed examples (for an interesting example see Wilkinson, 1961), a more realistic, though not rigorous, bound for practical purposes contains a factor of the order of unity rather than 2^{n-1}.

Most of the above discussion relates to forward error analysis in which a measure of the closeness of the computed solution to the actual solution is sought. For many algorithms it is more meaningful and relevant to use a backward error analysis. In such an analysis the solution obtained is interpreted as the exact solution of a problem with data $\bar{\underset{\sim}{x}}$ which is (hopefully) only slightly different from $\underset{\sim}{x}$. Bounds upon $\|\bar{\underset{\sim}{x}}-\underset{\sim}{x}\|$ are then sought, which again indicate whether the algorithms can be considered as being numerically stable.

Many of the computational processes we discuss are accompanied by commented algorithms. These algorithms are intended to provide a definitive "interface" between a "casual" description of a computational process and its formal implementation in a high-level language such as Algol or Fortran. We believe that a reader knowledgeable in a high-level language would readily be able to code these algorithms. For commercial reasons we are unable to list actual codes in this work. However, all the algorithms presented here have been programmed in Algol 60, Fortran IV or Babel, an Algol-like language due to Scowen (1969). Apart from the relatively trivial illustrative algorithms, such as Algorithm 1.3.1 below, they have been tested carefully on a wide variety of both model and practical problems.

We use the algorithms as building blocks, just as procedures are used in Algol and subroutines in Fortran. For example, the relatively simple algorithms in Section 2.1 for solving triangular systems are needed by many of the more complicated algorithms for solving general linear systems in the subsequent sections of Chapter 2. In turn, the algorithms in Chapters 6 and 7 for spline interpolation and least-squares spline approximation make use of the algorithms for linear systems.

Each algorithm is described by a sequence of steps or stages. Most steps describe one or more of the following operations: assign a value to a variable; advance or return to a stated step if a condition is satisfied, execute the stated steps the stated number of times. These three types of step occur frequently. Occasionally we need to make use of a dummy statement (or null operation), ie a statement whose presence is necessary to describe unambiguously the flow of a computational process. For this null operation we borrow the term Continue from the Fortran language. Other types of step also appear; we believe that most of these are self-explanatory: qualification will be given where thought necessary. Where appropriate the algorithmic steps are interspersed by comments or remarks which help relate the various stages of the algorithm to those of the computational process being implemented. In particular, if a special storage strategy is employed, such as in the algorithms of Sections 2.12 to 2.14 for stepped-banded matrices, the algorithmic steps refer to the notation appropriate to the special strategy, whereas the comments refer to the natural storage notation.

As a very simple illustration of the form of our algorithms, the recurrence relation (1.2.33) for generating the Fibonacci numbers is described by Algorithm 1.3.1 below.

Algorithm 1.3.1: Generation of the Fibonacci numbers $f_0, f_1, \ldots, f_n$.

Comment: Initialization.

Step 1. Set $f_0 = 1$ and $f_1 = 1$.

Comment: Recur the defining relation for the Fibonacci numbers.

Step 2. For $r=2,3,\ldots,n$ form $f_r = f_{r-1} + f_{r-2}$.

CHAPTER 2

THE NUMERICAL SOLUTION OF LINEAR ALGEBRAIC EQUATIONS

Frequent use is made throughout this work of methods for the solution of systems of linear equations (Chapters 6, 8 and 10) and also for the least-squares solution of systems of over-determined linear equations (Chapters 7 and 10). Accordingly, this chapter is devoted to the description of numerically stable methods for solving such problems. We concentrate particularly upon the linear least-squares problem, since the solution of a system of linear equations can be considered as being included as a special case. The linear least-squares problems that arise from the use of polynomial splines as approximating functions tend to be highly structured, if a suitable basis for the spline is employed. The so-called observation matrix (Section 2.2) proves to have special properties in that many of its elements are zero and, moreover, the disposition of the non-zero elements can be characterized in a straightforward manner. Similar remarks apply to the systems of linear equations arising from spline interpolation problems.

In order to obtain efficient algorithms for solving these problems it is important to take advantage of the special structure of these matrices. Firstly, however, we outline a number of methods currently available for the solution of dense linear least-squares problems and consider subsequently ways in which they can be modified so that structured problems can be treated.

There are six methods in current use:

(i) Cholesky decomposition of the normal equations,

(ii) Gaussian elimination

(iii) Gram-Schmidt orthogonalization

(iv) Householder transformations

(v) Givens rotations

(vi) The singular value decomposition

applied to the observation matrix.

For our purposes the use of Givens rotations provesto be most appropriate. In order to establish this we give a brief description of each approach, together with its merits and demerits.

In an attempt to obtain the utmost numerical stability, the methods applied to the observation matrix are sometimes implemented so as to include a column-interchange (pivoting) strategy (see, for example, Golub, 1965; Businger and Golub, 1965 and Peters and Wilkinson, 1970). Unfortunately, the interchanging of columns tends to destroy the nature of the zero-non-zero structure. Since in our work we wish to take full advantage of structure, we would be prepared to accept a slight loss of numerical stability if the avoidance of column interchanges led to significantly more efficient algorithms.

There is evidence both empirical and theoretical that the behaviour of the modified Gram-Schmidt method (see Section 2.6) is not improved by column interchanges. For instance, after obtaining considerable computational evidence, Rice (1966) concluded that interchanges result in a perceptible but small (even negligible) improvement. In a detailed theoretical floating-point error analysis Björck (1967) concluded that, regardless of whether or not interchanges are made, the errors in the computed solution are less than the errors resulting from relative perturbations in the observation matrix and right-hand side of $K(m,n)2^{-t}$. Here t is the number of bits in the mantissa of the floating-point word and K is a modest function of m and n (the respective numbers of rows and columns in the observation matrix). Similar conclusion can be expected to hold in respect of methods (iv) and (v) (Wilkinson, 1974).

Many of the numerical methods we describe are applicable equally to the square case (interpolation) and to the rectangular or over-determined case (least squares). However, there are advantages to be gained in terms of computational efficiency by employing elimination methods in the square case, and in terms of stability and simplicity by using orthogonalization methods in the over-determined case. Accordingly, most of the algorithms we present for these methods reflect these considerations.

In Section 2.1 we give algorithms for the solution of triangular systems, since these are required by many of the subsequent algorithms for more general systems. In Section 2.2 we introduce the linear least-squares problem and describe in Section 2.3 the normal-equations approach to its solution. Elimination methods are discussed in Section 2.4 and in Section 2.5 the use of orthogonal transformations is considered. Particular methods for orthogonal transformations, viz modified Gram-Schmidt, Householder and Givens rotations are described in Sections 2.6, 2.7 and 2.8. Modern variants of Givens rotations are presented in Section 2.9 and a comparison of the various methods for orthogonal triangularization is made in Section 2.10. In Section 2.11 stepped-banded matrices are defined and in Sections 2.12, 2.13 and 2.14 methods based upon Gaussian elimination, elementary transformations and orthogonal transformations for solving systems with stepped-banded matrices are presented. The powerful singular value decomposition is considered in Section 2.15 and, finally, in Section 2.16 perturbation bounds for the solution of linear systems are given.

2.1 The solution of triangular systems

Most of the numerical methods we describe for solving the frequently over-determined linear system

$$\underset{\sim}{A}\underset{\sim}{x} = \underset{\sim}{b} , \qquad (2.1.1)$$

where A is a given m by n matrix and b is a given m-vector, firstly reduce the system to upper triangular form. This reduction is usually carried out by pre-multiplying both sides of (2.1.1) by a sequence of transformation matrices chosen to have the effect of annihilating in a systematic manner the sub-diagonal elements of A. Triangular systems also arise in our work in various other ways.

We describe algorithms for solving three types of triangular system that are of particular importance. We denote the general triangular system by

$$R x = \theta , \qquad (2.1.2)$$

where R is an upper-triangular matrix of order n by n and θ an n-vector. It is assumed henceforth that R is non-singular, ie the elements on the main diagonal of R are non-zero. Any implementation of our algorithms would of course test either implicitly or explicitly whether these elements were indeed non-zero.

We consider first the simplest case where R is dense, ie all or most of the super-diagonal elements of R are non-zero. In the trivial algorithm below, a natural storage strategy is assumed, ie that element r_{ij} $(j \geqslant i)$ of R is stored in location (i,j) of an n by n array. Locations (i,j) $(j < i)$ of this array are not used.

<u>Algorithm 2.1.1</u>: Solution of the dense upper triangular system $Rx = \theta$. in the case where R is stored in natural form.

Step 1. For $j = n, n-1, \ldots, 1$ compute

$$x_j = (\theta_j - \sum_{k=j+1}^{n} r_{jk} x_k)/r_{jj} .$$

In this and subsequent algorithms we adopt the convention that there is no contribution from a sum having a lower limit that exceeds the upper limit.

In order to minimize storage requirements, some of our algorithms store the diagonal and super-diagonal elements of R sequentially by rows in a vector of length $\frac{1}{2}n(n+1)$. In Algorithm 2.1.2 this storage strategy is assumed. Algorithm 2.1.3 is similar to Algorithm 2.1.3 except that R is taken to be <u>unit</u> upper triangular; in this case only the super-diagonal elements are stored, again sequentially by rows, in a vector of length $\frac{1}{2}n(n-1)$.

Algorithm 2.1.2: Solution of the dense upper triangular system $\underset{\sim}{R}\underset{\sim}{x} = \underset{\sim}{\theta}$, in the case where the diagonal and super-diagonal elements of $\underset{\sim}{R}$ are stored sequentially by rows.

Step 1. For $j = n, n-1, \ldots, 1$ execute Steps 2-7.

Step 2. Set $l = (j-1)(2n+2-j)/2+1$.

Comment: r_{jj} is stored as the l th element of the vector.

Step 3. Set $y = r_l$ and $z = \theta_j$.

Step 4. For $k = j+1, j+2, \ldots, n$ execute Steps 5-6.

Step 5. Replace l by $l+1$.

Step 6. Replace z by $z-r_l x_k$.

Step 7. Set $x_j = z/y$.

Algorithm 2.1.3: Solution of the dense unit upper triangular system $\underset{\sim}{R}\underset{\sim}{x} = \underset{\sim}{\theta}$, in the case where the super-diagonal elements of $\underset{\sim}{R}$ are stored sequentially by rows.

Step 1. For $j = n, n-1, \ldots, 1$ execute Steps 2-7.

Step 2. Set $l = (j-1)(2n-j)/2$.

Step 3. Set $z = \theta_j$.

Step 4. For $k = j+1, j+2, \ldots, n$ execute Steps 5-6.

Step 5. Replace l by $l+1$.

Step 6. Replace z by $z-r_l x_k$.

Step 7. Set $x_j = z$.

Particular attention will be paid to the solution of systems where the matrices are stepped-banded in form (for a definition see Section 2.11). The resulting triangular systems have matrices that are band upper triangular. Algorithm 2.1.4 solves the system (2.1.2) in the case where $\underset{\sim}{R}$ has $q-1$ super-diagonals. The strategy employed is to store the diagonal and super-diagonals of $\underset{\sim}{R}$ as the successive columns of an n by q array. This condensed storage strategy is illustrated in the case $n = 6$, $q = 3$ in

Fig 2.1.1 (* denotes an unused storage location).

$$\begin{array}{cccccc}
r_{11} & r_{12} & r_{13} & & & \\
 & r_{22} & r_{23} & r_{24} & & \\
 & & r_{33} & r_{34} & r_{35} & \\
 & & & r_{44} & r_{45} & r_{46} \\
 & & & & r_{55} & r_{56} \\
 & & & & & r_{66}
\end{array}
\qquad\qquad
\begin{array}{ccc}
r_{11} & r_{12} & r_{13} \\
r_{22} & r_{23} & r_{24} \\
r_{33} & r_{34} & r_{35} \\
r_{44} & r_{45} & r_{46} \\
r_{55} & r_{56} & * \\
r_{66} & * & *
\end{array}$$

Natural storage — Condensed storage

Fig 2.1.1 Natural and condensed storage for an upper band triangular matrix in the case $n = 6$, $q = 3$.

<u>Algorithm 2.1.4</u>: Solution of the upper band triangular system $\underset{\sim}{R}\underset{\sim}{x} = \underset{\sim}{\theta}$, in the case where the diagonal and super-diagonals of $\underset{\sim}{R}$ are stored successively in columns.

Step 1. For $i = n, n-1, \ldots, 1$ execute Steps 2-3.

Step 2. Set $j = \min(n-i+1, q)$.

Step 3. Form $x_i = (\theta_i - \sum_{k=2}^{j} r_{ik} x_{k+i-1})/r_{i1}$.

2.2 <u>The linear least-squares problem</u>

Consider the linear least-squares problem

$$\min_{\underset{\sim}{x}} \ \underset{\sim}{r}^T \underset{\sim}{r} \ , \tag{2.2.1}$$

where

$$\underset{\sim}{r} = \underset{\sim}{A}\underset{\sim}{x} - \underset{\sim}{b} \ . \tag{2.2.2}$$

In (2.2.2), $\underset{\sim}{A}$, a prescribed <u>observation</u> or <u>design</u> matrix, is an m by n matrix of rank k ($k \leq n \leq m$), $\underset{\sim}{b}$ is a prescribed <u>right-hand-side vector</u> of length m, $\underset{\sim}{r}$ is the <u>residual vector</u> and $\underset{\sim}{x}$ the <u>solution vector</u>. In the problems to be considered k is usually, but not always, equal to n.

The following properties of least-squares solutions are known (see, for example, Peters and Wilkinson, 1970):

(i) <u>All</u> least-squares solutions satisfy the so-called <u>normal equations</u>

$$\underset{\sim}{A}^T\underset{\sim}{A}\underset{\sim}{x} = \underset{\sim}{A}^T\underset{\sim}{b} . \tag{2.2.3}$$

(ii) If k=n then the system (2.2.3) has a unique solution.

(iii) If $k < n$ then of all solutions satisfying (2.2.3) there is only one, known as the <u>minimal norm solution</u>, which minimizes $z = (\underset{\sim}{x}^T\underset{\sim}{x})^{\frac{1}{2}}$, the Euclidean length of the solution vector.

One method for the solution of (2.2.1) is based upon the direct solution of the system (2.2.3). The other five methods are all based upon an initial factorization of the form

$$\underset{\sim}{A} = \underset{\sim}{G}\underset{\sim}{H} , \tag{2.2.4}$$

where $\underset{\sim}{G}$ is an m by k matrix, $\underset{\sim}{H}$ a k by n matrix and both $\underset{\sim}{G}$ and $\underset{\sim}{H}$ are of rank k. Evidently this factorization is not unique since we may write

$$\underset{\sim}{A} = \underset{\sim}{G}'\underset{\sim}{H}' , \tag{2.2.5}$$

where

$$G' = GF^{-1} , \tag{2.2.6}$$

$$\underset{\sim}{H}' = \underset{\sim}{F}\underset{\sim}{H} \tag{2.2.7}$$

and $\underset{\sim}{F}$ is any k by k matrix of rank k.

The substitution of (2.2.4) into (2.2.3) yields

$$\underset{\sim}{H}^T\underset{\sim}{G}^T\underset{\sim}{G}\underset{\sim}{H}\underset{\sim}{x} = \underset{\sim}{H}^T\underset{\sim}{G}^T\underset{\sim}{b} . \tag{2.2.8}$$

The pre-multiplication of both sides of (2.2.8) by $\underset{\sim}{H}$ then yields

$$\underset{\sim}{H}\underset{\sim}{H}^T\underset{\sim}{G}^T\underset{\sim}{G}\underset{\sim}{H}\underset{\sim}{x} = \underset{\sim}{H}\underset{\sim}{H}^T\underset{\sim}{G}^T\underset{\sim}{b} \ . \tag{2.2.9}$$

Now both $\underset{\sim}{H}\underset{\sim}{H}^T$ and $\underset{\sim}{G}^T\underset{\sim}{G}$ are k by k matrices of rank k, since $\underset{\sim}{G}$ and $\underset{\sim}{H}$ are themselves of rank k. Hence the pre-multiplication of both sides of (2.2.9) by $(\underset{\sim}{G}^T\underset{\sim}{G})^{-1}(\underset{\sim}{H}\underset{\sim}{H}^T)^{-1}$ yields

$$\underset{\sim}{H}\underset{\sim}{x} = (\underset{\sim}{G}^T\underset{\sim}{G})^{-1}\underset{\sim}{G}^T\underset{\sim}{b} \ . \tag{2.2.10}$$

Since $\underset{\sim}{H}$ is of rank k, a particular solution (which may be verified by inspection) of the equation

$$\underset{\sim}{H}\underset{\sim}{x} = \underset{\sim}{v} \ , \tag{2.2.11}$$

where $\underset{\sim}{v}$ is any given vector of length k, is

$$\underset{\sim}{x} = \underset{\sim}{H}^T(\underset{\sim}{H}\underset{\sim}{H}^T)^{-1} \ \underset{\sim}{v}. \tag{2.2.12}$$

Thus a particular solution of (2.2.10) is

$$\underset{\sim}{x} = \underset{\sim}{H}^T(\underset{\sim}{H}\underset{\sim}{H}^T)^{-1}(\underset{\sim}{G}^T\underset{\sim}{G})^{-1}\underset{\sim}{G}^T\underset{\sim}{b} \ . \tag{2.2.13}$$

Peters and Wilkinson (1970) show that (2.2.13) is in fact the <u>minimal least-squares solution</u>. The matrix

$$\underset{\sim}{A}^{\dagger} = \underset{\sim}{H}^T(\underset{\sim}{H}\underset{\sim}{H}^T)^{-1}(\underset{\sim}{G}^T\underset{\sim}{G})^{-1}\underset{\sim}{G}^T \tag{2.2.14}$$

is termed the <u>pseudo-inverse</u> of $\underset{\sim}{A}$.

In the full-rank case k=n, $\underset{\sim}{H}$ is an n by n matrix of rank n and accordingly (2.2.13) reduces to

$$\underset{\sim}{x} = \underset{\sim}{H}^{-1}(\underset{\sim}{G}^T\underset{\sim}{G})^{-1}\underset{\sim}{G}^T\underset{\sim}{b} \ . \tag{2.2.15}$$

2.3 Cholesky decomposition of the normal equations

One method of computing the linear least-squares solution in the full-rank case k=n is suggested by equations (2.2.3). In this approach we form the n by n matrix $\underset{\sim}{C}$ and the n-vector $\underset{\sim}{d}$, where

$$\underset{\sim}{C} = \underset{\sim}{A}^T\underset{\sim}{A} \tag{2.3.1}$$

and

$$\underset{\sim}{d} = \underset{\sim}{A}^T\underset{\sim}{b} , \tag{2.3.2}$$

and then solve

$$\underset{\sim}{C}\underset{\sim}{x} = \underset{\sim}{d} . \tag{2.3.3}$$

Since $\underset{\sim}{A}$ is of rank n, $\underset{\sim}{C}$ is also of rank n. Moreover, $\underset{\sim}{C}$ is positive definite. In such a case, Cholesky decomposition (Wilkinson, 1965: p 229 <u>et seq</u>) may be used to give the factor $\underset{\sim}{R}$ in

$$\underset{\sim}{C} = \underset{\sim}{R}^T\underset{\sim}{R} , \tag{2.3.4}$$

where $\underset{\sim}{R}$ is an upper triangular matrix of order n and rank n. The solution $\underset{\sim}{x}$ may then be obtained by solving the triangular systems

$$\underset{\sim}{R}^T\underset{\sim}{y} = \underset{\sim}{d} \tag{2.3.5}$$

and

$$\underset{\sim}{R}\underset{\sim}{x} = \underset{\sim}{y} . \tag{2.3.6}$$

The main disadvantage of this approach is that in forming $\underset{\sim}{C} = \underset{\sim}{A}^T\underset{\sim}{A}$ some of the information contained in $\underset{\sim}{A}$ may be lost. The following example due to Läuchli (1961) (also see Golub, 1965 and Björck, 1967) illustrates this point very well. Let

$$\underset{\sim}{A} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ \delta & 0 & 0 & 0 & 0 \\ 0 & \delta & 0 & 0 & 0 \\ 0 & 0 & \delta & 0 & 0 \\ 0 & 0 & 0 & \delta & 0 \\ 0 & 0 & 0 & 0 & \delta \end{bmatrix}. \tag{2.3.7}$$

Then

$$\underset{\sim}{C} = \underset{\sim}{A}^T\underset{\sim}{A} = \begin{bmatrix} 1+\delta^2 & 1 & 1 & 1 & 1 \\ 1 & 1+\delta^2 & 1 & 1 & 1 \\ 1 & 1 & 1+\delta^2 & 1 & 1 \\ 1 & 1 & 1 & 1+\delta^2 & 1 \\ 1 & 1 & 1 & 1 & 1+\delta^2 \end{bmatrix}. \tag{2.3.8}$$

It is easily verified that if $\delta \neq 0$ then the rank of $\underset{\sim}{C}$ is five, since the eigenvalues of $\underset{\sim}{C}$ are $5+\delta^2$, δ^2, δ^2, δ^2 and δ^2. Now consider the computation of the elements of $\underset{\sim}{C}$. Even if this computation is exact, apart from a final rounding to t binary digits, then $1+\delta^2$ will be rounded to unity for all δ such that $|\delta| \leq 2^{-t/2}$. In such cases the exact eigenvalues of the computed $\underset{\sim}{C}$ will be 5, C, 0, 0 and 0 and the corresponding rank will be unity. Thus, however accurately the Cholesky decomposition is carried out, it is impossible to solve the system (2.3.3).

There are some problems, however, where the approach of this section is effective. These problems correspond, at least in the context of data approximation, to the choice of a "nearly-orthogonal" set of basis functions, together with an appropriate set of data points.

2.4 Gaussian elimination

We now outline a method based upon Gaussian elimination for the full-rank case k=n. Peters and Wilkinson (1970), who appear to have been the first to use the method, give a more detailed description for the general case $k \leq n$.

It is well known that in the case m=n the n by n matrix $\underset{\sim}{A}$ can be factorized in the form

$$\underset{\sim}{A} = \underset{\sim}{L}\underset{\sim}{U} , \tag{2.4.1}$$

where $\underset{\sim}{L}$ is lower-triangular with unit diagonal elements and $\underset{\sim}{U}$ is upper triangular. A common way to obtain the factorization in a numerically stable manner is to employ Gaussian elimination with partial pivoting (Wilkinson, 1965: p200 et seq). The partial pivoting strategy means that in general we obtain an $\underset{\sim}{L}\underset{\sim}{U}$ decomposition of a matrix $\underset{\sim}{A}'$, where $\underset{\sim}{A}'$ is derived from $\underset{\sim}{A}$ by suitably permuting its rows, ie

$$\underset{\sim}{A}' = \underset{\sim}{P}\underset{\sim}{A} = \underset{\sim}{L}\underset{\sim}{U} , \tag{2.4.2}$$

where $\underset{\sim}{P}$ is a permutation matrix.

In the case m>n it is also possible to use Gaussian elimination to obtain a factorization of the m by n matrix $\underset{\sim}{A}$. We still obtain a decomposition of the form (2.4.2), but now $\underset{\sim}{L}$ is an m by n unit lower-trapezoidal matrix (Peters and Wilkinson, 1970), ie $\underset{\sim}{L}$ is of the form illuatrated in Fig 2.4.1 for the case m=6, n=4.

$$\underset{\sim}{L} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ l_{21} & 1 & 0 & 0 \\ l_{31} & l_{32} & 1 & 0 \\ l_{41} & l_{42} & l_{43} & 1 \\ l_{51} & l_{52} & l_{53} & l_{54} \\ l_{61} & l_{62} & l_{63} & l_{64} \end{bmatrix} .$$

Fig 2.4.1 A unit lower-trapezoidal matrix of order 6 by 4.

For notational convenience suppose that the rows of $\underset{\sim}{A}$ have initially been permuted so that no interchanges are subsequently necessary, ie $\underset{\sim}{P} = \underset{\sim}{I}$, the unit matrix. Then identifying $\underset{\sim}{G}$ with $\underset{\sim}{L}$ and $\underset{\sim}{H}$ with $\underset{\sim}{U}$ in (2.2.15) gives

$$x = U^{-1}(L^TL)^{-1}d \, , \tag{2.4.3}$$

where

$$d = L^Tb \, . \tag{2.4.4}$$

The solution vector x in (2.4.3) may then be computed as follows. After forming the factors L and U, we form the vector d in (2.4.4) and the n by n matrix

$$M = L^TL \, . \tag{2.4.5}$$

Since M is symmetric positive definite, it possesses the Cholesky decomposition

$$M = VV^T \, , \tag{2.4.6}$$

where V is upper triangular. After forming V, the intermediate vectors z, y and the solution vector x are obtained from the solutions of the triangular sets of equations

$$Vz = d \, , \tag{2.4.7}$$

$$V^Ty = z \, , \tag{2.4.8}$$

$$Ux = y \, . \tag{2.4.9}$$

In the case where A is square, ie m=n, then the steps involving the formation and the Cholesky decomposition of M are unnecessary. The solution to (2.2.1) in this case is also that of

$$Ax = b \, , \tag{2.4.10}$$

which can be found by the use of (2.4.1), ie by solving

$$Ly = b \tag{2.4.11}$$

and

$$Ux = y \, . \tag{2.4.12}$$

It may be thought that, since this method forms $\underset{\sim}{M} = \underset{\sim}{L}^T\underset{\sim}{L}$ and then the Cholesky decomposition of $\underset{\sim}{M}$, the stability problems associated with the normal-equations approach are still present. However, Peters and Wilkinson (1970) state that pivoting usually causes $\underset{\sim}{L}$ to be well-conditioned and any ill-condition in $\underset{\sim}{A}$ is wholly reflected in $\underset{\sim}{U}$. Thus the squaring of the condition number, as a result of forming $\underset{\sim}{A}^T\underset{\sim}{A}$ directly, is avoided.

We present, as Algorithm 2.4.1 below, an algorithmic statement of the method of this section. In this implementation $\underset{\sim}{A}$ is overwritten by $\underset{\sim}{L}$ and $\underset{\sim}{U}$, with the main diagonal and the super-diagonals containing the elements of $\underset{\sim}{U}$ and the sub-diagonals the sub-diagonal elements of $\underset{\sim}{L}$ (the main diagonal of $\underset{\sim}{L}$ is not stored since all its elements have the value unity). The vector $\underset{\sim}{d}$ is overwritten on $\underset{\sim}{b}$. The sub-diagonal elements of the symmetric matrix $\underset{\sim}{M}$ then overwrite the sub-diagonals of $\underset{\sim}{A}$ and the main diagonal of $\underset{\sim}{M}$ is formed in an n-vector $\underset{\sim}{p}$. The intermediate vectors $\underset{\sim}{y}$ and $\underset{\sim}{z}$ are stored in the locations ultimately used for $\underset{\sim}{x}$.

<u>Algorithm 2.4.1</u>: LU factorization and linear least-squares solution using Gaussian elimination with partial pivoting.

Comment: $\underset{\sim}{L}$ and $\underset{\sim}{U}$ are formed in Steps 1-8.

Step 1. For $l = 1, 2, \ldots, n$ execute Steps 2-8.

Step 2. Determine the smallest value of k such that

$$|a_{kl}| \geqslant |a_{il}| \quad (i = l, l+1, \ldots, m).$$

Step 3. If $k = l$ advance to Step 6.

Step 4. For $j = 1, 2, \ldots, n$ interchange the values of a_{kj} and a_{lj}.

Step 5. Interchange the values of b_k and b_l.

Step 6. For $i = l+1, l+2, \ldots, m$ execute Steps 7-8.

Step 7. Replace a_{il} by a_{il}/a_{ll}.

Step 8. For $j = l+1, l+2, \ldots, n$ replace a_{ij} by $a_{ij}-a_{il}a_{lj}$.

...

Comment: Branch according to whether the system is square or rectangular.

Step 9. If $m > n$ advance to Step 12.

Comment: In the square case the formation of $\underset{\sim}{M}$ and $\underset{\sim}{d}$ and the solution of $\underset{\sim}{M}\underset{\sim}{y} = \underset{\sim}{d}$ are replaced by the solution of $\underset{\sim}{L}\underset{\sim}{y} = \underset{\sim}{b}$.

Step 10. For $i = 1, 2, \ldots, n$ compute $x_i = b_i - \sum_{k=1}^{i-1} a_{ik} x_k$.

Step 11. Advance to Step 26.

Step 12. Comment: Form $\underset{\sim}{d} = \underset{\sim}{L}^T \underset{\sim}{b}$.

Step 13. For $j = 1, 2, \ldots, n$ replace b_j by $b_j + \sum_{i=j+1}^{m} b_i a_{ij}$.

Comment: $\underset{\sim}{M} = \underset{\sim}{L}^T \underset{\sim}{L}$ is formed in Steps 14-18.

Step 14. For $i = 1, 2, \ldots, n$ execute Steps 15-18.

Step 15. For $j = 1, 2, \ldots, i$ execute Steps 16-18.

Step 16. Set $g = a_{ij}$ (if $i \neq j$) or 1 (otherwise).

Step 17. Replace g by $g + \sum_{l=i+1}^{m} a_{li}\, a_{lj}$.

Step 18. Set $a_{ij} = g$ (if $i \neq j$) or $p_j = g$ (otherwise).

Comment: $\underset{\sim}{V}$ is formed in Steps 19-23.

Step 19. For $j = 1, 2, \ldots, n$ execute Steps 20-23.

Step 20. For $i = j, j+1, \ldots, n$ execute Steps 21-23.

Step 21. Set $g = a_{ij}$ (if $i \neq j$) or p_j (otherwise).

Step 22. Replace g by $g - \sum_{k=1}^{j-1} a_{jk}\, a_{ik}$.

Step 23. Set $p_j = h = g^{-\frac{1}{2}}$ (if $i = j$) or $a_{ij} = hg$ (otherwise).

Comment: Solve $\underset{\sim}{V}\underset{\sim}{z} = \underset{\sim}{d}$.

Step 24. For $i = 1, 2, \ldots, n$ compute $x_i = p_i \, (b_i - \sum_{k=1}^{i-1} a_{ik}\, x_k)$.

Comment: Solve $\underset{\sim}{V}^T \underset{\sim}{y} = \underset{\sim}{z}$.

Step 25. For $i = n, n-1, \ldots, 1$ replace x_i by $p_i \, (x_i - \sum_{k=i+1}^{n} a_{ki}\, x_k)$.

Comment: Solve $\underset{\sim}{U}\underset{\sim}{x} = \underset{\sim}{y}$.

Step 26. For $i = n, n-1, \ldots, 1$ replace x_i by $(x_i - \sum_{k=i+1}^{n} a_{ik}\, x_k)/a_{ii}$.

A variant of the method of this section for $m \geqslant n$, proposed by Cline (1973), avoids the formation of $\underset{\sim}{M}$ in (2.4.5). In Cline's method $\underset{\sim}{L}$ is decomposed into

$$\underset{\sim}{L} = \underset{\sim}{Q} \begin{bmatrix} \underset{\sim}{I} \\ \underset{\sim}{0} \end{bmatrix} \underset{\sim}{L}' , \tag{2.4.13}$$

where $\underset{\sim}{Q}$ is an m by m orthogonal (unitary) matrix (Section 2.5) and $\underset{\sim}{L}'$ is n by n lower triangular. Cline shows how to construct $\underset{\sim}{L}'$ (it is not necessary to form $\underset{\sim}{Q}$ explicitly) using Householder transformations (Section 2.7). Now, since

$$\underset{\sim}{L}^T\underset{\sim}{L} = (\underset{\sim}{L}')^T \begin{bmatrix} \underset{\sim}{I} & \underset{\sim}{0} \end{bmatrix} \underset{\sim}{Q}^T\underset{\sim}{Q} \begin{bmatrix} \underset{\sim}{I} \\ \underset{\sim}{0} \end{bmatrix} \underset{\sim}{L}' = (\underset{\sim}{L}')^T\underset{\sim}{L}', \tag{2.4.14}$$

(2.4.3) reduces to

$$\underset{\sim}{x} = \underset{\sim}{U}^{-1} \left\{ (\underset{\sim}{L}')^T\underset{\sim}{L}' \right\}^{-1} \underset{\sim}{d}, \tag{2.4.15}$$

from which $\underset{\sim}{x}$ may be obtained, via intermediate vectors $\underset{\sim}{z}$ and $\underset{\sim}{y}$, from the solutions of the triangular sets of equations

$$(\underset{\sim}{L}')^T\underset{\sim}{z} = \underset{\sim}{d}, \tag{2.4.16}$$

$$\underset{\sim}{L}'\underset{\sim}{y} = \underset{\sim}{z} , \tag{2.4.17}$$

$$\underset{\sim}{U}\underset{\sim}{x} = \underset{\sim}{y} . \tag{2.4.18}$$

Cline shows that, if terms of order mn and n^2 are ignored compared with those of mn^2 and n^3, on a multiplication count his method is faster than the normal-equations approach if $m < 4n/3$ and faster than the use of Householder transformations if $m < 5n/3$.

2.5 The use of orthogonal transformations

That orthogonal transformations can usefully be employed in the solution of linear least-squares problems appears to have been first proposed by Householder (1958). However, it was not until seven years later that Golub (1965) and Golub and Kahan (1965) gave detailed expositions of the application of orthogonal (Householder) transformations to least-squares problems. The joint work was concerned with the more sophisticated singular value decomposition (SVD); we shall defer discussion of the SVD until Section 2.15. A further seven years later the work of Gentleman (1972, 1973) showed that Givens rotations could also be used to advantage in solving such problems. This and the subsequent work by Hammarling (1974) and Moler (1974) gave a new impetus to the use of Givens rotations in that they showed that the amount of arithmetic could be reduced to that of the method of Householder transformations, whilst still preserving numerical stability. A third method, the Gram-Schmidt factorization, can also usefully be classified with the Householder and Givens methods.

Suppose an orthogonal matrix $\underset{\sim}{Q}$ of order m by k can be found to yield the factorization

$$\underset{\sim}{A} = \underset{\sim}{Q}\underset{\sim}{R} , \tag{2.5.1}$$

where $\underset{\sim}{R}$ is an upper-trapezoidal matrix of order k by n. The identification of $\underset{\sim}{Q}$ with $\underset{\sim}{G}$ and $\underset{\sim}{R}$ with $\underset{\sim}{H}$ in (2.2.4) gives, upon using (2.2.13),

$$\underset{\sim}{x} = \underset{\sim}{R}^T(\underset{\sim}{R}\underset{\sim}{R}^T)^{-1}(\underset{\sim}{Q}^T\underset{\sim}{Q})^{-1}\underset{\sim}{Q}^T\underset{\sim}{b} . \tag{2.5.2}$$

Now if the transformations applied to $\underset{\sim}{A}$ to yield $\underset{\sim}{R}$ are also applied to $\underset{\sim}{b}$, then

$$\underset{\sim}{b} = \underset{\sim}{Q}\underset{\sim}{c} , \tag{2.5.3}$$

say, where $\underset{\sim}{c}$ is the transformed vector. The substitution of (2.5.3) into

(2.5.2) yields

$$\underset{\sim}{x} = \underset{\sim}{R}^T(\underset{\sim}{R}\underset{\sim}{R}^T)^{-1}\underset{\sim}{c} \ . \tag{2.5.4}$$

We now form the k by k symmetric positive definite matrix

$$\underset{\sim}{M} = \underset{\sim}{R}\underset{\sim}{R}^T \tag{2.5.5}$$

and, as in the Gaussian elimination algorithm (Section 2.4), take the Cholesky decomposition

$$\underset{\sim}{M} = \underset{\sim}{V}\underset{\sim}{V}^T , \tag{2.5.6}$$

where $\underset{\sim}{V}$ is upper triangular. Then intermediate vectors $\underset{\sim}{z}$ and $\underset{\sim}{y}$ may be obtained from the triangular sets of equations

$$\underset{\sim}{V}\underset{\sim}{z} = \underset{\sim}{c} \tag{2.5.7}$$

and

$$\underset{\sim}{V}^T\underset{\sim}{y} = \underset{\sim}{z} \tag{2.5.8}$$

and finally the solution $\underset{\sim}{x}$ from

$$\underset{\sim}{x} = \underset{\sim}{R}^T\underset{\sim}{y} \ . \tag{2.5.9}$$

In the full-rank case k=n, (2.5.4) reduces to

$$\underset{\sim}{x} = \underset{\sim}{R}^{-1}\underset{\sim}{c} , \tag{2.5.10}$$

a single triangular system.

There are three methods currently available for carrying out the factorization (2.5.1). These methods are (i) modified Gram-Schmidt, (ii) Householder transformations and (iii) Givens rotations. We give brief descriptions of these methods for the case of full rank, ie k=n. Their extension to the general case $k \leqslant n$ is straightforward (Peters and Wilkinson, 1970); we shall not concern ourselves with the specific details here.

It will be noticed that in the Householder and Givens methods, the matrix $\underset{\sim}{Q}$ is in fact m by m rather than m by k. However, in the product $\underset{\sim}{A} = \underset{\sim}{Q}\underset{\sim}{R}$, the last m-k columns of $\underset{\sim}{Q}$ play no part and hence we may write

$$\underset{\sim}{A} = (\underset{\sim}{Q}\underset{\sim}{I}_{mk})\underset{\sim}{R} , \tag{2.5.11}$$

where $\underset{\sim}{I}_{mk}$ consists of the first k columns of the m by m unit matrix, which is compatible dimensionally with (2.2.4).

Note that the expressions (2.5.4) and (2.5.10) for the solution vector $\underset{\sim}{x}$ do not involve the orthogonal matrix $\underset{\sim}{Q}$. In fact the Householder and Givens methods do not even form $\underset{\sim}{Q}$ explicitly. The Gram-Schmidt method does in fact form $\underset{\sim}{Q}$ column by column, but as soon as a column has been utilized it may be discarded before the next column is formed and hence an extra storage space of only one m-vector is required.

Nearly always in our discussions $\underset{\sim}{Q}$ will be orthonormal ($\underset{\sim}{Q}^T\underset{\sim}{Q} = \underset{\sim}{I}$), rather than merely orthogonal ($\underset{\sim}{Q}^T\underset{\sim}{Q}$ diagonal). However, in accordance with custom we shall refer to an orthonormal $\underset{\sim}{Q}$ as orthogonal; we shall make clear cases where $\underset{\sim}{Q}$ is not orthonormal.

2.6 The modified Gram-Schmidt method

In this method the matrix $\underset{\sim}{Q}$ is determined explicitly. Let $\underset{\sim}{q}_j$ denote the jth column of $\underset{\sim}{Q}$. The computational process consists of n major steps in which the matrix $\underset{\sim}{A} = \underset{\sim}{A}^{(1)}$ is transformed successively to $\underset{\sim}{A}^{(2)}$, $\underset{\sim}{A}^{(3)}$, ..., $\underset{\sim}{A}^{(n+1)} = \underset{\sim}{R}$. At the beginning of the lth step (l = 1, 2, ..., n),

$$\underset{\sim}{A}^{(l)} = \left[\underset{\sim}{q}_1, \underset{\sim}{q}_2, \ldots, \underset{\sim}{q}_{l-1}, \underset{\sim}{a}_l^{(l)}, \underset{\sim}{a}_{l+1}^{(l)}, \ldots, \underset{\sim}{a}_n^{(l)}\right] , \tag{2.6.1}$$

where $\underset{\sim}{q}_1, \underset{\sim}{q}_2, \ldots, \underset{\sim}{q}_{l-1}$ are a set of orthonormalized vectors and $\underset{\sim}{a}_l^{(l)}$, $\underset{\sim}{a}_{l+1}^{(l)}, \ldots, \underset{\sim}{a}_n^{(l)}$ are modified versions of the corresponding columns $\underset{\sim}{a}_l^{(1)}$, $\underset{\sim}{a}_{l+1}^{(1)}, \ldots, \underset{\sim}{a}_n^{(1)}$ of the original matrix $\underset{\sim}{A}^{(1)}$.

The lth major step consists of (i) making the lth column a unit vector by replacing $\underset{\sim}{a}_l^{(l)}$ by

$$\underset{\sim}{q}_l = \underset{\sim}{a}_l^{(l)} / \left\| \underset{\sim}{a}_l^{(l)} \right\|_2 , \tag{2.6.2}$$

followed by (ii) n-l minor steps, the jth of which (j = l+1, l+2, ..., n) involves the orthogonalization of the jth column with respect to the lth column. It is easily verified that the computation

$$\underset{\sim}{a}_j^{(l+1)} = \underset{\sim}{a}_j^{(l)} - r_{lj}\underset{\sim}{q}_l , \tag{2.6.3}$$

where

$$r_{lj} = \underset{\sim}{q}_l^T \underset{\sim}{a}_j^{(l)} , \tag{2.6.4}$$

yields the required orthogonalization.

The same transformations are applied to the vector $\underset{\sim}{b}$, ie $\underset{\sim}{b}$ is treated just as if it were another column of $\underset{\sim}{A}$.

A formal statement of the complete process is given by Algorithm 2.6.1 below. During the lth major step column $\underset{\sim}{q}_l$ of $\underset{\sim}{Q}$ is held in the m-vector $\underset{\sim}{p}$. In this and subsequent algorithms we suppress superscripts and write eg

$$\text{Replace } a_{kj} \text{ by } a_{kj} - r_{lj}p_k \tag{2.6.5}$$

(as in Step 6 of Algorithm 2.6.1), rather than

$$a_{kj}^{(l+1)} = a_{kj}^{(l)} - r_{lj}p_k . \tag{2.6.6}$$

Apart from the advantage of brevity, the superscript-free notation also implies forcibly (as we wish to imply) that the old value of a_{kj}, ie $a_{kj}^{(l)}$, is <u>overwritten</u> by the new value, ie $a_{kj}^{(l+1)}$.

Algorithm 2.6.1: Orthogonal triangularization and linear least-squares solution using the modified Gram-Schmidt process.

Comment: The lth major step is described by Steps 2-8.

Step 1: For $l = 1, 2, \ldots, n$ execute Steps 2-8.

Comment: Column l is made a unit vector in Steps 2-3.

Step 2: Compute $r_{ll} = \left(\sum_{k=1}^{m} a_{kk}^2 \right)^{\frac{1}{2}}$.

Step 3: For $k = 1, 2, \ldots, m$ set $p_k = a_{kl}/r_{ll}$.

Comment: Row l of $\underset{\sim}{R}$ is formed and columns $l+1, l+2, \ldots, n$ of $\underset{\sim}{A}$ are orthogonalized with respect to column l in Steps 4-6.

Step 4: For $j = l+1, l+2, \ldots, n$ execute Steps 5-6.

Step 5: Form $r_{lj} = \sum_{k=1}^{m} p_k a_{kj}$.

Step 6: For $k = 1, 2, \ldots, m$ replace a_{kj} by $a_{kj} - r_{lj}p_k$.

Comment: Similar operations are applied to the right-hand side in Steps 7-8.

Step 7: Form $c_l = \sum_{k=1}^{m} p_k b_k$.

Step 8: For $k = 1, 2, \ldots, m$ replace b_k by $b_k - p_k c_l$.

Step 9: Use Algorithm 2.1.1 to solve $\underset{\sim}{R}\underset{\sim}{x} = \underset{\sim}{c}$.

The process described here is termed the modified Gram-Schmidt process to distinguish it from the classical procedure. The classical procedure differs from the modified method in that $\underset{\sim}{a}_j^{(1)}$ rather than $\underset{\sim}{a}_j^{(l)}$ is used to determine r_{lj}, with similar considerations applying to the right-hand side. Mathematically, the processes are identical; computationally, they behave very differently, the classical method being extremely unstable and the modified process a very reliable technique. In fact, the modified process is more convenient in practice than the classical procedure, since the initial columns $\underset{\sim}{a}_j^{(1)}$ do not have to be preserved but, as in Algorithm 2.6.1, can be overwritten by subsequent columns $\underset{\sim}{a}_j^{(l)}$.

In the above description of the modified Gram-Schmidt process, it is necessary to compute square roots. Modifications to the method have been proposed (Bauer, 1965; Björck, 1967) which obviate the necessity to compute these square roots. The basic idea is to form the decomposition

$$\underset{\sim}{A} = \hat{\underset{\sim}{Q}}\hat{\underset{\sim}{R}} , \tag{2.6.7}$$

where $(\hat{\underset{\sim}{Q}})^T\hat{\underset{\sim}{Q}}$ is diagonal, ie $\hat{\underset{\sim}{Q}}$ is generally orthogonal rather than orthonormal, and $\hat{\underset{\sim}{R}}$ is unit upper triangular. Equivalently, this decomposition may be expressed as

$$\underset{\sim}{A} = \underset{\sim}{Q}\underset{\sim}{D}^{\frac{1}{2}}\hat{\underset{\sim}{R}} , \tag{2.6.8}$$

where $\underset{\sim}{Q}$ is orthonormal and $\underset{\sim}{D} = \text{diag}\left\{d_1, d_2, \ldots, d_n\right\}$ is the diagonal matrix $(\hat{\underset{\sim}{Q}})^T\hat{\underset{\sim}{Q}}$. In terms of this decomposition the lth major step consists of (i) forming $d_l = \left\| \underset{\sim}{a}_l^{(l)} \right\|_2^2$, followed by (ii) n-1 minor steps, the jth of which ($j = l+1, l+2, \ldots, n$) involves forming $\underset{\sim}{a}_j^{(l+1)} = \underset{\sim}{a}_j^{(l)} - \hat{r}_{lj}\hat{\underset{\sim}{q}}_l$, where $\hat{r}_{lj} = (\hat{\underset{\sim}{q}}_j)^T \underset{\sim}{a}_j^{(l)}/d_l$. Algorithm 2.6.1 is readily modified to use this alternative decomposition.

The modified Gram-Schmidt algorithm and its square-root free variant are extremely satisfactory in practice. Indeed, in discussing the basic process (upon which Algorithm 2.6.1 is based), Peters and Wilkinson (1970) state that "Evidence is accumulating that the modified Gram-Schmidt gives better results than Householder in spite of the fact that the latter guarantees almost exact orthogonality of the columns of $\underset{\sim}{Q}$ while this is by no means true of the modified Gram-Schmidt procedure when $\underset{\sim}{A}$ has ill-conditioned columns. The reasons for this phenomenum appear not to have been elucidated yet." Despite this point in its favour the modified Gram-Schmidt process is not particularly appropriate in spline approximation problems. There are two reasons for this. Firstly, the organization of the modified Gram-Schmidt process is usually such that the complete matrix A is required in store at the start of the process; it does seem possible, however, to modify the method so that $\underset{\sim}{A}$ can be processed stably in a row-by-row manner, but it is an open question whether one will still have an efficient algorithm. Secondly, and more importantly, even if $\underset{\sim}{A}$ initially has a high proportion of zero elements, fill-in, ie the replacement of zero by non-zero elements, tends to occur so rapidly that little or no advantage can be taken of the structure of $\underset{\sim}{A}$.

2.7 The method of Householder transformations

Like the modified Gram-Schmidt method, the method of Householder transformations (Golub, 1965; Businger and Golub, 1965) consists of n major steps in which the matrix $\underset{\sim}{A} = \underset{\sim}{A}^{(1)}$ is transformed successively to $\underset{\sim}{A}^{(2)}, \underset{\sim}{A}^{(3)}, \ldots, \underset{\sim}{A}^{(n+1)} = \underset{\sim}{R}$. However, unlike the modified Gram-Schmidt method, it is unnecessary (except in special applications) to determine $\underset{\sim}{Q}$ explicitly.

At the beginning of the kth step, $\underset{\sim}{A}^{(k)}$ has the property that $a_{ij}^{(k)} = 0$ $(i = j+1, j+2, \ldots, m;\ j = 1, 2, \ldots, k-1)$, ie the first k-1 columns of $\underset{\sim}{A}^{(k)}$ are in "upper-triangular form". The kth step consists formally of pre-multiplying $\underset{\sim}{A}^{(k)}$ by the matrix

$$\underset{\sim}{P}^{(k)} = \underset{\sim}{I} - 2\underset{\sim}{w}^{(k)}(\underset{\sim}{w}^{(k)})^T / \left\| \underset{\sim}{w}^{(k)} \right\|_2^2 \tag{2.7.1}$$

to produce $\underset{\sim}{A}^{(k+1)}$.

Since

$$\begin{aligned}
(\underset{\sim}{P}^{(k)})^T\underset{\sim}{P}^{(k)} &= \left\{ \underset{\sim}{I} - 2\underset{\sim}{w}^{(k)}(\underset{\sim}{w}^{(k)})^T / \left\| \underset{\sim}{w}^{(k)} \right\|_2^2 \right\}^2 \\
&= \underset{\sim}{I} - 4\underset{\sim}{w}^{(k)}(\underset{\sim}{w}^{(k)})^T / \left\| \underset{\sim}{w}^{(k)} \right\|_2^2 \\
&\quad + 4\underset{\sim}{w}^{(k)} \left\{ (\underset{\sim}{w}^{(k)})^T\underset{\sim}{w}^{(k)} \right\} (\underset{\sim}{w}^{(k)})^T / \left\| \underset{\sim}{w}^{(k)} \right\|_2^4 \\
&= \underset{\sim}{I} - 4\underset{\sim}{w}^{(k)}(\underset{\sim}{w}^{(k)})^T / \left\| \underset{\sim}{w}^{(k)} \right\|_2^2 + 4\underset{\sim}{w}^{(k)}(\underset{\sim}{w}^{(k)})^T / \left\| \underset{\sim}{w}^{(k)} \right\|_2^2 \\
&= \underset{\sim}{I} \ ,
\end{aligned} \tag{2.7.2}$$

it follows that $\underset{\sim}{P}^{(k)}$ is orthogonal and symmetric.

The vector $\underset{\sim}{w}^{(k)}$ is chosen to annihilate the elements $a_{ik}^{(k)}$ (i = k+1, k+2, ..., m). It is easily verified that the vector $\underset{\sim}{w}^{(k)}$ defined by

$$w_i^{(k)} = \begin{cases} 0 & (i = 1, 2, \ldots, k-1) \\ \mathrm{sgn}\,(a_{kk}^{(k)})(\sigma^{(k)} + a_{kk}^{(k)}) & (i = k) \\ a_{ik}^{(k)} & (i = k+1, k+2, \ldots, m)\ , \end{cases} \tag{2.7.3}$$

where

$$(\sigma^{(k)})^2 = \sum_{i=k}^{m} (a_{ik}^{(k)})^2 \tag{2.7.4}$$

and

$$\mathrm{sgn}(x) = \begin{cases} -1 & (x < 0) \\ 0 & (x = 0) \\ +1 & (x > 0)\ , \end{cases} \tag{2.7.5}$$

possesses the above annihilation property.

Now since

$$\left\| \underset{\sim}{w}^{(k)} \right\|_2^2 = (\sigma^{(k)})^2 + 2\sigma^{(k)} \left| a_{kk}^{(k)} \right| + \sum_{i=k}^{m} (a_{ik}^{(k)})^2$$

$$= 2\sigma^{(k)}(\sigma^{(k)} + \left| a_{kk}^{(k)} \right|), \tag{2.7.6}$$

we may write

$$\underset{\sim}{P}^{(k)} = \underset{\sim}{I} - \beta^{(k)} \underset{\sim}{w}^{(k)} (\underset{\sim}{w}^{(k)})^T\ , \tag{2.7.7}$$

where

$$\beta^{(k)} = \left\{ \sigma^{(k)} (\sigma^{(k)} + \left| a_{kk}^{(k)} \right|) \right\}^{-1}\ . \tag{2.7.8}$$

It is both inefficient and unnecessary to compute $\underset{\sim}{P}^{(k)}$ explicitly; rather, we compute first

$$(\underset{\sim}{y}^{(k)})^T = \beta^{(k)}(\underset{\sim}{w}^{(k)})^T \underset{\sim}{A}^{(k)} \qquad (2.7.9)$$

and then

$$\underset{\sim}{A}^{(k+1)} = \underset{\sim}{A}^{(k)} - \underset{\sim}{w}^{(k)}(\underset{\sim}{y}^{(k)})^T . \qquad (2.7.10)$$

As with the modified Gram-Schmidt method, the same transformations are applied to the vector $\underset{\sim}{b} = \underset{\sim}{b}^{(1)}$ to yield successive vectors $\underset{\sim}{b}^{(2)}$, $\underset{\sim}{b}^{(3)}$, ..., $\underset{\sim}{b}^{(n+1)} = \underset{\sim}{c}$. The right-triangular system

$$\underset{\sim}{R}\underset{\sim}{x} = \underset{\sim}{c} \qquad (2.7.11)$$

is then solved for $\underset{\sim}{x}$.

Algorith 2.7.1 below is a statement of the method of this section. The initial matrix $\underset{\sim}{A} = \underset{\sim}{A}^{(1)}$ is successively overwritten by $\underset{\sim}{A}^{(2)}$, $\underset{\sim}{A}^{(3)}$, ..., $\underset{\sim}{A}^{(n+1)} = \underset{\sim}{R}$. Likewise, the right-hand side $\underset{\sim}{b} = \underset{\sim}{b}^{(1)}$ is successively overwritten by $\underset{\sim}{b}^{(2)}$, $\underset{\sim}{b}^{(3)}$, ..., $\underset{\sim}{b}^{(n+1)} = c$.

Algorithm 2.7.1: Orthogonal triangularization and linear least-squares solution using Householder transformations.

Comment: The kth major step is described by Steps 2-11.

Step 1. For $k = 1, 2, \ldots, n$ execute Steps 2-11.

Comment: The parameters of $\underset{\sim}{P}^{(k)}$ are formed in Steps 2-6.

Step 2. Form $\sigma = \left(\sum_{i=k}^{m} a_{ik}^2 \right)^{\frac{1}{2}}$.

Step 3. Form $\alpha = \sigma + |a_{kk}|$.

Step 4. Form $w_k = \alpha \operatorname{sgn}(a_{kk})$.

Step 5. For $i = k+1, k+2, \ldots, m$ set $w_i = a_{ik}$.

Step 6. Form $\beta = (\alpha\sigma)^{-1}$ and replace a_{kk} by α .

Comment: The transformation $\underset{\sim}{P}^{(k)}\underset{\sim}{A}^{(k)}$ is carried out in Steps 7-9.

Step 7. For $j = k+1, k+2, \ldots, n$ execute Steps 8-9.

Step 8. Compute $y = \beta \sum_{i=k}^{m} w_i a_{ij}$.

Step 9. For $i = k, k+1, \ldots, m$ replace a_{ij} by $a_{ij} - yw_i$.

Comment: The same transformation is applied to the right-hand side in Steps 10-11.

Step 10. Compute $y = \beta \sum_{i=k}^{m} w_i b_i$.

Step 11. For $i = k, k+1, \ldots, m$ replace b_i by $b_i - yw_i$.

Step 12. Use Algorithm 2.1.1 to solve $\underset{\sim}{R}\underset{\sim}{x} = \underset{\sim}{c}$ ($\underset{\sim}{R}$ stored in $\underset{\sim}{A}$, $\underset{\sim}{c}$ in $\underset{\sim}{b}$).

2.8 Classical plane rotations

In this section we consider classical Givens plane rotations and give two algorithms based on their use for the orthogonal triangularization of an m by n matrix $\underset{\sim}{A}$ of rank n and for the least-squares solution of the over-determined linear system $\underset{\sim}{A}\underset{\sim}{x} = \underset{\sim}{b}$. In Section 2.9 we examine variants of the modern form of plane rotations without square roots and again give algorithms based on their use for the orthogonal triangularization of $\underset{\sim}{A}$. In Section 2.10 we compare the use of both classical and modern forms of plane rotations with the more common methods described in Sections 2.6 and 2.7, and give reasons why we believe that plane rotations are particularly appropriate for solving the types of least-squares systems arising from spline approximation problems.

Consider the pre-multiplication of the m by n matrix $\underset{\sim}{A}$ by the orthogonal m by m matrix

$$\underset{\sim}{Q}_{ij} = \begin{array}{cc} & \begin{array}{ccccccccccc} & & & i & & & & j & & & \end{array} \\ \begin{array}{c} \\ \\ \\ i \\ \\ \\ \\ j \\ \\ \\ \\ \end{array} & \left[\begin{array}{ccccccccccc} 1 &&&&&&&&&& \\ &\ddots&&&&&&&&& \\ &&1&&&&&&&& \\ &&&c&&&&s&&& \\ &&&&1&&&&&& \\ &&&&&\ddots&&&&& \\ &&&&&&1&&&& \\ &&&-s&&&&c&&& \\ &&&&&&&&1&& \\ &&&&&&&&&\ddots& \\ &&&&&&&&&&1 \end{array}\right] \end{array}, \qquad (2.8.1)$$

where c and s denote cos θ and sin θ, respectively, and θ is chosen such that the element in position (j, i) of the matrix

$$\underset{\sim}{A}' = \underset{\sim}{Q}_{ij}\underset{\sim}{A} \tag{2.8.2}$$

is zero. It is straightforward to verify that the appropriate values of c and s are given by

$$c = a_{ii}/h , \tag{2.8.3}$$

$$s = a_{ji}/h , \tag{2.8.4}$$

where

$$h = (a_{ii}^2 + a_{ji}^2)^{\frac{1}{2}} . \tag{2.8.5}$$

Here we assume that $a_{ji} \neq 0$, which ensures that h is non-zero. Note that if a_{ji} is already zero then no rotation is needed, in which case we take c = 1 and s = 0 (even if $a_{ii} = 0$), so that $\underset{\sim}{Q}_{ij}$ reduces to the unit matrix. Only rows i and j of $\underset{\sim}{A}$ are altered by the transformation, the effect being to replace row i by (c × row i + s × row j) and row j by (c × row j − s × row i). It follows therefore that if both rows i and j have zeros in the same column position, then these zeros are undisturbed by the process.

The rotation can be described completely by (2.8.3), (2.8.4) and (2.8.5), together with the expressions

$$a'_{ii} = h \tag{2.8.6}$$

$$\left.\begin{aligned} a'_{ik} &= ca_{ik} + sa_{jk} \\ a'_{jk} &= -sa_{ik} + ca_{jk} \end{aligned}\right\} (k = i+1, i+2, \ldots, n). \tag{2.8.7}$$

We term (2.8.7) a <u>4-multiplication rule</u>, since for each value of k four multiplications are required to evaluate a'_{ik} and a'_{jk} from a_{ik} and a_{jk}.

Wilkinson (1965: 131 et seq) has shown that the 4-multiplication rule is unconditionally stable in that if a_{ik} and a_{jk} $(k = i, i+1, \ldots, n)$ are specified then

$$\left\| \begin{matrix} fl(a'_{ik}) - a'_{ik} \\ \\ fl(a'_{jk}) - a'_{jk} \end{matrix} \right\|_2 \leqslant 6 \left\| \begin{matrix} a_{ik} \\ \\ a_{jk} \end{matrix} \right\|_2 2^{-t} , \tag{2.8.8}$$

where the factor of 6 is, according to Wilkinson, extremely generous. Since the Euclidean norm is invariant with respect to orthogonal transformation, the right-hand side of (2.8.8) can, apart from a multiplicative factor of $(1+\varepsilon)^6$ $(|\varepsilon| \leqslant 2^{-t})$, be replaced by

$$6 \left\| \begin{matrix} a'_{ik} \\ \\ a'_{jk} \end{matrix} \right\|_2 2^{-t} . \tag{2.8.9}$$

Thus making the very mild assumption that the factor $(1+\varepsilon)^6$ can be absorbed into the "generous" factor of 6, the relative error in the 4-multiplication rule is bounded in modulus by $(6)2^{-t}$.

Pre-multiplication of $\underset{\sim}{A}$ by $\underset{\sim}{Q}_{ij}$ is termed a rotation in the (i,j)-plane. We also refer to this pre-multiplication as the rotation of row j into row i.

Two variants of the class of methods which employ classical plane rotations may be described as follows.

In the first method, which we term triangularization by columns, there are n major steps. The kth major step $(k = 1, 2, \ldots, n)$ consists of m-k minor steps, the ith of which $(i = k+1, k+2, \ldots, m)$ has the effect of reducing element a_{ik} to zero, whilst preserving zeros established in previous steps.

In the second method, which we shall refer to as triangularization by rows, there are m-1 major steps. The ith major step ($i = 2, 3, \ldots, m$) consists of at most n minor steps, the kth of which ($k = 1, 2, \ldots, \min(i-1, n)$) has the effect of annihilating element a_{ik}, whilst preserving previously-established zeros.

In either of these two methods a minor step consists of a single plane rotation. In that the sub-diagonal elements in successive columns are reduced to zero, the first method is analogous to the modified Gram-Schmidt and the Householder methods. On the other hand, the second of the two methods, in which elements to the left of the main diagonal in successive rows are annihilated, has no natural correspondence with the other orthogonalization methods.

If, in either of the two methods, the same sequence of rotations is performed upon the vector $\underset{\sim}{b}$ (by treating it essentially as another column of $\underset{\sim}{A}$), then the least-squares solution is given by the solution of the system $\underset{\sim}{R}\underset{\sim}{x} = \underset{\sim}{c}$, where $\underset{\sim}{R}$ denotes the triangle ultimately produced in the first n rows of $\underset{\sim}{A}$ and $\underset{\sim}{c}$ the first n elements of the transformed vector $\underset{\sim}{b}$.

We now present algorithms based on these two methods.

Algorithm 2.8.1: Orthogonal triangularization by columns and linear least-squares solution using classical plane rotations.

Comment: The kth major step is described by Steps 2-11.

Step 1. For $k = 1, 2, \ldots, n$ execute Steps 2-11.

Comment: The ith minor step is described by Steps 3-10.

Step 2. For $i = k+1, k+2, \ldots, m$ execute Steps 3-10.

Comment: A rotation is skipped if a_{ik} is already zero.

Step 3. If $a_{ik} = 0$ advance to Step 10.

Comment: The plane rotation annihilating a_{ik} is applied in Steps 4-7.

Step 4. Compute $h = (a_{kk}^2 + a_{ik}^2)^{\frac{1}{2}}$. Form $c = a_{kk}/h$ and $s = a_{ik}/h$.

Replace a_{kk} by h.

Step 5. For $j = k+1, k+2, \ldots, n$ execute Steps 6-7.

Step 6. Set $y = a_{kj}$ and $z = a_{ij}$.

Step 7. Replace a_{kj} by $cy + sz$ and a_{ij} by $cz - sy$.

Comment: The same rotation is applied to the right-hand side in Steps 8-9.

Step 8. Set $y = b_k$ and $z = b_i$.

Step 9. Replace b_k by $cy + sz$ and b_i by $cz - sy$.

Step 10. Continue.

Step 11. Continue.

Step 12. Use Algorithm 2.1.1 to solve $\underset{\sim}{R}\underset{\sim}{x} = \underset{\sim}{c}$ ($\underset{\sim}{R}$ stored in $\underset{\sim}{A}$, $\underset{\sim}{c}$ in $\underset{\sim}{b}$).

Algorithm 2.8.2: Orthogonal triangularization by rows and linear least-squares solution using classical plane rotations.

Comment: The ith major step is described by Steps 2-11.

Step 1. For $i = 2, 3, \ldots, m$ execute Steps 2-11.

Comment: The kth minor step is described by Steps 3-10.

Step 2. For $k = 1, 2, \ldots, \min(i-1, n)$ execute Steps 3-10.

Steps 3-12. As Steps 3-12 of Algorithm 2.8.1.

The above methods for orthogonal triangularization by columns and by rows have their analogues when modern forms of plane rotations (Section 2.9) are used. The main differences relate to the nature of the arithmetic operations within individual rotations, the overall strategies, ie the orders in which the sub-diagonal elements are annihilated, being unchanged.

2.9 Modern plane rotations

The more general factorization

$$\underset{\sim}{A} = \underset{\sim}{Q}\underset{\sim}{D}^{\frac{1}{2}}\hat{\underset{\sim}{R}} , \qquad (2.9.1)$$

considered in Section 2.6 in terms of the modified Gram-Schmidt method, can also be formed using a generalization of the method of plane rotations. As in Section 2.6, $\underset{\sim}{D}$ is a diagonal matrix with non-negative elements, $\underset{\sim}{Q}$ is orthogonal and $\hat{\underset{\sim}{R}}$ upper-triangular. The factorization (2.9.1) has more degrees of freedom associated with it than the usual factorization $\underset{\sim}{A} = \underset{\sim}{Q}\underset{\sim}{R}$. These degrees of freedom may be used to advantage in a number of ways. The factorization evidently includes the classical form as a special case, viz when $\underset{\sim}{D} = \underset{\sim}{I}$. $\hat{\underset{\sim}{R}}$ can be made unit upper-triangular by setting the diagonal elements of $\underset{\sim}{D}$ equal to the squares of the diagonal elements of $\underset{\sim}{R}$. Other choices of $\underset{\sim}{D}$ and $\hat{\underset{\sim}{R}}$ enable not only the square roots in the plane rotation method to be avoided, but also the number of multiplications to be reduced by either 25% or 50% (Gentleman, 1972, 1973; Hammarling, 1974). The 50% reduction makes Givens rotations as attractive arithmetically as Householder transformations and the modified Gram-Schmidt method for solving linear least-squares problems.

To examine the generalized class of rotations, suppose that immediately before the rotation,

$$\underset{\sim}{A} = \underset{\sim}{D}^{\frac{1}{2}}\underset{\sim}{G} \qquad (2.9.2)$$

and that after the rotation

$$\underset{\sim}{A}' = \underset{\sim}{Q}\underset{\sim}{A} \qquad (2.9.3)$$

we have

$$\underset{\sim}{A}' = (\underset{\sim}{D}')^{\frac{1}{2}}\underset{\sim}{G}' . \qquad (2.9.4)$$

Both $\underset{\sim}{D}$ and $\underset{\sim}{D}'$ denote diagonal matrices with non-negative elements. We wish to determine formulae for computing those elements of $\underset{\sim}{D}'$ and $\underset{\sim}{G}'$ changed by the transformation in terms of those of $\underset{\sim}{D}$ and $\underset{\sim}{G}$. Now since

$$a_{ij} = d_i^{\frac{1}{2}} g_{ij} \tag{2.9.5}$$

and

$$a'_{ij} = (d'_i)^{\frac{1}{2}} g'_{ij} \quad , \tag{2.9.6}$$

the counterparts of (2.8.3) to (2.8.7) are

$$c = d_i^{\frac{1}{2}} g_{ii}/h \ , \tag{2.9.7}$$

$$s = d_j^{\frac{1}{2}} g_{ji}/h \ , \tag{2.9.8}$$

$$h = (d_i g_{ii}^2 + d_j g_{ji}^2)^{\frac{1}{2}} \ , \tag{2.9.9}$$

$$g'_{ii} = h/(d'_i)^{\frac{1}{2}} \ , \tag{2.9.10}$$

$$\left.\begin{aligned} g'_{ik} &= (d_i g_{ii} g_{ik} + d_j g_{ji} g_{jk}) / \left\{ h(d'_i)^{\frac{1}{2}} \right\} \\ g'_{jk} &= d_i^{\frac{1}{2}} d_j^{\frac{1}{2}} (g_{ii} g_{jk} - g_{ji} g_{ik}) / \left\{ h(d'_j)^{\frac{1}{2}} \right\} \end{aligned}\right\} (k = i+1,\ i+2,\ \ldots,\ n). \tag{2.9.11}$$

Suppose that $\underset{\sim}{D}$ and $\underset{\sim}{G}$ are given, and that we have freedom just in our choice of $\underset{\sim}{D}'$.

Gentleman (1972, 1973) chooses

$$d'_i = h^2 \ , \tag{2.9.12}$$

$$d'_j = d_i d_j / h^2 \ . \tag{2.9.13}$$

Then (2.9.10) and (2.9.11) become

$$g'_{ii} = 1, \tag{2.9.14}$$

$$\left.\begin{aligned} g'_{ik} &= \left(\frac{d_i g_{ii}}{h^2}\right) g_{ik} + \left(\frac{d_j g_{ji}}{h^2}\right) g_{jk} \\ g'_{jk} &= g_{ii} g_{jk} - g_{ji} g_{ik} \end{aligned}\right\} \quad (k = i+1,\ i+2,\ \ldots,\ n). \qquad (2.9.15)$$

But if previous rotations made $g_{ii} = 1$ then (2.9.15) reduces to

$$\left.\begin{aligned} g'_{ik} &= \left(\frac{d_i}{h^2}\right) g_{ik} + \left(\frac{d_j g_{ji}}{h^2}\right) g_{jk} \\ g'_{jk} &= g_{jk} - g_{ji} g_{ik} \end{aligned}\right\} \quad (k = i+1,\ i+2,\ \ldots,\ n). \qquad (2.9.16)$$

As a consequence, Gentleman's rotation is defined by the relations

$$d'_i = d_i + d_j g_{ji}^2 , \qquad (2.9.17)$$

$$d'_j = d_i d_j / d'_i , \qquad (2.9.18)$$

$$\hat{c} = d_i / d'_i , \qquad (2.9.19)$$

$$\hat{s} = d_j g_{ji} / d'_i , \qquad (2.9.20)$$

$$\left.\begin{aligned} g'_{ik} &= \hat{c} g_{ik} + \hat{s} g_{jk} \\ g'_{jk} &= g_{jk} - g_{ji} g_{ik} \end{aligned}\right\} \quad (k = i+1,\ i+2,\ \ldots,\ n). \qquad (2.9.21)$$

This is a <u>3-multiplication rule</u>.

Gentleman (1973) has shown that his 3-multiplication rule is unconditionally numerically stable in that

$$\left\| \begin{matrix} fl\left\{(d'_i)^{\frac{1}{2}}\right\} fl(g'_{ik}) - (d'_i)^{\frac{1}{2}} g'_{ik} \\ \\ fl\left\{(d'_j)^{\frac{1}{2}}\right\} fl(g'_{jk}) - (d'_j)^{\frac{1}{2}} g'_{jk} \end{matrix} \right\|_2 \leqslant 7.5 \left\| \begin{matrix} d_i^{\frac{1}{2}} g_{ik} \\ \\ d_j^{\frac{1}{2}} g_{jk} \end{matrix} \right\|_2 2^{-t} . \qquad (2.9.22)$$

This result*, which should be compared with (2.8.8), shows that the 3-multiplication rule and the classical 4-multiplication rule have comparable stability.

Golub (private communication - see Gentleman, 1973) has pointed out that the arithmetic involved in relations (2.9.21) may be reduced by observing that

$$g'_{ik} = \hat{c}g_{ik} + \hat{s}(g'_{jk} + g_{ji}g_{ik}) \tag{2.9.23}$$

$$= g_{ik} + sg'_{jk} , \tag{2.9.24}$$

upon using (2.9.17), (2.9.19) and (2.9.20). Thus Golub's form of the rotation may be defined by the relations

$$d'_i = d_i + d_j g_{ji}^2 , \tag{2.9.25}$$

$$d'_j = d_i d_j / d'_i , \tag{2.9.26}$$

$$\hat{s} = d_j g_{ji} / d'_i , \tag{2.9.27}$$

$$\left.\begin{aligned} g'_{jk} &= g_{jk} - g_{ji}g_{ik} \\ g'_{ik} &= g_{ik} + sg'_{jk} \end{aligned}\right\} \quad (k = i+1,\ i+2,\ \ldots,\ n). \tag{2.9.28}$$

This is a <u>2-multiplication rule</u> .

Gentleman (1973) has carried out a floating-point error analysis of Golub's rule and has shown that

---

* Gentleman (1974) has subsequently shown that the factor of 7.5 in (2.9.22) may be improved to a value of 4.5.

$$\left\| \begin{matrix} fl\left\{(d_i')^{\frac{1}{2}}\right\} fl(g_{ik}') - (d_i')^{\frac{1}{2}} g_{ik}' \\ \\ fl\left\{(d_j')^{\frac{1}{2}}\right\} fl(g_{jk}') - (d_j')^{\frac{1}{2}} g_{jk}' \end{matrix} \right\|_2 \leqslant K(d_i, d_i') \left\| \begin{matrix} d_i^{\frac{1}{2}} g_{ik} \\ \\ d_j^{\frac{1}{2}} g_{jk} \end{matrix} \right\|_2 2^{-t}, \qquad (2.9.29)$$

where

$$\left\{K(d_i, d_i')\right\}^2 = (4.52)^2 + \left\{4.52 + 8.04(d_i'/d_i)^{\frac{1}{2}}\right\}^2, \qquad (2.9.30)$$

from which it is clear that the stability of the rule depends upon the relative magnitudes of d_i' and d_i. Gentleman (1973) states that this 2-multiplication rule is "numerically unstable, producing terrible results for least squares problems with very well conditioned design matrices". Hammarling (1974) gives a simple example to illustrate this point. Gentleman suggests that since the instability can readily be detected, simply by examining the ratio d_i'/d_i, then we can cut cost <u>and</u> preserve stability by using the 2-multiplication rule if $d_i'/d_i \leqslant 100$, say, and the 3-multiplication rule otherwise. If this strategy is employed then relation (2.9.22) holds with 7.5 replaced by 85.04.

Hammarling (1974) has considered choices of d_i' and d_j' that lead directly to 2-multiplication rules. The choice

$$d_i' = d_j^2 g_{ji}^2 / h^2, \qquad (2.9.31)$$

$$d_j' = d_i d_j g_{ii}^2 / h^2 \qquad (2.9.32)$$

reduces (2.9.11) to

$$\left.\begin{aligned} g_{ik}' &= g_{ik} + \left(\frac{d_j g_{ji}}{d_i g_{ii}}\right) g_{jk} \\ g_{jk}' &= g_{jk} - \left(\frac{g_{ji}}{g_{ii}}\right) g_{ik} \end{aligned}\right\} \quad (k = i+1, i+2, \ldots, n). \qquad (2.9.33)$$

According to Hammarling, the other choices of d_i' and d_j', of which there are five in all, lead to similar relations. Hammarling's rotation may be defined by the relations

$$\hat{u} = g_{ji}/g_{ii} , \tag{2.9.34}$$

$$\hat{s} = \hat{u}d_j/d_i , \tag{2.9.35}$$

$$d_i' = d_i/(1 + \hat{s}\hat{u}) , \tag{2.9.36}$$

$$d_j' = d_j/(1 + \hat{s}\hat{u}) , \tag{2.9.37}$$

$$\left.\begin{aligned} g_{ik}' &= g_{ik} + \hat{s}g_{jk} \\ g_{jk}' &= g_{jk} - \hat{u}g_{ik} \end{aligned}\right\} (k = i+1,\ i+2,\ \ldots,\ n). \tag{2.9.38}$$

Although Hammarling demonstrates the stability of his rule, he states that there is clearly some danger of underflow in d_i' and d_j' when a sequence of rotations is involved. He suggests, without giving specific details, that this danger may be avoided by storing the exponents of $\underset{\sim}{D}$ separately, by normalizing occasionally or by performing row interchanges. Moler (1974) has recently given details of a row interchange strategy which reduces the danger of underflow in the 2-multiplication rule. However, even in Moler's algorithm underflow <u>can</u> occur and hence periodic testing should be incorporated to see whether scaling is required.

Before we give algorithms for the modern Givens rules we describe an algorithm for orthogonal triangularization by rows using classical plane rotations which has storage requirements independent of m. The basic idea is due to Gentleman (1972). Now the solutions to the problems of minimizing $\underset{\sim}{r}^T\underset{\sim}{r}$, where $\underset{\sim}{r}$ is given by (2.2.2), or by

$$\underset{\sim}{r} = \begin{bmatrix} \underset{\sim}{O} \\ \underset{\sim}{A} \end{bmatrix} \underset{\sim}{x} - \begin{bmatrix} \underset{\sim}{0} \\ \underset{\sim}{b} \end{bmatrix} ,$$

are evidently identical. Thus we can determine the required least-squares solution by initializing $\underset{\sim}{R}$ and $\underset{\sim}{\theta}$ to zero, and then rotating each successive row of $(\underset{\sim}{A} \mid \underset{\sim}{b})$ into $(\underset{\sim}{R} \mid \underset{\sim}{\theta})$. This scheme has the advantage that the only storage required, assuming that each row of $(\underset{\sim}{A} \mid \underset{\sim}{b})$ can be input or formed when needed, is $\frac{1}{2}n(n+1)$ words for $\underset{\sim}{R}$, n for $\underset{\sim}{\theta}$ and n+1 for the current row of $(\underset{\sim}{A} \mid \underset{\sim}{b})$. Thus the total storage for such a scheme is $\frac{1}{2}n(n+5)+O(1)$ words. This is to be compared with the storage required for the column by column process which requires $mn+O(m)+O(n)$ words.

A worthwhile saving in arithmetic can be made if a rotation involving a null row is treated specially. Suppose, in the notation of Section 2.8, that row i is null and that $a_{ji} \neq 0$ (if $a_{ji} = 0$ a rotation is not required). Then from (2.8.3), (2.8.4) and (2.8.5), $c = 0$ and $s = 1$, with the result that (2.8.6) and (2.8.7) reduce to

$$a'_{ii} = a_{ji}, \tag{2.9.39}$$

$$\left.\begin{array}{l} a'_{ik} = a_{jk} \\ \\ a'_{jk} = 0 \end{array}\right\} \quad (k = i+1, i+2, \ldots, n). \tag{2.9.40}$$

Thus, since $a'_{ji} = 0$, the effect of the rotation is to interchange rows i and j. Note that there is an ambiguity associated with the sign of s. Here we have chosen $s = +1$. The choice $s = -1$ could also be made, the only difference being that all values in row i are negated. In either case no further rotations involving row j are necessary since it is now null. This refinement and its counterparts for the modern Givens rules have been incorporated in Algorithms 2.9.1 to 2.9.4 below.

In Algorithm 2.9.1 the upper-triangular matrix $\underset{\sim}{R}$ is stored by rows in the vector r_l $(l = 1, 2, \ldots, \frac{1}{2}n(n+1))$. The ith row of $(\underset{\sim}{A} \mid \underset{\sim}{b})$ is read into or formed in locations v_j $(j = 1, 2, \ldots, n)$ and $\underset{\sim}{b}$ and the associated weight (often unity) in w. The minimum sum of squares is formed in σ.

Algorithm 2.9.1: Orthogonal triangularization by rows and linear least-squares solution using classical plane rotations (vector storage strategy).

Comment: $\underset{\sim}{R}$, $\underset{\sim}{\theta}$ and σ are initialized to zero.

Step 1. For $l = 1, 2, \ldots, \frac{1}{2}n(n+1)$ set $r_l = 0$ and for $j = 1, 2, \ldots, n$ set $\theta_j = 0$. Set $\sigma = 0$.

Comment: The ith major step is described by Steps 3-30.

Step 2. For $i = 1, 2, \ldots, m$ execute Steps 3-30.

Comment: The ith row of $(\underset{\sim}{A} \,|\, \underset{\sim}{b})$ and the corresponding weight are read or formed.

Step 3. Read or form the ith row $w, v_1, v_2, \ldots, v_n, b$.

Comment: No operations on row i are required if w is zero.

Step 4. If $w = 0$ advance to Step 30.

Comment: The weight is incorporated in row i in Steps 5-8.

Step 5. If $w = 1$ advance to Step 9.

Step 6. Set $z = w^{\frac{1}{2}}$.

Step 7. For $j = 1, 2, \ldots, n$ replace v_j by zv_j.

Step 8. Replace b by zb.

Comment: The jth minor step is described by Steps 10-28.

Step 9. For $j = 1, 2, \ldots, n$ execute Steps 10-28.

Comment: A rotation is skipped if a_{ij} is already zero.

Step 10. If $v_j = 0$ advance to Step 28.

Comment: Element r_{jj} is stored as r_l.

Step 11. Set $l = (j-1)(2n+2-j)/2+1$.

Comment: The algorithm branches according to whether r_{jj} is zero or non-zero.

Step 12. If $r_l \neq 0$ advance to Step 19.

Comment: In the case $r_{jj} = 0$ row j of $(R \,|\, \theta)$ is replaced by row i of $(\underset{\sim}{A} \,|\, \underset{\sim}{b})$ in Steps 13-17.

Step 13. Set $r_l = v_j$.

Step 14. For $k = j+1, j+2, \ldots, n$ execute Steps 15–16.

Step 15. Replace l by $l+1$.

Step 16. Replace r_l by v_k.

Step 17. Replace θ_j by b.

Comment: No further rotations involving row i of $(A|b)$ are required.

Step 18. Advance to Step 30.

Comment: In the case $r_{jj} \neq 0$ a conventional rotation to annihilate a_{ij} is carried out in Steps 19–28.

Step 19. Set $g = (r_l^2 + v_j^2)^{\frac{1}{2}}$.

Step 20. Set $c = r_l/g$, $s = v_j/g$.

Step 21. Set $r_l = g$.

Step 22. For $k = j+1, j+2, \ldots, n$ execute Steps 23–25.

Step 23. Replace l by $l+1$.

Step 24. Set $y = r_l$ and $z = v_k$.

Step 25. Replace r_l by $cy+sz$ and v_k by $cz-sy$.

Step 26. Set $y = \theta_j$ and $z = b$.

Step 27. Replace θ_j by $cy+sz$ and b by $cz-sy$.

Step 28. Continue.

Comment: The residual sum of squares is updated.

Step 29. Replace σ by $\sigma + b^2$.

Step 30. Continue.

Step 31. Use Algorithm 2.1.2 to solve $\underset{\sim}{R}\underset{\sim}{x} = \underset{\sim}{\theta}$.

Although there is no possibility of "element growth" with orthogonalization methods, another problem may arise. In the classical Givens method this problem is associated with the computation of the parameters c and s from (2.8.3), (2.8.4) and (2.8.5). Even if the values of a_{ii} and a_{ji} are well within the number range of the machine, overflow or underflow may result when they are squared*. On the KDF9 computer, for example, the number range is 2^{-2^7} to 2^{2^7}, ie approximately 10^{-38} to 10^{38}. Thus if $|a_{ii}|$ or $|a_{ji}| < 10^{-20}$ (say) underflow will occur and if $|a_{ii}|$ or $|a_{ji}| > 10^{20}$ overflow will result. Overflow is serious in that the computation will be halted, but underflow is dangerous (unless both a_{ii}^2 and a_{ji}^2 underflow) in that on many machines the computation will continue without warning and erroneous results produced. The situation is easily remedied, however, by using, instead of (2.8.3), (2.8.4) and (2.8.5),

$$c = \operatorname{sgn} a_{ii} \left\{1 + (a_{ji}/a_{ii})^2\right\}^{-\frac{1}{2}}, \tag{2.9.41}$$

$$s = ca_{ji}/a_{ii} \tag{2.9.42}$$

if $|a_{ji}| \leq |a_{ii}|$, and

$$s = \operatorname{sgn} a_{ji} \left\{1 + (a_{ii}/a_{ji})^2\right\}^{-\frac{1}{2}}, \tag{2.9.43}$$

$$c = sa_{ii}/a_{ji} \tag{2.9.44}$$

if $|a_{ji}| > |a_{ii}|$. The values of a'_{ii} is then formed from

$$a'_{ii} = \begin{cases} a_{ii}/c & (|a_{ji}| \leq |a_{ii}|) \\ \\ a_{ji}/s & (|a_{ji}| > |a_{ii}|) \end{cases}, \tag{2.9.45}$$

rather than from (2.8.6). We have assumed of course that a_{ii} and a_{ji} are non-zero, since if either is zero special action is taken anyway.

---

* The problem is also present in the modified Gram-Schmidt and Householder methods, where the 2-norms of certain vectors have to be formed. One way of overcoming the problem in these cases is, before forming the sum of the squares of the vector elements, to search for the element of largest magnitude and to divide each element by this value. The 2-norm so obtained is then multiplied by the modulus of the element of largest magnitude.

Algorithm 2.9.1 is modified accordingly by replacing Steps 19-21 by those in Algorithm 2.9.2 below.

<u>Algorithm 2.9.2</u>: Orthogonal triangularization by rows and linear least-squares solution using classical plane rotations with overflow/underflow prevention (vector storage strategy).

Steps 1-18. As Steps 1-18 of Algorithm 2.9.1.

Step 19. If $|r_1| \geqslant |v_j|$ advance to Step 21.1.

Step 20.1. Set $g = r_1/v_j$.

Step 20.2. Set $s = \operatorname{sgn} v_j \; (1 + g^2)^{-\frac{1}{2}}$.

Step 20.3. Set $c = sg$ and replace r_1 by v_j/s.

Step 20.4. Advance to Step 22.

Step 21.1. Set $g = v_j/r_1$.

Step 21.2. Set $c = \operatorname{sgn} r_1 \; (1+g^2)^{-\frac{1}{2}}$.

Step 21.3. Set $s = cg$ and replace r_1 by r_1/c.

Steps 22-31. As Steps 22-31 of Algorithm 2.9.1.

Further algorithms based on classical plane rotations, presented in subsequent sections, can also be modified in this way.

In Algorithm 2.9.3, which implements the Gentleman 3-multiplication rule, successive rows of $\underset{\sim}{W}^{\frac{1}{2}}(\underset{\sim}{A}|\underset{\sim}{b})$, where $\underset{\sim}{W} = \operatorname{diag}(w_1, w_2, \ldots, w_m)$ denotes a matrix of non-negative weights, are rotated into $\underset{\sim}{D}^{\frac{1}{2}}(\underset{\sim}{G}|\underset{\sim}{h})$. Here $\underset{\sim}{h}$ is related to the right-hand side $\underset{\sim}{\theta}$ in the classical Givens method by $\underset{\sim}{\theta} = \underset{\sim}{D}^{\frac{1}{2}}\underset{\sim}{h}$. $\underset{\sim}{D}$, the super-diagonals of $\underset{\sim}{G}$, $\underset{\sim}{h}$ and σ (in which the minimum sum of squares is formed) are all initialized to zero. $\underset{\sim}{D}$ is stored in $d_j (j = 1, 2, \ldots, n)$ and the super-diagonals of $\underset{\sim}{G}$ by rows in $g_l (l = 1, 2, \ldots, \frac{1}{2}n(n-1))$. Much of the remaining notation is similar to that in Algorithm 2.9.1.

Algorithm 2.9.3: Orthogonal triangularization by rows and linear least-squares solution using modern plane rotations (Gentleman's 3-multiplication rule) with vector storage.

Comment: $\underset{\sim}{D}$, $\underset{\sim}{h}$, σ and the super-diagonals of $\underset{\sim}{G}$ are initialized to zero.

Step 1. For $l = 1, 2, \ldots, \frac{1}{2}n(n-1)$ set $g_l = 0$. For $j = 1, 2, \ldots, n$ set $d_j = 0$ and $h_j = 0$. Set $\sigma = 0$.

Comment: The ith major step is described by Steps 3-27.

Step 2. For $i = 1, 2, \ldots, m$ execute Steps 3-27.

Comment: The ith row of $(\underset{\sim}{A}|\underset{\sim}{b})$ and the corresponding weight are read or formed.

Step 3. Read or form the current (ith) row $w, v_1, v_2, \ldots, v_n, b$.

Comment: The jth minor step is described by Steps 5-25.

Step 4. For $j = 1, 2, \ldots, n$ execute Steps 5-25.

Comment: No operations on row i are required if w is zero.

Step 5. If $w = 0$ advance to Step 27.

Comment: A rotation is skipped if a_{ij} is already zero.

Step 6. If $v_j = 0$ advance to Step 25.

Comment: Element g_{jj} is stored as g_l.

Step 7. Set $l = (j-1)(2n-j)/2$.

Comment: The algorithm branches according to whether d_j is zero or non-zero.

Step 8. If $d_j \neq 0$ advance to Step 15.

Comment: In the case $d_j = 0$ row j of $\underset{\sim}{D}^{\frac{1}{2}}(\underset{\sim}{G}|\underset{\sim}{h})$ is replaced by row i of $\underset{\sim}{W}^{\frac{1}{2}}(\underset{\sim}{A}|\underset{\sim}{b})$ in Steps 9-13.

Step 9. Replace d_j by wv_j^2.

Step 10. For $k = j+1, j+2, \ldots, n$ execute Steps 11-12.

Step 11. Replace l by $l+1$.

Step 12. Replace g_l by v_k/v_j.

Step 13. Replace h_j by b/v_j.

Comment: No further rotations involving row i of $(\underset{\sim}{A}|\underset{\sim}{b})$ are required.

Step 14. Advance to Step 27.

Comment: In the case $d_j \neq 0$ a 3-multiplication rule to annihilate a_{ij} is carried out in Steps 15-24.

Step 15. Set $y = d_j$ and $z = wv_j$.

Step 16. Replace d_j by $y + zv_j$.

Step 17. Set $\hat{c} = y/d_j$ and $\hat{s} = z/d_j$.

Step 18. Replace w by $\hat{c}w$.

Step 19. For $k = j+1, j+2, \ldots, n$ execute Steps 20-22.

Step 20. Replace l by $l+1$.

Step 21. Set $y = g_l$ and $z = v_k$.

Step 22. Replace g_l by $\hat{c}y+\hat{s}z$ and v_k by $z-v_j y$.

Step 23. Set $y = h_j$ and $z = b$.

Step 24. Replace h_j by $\hat{c}y+\hat{s}z$ and b by $b-v_j y$.

Step 25. Continue.

Comment: The residual sum of squares is updated.

Step 26. Replace σ by $\sigma + wb^2$.

Step 27. Continue.

Step 28. Use Algorithm 2.1.3 to solve $\underset{\sim}{G}\underset{\sim}{x} = \underset{\sim}{h}$.

In Algorithm 2.9.4 the extensions to Algorithm 2.9.3 to implement the hybrid 2- and 3-multiplication rule are incorporated.

Algorithm 2.9.4: Orthogonal triangularization by rows and linear least-squares solution using modern plane rotations (Gentleman's hybrid 2- and 3-multiplication rule) with vector storage.

Steps 1-18. As Steps 1-18 of Algorithm 2.9.3.

Step 18.1. If $100y \geqslant d_j$ advance to Step 24.2.

Steps 19-24. As Steps 19-24 of Algorithm 2.9.3.

Step 24.1. Advance to Step 25.

Step 24.2. For $k = j+1, j+2, \ldots, n$ execute Steps 24.3-24.5.

Step 24.3. Replace l by $l+1$.

Step 24.4. Replace v_k by $v_k - v_j g_l$.

Step 24.5. Replace g_l by $g_l + \hat{s} v_k$.

Step 24.6. Replace b by $b - v_j h_j$.

Step 24.7. Replace h_j by $h_j + \hat{s} b$.

Steps 25-28. As Steps 25-28 of Algorithm 2.9.3.

We do not present algorithms for the Hammarling 2-multiplication rules, since appropriate strategies to overcome the possibility of underflow are still being worked out.

2.10 A comparison of the plane-rotation methods with other methods based upon orthogonal transformations

Classical plane rotations appear to have been little-used for solving linear least-squares problems, despite the fact that their stability properties compare favourably with those of the modified Gram-Schmidt and Householder methods. The main reason for this lack of use is the unfavourable amount of arithmetic required by classical rotations compared with other orthogonalization methods (Wilkinson, 1965: 244-247). For example, if $m \gg n$, classical plane rotations require about $2mn^2$ long operations to triangularize an m by n matrix, whereas the other two methods each take about mn^2 long operations (these numbers are to be compared with the less satisfactory method of normal equations which takes about $\frac{1}{2}mn^2$ long operations). As a result, nearly all numerically stable methods for solving dense linear least-squares problems that have been developed in recent years use either the modified Gram-Schmidt method or Householder transformations. Moreover, as a further consequence of the unfavourable amount of arithmetic required by classical plane rotations, a number of authors (eg Reid, 1967; Hanson and Lawson, 1969) have preferred to develop extensions of Householder's method for exploiting structured problems or for updating linear least-squares solutions. Of course, the recent appearance of the modern versions of plane rotations requiring fewer arithmetic operations will almost certainly give rise to a greater concentration upon their use.

Despite the above arguments which, superficially at least, seem quite reasonable, we believe there are a number of reasons (here we give four) why the plane-rotation methods (even in their classical form) discussed in Sections 2.8 and 2.9 have advantages over methods such as modified Gram-Schmidt or Householder for solving either dense or structured linear least-suqares problem.

Firstly, the matrix can be orthogonally triangularized row-by-row, thus enabling the complete process (in the dense case) to be carried out in a storage space of $\frac{1}{2}n(n+1)$ words for the upper-triangular matrix, n words for the right-hand side, and n words for the current row of $\underset{\sim}{A}$, giving a total of $\frac{1}{2}n(n+5)$ words (see Algorithms 2.9.1, 2.9.2, 2.9.3 and 2.9.4). Note that this storage space is independent of m, and thus very large problems can be solved, as long as there is sufficient store available for $\frac{1}{2}n(n+5)$ elements plus, of course, the program itself (on the English Electric KDF9 computer, for example, with its 32K-word core store, this implies that m is unlimited and n may be well over 200.

Secondly, in performing a single plane rotation (as opposed to a single Householder transformation or a step of the modified Gram-Schmidt method), considerable advantage can frequently be taken of the zero-non-zero structure of the matrix, ie unnecessary arithmetic operations upon zero elements can be avoided, and further economics in storage can consequently be made. An important instance, mentioned in Section 2.8, is when the two rows involved in a rotation have zero elements in corresponding column positions, in which case no arithmetic need be performed upon these elements.

Thirdly, an aspect of scientific computation often overlooked is that when many numerical methods are programmed in a high-level language such as Algol or Fortran, the actual proportion of time spent in the execution of <u>purely arithmetic</u> statements is frequently a small percentage of the total time. The bulk of the time is often spent in referencing (either fetching or storing) array variables, for- or DO- loop overheads etc (see, for example, Wichmann, 1973). For instance, Algol 60 implementations of orthogonalization methods (modified Gram-Schmidt, Householder or plane rotations) for the solution of linear systems spend typically only about 10% of the total time executing purely arithmetic statements (see later in this section). Consequently, even a substantial saving in the number of multiplications has only a marginal relative effect on the total execution time. Therefore, the main consideration is flexibility: a method such as plane rotations that enables structure to be exploited in a more straightforward and efficient manner is frequently to be preferred.

Fourthly, another factor, though not quite so important in the light of the comments in the previous paragraph, is that the generalized forms of plane rotation discussed in Section 2.9 enable the number of multiplications to be reduced by a quarter or even by one-half. In the latter case the amount of arithmetic is about that of the other orthogonalization methods.

To reinforce our claims relating to the proportion of time spent on purely arithmetic operations and to demonstrate the little-known competitiveness of the plane-rotation methods we discuss in detail the "inner loops" of the modified Gram-Schmidt, Householder and plane rotation methods. For simplicity we shall assume that $m \gg n$. For purposes of comparison we present code segments, each written in Algol 60, for these methods. We have made a serious attempt to code each of these computations as efficiently as possible in order that our comparison shall be a fair one. We then apply to these code segments the method of Wichmann (1973) for estimating the execution speed of Algol programs. In Wichmann's approach a weight (representing a number of computational time units) is assigned to each identifier, constant or delimeter in a program. This weight is independent of the computer or the compiler and represents an average based on a number of existing Algol compilers.

The "inner loop" of the modified Gram-Schmidt method (Steps 5 and 6 of Algorithm 2.6.1) is really in two parts, code for which is

First part:

```
rlj := 0;
for k := 1 step 1 until m do
    rlj := rlj + p[k] × a[k,j];
```

Second part:

```
for k := 1 step 1 until m do
    a[k,j] := a[k,j] - rlj × p[k];
```

Note that a further advantage accrues from storing element q_{kl} in position k of the one-dimensional array p: one-dimensional array elements can be referenced faster than two-dimensional array elements. Also, for purposes of the summation the values of r_{lj} is accumulated as the simple variable rlj. We incorporate similar ideas in the codes for the other methods.

The time for the kth cycle ($k = 1, 2, \ldots, m$) in the first part is

$$T_1 = L + M + A + 2V_0 + V_1 + V_2, \tag{2.10.1}$$

where

L = time for loop control (ie incrementing and testing of counter k),

M = floating-point multiplication time,

A = floating-point addition or subtraction time,

V_i = time to reference a variable with i subscripts.

In the ensuing analyses we shall also need

S = time for floating-point square root.

The weights obtained by Wichmann (1973) for these parameters were $L = 14$, $M = 2$, $A = 1$, $V_i = 1+4i$ and $S = 50$ computational units (1 computational unit (c.u.) = 8.3 μsec on KDF9, 0.85 μsec on CDC 6600, etc). Using these values

$$T_1 = 14 + 2 + 1 + (2)(1) + 5 + 9 = 33 \text{ c.u.} \tag{2.10.2}$$

Thus the total time for the first part is $33m + O(1)$ c.u., the $O(1)$ term stemming from loop set-up costs. Similarly, for the kth cycle ($k = 1, 2, \ldots, m$) of the second part we obtain a time of

$$T_2 = L + M + A + V_0 + V_1 + 2V_2 \tag{2.10.3}$$

$$= 14 + 2 + 1 + 1 + 5 + (2)(9) = 41 \text{ c.u.} \tag{2.10.4}$$

So the total time for the second part is $41m + O(1)$ c.u. The total time spent in the two parts is therefore $74m + O(1)$ c.u. Thus, since the above two parts are executed about $\frac{1}{2}n^2$ times in all, the overall time for the modified Gram-Schmidt method is $37\ mn^2$ c.u., ignoring terms of lower order. Note that of these $37\ mn^2$ c.u., only $3\ mn^2$ (8%) are purely arithmetical and only $2\ mn^2$ (5%) involve multiplications.

We now examine the "inner loop" of the Householder method (Steps 8 and 9 of Algorithm 2.7.1). Again there are two parts, codes for which are

First part:

```
y := 0;
for i := k step 1 until m do
    y := y + w[i] × a[i,j];
y := beta × y;
```

Second part:

```
for i := k step 1 until m do
    a[i,j] := a[i,j] - y × w[i];
```

We see, by comparison with the code for the modified Gram-Schmidt method, that the codes are very similar in form, the main difference being the initial values of the for-loop counters. Accordingly, the total time for the two parts is 74 (m-k) + O(1) c.u. Now the above code is executed for values of j from k+1 to n and for values of k from 1 to n (see Steps 7 and 1 of Algorithm 2.7.1). Thus the overall time for the method of Householder transformations is

$$\sum_{k=1}^{n} \sum_{j=k+1}^{n} 74(m-k) = 37\ mn^2 \text{ c.u.,} \tag{2.10.5}$$

ignoring terms of lower order. Again, as with the modified Gram-Schmidt method, only 8% of this time is purely arithmetic and only 5% involves multiplications.

We now turn to classical plane rotations. Code for the "inner loop" of the triangularization by columns method, based on Steps 5, 6 and 7 of Algorithm 2.8.1, is

```
for j := k+1 step 1 until n do
begin
    y := a[k,j];
    z := a[i,j];
    a[k,j] := c×y + s×z;
    a[i,j] := c×z - s×y
end j;
```

The time for the jth cycle is

$$L + 4M + 2A + 10V_0 + 4V_2 \quad (2.10.6)$$

$$= 14 + (4)(2) + (2)(1) + (10)(1) + (4)(9) = 70 \text{ c.u.} \quad (2.10.7)$$

The total time for the inner loop is therefore 70 (n-k) + O(1) c.u. This inner loop is executed for values of i from k+1 to m and values of k from 1 to n, giving an overall time of

$$\sum_{k=1}^{n} \sum_{i=k+1}^{m} 70(n-k) = 35 \; mn^2 \text{ c.u.}, \quad (2.10.8)$$

ignoring terms of lower order. An identical time is taken by the triangularization by rows process. Note that 10/70 = 14% of the total time is spent on purely arithmetic operations and 8/70 = 11% of the total time involves multiplications.

Finally, we examine the "inner loop" of Gentleman's 3-multiplication rule (cf relations (2.9.21)), code for which, if gji denotes the values of g_{ji},

```
for k := i+1 step 1 until n do
begin
    y := g[i,k];
    z := g[j,k];
    g[i,k] := ccap × y + scap × z;
    g[j,k] := z - gji × y
end k;
```

The time for the kth cycle is

$$L + 3M + 2A + 9V_0 + 4V_2 \quad (2.10.9)$$

$$= 14 + (3)(2) + (2)(1) + (9)(1) + (4)(9) = 67 \text{ c.u.} \quad (2.10.10)$$

Thus the overall time is approximately 33.5 mn^2 c.u., of which 8/67 = 12% is spent on purely arithmetic operations and 6/67 = 9% on multiplications.

All the above times, together with those for the Golub-Hammarling class of rules which are derived in a similar way, are summarised in Table 2.10.1. It should be emphasized that the values in this table apply to the "average" Algol 60 compiler. Corresponding values for other high-level languages, such as Fortran, may well be different.

Method	Number of multns.	Time in computational units	Ratio of times	Proportion of time spent on	
				purely arithmetic operations	multns.
Modified Gram-Schmidt	mn^2	$37mn^2$	1.00	8%	5%
Householder	mn^2	$37mn^2$	1.00	8%	5%
Classical plane rotations	$2mn^2$	$35mn^2$ $29mn^2$	0.95 0.78	14% 17%	11% 14%
Modern plane rotations (Gentleman's 3-multn. rule)	$\frac{3}{2}mn^2$	$33.5mn^2$ $27.5mn^2$	0.91 0.74	12% 15%	9% 11%
Modern plane rotations (Golub-Hammarling 2-multn. rules)	mn^2	$32mn^2$ $26mn^2$	0.86 0.70	9% 12%	6% 8%

Table 2.10.1 A comparison of the theoretical computation times of several methods for the orthogonal triangularization of an m by n matrix ($m \gg n$). For the plane-rotation methods the upper of the two entries in Columns 3-6 applies to array storage and the lower to vector storage.

2.11 Stepped-banded matrices

Matrices of a special form, which we term stepped-banded matrices, are introduced in this section. These matrices, which are a generalization of band matrices, arise in problems of interpolation and approximation in one, two or more independent variables by linear combinations of basis functions having restricted support (see Chapters 6, 7 and 10).

A stepped-banded matrix $\underset{\sim}{A}$ is defined as follows. Let $\underset{\sim}{A}$ be an m by n matrix. Let q be an integer such that $1 \leqslant q \leqslant n$. Let $p_o, p_1, \ldots, p_{n-q+1}$ be a set of non-negative integers which satisfy

$$0 = p_o < p_1 \leqslant p_2 \leqslant p_3 \quad \cdots \leqslant p_{n-q} < p_{n-q+1} = m. \qquad (2.11.1)$$

Suppose $\underset{\sim}{A}$ can be subdivided into n-q+1 blocks such that the kth block (k = 1, 2, ..., n-q+1) consists of rows $p_{k-1}+1$ to p_k and has non-zero elements only in columns k to k-q+1 (note that the block is empty if $p_{k-1} = p_k$). Such a matrix is termed a stepped-banded matrix of bandwidth q. Fig. 2.11.1 illustrates a stepped-banded matrix of order 12 by 8 with bandwidth q = 4, having $p_1 = 2$, $p_2 = 5$, $p_3 = p_4 = 9$.

$$\left[\begin{array}{cccccccc}
X & X & X & X & & & & \\
X & X & X & X & & & & \\
\hline
 & X & X & X & X & & & \\
 & X & X & X & X & & & \\
 & X & X & X & X & & & \\
\hline
 & & X & X & X & X & & \\
 & & X & X & X & X & & \\
 & & X & X & X & X & & \\
 & & X & X & X & X & & \\
\hline\hline
 & & & & X & X & X & X \\
 & & & & X & X & X & X \\
 & & & & X & X & X & X
\end{array}\right]$$

Fig. 2.11.1 A stepped-banded matrix with m = 12, n = 8 and q = 4.

Evidently $\underset{\sim}{A}$ can be held in condensed form in a rectangular array of size m by q, if the n-q values of the integers p_1, p_2, ..., p_{n-q} are also stored. In this condensed form of storage a_{ij}, if it lies in the kth block, is stored in location (i, j-k+1).

2.12 Triangularization of stepped-banded matrices using Gaussian elimination

Let $\underset{\sim}{A}$ be an m by n stepped-banded matrix as defined in Section 2.11. We consider the $\underset{\sim}{LU}$ factorization (cf Section 2.4) of $\underset{\sim}{A}$ using Gaussian elimination. The process to be described generalizes the algorithm of Martin and Wilkinson (1967) for the factorization of uniformly-banded square matrices.

The algorithm consists of n-1 major steps, the kth of which (k = 1, 2, ..., n-1) involves the elimination of the sub-diagonal elements in the kth column of $\underset{\sim}{A}$. Before the start of the kth major step, the first k-1 rows of $\underset{\sim}{A}$ are in upper band triangular form with (at most) q-1 super-diagonals. The final matrix also takes the form of an upper banded triangle of bandwidth q.

The configuration at the start of the kth major step is illustrated in Fig. 2.12.1 for the case m = 12, n = 10, q = 4, $p_1 = 2$, $p_2 = 4$, $p_3 = 5$, $p_4 = 7$, $p_5 = 8$, $p_6 = 9$, k = 4. In the kth major step there are (at most) $p_k - k$ sub-diagonal elements to be eliminated (here we define $p_k = m$ if $k > n-q$). The kth major step consists of (i) determining j, the smallest value of i for which $|a_{jk}| \geq |a_{ik}|$ $(i = k, k+1, \ldots, p_k)$, (ii) interchanging rows k and j if $k \neq j$ and (iii) p_k-k minor steps, the ith of which $(i = k+1, k+2, \ldots, p_k)$ involves forming $m_{ik} = a_{ik}/a_{kk}$ (N.B. $|m_{ik}| \leq 1$) and replacing row i by row $i-m_{ik} \times$ row k.

$$
\begin{array}{l|cccccccccc}
 & X & X & X & X & & & & & & \\
 & X & X & X & X & X & & & & & \\
 & & X & X & X & X & X & & & & \\
 & & X & X & X & X & X & & & & \\
\text{Rows involved} & & & X & X & X & X & & & & \\
\text{in 4th major} & & & & X & X & X & X & & & \\
\text{step} & & & & X & X & X & X & & & \\
 & & & & & X & X & X & X & & \\
 & & & & & & X & X & X & X & \\
 & & & & & & & X & X & X & X \\
 & & & & & & & X & X & X & X \\
 & & & & & & & X & X & X & X
\end{array}
$$

Fig. 2.12.1 The configuration at the start of the 4th major step in the LU factorization by Gaussian elimination with partial pivoting of a stepped-banded matrix with $m = 12$, $n = 10$ and $q = 4$. X denotes a (usually) non-zero element. X denotes an element that has been reduced to zero.

In practice an economized form of storage is used in which A is stored as an m by q rectangular array as described in Section 2.11; for further details see Algorithm 2.12.1 below.

Note that, since at any stage of the reduction there are at most q elements in any row, stage (iii) involves at most $q(p_k-k)$ long operations and hence the total number of long operations M is bounded by

$$M = \sum_{k=1}^{n-1} q\,(p_k-k). \tag{2.12.1}$$

It is easily established that if A is of rank n then an upper bound for p_k $(k = 1, 2, \ldots, n-q)$ is $m-n+q+k-1$. Moreover, $p_k = m$ for $k = n-q+1$, $n-q+2, \ldots, n-1$. Thus

$$M \leqslant q \sum_{k=1}^{n-q} p_k + q \sum_{k=n-q+1}^{n-1} p_k - q \sum_{k=1}^{n-1} k$$

$$= q \sum_{k=1}^{n-q} (m-n+q+k-1) + q \sum_{k=n-q+1}^{n-1} m - \tfrac{1}{2}qn(n-1) \qquad (2.12.2)$$

$$< qn(m-n+q). \qquad (2.12.3)$$

Note that in the square case m=n this bound reduces to

$$M < nq^2. \qquad (2.12.4)$$

These bounds are somewhat pessimistic however. A more realistic estimate is given by assuming that each block has roughly the same number of rows. Then p_k ($k = 1, 2, \ldots, n-q$) has the approximate value of $mk/(n-q+1)$. In this case

$$M \doteq q \sum_{k=1}^{n-q} mk/(n-q+1) + q(q-1)m - \tfrac{1}{2}qn(n-1)$$

$$= \tfrac{1}{2}q\left\{m(q-2) + n(m-n+1)\right\} , \qquad (2.12.5)$$

which for m=n reduces to

$$M = \tfrac{1}{2}q(q-1)n . \qquad (2.12.6)$$

These two more realistic bounds are about half of the above rigorous bounds.

Another approach to the solution of stepped-banded systems in the square case is based on the observation that the non-zero elements of a square non-singular stepped-banded matrix can be contained wholly within a uniformly-banded matrix with q-1 super-diagonals and q-1 sub-diagonals. A uniformly-banded matrix with these dimensions requires for its factorization about $2n(q-\frac{1}{2})(q-1)$ long operations, ie about twice as many operations as the above rigorous bound or about four times as many as the realistic bound. Thus the application of a standard algorithm for uniformly-banded systems could be employed but computationally it would be probably four times as expensive.

Having reduced A to LU form, the system $Ax = b$ may be solved in the square case m=n by solving two banded triangular sets of equations, or in the least-squares case $m > n$ by applying the method of Section 2.4, taking full advantage of the banded nature of L and U. Alternatively, in the square case, if the elimination steps performed on A are also performed on b to produce a new vector d then it is merely necessary to solve the single band triangular system $Ux = d$. Algorithm 2.12.1 below, for the case m=n, in which A is stored in condensed form, is based upon this alternative approach.

Some features of the algorithm are as follows. Immediately after the element a_{ik} has been eliminated, the new value of a_{ij} $(j > k)$ is stored in location (i, j-1). This strategy ensures that all non-zero elements remain within the confines of the n by q array and, in particular, that successive diagonals of the resulting band triangle are stored in successive columns of the rectangular array. During the kth major step at most q blocks are involved. Thus, if required, the matrix can be brought into store block by block as the elimination proceeds. In particular, the kth block is not processed until the kth major cycle.

Algorithm 2.12.1: Solution of a square stepped-banded linear system using Gaussian elimination with partial pivoting (economized storage strategy).

Comment: The kth major step is described by Steps 2-18.

Step 1. For k = 1, 2, ..., n execute Steps 2-18.

Comment: Set l to the number of the last row involved in the kth major step.

Step 2. Set $l = p_k$ (if $k \leq n-q$) or n (otherwise).

Comment: The row number, j, of the element with maximum modulus in column k is determined in Steps 3-5.

Step 3. Set $z = |a_{k1}|$ and $j = k$.

Step 4. For $i = k+1, k+2, \ldots, l$ execute Step 5.

Step 5. If $|a_{i1}| > z$ replace z by $|a_{i1}|$ and j by i.

Comment: A row interchange is not required if $j = k$.

Step 6. If $j = k$ advance to Step 14.

Comment: Rows j and k are interchanged in Steps 7-13.

Step 7. For $u = 1, 2, \ldots, q$ execute Steps 8-10.

Step 8. Set $z = a_{ku}$.

Step 9. Replace a_{ku} by a_{ju}.

Step 10. Replace a_{ju} by z.

Step 11. Set $z = b_k$.

Step 12. Replace b_k by b_j.

Step 13. Replace b_j by z.

Comment: The ith minor step is described by Steps 15-18.

Step 14. For $i = k+1, k+2, \ldots, l$ execute Steps 15-18.

Step 15. Set $z = a_{i1}/a_{k1}$.

Step 16. For $u = 2, 3, \ldots, q$ replace $a_{i,u-1}$ by $a_{iu} - za_{ku}$.

Step 17. Set $a_{iq} = 0$.

Step 18. Replace b_i by $b_i - zb_k$.

Step 19. Use Algorithm 2.1.4 to solve $\underset{\sim}{U}\underset{\sim}{x} = \underset{\sim}{b}$ ($\underset{\sim}{U}$ stored in $\underset{\sim}{A}$).

2.13 Triangularization of stepped-banded matrices using stabilized elementary transformations

We now describe a method employing stabilized elementary matrices for the triangularization of a stepped-banded matrix $\underset{\sim}{A}$. As with the Gaussian elimination method of Section 2.12 the method takes full advantage of the structure of $\underset{\sim}{A}$ in that only very few arithmetic operations are performed on zero elements of $\underset{\sim}{A}$. The method has the further practical advantage that the rows of $\underset{\sim}{A}$ are processed <u>sequentially</u>, ie each row <u>in turn</u> may be computed or read from an input device, and then processed fully before the next row is so treated. Thus, matrices with an indefinitely large number of rows may be triangularized. The only restriction is that a storage space of roughly nq locations must be available. A parallel of the method which uses plane rotations to effect an orthogonal triangularization is given in Section 2.14.

The matrix $\underset{\sim}{M}_{ij}$, equal to the identity matrix apart from the element in position (i, j) ($i \neq j$) which is $-m_{ij}$, is termed an <u>elementary matrix</u> (Wilkinson, 1965: p164 <u>et seq</u>). If $|m_{ij}| \leqslant 1$ then $\underset{\sim}{M}_{ij}$ is termed a <u>stabilized elementary matrix</u>. The effect of pre-multiplying the matrix $\underset{\sim}{A}$ by $\underset{\sim}{M}_{ij}$ is to replace row i by row i $- m_{ij} \times$ row j and to leave the remaining rows undisturbed. The inverse of $\underset{\sim}{M}_{ij}$ is easily verified to be equal to the identity matrix apart from the element in position (i, j) which is $+m_{ij}$.

The triangularization process consists of m major steps, the ith of which ($i = 2, 3, \ldots, m$) involves the elimination, by employing a sequence of stabilized elementary matrices, of the elements in row i of $\underset{\sim}{A}$ that lie to the left of the main diagonal. Immediately before the start of the ith major step, the first i-1 rows of $\underset{\sim}{A}$ are in upper band triangular form with at most q-1 super-diagonals. The configuration at the start of the ith major step is illustrated in Fig. 2.13.1 for the case $q = 4$, $p_1 = 3$, $p_2 = 5$, $p_3 \geqslant 8$, $i = 8$.

The ith major step involves initially the determination of the smallest integer k such that $i \leqslant p_k$, followed by (at most) q minor steps, the jth of which ($j = k, k+1, \ldots, i-2$) is executed only if $a_{ij} \neq 0$ and consists of (i) interchanging rows i and j if $|a_{ij}| > |a_{jj}|$, (ii) forming $m_{ij} = a_{ij}/a_{ii}$ and (iii) replacing row i by row i $- m_{ij} \times$ row j. The interchange in stage (i) is necessary if $|a_{ij}| > |a_{jj}|$ to ensure that $|m_{ij}| \leq 1$ and hence that the elementary transformation defined by stages (ii) and (iii) is stabilized. A full description of the complete process in the case $m = n$, including the treatment of a right-hand side $\underset{\sim}{b}$, is given as Algorithm 2.13.1 below. In this algorithm $\underset{\sim}{R}$ is formed in an n by q array, the successive diagonals of $\underset{\sim}{R}$ being stored as successive columns in the array. The ith rows of $(\underset{\sim}{A}|\underset{\sim}{b})$ $(i = 1, 2, \ldots, n)$ are assumed to be supplied successively in locations $v_1, v_2, \ldots, v_q, u$.

Two refinements that result in a worthwhile saving in computation are incorporated in Algorithm 2.13.1. The first refinement involves, in the case $|v_j| > |r_{jj}|$, replacing the explicit row interchange and the following elimination step by a simple strategy which combines these operations and thus reduces the overheads associated with loop control and the accessing of array variables (cf Section 2.10). The second refinement takes advantage of any zero elements on the diagonal of $\underset{\sim}{R}$. If v_j is about to be eliminated and r_{jj} is zero then the jth row of $\underset{\sim}{R}$ (ie a null row) is interchanged with the current row. The remaining rotations associated with the new current (now null) row are then skipped.

$$
\begin{array}{cccccc}
X & X & X & X & & \\
X & X & X & X & X & \\
X & X & X & X & X & X \\
 & X & X & X & X & X \\
 & X & X & X & X & X \\
 & & X & X & X & X \\
 & & X & X & X & X \\
 & & X & X & X & X
\end{array}
$$

Fig. 2.13.1 The configuration at the start of the 8th major step in the $\underset{\sim}{L}\underset{\sim}{U}$ factorization by stabilized elementary matrices of a stepped-banded matrix with $q = 4$, $p_1 = 3$, $p_2 = 5$, $p_3 \geq 8$. X and (X) as in Fig. 2.12.1.

With very minor changes, plane rotations can be used in place of stabilized elementary transformations in order to effect a $\underset{\sim}{QR}$ rather than an $\underset{\sim}{LU}$ decomposition (See Section 2.14).

In the square case $m = n$ the solution of $\underset{\sim}{A}\underset{\sim}{x} = \underset{\sim}{b}$ then reduces to the solution of the triangular system $\underset{\sim}{R}\underset{\sim}{x} = \underset{\sim}{\theta}$.

In the general case $m \geqslant n$ the least-squares solution can be obtained using the method described in Section 2.4. It is necessary to form the unit lower trapezoidal matrix $\underset{\sim}{L}$, which has the same sub-diagonal structure as that of $\underset{\sim}{A}$. In fact, $\underset{\sim}{L}$ is easily formed as the product of the inverses of the stabilized elementary transformations computed during the reduction.

<u>Algorithm 2.13.1</u>: Solution of a square stepped-banded linear system using stabilized elementary transformations (economized storage strategy).

Comment: k is the number of the current block being processed.

Step 1. Set $k = 1$.

Comment: $\underset{\sim}{R}$ and $\underset{\sim}{\theta}$ are initialized to zero in Steps 2-4.

Step 2. For $i = 1, 2, \ldots, n$ execute Steps 3-4.

Step 3. For $j = 1, 2, \ldots, q$ set $r_{ij} = 0$.

Step 4. Set $\theta_i = 0$.

Comment: The ith major step, in which row i is processed, is described by Steps 6-31.

Step 5. For $i = 1, 2, \ldots, n$ execute Steps 6-31.

Comment: The current block number is updated in Steps 6-7.

Step 6. If $i \leqslant p_k$ advance to Step 8.

Step 7. Replace k by k+1 and return to Step 6.

Comment: The ith row of $(\underset{\sim}{A} \mid \underset{\sim}{b})$ is read or formed.

Step 8. Read or form the current (ith) row $v_1, v_2, \ldots, v_q, u$.

Comment: The jth minor step, in which $a_{i,k+j-1}$ (v_j) is eliminated, is described by Steps 10-30.

Step 9. For $j = 1, 2, \ldots, q$ execute Steps 10-30.

Comment: A transformation is skipped if $a_{i,k+j-1}$ is already zero.

Step 10. If $v_j = 0$ advance to Step 30.

Comment: Special action is taken if $r_{k+j-1,k+j-1}$ $(r_{k+j-1,1})$ is zero.

Step 11. If $r_{k+j-1,1} = 0$ advance to Step 27.

Comment: A test is made to see whether a row interchange is required.

Step 12. If $|v_j| \leqslant |r_{k+j-1,1}|$ advance to Step 23.

Comment: A transformation with implicit row interchange is carried out in Steps 13-21.

Step 13. Compute $\mu = r_{k+j-1,1}/v_j$.

Step 14. Replace $r_{k+j-1,1}$ by v_j.

Step 15. For $l = j+1, j+2, \ldots, q$ execute Steps 16-18.

Step 16. Set $z = v_l$.

Step 17. Replace v_l by $r_{k+j-1,\, l-j+1} - \mu z$.

Step 18. Replace $r_{k+j-1,l-j+1}$ by z.

Step 19. Set $z = u$.

Step 20. Replace u by $\theta_{k+j-1} - \mu z$.

Step 21. Replace θ_{k+j-1} by z.

Step 22. Advance to Step 30.

Comment: A transformation without interchange is carried out in Steps 23-25.

Step 23. Compute $\mu = v_j/r_{k+j-1,1}$.

Step 24. For $l = j+1, j+2, \ldots, q$ replace v_l by $v_l - \mu r_{k+j-1,l-j+1}$.

Step 25. Replace u by $u - \mu\theta_{k+j-1}$.

Step 26. Advance to Step 30.

Comment: The $(k+j-1)$th row of $(\underset{\sim}{R} \mid \underset{\sim}{\theta})$ is replaced by the current row in Steps 26-27.

Step 27. For $l = j, j+1, \ldots, q$ replace $r_{k+j-1,l-j+1}$ by v_l.

Step 28. Replace θ_{k+j-1} by u.

Step 29. Advance to Step 31.

Step 30. Continue.

Step 31. Continue.

Step 32. Use Algorithm 2.1.4 to solve $\underset{\sim}{R}\underset{\sim}{x} = \underset{\sim}{0}$.

2.14 Orthogonal triangularization of stepped-banded matrices using plane rotations

We now consider the triangularization by classical plane rotations of a stepped-banded matrix $\underset{\sim}{A}$. The method follows very closely the algorithm based on stabilized elementary transformations treated in Section 2.13 and shares similar advantages. However, there are two further advantages, not enjoyed by the method of Section 2.13. The first is that there is no possibility of severe element growth since the Euclidean norm of each column of $\underset{\sim}{A}$ remains essentially constant. The second is that, if the same operations are applied to the right-hand side, it is not necessary to store details of the transformation matrices themselves.

The process is identical to that of Section 2.13 except that we allow $m \geqslant n$ rather than $m = n$ and the jth minor step of the ith major step involves a plane rotation rather than a stabilized elementary transformation to annihilate a_{ij}:

If $a_{ij} = 0$ do nothing; otherwise

(i) Compute $d = (a_{jj}^2 + a_{ij}^2)^{\frac{1}{2}}$, $c = a_{jj}/d$, $s = a_{ij}/d$.

(ii) Replace row j by $c \times$ row j + $s \times$ row i and row i by $c \times$ row i − $s \times$ row j.

A full description of the complete process is given as Algorithm 2.14.1 below. Algorithm 2.14.1 can also be viewed as an adaptation of Algorithm 2.9.1 to stepped-banded systems. Similar adaptions of Algorithms 2.9.3 and 2.9.4 enable Gentleman's rules to be applied to such systems.

Algorithm 2.14.1: Orthogonal triangularization by rows and linear least-squares solution of a stepped-banded system using classical plane rotations (economized storage strategy).

Comment: k is the number of the current block being processed.

Step 1. Set $k = 1$ and $\sigma = 0$.

Comment: $\underset{\sim}{R}$ and $\underset{\sim}{Q}$ are initialized to zero in Steps 2-4.

Step 2. For $i = 1, 2, \ldots, n$ execute Steps 3-4.

Step 3. For $j = 1, 2, \ldots, q$ set $r_{ij} = 0$.

Step 4. Set $\theta_i = 0$.

Comment: The ith major step, in which row i is processed, is described by Steps 6-30.

Step 5. For $i = 1, 2, \ldots, m$ execute Steps 6-30.

Comment: The current block number is updated in Steps 6-7.

Step 6. If $i \leqslant p_k$ advance to Step 8.

Step 7. Replace k by k+1 and return to Step 6.

Comment: The ith row of $(\underset{\sim}{A} \mid \underset{\sim}{b})$ and the corresponding weight are read or formed.

Step 8. Read or form the current (ith) row $w, v_1, v_2, \ldots, v_q, u$.

Comment: No operations on row i are required if w is zero.

Step 9. If $w = 0$ advance to Step 30.

Comment: The weight is incorporated in row i in Steps 10-13.

Step 10. If $w = 1$ advance to Step 14.

Step 11. Set $z = w^{\frac{1}{2}}$.

Step 12. For $j = 1, 2, \ldots, q$ replace v_j by zv_j.

Step 13. Replace u by zu.

Comment: The jth minor step, in which $a_{i,k+j-1}$ (v_j) is eliminated, is described by Steps 15-28.

Step 14. For $j = 1, 2, \ldots, q$ execute Steps 15-28.

Comment: A rotation is skipped if $a_{i,k+j-1}$ is already zero.

Step 15. If $v_j = 0$ advance to Step 28.

Comment: The algorithm branches according to whether $r_{k+j-1,k+j-1}$ is zero or non-zero.

Step 16. If $r_{k+j-1,1} \neq 0$ advance to Step 20.

Comment: In the case $r_{k+j-1,k+j-1} = 0$ row k+j-1 of $(\underset{\sim}{R} \mid \underset{\sim}{\theta})$ is replaced by row i of $\underset{\sim}{W}^{\frac{1}{2}}(\underset{\sim}{A} \mid \underset{\sim}{b})$ in Steps 17-18.

Step 17. For $l = j, j+1, \ldots, q$ set $r_{k+j-1,l-j+1} = v_l$.

Step 18. Set $\theta_{k+j-1} = u$.

Comment: No further rotations involving row i of $\underset{\sim}{W}^{\frac{1}{2}}(\underset{\sim}{A} \mid \underset{\sim}{b})$ are required.

Step 19. Advance to Step 30.

Comment: In the case $r_{k+j-1,k+j-1} \neq 0$ a conventional rotation to annihilate $a_{i,k+j-1}$ is carried out in Steps 20-27.

Step 20. Set $g = (r^2_{k+j-1,k+j-1} + v_j^2)^{\frac{1}{2}}$.

Step 21. Set $c = r_{k+j-1,1}/g$ and $s = v_j/g$.

Step 22. Set $r_{k+j-1,1} = g$.

Step 23. For $l = j+1, j+2, \ldots, q$ execute Steps 24-25.

Step 24. Set $y = r_{k+j-1,l-j+1}$ and $z = v_l$.

Step 25. Replace $r_{k+j-1,l-j+1}$ by $cy+sz$ and v_l by $cz-sy$.

Step 26. Set $y = \theta_{k+j-1}$ and $z = u$.

Step 27. Replace θ_{k+j-1} by $cy+sz$ and u by $cz-sy$.

Step 28. Continue.

Comment: The residual sum of squares is updated.

Step 29. Replace σ by $\sigma + u^2$.

Step 30. Continue.

Step 31. Use Algorithm 2.1.4 to solve $\underset{\sim}{R}\underset{\sim}{x} = \underset{\sim}{\theta}$.

Reid (1967) and Hanson and Lawson (1969) have also described methods for the orthogonal triangularization of stepped-banded matrices. These methods utilize a special sequence of Householder transformations which avoid operations involving zero elements wherever reasonably possible.

However, these methods involve more complicated strategies than that described here, with the consequence that the resulting codes are somewhat longer.

2.15 The singular value decomposition

The most powerful tool for analysing linear least-squares problems is the singular value decomposition (SVD). Golub and Kahan (1965) appear to have been the first to describe in detail a computational scheme for the SVD, but they refer to the complicated nature of the algorithm they proposed. We confine ourselves to a brief discussion of a modern variant of the algorithm. This variant is due to Golub and Reinsch (1970) and constitutes an improvement of an earlier algorithm, based on the Golub-Kahan paper, due to Golub and Businger (1967). We also discuss a refinement of the Golub-Reinsch algorithm which demonstrates that the SVD can, in an important practical case, be made to operate in roughly half the number of multiplications. Moreover, the refinement enables structured problems to be solved very much more efficiently. It is assumed throughout this section that $m \geqslant n$. There is no loss of generality in this assumption since if $m < n$ we can work with $\underset{\sim}{A}^T$ rather than $\underset{\sim}{A}$.

If $\underset{\sim}{A}$ is an m by n matrix with $m \geqslant n$ there exist matrices $\underset{\sim}{P}$, $\underset{\sim}{S}$ and $\underset{\sim}{Q}$ such that

$$\underset{\sim}{A} = \underset{\sim}{P}\underset{\sim}{S}\underset{\sim}{Q}^T, \qquad (2.15.1)$$

where $\underset{\sim}{P}$ and $\underset{\sim}{Q}$ are orthonormal with respective dimensions m by m and n by n and $\underset{\sim}{S}$ is an m by n matrix with non-zero elements only on the main diagonal. A constructive proof of the existence of the decomposition (2.15.1) is given by Golub and Kahan (1965). The diagonal elements s_i ($i = 1, 2, \ldots, n$) of $\underset{\sim}{S}$ are termed the singular values of $\underset{\sim}{S}$ and, by suitably permuting the columns of $\underset{\sim}{P}$ and $\underset{\sim}{Q}$, may be ordered such that

$$s_1 \geqslant s_2 \geqslant \cdots \geqslant s_k > s_{k+1} = s_{k+2} = \cdots = s_n = 0, \qquad (2.15.2)$$

where k is the rank of $\underset{\sim}{A}$.

A particular advantage of the decomposition (2.15.1) is that it enables the over-determined linear system

$$\underset{\sim}{A}\underset{\sim}{x} = \underset{\sim}{b} \qquad (2.15.3)$$

to be de-coupled, ie to be expressed as the over-determined system

$$\underset{\sim}{S}\underset{\sim}{y} = \underset{\sim}{c}\ , \qquad (2.15.4)$$

where

$$\underset{\sim}{x} = \underset{\sim}{Q}\underset{\sim}{y} \qquad (2.15.5)$$

and

$$\underset{\sim}{b} = \underset{\sim}{P}\underset{\sim}{c} \qquad (2.15.6)$$

are orthonormal changes of variables.

Now $\underset{\sim}{S}$ has the decomposition

$$\underset{\sim}{S} = \underset{\sim}{G}\underset{\sim}{H}\ , \qquad (2.15.7)$$

where

$$\underset{\sim}{G} = \begin{bmatrix} \hat{\underset{\sim}{S}} \\ \underset{\sim}{0} \end{bmatrix} \qquad (2.15.8)$$

and

$$\underset{\sim}{H} = \begin{bmatrix} \underset{\sim}{I} & \underset{\sim}{0} \end{bmatrix} \qquad (2.15.9)$$

are respectively m by k and k by n matrices of rank k and $\hat{\underset{\sim}{S}}$ is the diagonal matrix with non-zero diagonal elements s_i (i = 1, 2, ..., k). The use of (2.2.14) then gives, as the pseudo-inverse of $\underset{\sim}{S}$,

$$\underset{\sim}{S}^{\dagger} = \begin{bmatrix} \hat{\underset{\sim}{S}}^{-1} & \underset{\sim}{0} \\ \underset{\sim}{0} & \underset{\sim}{0} \end{bmatrix} . \qquad (2.15.10)$$

Thus $S^\dagger$ is an n by m matrix whose only non-zero elements are given by

$$s_i^\dagger = \begin{cases} s_i^{-1} & (s_i \neq 0) \\ 0 & (s_i = 0) \,. \end{cases} \tag{2.15.11}$$

The least-squares solution of (2.15.3) can then be computed from (2.15.5) where

$$y = S^\dagger c \tag{2.15.12}$$

from (2.15.4) and

$$c = P^T b \tag{2.15.13}$$

from (2.15.6).

It remains to describe the manner in which P, S and Q are computed. In the Golub-Reinsch algorithm there are two main stages.

In the first stage two sequences of Householder transformations (cf Section 2.7)

$$P^{(k)} = I - 2u^{(k)}u^{(k)T} \quad (k = 1, 2, \ldots, n) \tag{2.15.14}$$

and

$$Q^{(k)} = I - 2v^{(k)}v^{(k)T} \quad (k = 1, 2, \ldots, n-2) \,, \tag{2.15.15}$$

where $\left\|u^{(k)}\right\|_2 = \left\|v^{(k)}\right\|_2 = 1$, are applied to A from the left and from the right in such a way that

$$P^{(n)} \ldots P^{(2)}P^{(1)}AQ^{(1)}Q^{(2)} \ldots Q^{(n-2)} = B \,, \tag{2.15.16}$$

an upper bidiagonal matrix. The transformation matrices $P^{(k)}$ and $Q^{(k)}$ are computed so that $P^{(k)}$ annihilates the sub-diagonal elements in column k, ie the elements $a_{ik}^{(k+\frac{1}{2})}$ $(i = k+1, k+2, \ldots, m)$, without destroying previously established zeros, and $Q^{(k)}$ annihilates the elements to the right of the leading super-diagonal in row k, ie the elements $a_{kj}^{(k+1)}$ $(j = k+2, k+3, \ldots, n)$, again without destroying previously established zeros. The superscripts here relate to the order in which the transformations are executed. Specifically,

$$\underset{\sim}{A}^{(k+\frac{1}{2})} = \underset{\sim}{P}^{(k)}\underset{\sim}{A}^{(k)} \quad (k = 1, 2, \ldots, n) \tag{2.15.17}$$

and

$$\underset{\sim}{A}^{(k+1)} = \underset{\sim}{A}^{(k+\frac{1}{2})}\underset{\sim}{Q}^{(k)} \quad (k = 1, 2, \ldots, n-2), \tag{2.15.18}$$

where $\underset{\sim}{A}^{(1)} = \underset{\sim}{A}$. Throughout the SVD algorithm it is convenient to apply the same left transformations to $\underset{\sim}{b}$. The final vector thus obtained is then the vector $\underset{\sim}{c}$ in (2.15.12) and (2.15.13).

In the second stage $\underset{\sim}{B}$ is reduced iteratively to diagonal form using a special form of the QR algorithm (Francis, 1961/2) with shifts for computing the eigenvalues and eigenvectors of a symmetric matrix.

An operation count establishes that about $2mn^2 - \frac{2}{3}n^3$ long operations, half of which are associated with the left and half with the right transformations, are required to reduce $\underset{\sim}{A}$ to bidiagonal form. The precise number of operations for the diagonalization phase cannot be predicted but, because of the extremely rapid convergence of the QR algorithm with shifts, can be expected to be roughly $4n^3$ (Lawson and Hanson, 1974). It is usually necessary to accumulate the right transformations so that the orthonormal change of variables (2.15.5) is available explicitly for subsequent computation; this accumulation takes about $\frac{2}{3}n^3$ long operations. Thus the complete algorithm takes about $2mn^2 + 4n^3$ long operations. In particular, if $m \gg n$, the SVD will be roughly twice as expensive as a conventional least-squares solution by orthogonalization.

We now consider a refinement of the Golub-Reinsch algorithm. In place of the reduction to bidiagonal form using alternately left and right transformations, firstly reduce $\underset{\sim}{A}$ to upper triangular form and <u>then</u> to bidiagonal form. The first of these two stages can be carried out using any of the methods of orthogonal triangularization, such as Steps 1-11 of Algorithm 2.7.1 (including those associated with the right-hand side b), discussed earlier. The second stage can be carried out by applying the Golub-Reinsch bidiagonalization scheme solely to the n by n matrix containing the right triangle. The total work is easily verified to be about $mn^2 + 5n^3$ (ie $mn^2 - \frac{1}{3}n^3$ for the triangularization, $\frac{2}{3}n^3$ for the bidiagonalization, $\frac{2}{3}n^3$ for accumulating the right transformations and about $4n^3$ for the diagonalization).

The refinement discussed above is important not only because it enables the arithmetic work roughly to be halved in the case $m \gg n$, but also because it enables structured systems, such as ones with stepped-banded matrices, to be solved particularly efficiently. If the original Golub-Reinsch scheme is applied to a stepped-banded system of bandwidth q, there is little that can be done without extensive reorganization to save arithmetic operations and thus the number of long operations remains essentially $2mn^2 + 4n^3$. With the scheme based on the initial triangularization the total work can be reduced to about $mq^2 + n^2q + \frac{14}{3}n^3$ (ie mq^2 for the triangularization, n^2q for the bidiagonalisation, $\frac{2}{3}n^3$ for accumulating the right transformations and about $4n^3$ for the diagonalization). The term n^2q is usually insignificant compared with $\frac{14}{3}n^3$ and hence the total work is essentially $mq^2 + 5n^3$ (say) long operations. This refinement to the SVD therefore becomes particularly significant if $m \gg n$. For instance, consider the values (typical in cubic-spline approximation problems) $m = 100$, $n = 10$ and $q = 4$. The Golub-Reinsch algorithm takes about 24,000 long operations, whereas the refinement requires about one-quarter of this number. If m is extremely large compared with n, the savings are even more substantial. For instance, if $m = 1000$, $n = 10$ and $q = 4$, the Golub-Reinsch algorithm takes about 200,000 long operations and the refinement about $\frac{1}{10}$ of this number.

We do not advocate the general use of the SVD in situations where the matrix $\underset{\sim}{A}$ arises from a well-chosen set of basis functions and a sensible choice of data. Rather, we view the SVD as a tool to employ in special circumstances, such as when we wish to investigate how "well-posed" is a particular formulation of a problem, or sometimes to obtain a reliable estimate of the rank of the observation matrix, even if the problem itself is well-posed (Chapter 10).

We make use of the SVD in Chapter 7 to establish that the choice of B-splines for the basis functions in spline approximation gives rise, in a wide variety of practical circumstances, to an extremely well-posed formulation of the problem. In particular, we use the SVD to estimate the sensitivity of the B-spline coefficients and hence the spline itself to perturbations in the data (cf Section 2.16).

2.16 Perturbation bounds for the solution of linear systems

In solving the linear system

$$\underset{\sim}{A}\underset{\sim}{x} = \underset{\sim}{b} , \qquad (2.16.1)$$

where $\underset{\sim}{A}$ is a real m by n matrix ($m \geqslant n$), it is frequently of some importance to examine the sensitivity of the solution $\underset{\sim}{x}$ to perturbations in $\underset{\sim}{A}$ and/or $\underset{\sim}{b}$. This question of sensitivity is of particular relevance in cases where the system (2.16.1) arises from problems of interpolation or least-squares approximation (Chapters 6, 7 and 10). In these problems $\underset{\sim}{A}$ corresponds to the matrix of n basis functions evaluated at m data points, and $\underset{\sim}{b}$ to a set of m values of a dependent variable. $\underset{\sim}{A}$ will inevitably contain errors resulting from roundings in the floating-point operations needed to evaluate the basis functions. $\underset{\sim}{b}$ will contain errors corresponding, in the case of mathematical data, to the truncation or rounding of non-computer-representable numbers or, in the more common case of experimental data, to the finite precision of such data. Accordingly, we wish to examine the effect on $\underset{\sim}{x}$ of replacing $\underset{\sim}{A}$ by $\underset{\sim}{A}+\delta\underset{\sim}{A}$ and $\underset{\sim}{b}$ by $\underset{\sim}{b}+\delta\underset{\sim}{b}$, where $\delta\underset{\sim}{A}$ and $\delta\underset{\sim}{b}$ denote respectively perturbations in $\underset{\sim}{A}$ and $\underset{\sim}{b}$. For the square case $m = n$ we follow Wilkinson (1965, pp 189 et seq) and for the over-determined case $m \geqslant n$ we follow Lawson and Hanson (1974).

In the case $m = n$, suppose $\underset{\sim}{A}$ is non-singular, and consider the solution $\underset{\sim}{x} + \delta\underset{\sim}{x}$ of

$$(\underset{\sim}{A} + \delta\underset{\sim}{A})(\underset{\sim}{x} + \delta\underset{\sim}{x}) = \underset{\sim}{b} + \delta\underset{\sim}{b} \ . \tag{2.16.2}$$

Expanding (2.16.2) and subtracting (2.16.1) gives

$$\delta\underset{\sim}{A}\underset{\sim}{x} + (\underset{\sim}{A} + \delta\underset{\sim}{A})\delta\underset{\sim}{x} = \delta\underset{\sim}{b} \ . \tag{2.16.3}$$

Thus

$$\underset{\sim}{A}(\underset{\sim}{I} + \underset{\sim}{A}^{-1}\delta\underset{\sim}{A})\delta\underset{\sim}{x} = \delta\underset{\sim}{b} - \delta\underset{\sim}{A}\underset{\sim}{x} \ , \tag{2.16.4}$$

from which

$$\delta\underset{\sim}{x} = (\underset{\sim}{I} + \underset{\sim}{A}^{-1}\delta\underset{\sim}{A})^{-1}\underset{\sim}{A}^{-1}(\delta\underset{\sim}{b} - \delta\underset{\sim}{A}\underset{\sim}{x}) \tag{2.16.5}$$

if $\underset{\sim}{G} = \underset{\sim}{I} + \underset{\sim}{A}^{-1}\delta\underset{\sim}{A}$ is non-singular. The non-singularity of $\underset{\sim}{G}$ is ensured if

$$\|\underset{\sim}{A}^{-1}\delta\underset{\sim}{A}\| \leqslant \|\underset{\sim}{A}^{-1}\|\,\|\delta\underset{\sim}{A}\| \leqslant 1 \ , \tag{2.16.6}$$

a condition we shall assume to apply. Thus

$$\|\delta\underset{\sim}{x}\| \leqslant \frac{\|\underset{\sim}{A}^{-1}\|}{1 - \|\underset{\sim}{A}^{-1}\|\,\|\delta\underset{\sim}{A}\|}\left(\|\delta\underset{\sim}{b}\| + \|\delta\underset{\sim}{A}\|\,\|\underset{\sim}{x}\|\right). \tag{2.16.7}$$

We may express (2.16.7) in the form of <u>the relative error bound</u>

$$\frac{\|\delta\underset{\sim}{x}\|}{\|\underset{\sim}{x}\|} \leqslant \frac{\kappa(\underset{\sim}{A})}{1 - \kappa(\underset{\sim}{A})\frac{\|\delta\underset{\sim}{A}\|}{\|\underset{\sim}{A}\|}}\left(\frac{\|\delta\underset{\sim}{A}\|}{\|\underset{\sim}{A}\|} + \frac{\|\delta\underset{\sim}{b}\|}{\|\underset{\sim}{A}\|\,\|\underset{\sim}{x}\|}\right), \tag{2.16.8}$$

where

$$\varkappa(\underset{\sim}{A}) = \|\underset{\sim}{A}\|\|\underset{\sim}{A}^{-1}\|. \tag{2.16.9}$$

From (2.16.1) we obtain

$$\|\underset{\sim}{x}\| \geqslant \|\underset{\sim}{b}\|/\|\underset{\sim}{A}\| \tag{2.16.10}$$

and hence (2.16.8) becomes

$$\frac{\|\delta\underset{\sim}{x}\|}{\|\underset{\sim}{x}\|} \leqslant \frac{\varkappa(\underset{\sim}{A})}{1 - \varkappa(\underset{\sim}{A})\frac{\|\delta\underset{\sim}{A}\|}{\|\underset{\sim}{A}\|}}\left(\frac{\|\delta\underset{\sim}{A}\|}{\|\underset{\sim}{A}\|} + \frac{\|\delta\underset{\sim}{b}\|}{\|\underset{\sim}{b}\|}\right). \tag{2.16.11}$$

In cases where $\varkappa(\underset{\sim}{A})\|\delta\underset{\sim}{A}\|/\|\underset{\sim}{A}\| \ll 1$, we see from (2.16.11) that the relative error $\|\delta\underset{\sim}{A}\|/\|\underset{\sim}{A}\|$ in $\underset{\sim}{A}$ plus the relative error $\|\delta\underset{\sim}{b}\|/\|\underset{\sim}{b}\|$ in $\underset{\sim}{b}$ is amplified by a factor of $\varkappa(\underset{\sim}{A})$ to produce a bound for the relative error $\|\delta\underset{\sim}{x}\|/\|\underset{\sim}{x}\|$ in $\underset{\sim}{x}$. The number $\varkappa(\underset{\sim}{A})$ is evidently <u>a measure of the sensitivity</u> of the solution of (2.16.1) in the case m = n with respect to perturbations in $\underset{\sim}{A}$ and $\underset{\sim}{b}$, and is commonly known as <u>the condition number of $\underset{\sim}{A}$ with respect to inversion</u>. We shall make particular use of the <u>spectral condition number</u> or <u>spectral norm</u> of $\underset{\sim}{A}$ defined by

$$\varkappa_2(\underset{\sim}{A}) = \|\underset{\sim}{A}\|_2\|\underset{\sim}{A}^{-1}\|_2. \tag{2.16.12}$$

Here $\|\underset{\sim}{A}\|_2$ is the square root of the maximum eigenvalue of $\underset{\sim}{A}^T\underset{\sim}{A}$ or, equivalently,

$$\|\underset{\sim}{A}\|_2 = \max_{\|\underset{\sim}{x}\|_2 = 1} \|\underset{\sim}{A}\underset{\sim}{x}\|_2, \tag{2.16.13}$$

where the 2-norm of a vector $\underset{\sim}{x}$ is defined by

$$\|\underset{\sim}{x}\|_2 = (\underset{\sim}{x}^T\underset{\sim}{x})^{\frac{1}{2}}. \tag{2.16.14}$$

It follows from the above definition that in terms of the (ordered) singular values of $\underset{\sim}{A}$ (Section 2.15),

$$\varkappa_2(\underset{\sim}{A}) = s_1/s_n. \tag{2.16.15}$$

Thus, having obtained the singular value decomposition of $\mathbf{A}$ we can compute immediately the spectral norm of $\mathbf{A}$ from (2.16.15) and then the required sensitivity of the solution of (2.16.1) from (2.16.11).

We now turn to the case $m \geqslant n$. Lawson and Hanson (1974) show that if $\mathbf{A}$ and $\mathbf{A} + \delta\mathbf{A}$ have the same rank k then the inequality corresponding to (2.16.7) is

$$\|\delta\mathbf{x}\| \leqslant \frac{\|\mathbf{A}^{\dagger}\|}{1 - \|\delta\mathbf{A}\|\,\|\mathbf{A}^{\dagger}\|}\left(\|\delta\mathbf{A}\|\,\|\mathbf{x}\| + \|\delta\mathbf{b}\| + \|\delta\mathbf{A}\|\,\|\mathbf{A}^{\dagger}\|\,\|\mathbf{r}\|\right), \quad (2.16.16)$$

where $\mathbf{A}^{\dagger}$ denotes the pseudo-inverse of $\mathbf{A}$, $\mathbf{r}$ the residual vector

$$\mathbf{r} = \mathbf{A}\mathbf{x} - \mathbf{b} \quad (2.16.17)$$

and $\mathbf{x}$ the minimal least-squares solution. Evidently, (2.16.16) reduces to (2.16.7) if $m = n = k$.

The result (2.16.16) may be expressed as

$$\frac{\|\delta\mathbf{x}\|}{\|\mathbf{x}\|} \leqslant \frac{\varkappa_2(\mathbf{A})}{1 - \varkappa_2(\mathbf{A})\frac{\|\delta\mathbf{A}\|}{\|\mathbf{A}\|}}\left(\frac{\|\delta\mathbf{A}\|}{\|\mathbf{A}\|} + \frac{\|\delta\mathbf{b}\|}{\|\mathbf{A}\|\,\|\mathbf{x}\|} + \frac{\|\delta\mathbf{A}\|}{\|\mathbf{A}\|}\varkappa_2(\mathbf{A})\frac{\|\mathbf{r}\|}{\|\mathbf{A}\|\,\|\mathbf{x}\|}\right), \quad (2.16.18)$$

where $\varkappa_2(A) = \|\mathbf{A}\|_2\|\mathbf{A}^{\dagger}\|_2$. $\varkappa_2(A)$ can be considered as a condition number for the rectangular matrix $\mathbf{A}$ (cf Golub and Wilkinson, 1966); it evidently reduces to the conventional spectral norm in the case $m = n = k$. Since $\|\mathbf{A}\|\,\|\mathbf{x}\| \geqslant \|\mathbf{A}\mathbf{x}\|$, (2.16.18) can be expressed as

$$\frac{\|\delta\mathbf{x}\|}{\|\mathbf{x}\|} \leqslant \frac{\varkappa_2(\mathbf{A})}{1 - \varkappa_2(\mathbf{A})\frac{\|\delta\mathbf{A}\|}{\|\mathbf{A}\|}}\left(\frac{\|\delta\mathbf{A}\|}{\|\mathbf{A}\|} + \frac{\|\delta\mathbf{b}\|}{\|\mathbf{A}\mathbf{x}\|} + \frac{\|\delta\mathbf{A}\|}{\|\mathbf{A}\|}\varkappa(\mathbf{A})\frac{\|\mathbf{r}\|}{\|\mathbf{A}\mathbf{x}\|}\right). \quad (2.16.19)$$

The result (2.16.19) reduces to (2.16.11) if $m = n = k$.

Now suppose that $\|\mathbf{r}\| \leqslant 0.1\|\mathbf{b}\|$, a result that will be true for nearly all least-squares solutions of practical importance (in any case it is a trivial matter to check whether this result holds). Then

$$\|\underset{\sim}{A}\underset{\sim}{x}\| = \|\underset{\sim}{b}+\underset{\sim}{r}\| \geqslant \|\underset{\sim}{b}\| - \|\underset{\sim}{r}\| \geqslant 0.9\|\underset{\sim}{b}\| \tag{2.16.20}$$

and hence (2.16.19) yields

$$\frac{\|\delta\underset{\sim}{x}\|}{\|\underset{\sim}{x}\|} \leqslant \frac{\kappa_2(\underset{\sim}{A})}{1-\kappa_2(\underset{\sim}{A})\frac{\|\delta\underset{\sim}{A}\|}{\|\underset{\sim}{A}\|}}\left(\frac{\|\delta\underset{\sim}{A}\|}{\|\underset{\sim}{A}\|} + \frac{10}{9}\frac{\|\delta\underset{\sim}{b}\|}{\|\underset{\sim}{b}\|} + \frac{10}{9}\frac{\|\delta\underset{\sim}{A}\|}{\|\underset{\sim}{A}\|}\kappa_2(\underset{\sim}{A})\frac{\|\underset{\sim}{r}\|}{\|\underset{\sim}{b}\|}\right). \tag{2.16.21}$$

We make use of (2.16.21) in Chapter 7.

CHAPTER 3

B-SPLINES AND THEIR NUMERICAL EVALUATION

Computations with splines are considered in this and in the remaining seven chapters of this work. It is crucially important that our choice of representation of splines and the way in which we manipulate the representation are such that the computations are numerically stable. One reason why we make such demands is that we require a high degree of confidence in our numerical results. We wish to be able to say, for instance, that the departures of an approximating spline from a set of data points are real and are not due to deficiencies in the representation or its use.

In Section 3.1 we define polynomial spline functions and associated concepts. B-splines and some of their properties are presented in Section 3.2. Algorithms based upon divided differences for the evaluation of B-splines are developed in Section 3.3. A recurrence relation for B-splines that is fundamental to much of our work is established in Section 3.4, where also recommended algorithms for B-spline evaluation and further properties of B-splines are presented. In Section 3.5 the values of the B-splines at the ends of the range are derived. The sum of and bounds for the values of normalized B-splines are derived in Section 3.6.

Error analyses of the algorithms of Sections 3.3 and 3.4 are given in Sections 3.7 to 3.9. In Section 3.10 the effects of perturbations in the knots and in the argument of the B-splines are discussed briefly. Some numerical examples are given in Section 3.11. Algorithms for evaluating all non-zero B-splines for a given argument are presented in Section 3.12. Finally, in Section 3.13 other methods for evaluating B-splines are discussed and compared with those recommended.

3.1 Definition of a spline function

Firstly, we define an n-extended partition. There are many equivalent definitions in the literature; that given here is essentially that due to de Boor and Fix (1973). Let n and N be prescribed positive integers. Let (a, b) be a finite or infinite interval on the real line. We say $\pi = \{x_1, x_2, \ldots, x_{N-1}\}$ is an n-extended partition of (a, b) if

(i) $a < x_1 \leqslant x_2 \leqslant \ldots \leqslant x_{N-1} < b$,

and

(ii) if d_i is the frequency with which the number $x = x_i$ appears among the x_j's, then $d_i \leqslant n$ $(i = 1, 2, \ldots, N-1)$.

Condition (ii) can be expressed equivalently as

(ii) $x_{i-n} < x_i$ $(i = n+1, n+2, \ldots, N-1)$. .

For example, if N = 8 the values depicted in Fig. 3.1.1 form a 3-extended partition, whereas those in Fig. 3.1.2 form a 4-extended (but not a 3-extended) partition. If $d_j = 1$ then x_j is termed a simple knot or a knot of multiplicity one; if $d_j > 1$ then x_j is termed a multiple knot or, more specifically, a knot of multiplicity d_j. We term the x_i $(i = 1, 2, \ldots, N-1)$ interior knots.

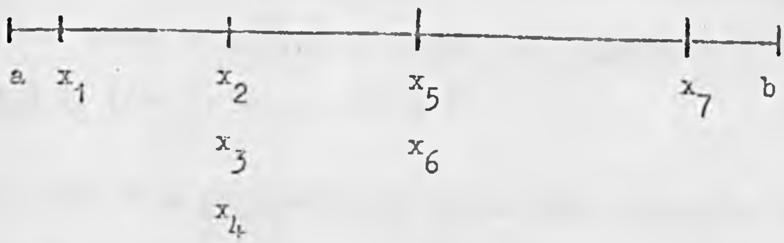

Fig. 3.1.1 A 3-extended partition

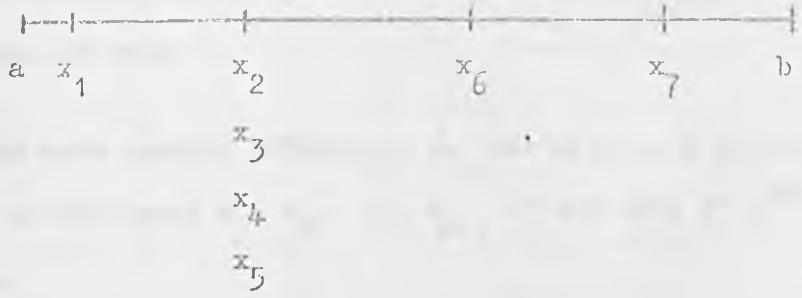

Fig. 3.1.2 A 4-extended partition

It is frequently useful, in cases where (a, b) is a finite interval, to augment the interior knots by further knots with the properties that $x_0 = a$, $x_N = b$, $x_i \leqslant a$ for $i < 0$, $x_i \geqslant b$ for $i > N$, and the complete set of knots form a non-decreasing sequence. We term x_o and x_N <u>end</u> or <u>boundary knots</u> and x_i $(i < 0)$ and x_i $(i > N)$ <u>exterior knots</u>. We call any knot set $\{x_i\}$ of this form a <u>standard knot set</u>. Any standard knot set with $x_i = a$ for $i \leqslant 0$ and $x_i = b$ for $i \geqslant N$ we call a <u>standard knot set with coincident end knots</u>. Carasso and Laurent (1969) appear to have been the first to suggest the use of coincident end knots, but they failed to point out the many practical advantages accruing from such a choice. These advantages become apparent in this and in subsequent chapters.

Let $\pi = \{x_1, x_2, \ldots, x_{N-1}\}$ be an n-extended partition of the finite or infinite interval $(a, b) \equiv (x_o, x_N)$. A function $s(x)$ is a <u>polynomial spline function</u> (or simply a <u>spline</u>) of <u>order</u> n (ie <u>degree</u> n-1) with the <u>knots</u> (or <u>joints</u>) x_i $(i = 1, 2, \ldots, N-1)$ if

(i) $s(x)$ is a polynomial of degree less than n in each of the intervals (x_{i-1}, x_i) $(i = 1, 2, \ldots, N)$.

(ii) $s(x) \in C^{n-2}(x_{i-1}, x_i)$ if $x_{i-1} < x_i$ $(i = 1, 2, \ldots, N)$,

(iii) $s^{(r)}(x_i-) = s^{(r)}(x_i+)$ $(i = 1, 2, \ldots, N-1;\ 0 \leqslant r < n-d_i)$.

Another definition of a spline is based upon the fact that the (n-1)th derivative of a spline of order n is a <u>step function</u> with discontinuities at the knots, and, conversely, the $(n-1)$th integral of step function is a spline of order n.

An even more concise definition is that $s(x)$, $x \in (a, b)$, is a spline of order n with knots $x_1, x_2, \ldots, x_{N-1}$ if and only if $s^{(n)}(x) = 0$ for all $x \notin \pi$.

Suppose all interior knots are simple. Then, since $s(x)$ is composed of N polynomial arcs of degree $< n$, it can evidently be described in terms of at most Nn linear parameters, together of course with the N-1 knots. However, because of condition (iii), this number of free linear parameters is reduced by the number of continuity conditions at the interior knots, ie by $(N-1)(n-1)$, to a total of at most $Nn - (N-1)(n-1) = N+n-1$ linear parameters. We obtain the same result if some or all of the knots are multiple. For, suppose there are r_1 simple knots, r_2 knots of multiplicity 2, ..., r_n knots of multiplicity n. Then $r_1 + 2r_2 + \ldots + nr_n = N-1$, the number of interior knots (including coincidences), and the number of (non-empty) intervals is $r_1 + r_2 + \ldots r_n+1$. The number of free parameters is therefore $n(r_1 + r_2 + \ldots + r_n+1)$ less the number of continuity conditions, ie $n(r_1 + r_2 + \ldots + r_n+1) - (n-1)r_1 - (n-2)r_2 - \ldots - r_{n-1}$ $= r_1 + 2r_2 + \ldots + nr_n + n = N+n-1$.

3.2 The definition of a B-spline

Let n be a positive integer. Define the truncated power function

$$x_+^{n-1} = \begin{cases} x^{n-1} & (x \geqslant 0) \\ 0 & (x < 0) \end{cases} \tag{3.2.1}$$

and

$$M_n(y\,;\,x) = (y-x)_+^{n-1} . \tag{3.2.2}$$

Suppose $x_{i-n}, x_{i-n+1}, \ldots, x_i$ are n+1 real numbers (knots) with $x_{i-n} \leqslant x_{i-n+1} \leqslant \cdots \leqslant x_i$ and $x_{i-n} < x_i$. Such a set of knots forms an n-extended partition of the real line (cf Section 3.1). Consider the divided difference of order n of the function (3.2.2) with respect to the variable y based on the arguments $y = x_{i-n}, x_{i-n+1}, \ldots, x_i$. Using a notation similar to that of Steffensen (1927) we denote this divided difference by $M_n(x_{i-n}, x_{i-n+1}, \ldots, x_i\,;\, x)$, which in unambiguous cases we shall abbreviate to $M_{ni}(x)$. If we let

$$w_{ni}(x) = (x-x_{i-n})(x-x_{i-n+1}) \ldots (x-x_i)\ , \qquad (3.2.3)$$

then in the particular case of <u>distinct</u> knots, ie where

$$x_{i-n} < x_{i-n+1} < \cdots < x_i,$$

an explicit expression (Greville, 1969) is

$$M_{ni}(x) = \sum_{r=i-n}^{i} \frac{(x_r-x)_+^{n-1}}{w'_{ni}(x_r)}\ , \qquad (3.2.4)$$

where the prime denotes differentiation with respect to x.

The truncated power function $(x_r-x)_+^{n-1}$ is evidently a spline of order n with a single knot at $x = x_r$, since it satisfies the conditions of Section 3.1 (with $a = -\infty$, $n = +\infty$). Thus, since the taking of divided differences is a linear operation, it follows that in the case of distinct knots $M_{ni}(x)$ is a linear combination of the functions $(x_r-x)_+^{n-1}$ ($r = i-n, i-n+1, \ldots, i$) and hence is a spline of order n with knots $x_{i-n}, x_{i-n+1}, \ldots, x_i$. This result can also be seen immediately from (3.2.4). For $x > x_i$, $M_{ni}(x)$ is identically zero, by virtue of (3.2.1), and for $x < x_{i-n}$, $M_{ni}(x)$ is simply the divided difference of order n of a polynomial of degree n-1 and hence vanishes identically. For $x_{i-n} < x < x_i$, $M_{ni}(x)$ has the property that it is strictly positive (Curry and Schoenberg, 1966). This property is proved by a simpler argument in Section 3.4 (Theorem 3.4.2). $M_{ni}(x)$ is termed a <u>B-spline</u> or <u>fundamental spline</u> of order n based on the knots $x_{i-n}, x_{i-n+1}, \ldots, x_i$. The B-splines were first introduced for the case of equally-spaced knots by Schoenberg (1946), and for the case of arbitrarily-spaced knots by Curry and Schoenberg (1966).

We have departed slightly from convention in our definition of B-splines in two ways. Firstly, our definition has the property (see Section 4.5) that

$$\int_{-\infty}^{\infty} M_{ni}(x)\,dx = 1/n \; , \tag{3.2.5}$$

whereas the usual definition (see, for example, Curry and Schoenberg, 1966) includes a multiplicative factor n so that the value of the integral is normalized to unity. We find the inclusion of this factor a hindrance, however, particularly when we come to derive in Section 3.4 a recurrence relation for the values of $M_{ni}(x)$. The factor can always be inserted for computational or other purposes as required. Secondly, we employ a double subscript in our abbreviated notation for B-splines, as opposed to the single subscript preferred by most authors. Our notation is necessary since we need to refer to B-splines of <u>various degrees</u> defined on <u>various knot sets</u>.

Recently, a very similar definition has been introduced independently by de Boor (1972); his $M_{ik}(x)$ is identical to our $M_{k,i+k}(x)$.

The <u>normalized B-spline</u> $N_{ni}(x)$ is defined (de Boor, 1972) by

$$N_{ni}(x) = (x_i - x_{i-n})M_{ni}(x) \tag{3.2.6}$$

$$\begin{aligned} &= M_n(x_{i-n+1}, x_{i-n+2}, x_i \; ; x) \\ &\quad - M_n(x_{i-n}, x_{i-n+1}, \ldots, x_{i-1} \; ; x) \; . \end{aligned} \tag{3.2.7}$$

If $\{x_{i-n}, x_{i-n+1}, \ldots, x_i\}$ forms an (n-1)-extended partition, ie if

$x_i > x_{i-n+1}$ or $x_{i-1} > x_{i-n}$, then it is readily verified that $M_{ni}(x)$ and $N_{ni}(x)$ are continuous functions. De Boor (1972) states (erroneously) that $M_{ni}(x)$ and $N_{ni}(x)$ are continuous if $\{x_{i-n}, x_{i-n+1}, \ldots, x_i\}$ form an n-extended partition. A counter-example to his statement is provided by the case $x_{i-n} = x_{i-n+1} = \ldots = x_{i-1} < x_i$ for which $M_{ni}(x)$ and $N_{ni}(x)$ are discontinuous at $x = x_{i-n}$. However, de Boor's result is true for the (open) interval $x_{i-n} < x < x_i$.

In the case $n = 1$, $M_{ni}(x)$ and $N_{ni}(x)$ are discontinuous at $x = x_{i-1}$ and at $x = x_i$. We assume, in accordance with (3.2.1) that

$$M_{1i}(x) = \begin{cases} (x_i - x_{i-1})^{-1} & (x_{i-1} \leqslant x < x_i) \\ 0 & (\text{otherwise}) \end{cases} \qquad (3.2.8)$$

and hence that

$$N_{1i}(x) = \begin{cases} 1 & (x_{i-1} \leqslant x < x_i) \\ 0 & (\text{otherwise}) \end{cases} \qquad . \qquad (3.2.9)$$

In Fig. 3.2.1 we illustrate the B-splines $N_{14}(x)$, $N_{24}(x)$, $N_{34}(x)$ and $N_{44}(x)$ defined upon the knots $x_0 = 0$, $x_1 = 0.3$, $x_2 = 0.45$, $x_3 = 0.65$ and $x_4 = 1$. In Fig. 3.2.2 we again illustrate $N_{14}(x)$, $N_{24}(x)$, $N_{34}(x)$ and $N_{44}(x)$, but with knots $x_0 = 0$, $x_1 = x_2 = x_3 = 0.4$ and $x_4 = 1$.

In Section 3.4 we state and prove a fundamental recurrence that relates B-splines of consecutive degrees. Most of the good error bounds we obtain and the numerically stable algorithms we develop stem from this and related results.

Many of the theorems we prove and the results we obtain for the unnormalized B-spline $M_{ni}(x)$ extend, in an obvious way, using (3.2.6), to the case of the normalized B-splines $N_{ni}(x)$.

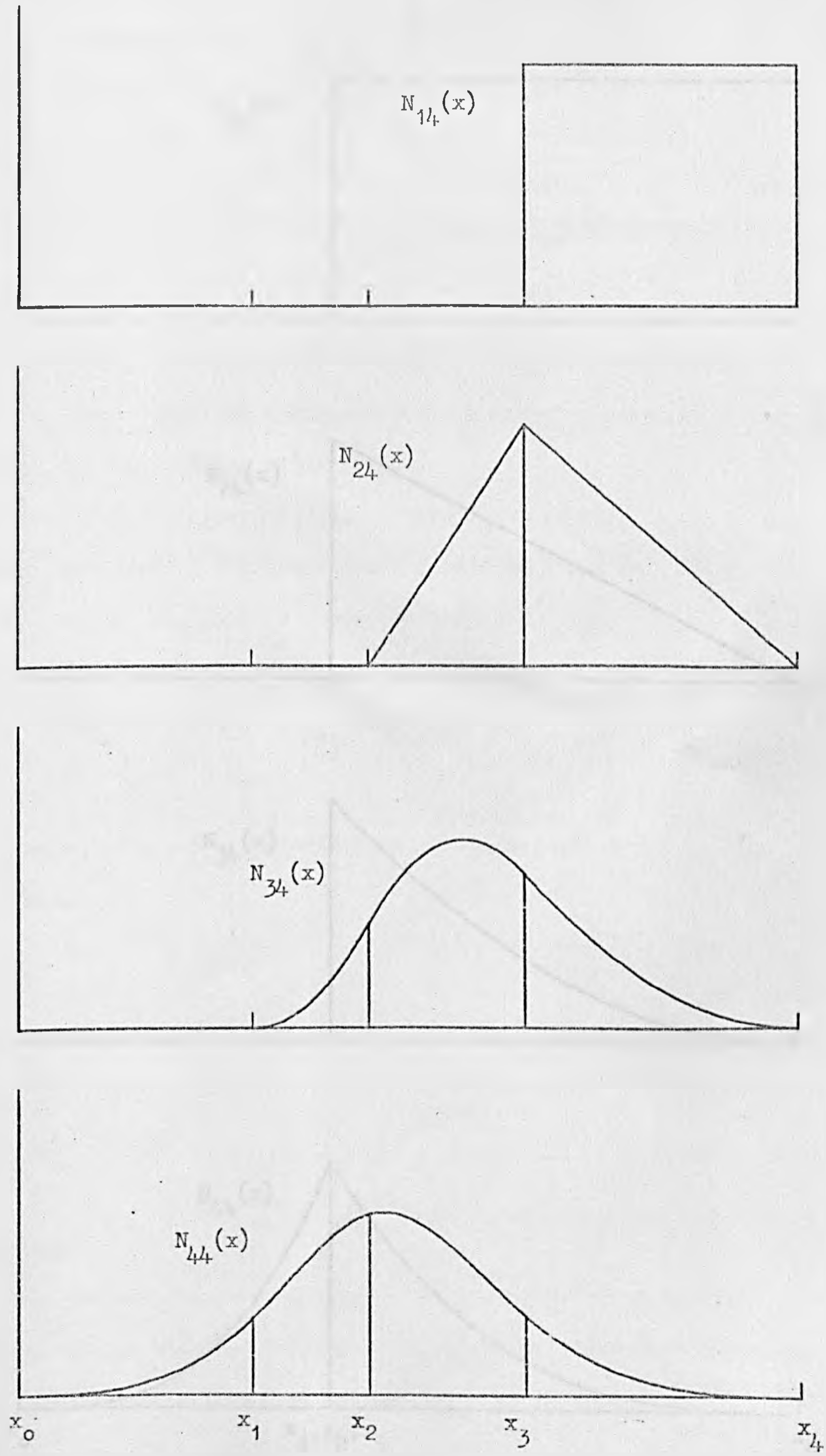

Fig. 3.2.1 The B-splines $N_{14}(x)$, $N_{24}(x)$, $N_{34}(x)$ and $N_{44}(x)$ with knots $x_0 = 0$, $x_1 = 0.3$, $x_2 = 0.45$, $x_3 = 0.65$ and $x_4 = 1$.

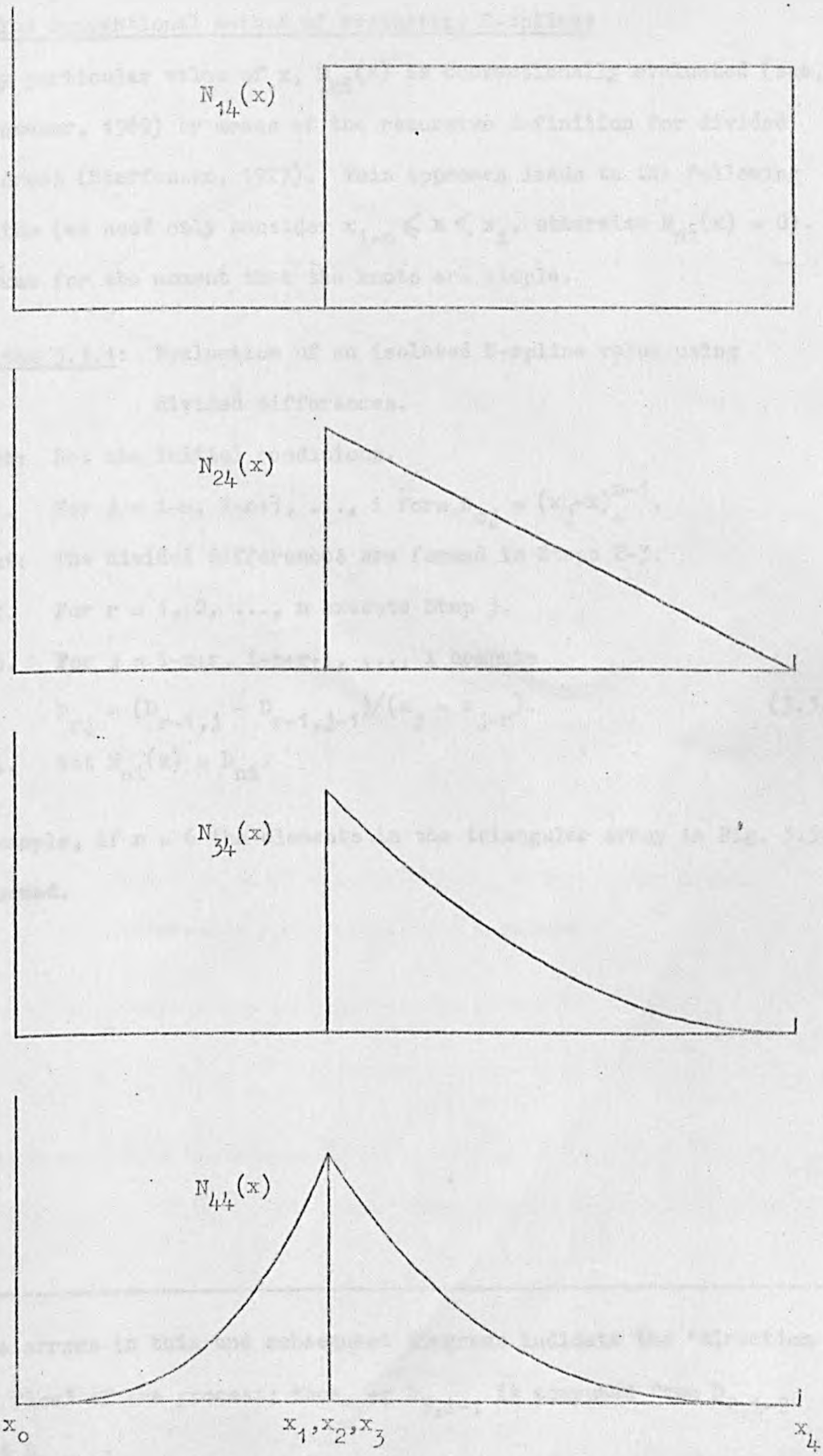

Fig. 3.2.2 The B-splines $N_{14}(x)$, $N_{24}(x)$, $N_{34}(x)$ and $N_{44}(x)$ with knots $x_o = 0$, $x_1 = x_2 = x_3 = 0.4$ and $x_4 = 1$.

3.3 The conventional method of evaluating B-splines

For any particular value of x, $M_{ni}(x)$ is conventionally evaluated (see, eg Schumaker, 1969) by means of the recursive definition for divided differences (Steffensen, 1927). This approach leads to the following algorithm (we need only consider $x_{i-n} \leqslant x < x_i$, otherwise $M_{ni}(x) = 0$). We assume for the moment that the knots are simple.

Algorithm 3.3.1: Evaluation of an isolated B-spline value using divided differences.

Comment: Set the initial conditions.

Step 1. For $j = i-n, i-n+1, \ldots, i$ form $D_{oj} = (x_j - x)_+^{n-1}$.

Comment: The divided differences are formed in Steps 2-3.

Step 2. For $r = 1, 2, \ldots, n$ execute Step 3.

Step 3. For $j = i-n+r, i-n+r+1, \ldots, i$ compute

$$D_{rj} = (D_{r-1,j} - D_{r-1,j-1})/(x_j - x_{j-r}). \tag{3.3.1}$$

Step 4. Set $M_{ni}(x) = D_{ni}$.

For example, if n = 6 the elements in the triangular array in Fig. 3.3.1* are formed.

---

* The arrows in this and subsequent diagrams indicate the "direction of flow" of the process; thus, eg $D_{3,i-1}$ is computed from $D_{2,i-2}$ and $D_{2,i-1}$.

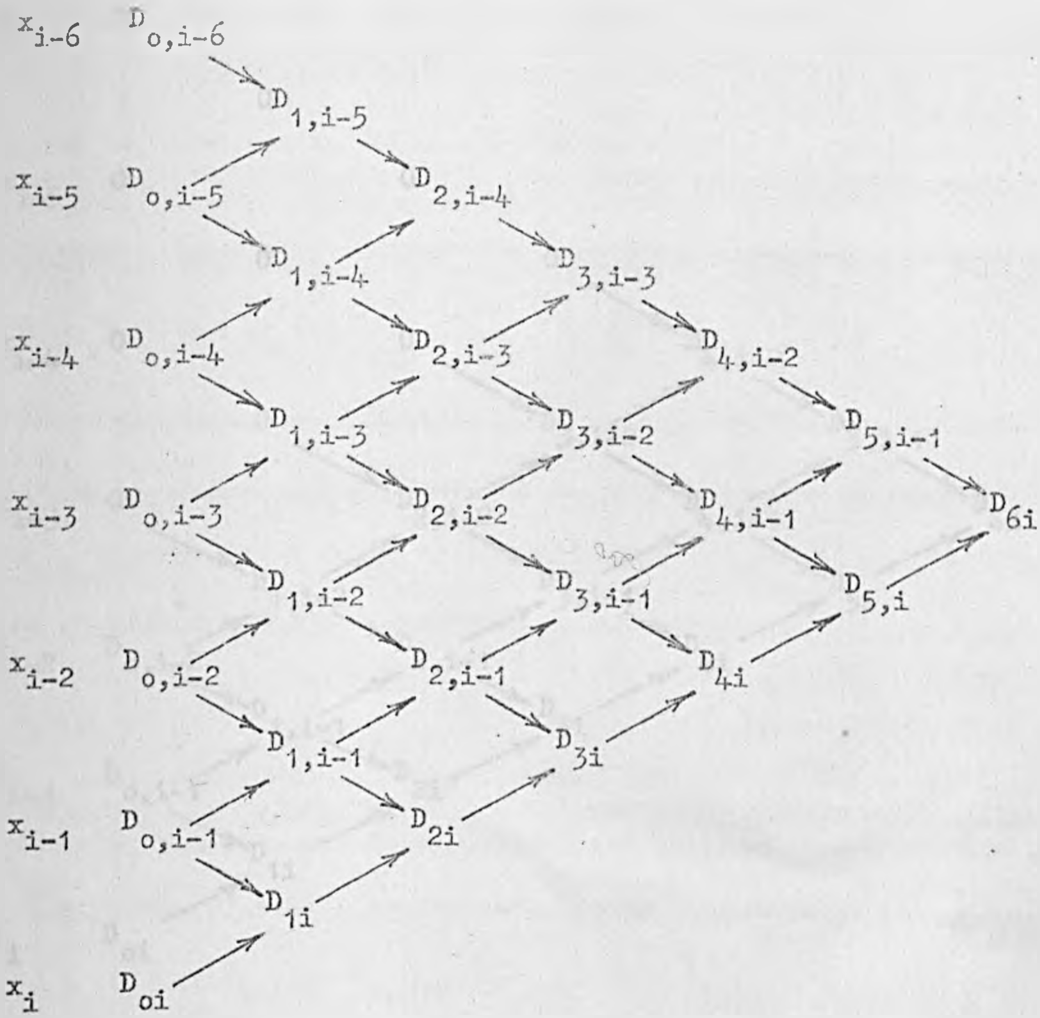

Fig. 3.3.1. Illustration of a computational scheme using divided differences for evaluating a B-spline.

In practice advantage can be taken of the property

$$D_{oj} = 0 \quad (x_j \leq x) , \tag{3.3.2}$$

in order to reduce the number of applications of (3.3.1). Thus if, for example, $x_{i-3} \leq x < x_{i-2}$, the above array takes the form indicated in Fig. 3.3.2.

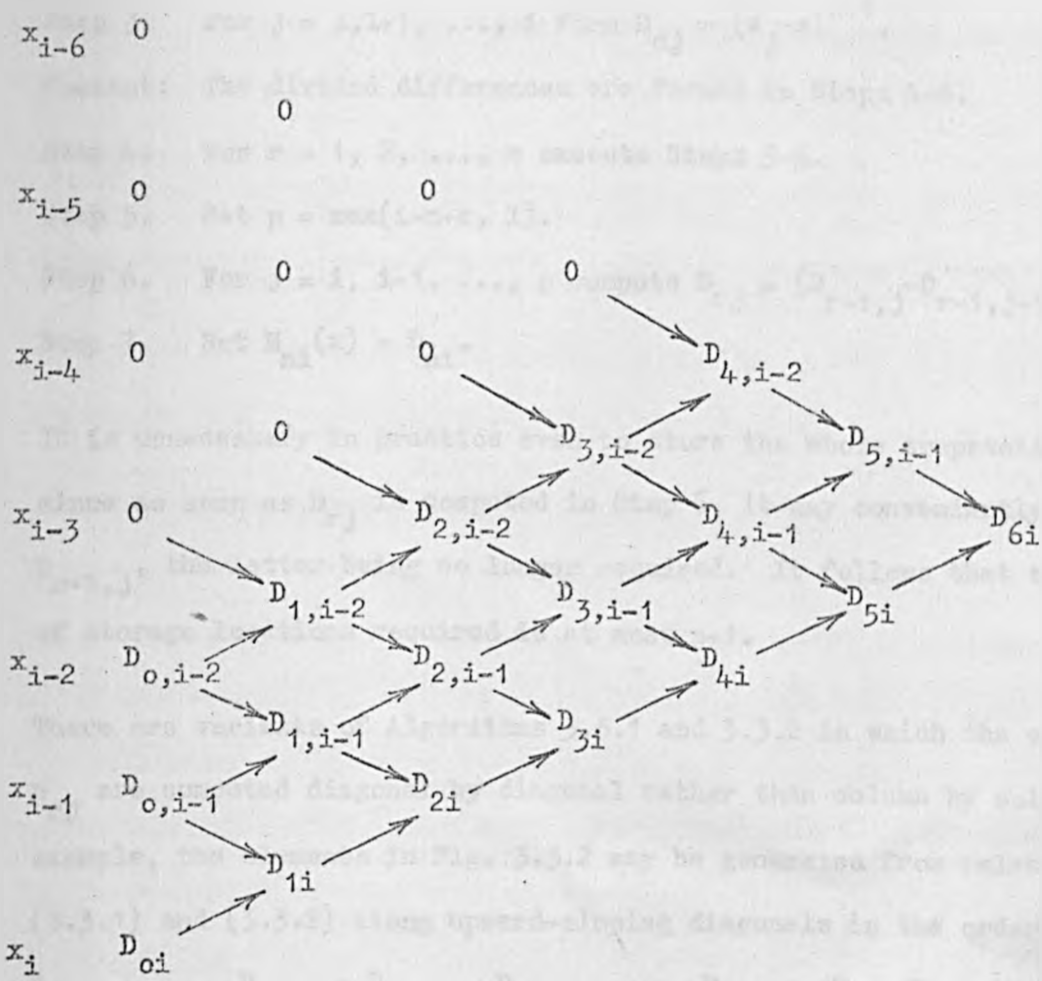

Fig. 3.3.2. Illustration of a more efficient scheme using divided differences for evaluating a B-spline.

In general it is necessary to compute and store only a trapezoidal array of non-zero elements. A modified version of the algorithm, taking advantage of (3.3.2) can be stated as follows:

<u>Algorithm 3.3.2</u>: Efficient evaluation of an isolated B-spline value using divided differences.

Comment: Find the interval containing x.

Step 1. Determine the unique integer l such that $x_{l-1} \leq x < x_l$.
Set $k = i-l$.

Comment: The initial conditions are set in Steps 2-3.

Step 2. For $r = 0, 1, 2, \ldots, n-k-1$ set $D_{r,l-1} = 0$.

Step 3. For $j = l, l+1, \ldots, i$ form $D_{oj} = (x_j - x)^{n-1}$.

Comment: The divided differences are formed in Steps 4-6.

Step 4. For $r = 1, 2, \ldots, n$ execute Steps 5-6.

Step 5. Set $p = \max(i-n+r, l)$.

Step 6. For $j = i, i-1, \ldots, p$ compute $D_{rj} = (D_{r-1,j} - D_{r-1,j-1})/(x_j - x_{j-r})$.

Step 7. Set $M_{ni}(x) = D_{ni}$.

It is unnecessary in practice even to store the whole trapezoidal array, since as soon as D_{rj} is computed in Step 6, it may conveniently overwrite $D_{r-1,j}$, the latter being no longer required. It follows that the number of storage locations required is at most n+1.

There are variants of Algorithms 3.3.1 and 3.3.2 in which the elements D_{rj} are computed diagonal by diagonal rather than column by column. For example, the elements in Fig. 3.3.2 may be generated from relations (3.3.1) and (3.3.2) along upward-sloping diagonals in the order $D_{o,i-2}$, $D_{1,i-2}, \ldots, D_{4,i-2}$; $D_{o,i-1}, D_{1,i-1}, \ldots, D_{5,i-1}$; $D_{oi}, D_{1i}, \ldots, D_{6i}$. Alternatively, along downward-sloping diagonals the successive elements D_{oi}; $D_{o,i-1}, D_{1i}$; $D_{o,i-2}, D_{1,i-1}, D_{2i}$; $D_{1,i-2}, D_{2,i-1}, D_{3i}$; ...; $D_{4,i-2}$, $D_{5,i-1}$, D_{6i} are generated. Computationally, there is little to choose between these various forms of the algorithm. They require similar amounts of computational effort and possess identical error-propagation characteristics.

The elements D_{rj} are all theoretically non-negative (Greville, 1969) and cancellation may therefore take place in computing (3.3.1) if $D_{r-1,j}$ and $D_{r-1,j-1}$ are of similar size. Hence the possibility exists of significant error growth in the computed values of the D_{rj} and hence of appreciable error in the computed value of $M_{ni}(x)$.

We expect therefore these algorithms based upon the use of divided differences to be unstable; this expectation is observed in practice, even for relatively "simple" examples (see Section 3.11). In particular, the algorithm breaks down completely in the case of multiple knots. In such cases the appropriate divided differences can be replaced by their limiting forms as derivatives, but even then very poor results are frequently obtained, as they are in cases of near-coincident knots. In Section 3.7 we use a running error analysis to give a posteriori bounds for the errors in the computed values of D_{rj}.

3.4. A recurrence relation for B-splines

We now state and prove a fundamental recurrence relation for B-splines; its use enables B-splines of order n to be evaluated from those of order n-1. The relation gives rise to a method for evaluating B-splines which we shall refer to subsequently as the method of convex combinations. This method and the method based on divided differences are analyzed in detail in the remainder of this chapter.

Theorem 3.4.1

The recurrence relation

$$M_{ni}(x) = \frac{(x-x_{i-n})M_{n-1,i-1}(x) + (x_i-x)M_{n-1,i}(x)}{x_i - x_{i-n}} \tag{3.4.1}$$

and its equivalent for normalized B-splines,

$$N_{ni}(x) = \left(\frac{x-x_{i-n}}{x_{i-1}-x_{i-n}}\right)N_{n-1,i-1}(x) + \left(\frac{x_i-x}{x_i-x_{i-n+1}}\right)N_{n-1,i}(x), \tag{3.4.2}$$

hold for all values of x.

Proof

A proof for the case of distinct knots, ie $x_{i-n} < x_{i-n+1} < \dots < x_i$, has been obtained (Cox, 1972) by making use of the explicit expression (3.2.4) for a B-spline. For the more general case, where the knots form an n-extended partition, ie $x_{i-n} \leqslant x_{i-n+1} \leqslant \dots \leqslant x_i$; $x_{i-n} < x_i$, the following more elegant proof has been given by de Boor (1972) who, independently of this work, also discovered the relation (3.4.1).

Leibnitz' formula for the nth divided difference of the function

$$h(y) = f(y)g(y) \tag{3.4.3}$$

in terms of the divided differences of f(y) and g(y) is

$$h(y_0,y_1,\ldots,y_n) = \sum_{j=0}^{n} f(y_0,y_1,\ldots,y_j)g(y_j,y_{j+1},\ldots,y_n). \tag{3.4.4}$$

The application of (3.4.4) to the function

$$h(y) \equiv M_n(y\ ;\ x) = (y-x)M_{n-1}(y\ ;\ x) \tag{3.4.5}$$

yields

$$M_n(x_{i-n},x_{i-n+1},\ldots,x_i\ ;\ x) = (x_{i-n}-x)M_{n-1}(x_{i-n},x_{i-n+1},\ldots,x_i\ ;\ x)$$

$$+\ 1.\ M_{n-1}(x_{i-n+1},x_{i-n+2},\ldots,x_i\ ;\ x)\ ,$$

since the divided differences of order greater than unity of the function y-x vanish. Thus, employing the properties of divided differences,

$$M_{ni}(x) = \left(\frac{x_{i-n}-x}{x_i-x_{i-n}}\right)\Big\{M_{n-1}(x_{i-n+1},x_{i-n+2},\ldots,x_i\ ;\ x)$$

$$-\ M_{n-1}(x_{i-n},x_{i-n+1},\ldots,x_{i-1}\ ;\ x)\Big\} + M_{n-1}(x_{i-n+1},x_{i-n+2},\ \ldots,\ x_i\ ;\ x)$$

$$= \frac{(x-x_{i-n})M_{n-1}(x_{i-n},x_{i-n+1},\ldots,x_{i-1}\ ;\ x)+(x_i-x)M_{n-1}(x_{i-n+1},x_{i-n+2},\ldots,x_i\ ;\ x)}{x_i - x_{i-n}}$$

$$= \frac{(x-x_{i-n})M_{n-1,i-1}(x)+(x_i-x)M_{n-1,i}(x)}{x_i - x_{i-n}}\ , \tag{3.4.6}$$

which establishes (3.4.1). Relation (3.4.2) then follows from (3.4.1) upon using (3.2.6). $\square$

We observe that (3.4.1) can be written as

$$M_{ni}(x) = \theta M_{n-1,i-1}(x) + (1-\theta)M_{n-1,i}(x) \ , \tag{3.4.7}$$

where

$$\theta = (x-x_{i-n})/(x_i-x_{i-n}) \ . \tag{3.4.8}$$

Now $M_{ni}(x) = 0$ for $x < x_{i-n}$ and $x > x_i$ (see Section 3.2). Hence, for the range of x over which $M_{ni}(x) \neq 0$, θ lies between 0 and 1. It follows that for $x_{i-n} \leq x < x_i$, $M_{ni}(x)$ is a <u>convex combination</u> of $M_{n-1,i-1}(x)$ and $M_{n-1,i}(x)$.

<u>Theorem 3.4.2</u>

For all $n > 0$ and all i,

$$M_{ni}(x),\ N_{ni}(x) \begin{cases} > 0 & (x_{i-n} < x < x_i) \\ = 0 & (x < x_{i-n},\ x_i < x) \ . \end{cases} \tag{3.4.9}$$

<u>Proof</u>

Assume the theorem is true for $n = r-1 > 0$ and all i, ie that $M_{r-1,i}(x) > 0$ for all i and $x_{i-r+1} < x < x_i$. Now consider relation (3.4.1) with n replaced by r. If $x_{i-r} < x < x_{i-1}$ then the term $(x-x_{i-r})M_{r-1,i-1}(x) > 0$. If $x_{i-r+1} < x < x_i$ then the term $(x_i-x)M_{r-1,i}(x) > 0$. If $x_{i-r} < x < x_i$ then at least one of these two terms is positive. It follows that $M_{ri}(x) > 0$ for $x_{i-r} < x < x_i$, ie the theorem is true for $n = r$. But the theorem is evidently true for $n = 1$, by virtue of (3.2.8). Hence, by induction, it is true for all n. $\square$

We refer subsequently to (3.4.9) as the <u>restricted</u> or <u>compact support property</u> of B-splines.

Note that in (3.4.9) we have omitted the end-points x_{i-n} and x_i. Normally, $M_{ni}(x)$ and $N_{ni}(x)$ are zero there too, but in the case $n = 1$ or if x_{i-n} is a knot of multiplicity n, it is straightforward to verify that they are <u>non-zero</u> at x_{i-n} as a consequence of (3.2.8), (3.4.1) and (3.2.6).

We now give an algorithm based upon (3.2.8) and (3.4.1) for evaluating $M_{ni}(x)$ (again we assume that $x_{i-n} \leqslant x < x_i$, otherwise $M_{ni}(x) = 0$) :

Algorithm 3.4.1: Evaluation of an isolated B-spline value using convex combinations.

Comment: Set the initial conditions.

Step 1. For $j = i-n+1, i-n+2, \ldots, i$, set $M_{1j} = M_{1j}(x)$.

Comment: B-splines are computed by convex combinations in Steps 2-3.

Step 2. For $r = 2, 3, \ldots, n$ execute Step 3.

Step 3. For $j = i-n+r, i-n+r+1, \ldots, i$ compute

$$M_{rj} = \frac{(x-x_{j-r})M_{r-1,j-1} + (x_j-x)M_{r-1,j}}{x_j - x_{j-r}} .$$

Step 4. Set $M_{ni}(x) = M_{ni}$.

For example, if $n = 6$, the elements in the following triangular array are computed:

$$\begin{tikzcd}[column sep=small, row sep=small]
M_{1,i-5} \arrow[dr] & & & & & \\
& M_{2,i-4} \arrow[dr] & & & & \\
M_{1,i-4} \arrow[ur] \arrow[dr] & & M_{3,i-3} \arrow[dr] & & & \\
& M_{2,i-3} \arrow[ur] \arrow[dr] & & M_{4,i-2} \arrow[dr] & & \\
M_{1,i-3} \arrow[ur] \arrow[dr] & & M_{3,i-2} \arrow[ur] \arrow[dr] & & M_{5,i-1} \arrow[dr] & \\
& M_{2,i-2} \arrow[ur] \arrow[dr] & & M_{4,i-1} \arrow[ur] \arrow[dr] & & M_{6i} \\
M_{1,i-2} \arrow[ur] \arrow[dr] & & M_{3,i-1} \arrow[ur] \arrow[dr] & & M_{5i} \arrow[ur] & \\
& M_{2,i-1} \arrow[ur] \arrow[dr] & & M_{4i} \arrow[ur] & & \\
M_{1,i-1} \arrow[ur] \arrow[dr] & & M_{3i} \arrow[ur] & & & \\
& M_{2i} \arrow[ur] & & & & \\
M_{1i} \arrow[ur] & & & & &
\end{tikzcd}$$

Fig.3.4.1. Illustration of a computational scheme using convex combinations for evaluating a B-spline.

As with the conventional divided difference algorithm, advantage can be taken of zero elements in the array in order to reduce the number of applications of (3.4.1). By making use of the relation (3.2.8) the above array takes, if for example $x_{i-3} \leq x < x_{i-2}$, the following form:

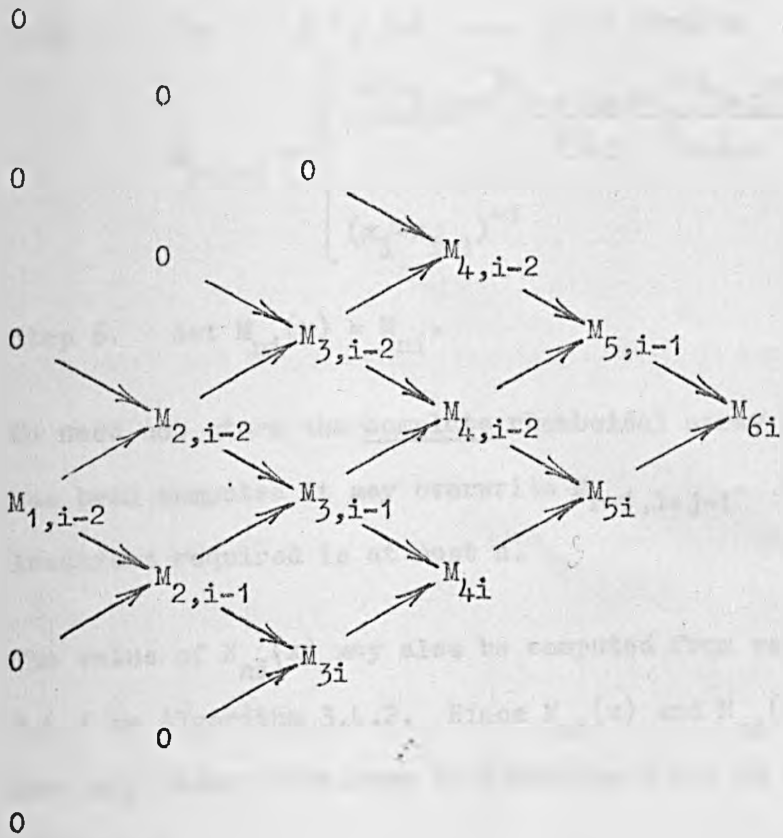

Fig. 3.4.2. Illustration of a more efficient scheme using convex combinations for evaluating a B-spline.

In general it is necessary to compute and store only a rhomboidal array of non-zero elements. A modified version of Algorithm 3.4.1, taking advantage of (3.2.8) can be stated as follows:

Algorithm 3.4.2: Efficient evaluation of an isolated B-spline value using convex combinations.

Comment: Find the interval containing x.

Step 1. Determine the unique integer l such that $x_{l-1} \leq x < x_l$.
Set $k = i-l$.

Comment: The initial conditions are set in Steps 2-3.

Step 2. For $r = 1, 2, \ldots, n-k-1$ set $M_{r,l-1} = 0$.

Step 3. For $r = 1, 2, \ldots, k$ set $M_{r,l+r} = 0$.

Comment: B-splines are computed by convex combinations in Steps 4-5.

Step 4. For $j = 0, 1, \ldots, k$ execute Step 5.

Step 5. For $r = j+1, j+2, \ldots, j+n-k$ compute

$$M_{r,l+j} = \begin{cases} \dfrac{(x-x_{l+j-r})M_{r-1,l+j-1}+(x_{l+j}-x)M_{r-1,l+j}}{x_{l+j}-x_{l+j-r}} & (r+j \neq 1) \\ (x_l-x_{l-1})^{-1} & (r+j = 1). \end{cases}$$

Step 6. Set $M_{ni}(x) = M_{ni}$.

We need not store the <u>complete</u> rhomboidal array since as soon as $M_{r,l+j}$ has been computed it may overwrite $M_{r-1,l+j-1}$. The number of storage locations required is at most n.

The value of $N_{ni}(x)$ may also be computed from variants of Algorithm 3.4.1 or Algorithm 3.4.2. Since $M_{ni}(x)$ and $N_{ni}(x)$ are related by (3.2.6), the only change necessary to Algorithm 3.4.1 is to omit the final division, producing

<u>Algorithm 3.4.3</u>: Evaluation of an isolated normalized B-spline value using convex combinations.

Comment: Set the initial conditions.

Step 1. For $j = i-n+1, i-n+2, \ldots, i$, set $M_{1j} = M_{1j}(x)$.

Comment: B-splines are computed by convex combinations in Steps 2-3.

Step 2. For $r = 2, 3, \ldots, n-1$ execute Step 3.

Step 3. For $j = i-n+r, i-n+r+1, \ldots i$ compute

$$M_{rj} = \frac{(x-x_{j-r})M_{r-1,j-1}+(x_j-x)M_{r-1,j}}{x_j - x_{j-r}}.$$

Step 4. Compute $N_{ni}(x) = (x-x_{i-n})M_{n-1,i-1} + (x_i-x)M_{n-1,i}$.

Algorithm 3.4.2 may also be modified similarly.

Before concluding this section we note that the elements in the array M_{rj} (or N_{rj}) can be computed column by column (as in Algorithms 3.4.1 and 3.4.3) or diagonal by diagonal (the diagonals either sloping upwards from left to right as in Algorithm 3.4.2, or downwards from left to right). As with the divided difference method, there is little to choose between these variants of the basic algorithm.

3.5 The values of B-splines at the ends of the range

At the ends of the range, B-splines defined upon a standard knot set with coincident end knots assume special values as established in Theorem 3.5.1 below.

Theorem 3.5.1

For B-splines of order n ($n \geqslant 1$), defined upon a standard knot set with coincident end knots,

$$N_{ni}(a) = N_{n,N+n-i}(b) = \begin{cases} 1 & (i = 1) \\ 0 & (i > 1) \end{cases}. \qquad (3.5.1)$$

Proof

The recurrence relation (3.4.1) yields

$$M_{ni}(a) = M_{n-1,i}(a) \quad (n > 1). \qquad (3.5.2)$$

But from (3.2.6), (3.2.8) and (3.2.9),

$$N_{1i}(a) = (x_1-a)M_{1i}(a) = \begin{cases} 1 & (i = 1) \\ 0 & (i > 1) \end{cases}, \qquad (3.5.3)$$

which in conjunction with (3.5.2) proves the theorem for $N_{ni}(a)$. In a similar manner we may prove the theorem for $N_{n,N+n-i}(b)$. $\square$

3.6 The sum of normalized B-splines and bounds for their values

It is important in problems of interpolation and least-squares approximation by splines (see, in particular, Chapters 6, 7 and 8) to know whether the matrices of basis functions are well-scaled. The value for the sum of normalized B-splines and the bounds for individual B-splines established in this section are particularly useful in such problems.

Let $\{\ldots, x_{-1}, x_0, x_1, \ldots\}$ be an n-extended partition of the real line.

Theorem 3.6.1

The normalized B-splines $N_{ni}(x)$ defined upon the knots $\ldots, x_{-1}, x_0, x_1, \ldots$ have the property

$$\sum_i N_{ni}(x) = 1 \tag{3.6.1}$$

for all x and all $n \geqslant 1$.

Proof

Summing (3.4.2) over i yields, for $n > 1$,

$$\sum_i N_{ni}(x) = \sum_i \left(\frac{x-x_{i-n}}{x_{i-1}-x_{i-n}}\right) N_{n-1,i-1}(x) + \sum_i \left(\frac{x_i-x}{x_i-x_{i-n+1}}\right) N_{n-1,i}(x). \tag{3.6.2}$$

Replacing i by i+1 in the first sum on the right-hand side of (3.6.2) gives

$$\sum_i N_{ni}(x) = \sum_i \left(\frac{x-x_{i-n+1}}{x_i-x_{i-n+1}}\right) N_{n-1,i}(x) + \sum_i \left(\frac{x_i-x}{x_i-x_{i-n+1}}\right) N_{n-1,i}(x)$$

$$= \sum_i N_{n-1,i}(x). \tag{3.6.3}$$

But from (3.2.9),

$$\sum_i N_{1i}(x) = 1. \tag{3.6.4}$$

Hence by induction the theorem is true for all x and all $n \geqslant 1$. $\square$

A generalization of (3.6.1) is considered in Chapter 5.

Theorem 3.6.2

$$\frac{1}{n} \leqslant \max_x N_{ni}(x) \leqslant 1 \qquad (n \geqslant 1). \tag{3.6.5}$$

Proof

We need only consider the interval $x_{i-n} \leqslant x \leqslant x_i$, since $N_{ni}(x)$ is zero outside this range. Now the average value of $N_{ni}(x)$ in this interval is, using (3.2.5), (3.2.6) and the compact support property,

$$\int_{x_{i-n}}^{x_i} N_{ni}(x)\,dx \Bigg/ \int_{x_{i-n}}^{x_i} dx = \left(\frac{x_i - x_{i-n}}{n}\right) / (x_i - x_{i-n}) = 1/n. \tag{3.6.6}$$

But the maximum value of a function over an interval must exceed (or at least be equal to) the average value of the function over the interval. Hence $\max_x N_{ni}(x) \geqslant \frac{1}{n}$. But $\max_x N_{ni}(x) \leqslant 1$ because of (3.6.1) and the non-negativity of the B-splines. Hence the theorem is proved. $\square$

Conjecture 3.6.1

$$\frac{1}{n-1} \leqslant \max_x N_{ni}(x) \leqslant 1 \qquad (n \geqslant 2). \tag{3.6.7}$$

We give proofs of this conjecture for the cases $n = 2$ and 3.

Theorem 3.6.3

For $n = 2$ and 3,

$$\frac{1}{n-1} \leqslant \max_x N_{ni}(x) \leqslant 1. \tag{3.6.8}$$

Proof for $n = 2$

Since $N_{2i}(x)$ is the only first-degree B-spline which is non-zero at $x = x_{i-1}$, the identity $\sum_j N_{2j}(x) = 1$ yields $N_{2i}(x_{i-1}) = 1$. Thus $\max_x N_{2i}(x) \geqslant 1$. But $N_{2i}(x) \leqslant 1$. Hence $\max_x N_{2i}(x) = 1$. $\square$

Proof for n = 3

The use of the recurrence relation (3.4.2), after setting $i = 3$ and transforming linearly the interval (x_0, x_3) to $(0, 1)$ with no loss of generality, yields

$$N_{33}(x) = \begin{cases} \dfrac{x^2}{x_1 x_2} & (0 \leqslant x < x_1) \\[2ex] \dfrac{x(x_2-x)/x_2+(1-x)(x-x_1)/(1-x_1)}{x_2 - x_1} & (x_1 \leqslant x < x_2) \\[2ex] \dfrac{(1-x)^2}{(1-x_1)(1-x_2)} & (x_2 \leqslant x < 1) \end{cases} \quad (3.6.8)$$

Since $N'_{33}(x)$ is continuous and increases monotonically from zero for $0 \leqslant x < x_1$, is linear for $x_1 \leqslant x < x_2$ and decreases monotonically to zero for $x_2 \leqslant x < 1$, it follows that $N_{33}(x)$ attains its (unique) maximum between x_1 and x_2. The maximizing value of x is given by

$$\frac{x_2-2x}{x_2} + \frac{1+x_1-2x}{1-x_1} = 0 \ , \qquad (3.6.9)$$

ie

$$x = \frac{x_2}{1+x_2-x_1} \ . \qquad (3.6.10)$$

Substituting this value into (3.6.8) gives

$$\max_x N_{33}(x) = 1/(1+x_2-x_1). \qquad (3.6.11)$$

Thus, since $0 \leqslant x_1 \leqslant x_2 < 1$, the result

$$\tfrac{1}{2} \leqslant \max_x N_{33}(x) \leqslant 1 \qquad (3.6.12)$$

follows. $\square$

3.7 A posteriori error bounds for the values of B-splines computed from divided differences

We derive in this section a posteriori error bounds for the values of B-splines computed from Algorithm 3.3.1 or from Algorithm 3.3.2 using a running error analysis (cf Section 1.2).

Theorem 3.7.1

Let the (simple) knots $x_{i-n}, x_{i-n+1}, \ldots, x_i$ and the argument x be given standard floating-point numbers (in Section 3.10 we return to the implications of this assumption). For the given value of x let $\bar{D}_{rj}$ denote the computed value of D_{rj} obtained from (3.3.1) if $r > 0$ or from $(x_j-x)_+^{n-1}$ if $r = 0$. Let

$$\delta D_{rj} = \bar{D}_{rj} - D_{rj} \, . \tag{3.7.1}$$

Then

$$\left|\delta D_{rj}\right| \leqslant 2^{-t_1} F_{rj}, \tag{3.7.2}$$

where F_{rj} is defined by the recurrence relation

$$F_{oj} = (n-1)\bar{D}_{oj} \, , \tag{3.7.3}$$

$$F_{rj} = \frac{(F_{r-1,j}+F_{r-1,j-1})+3\left|\bar{D}_{r-1,j}-\bar{D}_{r-1,j-1}\right|}{x_j - x_{j-r}} \quad (r > 0). \tag{3.7.4}$$

Proof

From (3.3.1),

$$\bar{D}_{rj} = fl\left(\frac{\bar{D}_{r-1,j}-\bar{D}_{r-1,j-1}}{x_j - x_{j-r}}\right)$$

$$= \left(\frac{\bar{D}_{r-1,j}-\bar{D}_{r-1,j-1}}{x_j - x_{j-r}}\right)(1+\varepsilon_1)(1+\varepsilon_2)(1+\varepsilon_3)$$

$$= \left(\frac{\bar{D}_{r-1,j}-\bar{D}_{r-1,j-1}}{x_j - x_{j-r}}\right)(1+3\varepsilon) \, , \tag{3.7.5}$$

where $|\varepsilon_i| \leqslant 2^{-t}$ $(i = 1, 2, 3)$ and $|e| < 2^{-t_1}$ and depend upon r and j. Hence, using (3.7.1)

$$D_{rj} + \delta D_{rj} = \frac{D_{r-1,j}-D_{r-1,j-1}+\delta D_{r-1,j}-\delta D_{r-1,j-1}+3e(\bar{D}_{r-1,j}-\bar{D}_{r-1,j-1})}{x_j - x_{j-r}} . \tag{3.7.6}$$

Subtraction of (3.3.1) from (3.7.6) gives

$$\delta D_{rj} = \frac{\delta D_{r-1,j}-\delta D_{r-1,j-1}+3e(\bar{D}_{r-1,j}-\bar{D}_{r-1,j-1})}{x_j - x_{j-r}} . \tag{3.7.7}$$

If F_{rj} $(r > 0)$ is defined by (3.7.4) and F_{oj} by (3.7.3), which is obtained by a simple error analysis of the computation of $(x_j-x)^{n-1}$, the theorem then follows. $\square$

In proving (3.7.3) it is assumed that D_{oj} is evaluated by forming $x_j-x = h$, say, and then computing h^{n-1} by repeated multiplication. Such an approach is in accordance with the Algol 60 report (Naur, 1963), and is therefore appropriate if the method is programmed in Algol 60. If n is sufficiently large it may be more accurate (and faster) to compute h^{n-1} from $\exp\left\{(n-1)\ln(h)\right\}$; we do not consider this approach here since any effects the alternative computation have on our general results are insignificant.

The computer itself can be made to determine the values of F_{rj}, since they depend solely on previous values together with the computed values of D_{rj}. In particular, $2^{-t_1}F_{ni}$ is a bound for the error in the computed value of $M_{ni}(x)$. In practice the computer makes rounding errors in computing the F_{rj} (cf Section 1.2). However, we see from a simple error analysis that the further contribution to the error incurred from a single computation of (3.7.4) is at most a multiplicative factor of $(1-2^{-t})^{-5}$. Since the contribution to the error incurred in computing F_{oj} from (3.7.3) is at most a factor $(1-2^{-t})^{-2}$, it follows that $(1-2^{-t})^{-5r-2}2^{-t_1}\bar{F}_{rj}$, where $\bar{F}_{rj}$ is the computed value of F_{rj}, gives a rigorous error bound for $|\delta D_{rj}|$. Now $(1-2^{-t})^{-5r-2} < 1.112$, by virtue of (1.1.12). Hence a simpler but marginally cruder bound for $|\delta D_{rj}|$ is $(1.112)2^{-t_1}\bar{F}_{rj}$ or $(1.179)2^{-t}\bar{F}_{rj}$.

These a posteriori bounds prove to be reasonably realistic (see the examples in Section 3.11). Moreover, since advantage is taken of any cancellations that occur during the course of the computation, these bounds are considerably better than those obtainable from an a priori error analysis (but see the special case discussed in Section 3.11).

3.8 A posteriori error bounds for the values of B-splines computed by the method of convex combinations

We derive in this section, again as the result of a running error analysis, a posteriori error bounds for B-splines computed from Algorithm 3.4.1 or from Algorithm 3.4.2. As before we assume that the $x_{i-n}, x_{i-n+1}, \ldots, x_i$ and the argument x are given floating-point numbers, and in Section 3.10 return to the consequences of this assumption.

Theorem 3.8.1

For the given argument x let $\bar{M}_{rj}(x)$ denote the value of $M_{rj}(x)$ computed from recurrences (3.4.1) and (3.2.8). Let

$$\delta M_{rj}(x) = \bar{M}_{rj}(x) - M_{rj}(x). \tag{3.8.1}$$

Then

$$\left|\delta M_{rj}(x)\right| \leqslant 2^{-t_1} H_{rj}, \tag{3.8.2}$$

where H_{rj} is defined by the recurrence relation

$$H_{1j} = 2\bar{M}_{1j}(x), \tag{3.8.3}$$

$$H_{rj} = \frac{(x-x_{j-r})\left\{H_{r-1,j-1}+5\bar{M}_{r-1,j-1}(x)\right\}+(x_j-x)\left\{H_{r-1,j}+5\bar{M}_{r-1,j}(x)\right\}}{x_j - x_{j-r}} \quad (r>1). \tag{3.8.4}$$

Proof

From (3.4.1),

$$\bar{M}_{rj}(x) = fl\left(\frac{(x-x_{j-r})\bar{M}_{r-1,j-1}(x)+(x_j-x)\bar{M}_{r-1,j}(x)}{x_j - x_{j-r}}\right) \quad (3.8.5)$$

$$= \left\{(x-x_{j-r})(1+\varepsilon_1)\bar{M}_{r-1,j-1}(x)(1+\varepsilon_2)+(x_j-x)(1+\varepsilon_3)\bar{M}_{r-1,j}(x)(1+\varepsilon_4)\right\} \times$$

$$\times(1+\varepsilon_5)(1+\varepsilon_6)(1+\varepsilon_7)/(x_j-x_{j-r}) , \quad (3.8.6)$$

where $|\varepsilon_i| \leqslant 2^{-t}$ $(i = 1, 2, \ldots, 7)$ and depend upon r and j. Thus

$$\bar{M}_{rj}(x) = \frac{(x-x_{j-r})\bar{M}_{r-1,j-1}(x)(1+5e_1)+(x_j-x)\bar{M}_{r-1,j}(x)(1+5e_2)}{x_j - x_{j-r}} , \quad (3.8.7)$$

where $|e_i| < 2^{-t_1}$ $(i = 1, 2)$ and depend upon r and j. Upon making use of (3.8.1) and (3.4.1), equation (3.8.7) gives

$$\delta M_{rj}(x) = \frac{(x-x_{j-r})\left\{\delta M_{r-1,j-1}(x)+5e_1\bar{M}_{r-1,j-1}(x)\right\}+(x_j-x)\left\{\delta M_{r-1,j}(x)+5e_2\bar{M}_{r-1,j}(x)\right\}}{x_j - x_{j-r}} \quad (3.8.8)$$

The theorem follows from (3.8.8) and a simple error analysis of the evaluation of (3.2.8). $\square$

Once again the computer can be used to determine the values of H_{rj} and in particular the value $2^{-t_1}H_{ni}$, which is a bound for the error in the computed value of $M_{ni}(x)$. As before the computer makes rounding errors in forming the H_{rj}. However, it is straightforward to verify that the contribution to the error incurred in computing H_{rj} from (3.8.4) is at most a factor $(1-2^{-t})^{-7}$. Since the error incurred in computing H_{1j} from (3.8.3) is at most a factor $(1-2^{-t})^{-1}$ it follows that $(1-2^{-t})^{6-7r}2^{-t_1}\bar{H}_{rj}$, where $\bar{H}_{rj}$ is the computed value of H_{rj}, gives a rigorous error bound for $|\delta M_{rj}|$. Again the simpler bound $|\delta M_{rj}| \leqslant (1.179)2^{-t}H_{rj}$ may be preferred.

Although the above analysis gives rise to extremely realistic <u>a posteriori</u> bounds for the errors in the computed values of the $M_{rj}(x)$, the need to compute such bounds, which roughly doubles the work involved in computing the $M_{rj}(x)$ alone, is obviated when account is taken of the non-negativity of the $\bar{M}_{rj}(x)$ (see Theorem 3.8.2 below). It is shown in Section 3.9 that realistic a posteriori bounds for the absolute errors in the $\bar{M}_{rj}(x)$ can be deduced immediately from the computed results. Moreover, a priori bounds for the relative errors are also derived in Section 3.9.

Theorem 3.8.2

Even in the presence of rounding errors the values of $M_{ni}(x)$ ($n \geqslant 1$, all i, $x_{i-n} < x < x_i$) computed in floating-point arithmetic from (3.4.1) are strictly positive.

Proof

The proof is similar to that of Theorem 3.4.2, except that the relevant recurrence is (3.8.7), rather than (3.4.1). Since, in (3.8.7), the terms $1+5e_1$ and $1+5e_2$ are both strictly positive, the theorem is proved. $\square$

We noted in Section 3.3 that the values of D_{rj} in the method employing divided differences are theoretically non-negative, but their computed values may be so inaccurate that they actually take negative values (see the examples in Section 3.11).

3.9 A priori error bounds for the values of B-splines computed by the method of convex combinations

In this section we establish a priori error bounds for the values of B-splines computed by the method of convex combinations. Firstly, however, we derive a readily-computable a posteriori error bound for the computed value of $M_{ni}(x)$.

Theorem 3.9.1

The values of H_{rj} defined by relations (3.8.3) and (3.8.4) satisfy the inequality

$$H_{rj} \leqslant (1-2^{-t})^{5(1-r)}(5r-3)\bar{M}_{rj}(x). \tag{3.9.1}$$

Proof

We first assume the theorem to be true for $r = s-1$, where $s > 0$, ie that

$$H_{s-1,j} \leqslant (1-2^{-t})^{5(2-s)}(5s-8)\bar{M}_{s-1,j}(x). \tag{3.9.2}$$

The substitution of (3.9.2) into the right-hand side of (3.8.4), after replacing r in the latter relation by s, then gives

$$H_{sj} \leqslant \left\{(1-2^{-t})^{5(2-s)}(5s-8)+5\right\}\left\{\frac{(x-x_{j-s})\bar{M}_{s-1,j-1}(x)+(x_j-x)\bar{M}_{s-1,j}(x)}{x_j - x_{j-s}}\right\}. \tag{3.9.3}$$

But it follows from (3.8.6) and Theorem 3.8.2 that

$$\bar{M}_{sj}(x) \geqslant (1-2^{-t})^5\left\{\frac{(x-x_{j-s})\bar{M}_{s-1,j-1}(x)+(x_j-x)\bar{M}_{s-1,j}(x)}{x_j - x_{j-s}}\right\}. \tag{3.9.4}$$

Hence

$$\begin{aligned} H_{sj} &\leqslant \left\{(1-2^{-t})^{5(2-s)}(5s-8)+5\right\}(1-2^{-t})^{-5}\bar{M}_{sj}(x) \\ &< (1-2^{-t})^{5(1-s)}(5s-3)\bar{M}_{sj}(x) . \end{aligned} \tag{3.9.5}$$

Thus (3.9.1) is true for $r = s$. But it is true for $r = 1$ by virtue of (3.8.3). Hence by induction it is true for all $r > 0$. $\square$

A slightly cruder bound, obtained by means of (1.1.12) is

$$H_{rj} \leqslant 1.112(5r-3)\bar{M}_{rj}(x) . \tag{3.9.6}$$

It follows from (3.8.2) that the computed value of $M_{ni}(x)$ differs from the true value by an amount not exceeding $1.112(5n-3)2^{-t_1}\bar{M}_{ni}(x)$, or, equivalently, $1.179(5n-3)2^{-t}\bar{M}_{ni}(x)$. We observe that this bound is computable, since it involves the value of $\bar{M}_{ni}(x)$, rather than the true (unknown) value $M_{ni}(x)$.

We now give a bound for the <u>relative error</u> in the computed value of $M_{ni}(x)$.

<u>Theorem 3.9.2</u>

$\delta M_{ni}(x)$ satisfies the <u>relative error bound</u>

$$\left|\delta M_{ni}(x)\right| / M_{ni}(x) \leqslant 1.337(5n-3)2^{-t}. \tag{3.9.7}$$

<u>Proof</u>

Now

$$\left|\delta M_{ni}(x)\right| \leqslant 1.179(5n-3)2^{-t}\bar{M}_{ni}(x) \tag{3.9.8}$$

$$= 1.179(5n-3)2^{-t}\left\{M_{ni}(x)+\delta M_{ni}(x)\right\} \tag{3.9.9}$$

and hence

$$\left|\delta M_{ni}(x)\right| \leqslant \frac{1.179(5n-3)2^{-t}M_{ni}(x)}{1-1.179(5n-3)2^{-t}} . \tag{3.9.10}$$

But from (1.1.7) it follows that the denominator is bounded from below by $1-0.1179 = 0.8821$. Hence

$$\left|\delta M_{ni}(x)\right| \leqslant 1.337(5n-3)2^{-t}M_{ni}(x) , \tag{3.9.11}$$

from which (3.9.7) follows immediately. □

The <u>a priori</u> bound (3.9.7) is remarkable in that it is <u>independent of the positions of the knots</u> (but see the comments in Section 3.10 on the effects of decimal to binary conversion). It follows for example that B-splines of order 15 or less can be evaluated with a loss of accuracy not exceeding 100 units in the least significant binary place. Such a result compares extremely favourably with the conventional method employing divided differences for which non-pathological examples of order very much smaller than 15 (see Section 3.11) can be constructed that yield no correct figures (on the KDF9 computer for which $t = 39$) in the results.

The counterparts of (3.9.6) and (3.9.7) in the case of normalized B-splines are, as a consequence of the omission of the final division when forming $N_{ni}(x)$ from $M_{n-1,i-1}(x)$ and $M_{n-1,i}(x)$ (see Algorithm 3.4.3),

$$H_{ni} \leqslant 1.112(5n-5)\bar{N}_{ni}(x) = 5.56(n-1)\bar{N}_{ni}(x) \qquad (3.9.12)$$

and

$$\left|\bar{N}_{ni}(x) - N_{ni}(x)\right| \leqslant 1.337(5n-5)2^{-t}N_{ni}(x) = 6.685(n-1)2^{-t}N_{ni}(x). \qquad (3.9.13)$$

3.10 The effects of perturbations in the data

There is one aspect of the problem that our analyses have so far not covered, viz the sensitivity of the computed values of $M_{ni}(x)$ with respect to perturbations in the data. By <u>data</u> in this context we mean the given values of the knots $x_{i-n}, x_{i-n+1}, \ldots, x_i$ and the argument x. A particular reason why the study of such perturbations is important is that our analyses are rigorous only for data that can be represented exactly as standard floating-point numbers. However, we may be comforted by the fact that our analyses <u>do</u> apply to the problem defined by the data <u>stored in the computer</u>. So it follows that the method of convex combinations solves accurately a problem with data perturbed slightly from that given. For most computers the perturbed (stored) data differs from that given by relative errors bounded in modulus by 2^{-t}.

<u>A posteriori</u> bounds relating to the <u>given</u> data, rather than the <u>stored</u> data, can be found if required by an extension of the running error analyses described in Section 3.8. The manner in which this analysis is carried out is straightforward but tedious and is not given here. One consequence of this analysis is that the bounds are now no longer independent of the knot spacing. However, unless the knot spacing is

highly non-uniform, the bounds for the stable method appear to depend only mildly upon the positions of the knots, although it is now no longer possible to quote <u>a priori</u> bounds. On the other hand the bounds (as well as the computed values themselves) for the conventional method seem to be very sensitive to small perturbations in the knots, which is a reflection of the inherent instability of that method.

3.11 Numerical examples

In order to compare numerically the conventional method (based upon divided differences) and the stable method (based upon convex combinations) we give a number of examples. For each case we consider the B-spline of a prescribed order n based on a given set of knots $x_{i-n}, x_{i-n+1}, \ldots, x_i$. The B-spline is evaluated by both methods (using Algorithms 3.3.2 and 3.4.2) at the positions of the interior knots $x_{i-n+1}, x_{i-n+2}, \ldots, x_{i-1}$. For the conventional method the error bound $(1.179)2^{-t}\bar{F}_{ni}$ and for the stable method the error bound $1.179(5n-3)2^{-t}\bar{M}_{ni}$ are also quoted.

<u>Example 3.11.1</u>

Degree 5. Knots 0, 1, 2, 3, 4, 5, 6 (Table 3.11.1). The values produced by the conventional method differ only slightly from those given by the stable method.

Table 3.11.1

Degree 5. Knots 0,1,2,3,4,5,6

x	Conventional method		Stable method	
	Value	Error bound	Value	Error bound
1	$1.38888\ 88892_{10}{-3}$	$3.52490_{10}{-10}$	$1.38888\ 88889_{10}{-3}$	$8.04221_{10}{-14}$
2	$3.61111\ 11111_{10}{-2}$	$9.18241_{10}{-11}$	$3.61111\ 11111_{10}{-2}$	$2.09097_{10}{-12}$
3	$9.16666\ 66667_{10}{-2}$	$1.81933_{10}{-11}$	$9.16666\ 66667_{10}{-2}$	$5.30786_{10}{-12}$
4	$3.61111\ 11111_{10}{-2}$	$2.22799_{10}{-12}$	$3.61111\ 11111_{10}{-2}$	$2.09097_{10}{-12}$
5	$1.38888\ 88889_{10}{-3}$	$6.85077_{10}{-14}$	$1.38888\ 88889_{10}{-3}$	$8.04221_{10}{-14}$

Example 3.11.2

Degree 21. Knots 0,1,2,...,22 (Table 3.11.2). For values of $x \geq 13$ the conventional method produces results of comparable accuracy to those of the stable method. However, as x is decreased from 13 to unity the conventional method becomes less and less accurate. Indeed, all values for $x<6$ have no correct figures at all. (Note that since this B-spline is symmetric about $x=11$ the conventional method could be used to give reliable results by replacing x by $22-x$ if $x<11$).

Table 3.11.2

Degree 21. Knots 0,1,2,...,22

x	Conventional method		Stable method	
	Value	Error bound	Value	Error bound
1	$-2.90493\ 62704_{10}-4$	$1.07302_{10}-1$	$8.89679\ 13924_{10}-22$	$2.04156_{10}-31$
2	$1.70851\ 18477_{10}-4$	$2.87569_{10}-2$	$1.86577\ 28133_{10}-15$	$4.28141_{10}-25$
3	$-5.09073\ 29585_{10}-5$	$7.22242_{10}-3$	$9.26531\ 08069_{10}-12$	$2.12612_{10}-21$
4	$8.78471\ 73001_{10}-6$	$1.69058_{10}-3$	$3.70854\ 13543_{10}-9$	$8.51003_{10}-19$
5	$5.18082\ 90829_{10}-8$	$3.66527_{10}-4$	$3.40296\ 26271_{10}-7$	$7.80881_{10}-17$
6	$1.07095\ 90738_{10}-5$	$7.30865_{10}-5$	$1.10732\ 92030_{10}-5$	$2.54100_{10}-15$
7	$1.59735\ 14258_{10}-4$	$1.32968_{10}-5$	$1.59595\ 80785_{10}-4$	$3.66226_{10}-14$
8	$1.15687\ 81534_{10}-3$	$2.18684_{10}-6$	$1.15690\ 83302_{10}-3$	$2.65477_{10}-13$
9	$4.55429\ 00755_{10}-3$	$3.21645_{10}-7$	$4.55428\ 59425_{10}-3$	$1.04508_{10}-12$
10	$1.01945\ 49442_{10}-2$	$4.17744_{10}-8$	$1.01945\ 49722_{10}-2$	$2.33935_{10}-12$
11	$1.33010\ 31228_{10}-2$	$4.71872_{10}-9$	$1.33010\ 31238_{10}-2$	$3.05220_{10}-12$
12	$1.01945\ 49726_{10}-2$	$4.55067_{10}-10$	$1.01945\ 49722_{10}-2$	$2.33935_{10}-12$
13	$4.55428\ 59422_{10}-3$	$3.66011_{10}-11$	$4.55428\ 59425_{10}-3$	$1.04508_{10}-12$
14	$1.15690\ 83302_{10}-3$	$2.37665_{10}-12$	$1.15690\ 83302_{10}-3$	$2.65477_{10}-13$
15	$1.59595\ 80785_{10}-4$	$1.18145_{10}-13$	$1.59595\ 80785_{10}-4$	$3.66226_{10}-14$
16	$1.10732\ 92030_{10}-5$	$4.08175_{10}-15$	$1.10732\ 92030_{10}-5$	$2.54100_{10}-15$
17	$3.40296\ 26271_{10}-7$	$8.26650_{10}-17$	$3.40296\ 26271_{10}-7$	$7.80881_{10}-17$
18	$3.70854\ 13543_{10}-9$	$7.37998_{10}-19$	$3.70854\ 13543_{10}-9$	$8.51003_{10}-19$
19	$9.26531\ 08068_{10}-12$	$1.73796_{10}-21$	$9.26531\ 08069_{10}-12$	$2.12612_{10}-21$
20	$1.86577\ 28133_{10}-15$	$3.48119_{10}-25$	$1.86577\ 28133_{10}-15$	$4.28141_{10}-25$
21	$8.89679\ 13924_{10}-22$	$1.65996_{10}-31$	$8.89679\ 13924_{10}-22$	$2.04156_{10}-31$

Example 3.11.3

Degree 3. Knots -10000, -9999, 0, 9999, 10000 (Table 3.11.3). This example is included to illustrate how erroneous the results of the conventional method can be for a degree as low as three, if the knot spacing is highly non-uniform. Such a case is important in practice since it is often of interest to investigate the case of near-coincident knots. We see that at $x=-9999$ the conventional method produces a negative value. Even at $x=0$, the peak of the B-spline, three figures have been lost. At $x=9999$ the result agrees with that of the stable method.

Table 3.11.3

Degree 3. Knots -10000, -9999, 0, 9999, 10000

x	Conventional method		Stable method	
	Value	Error bound	Value	Error bound
-9999	$-1.50012\ 05611_{10}-12$	$2.57409_{10}-11$	$2.50012\ 50063_{10}-13$	$9.11496_{10}-24$
0	$2.50012\ 49812_{10}-5$	$3.21865_{10}-12$	$2.50012\ 50062_{10}-5$	$9.11496_{10}-16$
9999	$2.50012\ 50063_{10}-13$	$8.04261_{10}-24$	$2.50012\ 50063_{10}-13$	$9.11496_{10}-24$

Example 3.11.4

Degree 9. Knots 2^j, $j=0,1,\ldots,10$ (Table 3.11.4)

Again the conventional method produces very inaccurate results (from the point of view of relative errors) for the small values of x. However, in terms of absolute error (measured with respect to the peak height), the conventional method appears perfectly adequate. Since in many applications such results would be quite acceptable we might expect that for examples similar to this the conventional method is satisfactory. Example 5 shows that this is not the case.

Table 3.11.4

Degree 9. Knots 1,2,4,8,16,32,64,128,256,512,1024

x	Conventional method		Stable method	
	Value	Error bound	Value	Error bound
2	$-1.60028\ 39327_{10}-14$	$9.53785_{10}-13$	$9.60166\ 98247_{10}-17$	$9.67807_{10}-27$
4	$1.80654\ 27507_{10}-12$	$9.11019_{10}-13$	$1.79167\ 15893_{10}-12$	$1.80593_{10}-22$
8	$1.97471\ 48111_{10}-\ 9$	$8.33280_{10}-13$	$1.97471\ 78228_{10}-\ 9$	$1.99043_{10}-19$
16	$3.81224\ 92651_{10}-\ 7$	$7.03203_{10}-13$	$3.81224\ 94453_{10}-\ 7$	$3.84258_{10}-17$
32	$1.72139\ 97270_{10}-\ 5$	$5.14271_{10}-13$	$1.72139\ 97251_{10}-\ 5$	$1.73510_{10}-15$
64	$1.95187\ 17159_{10}-\ 4$	$2.95574_{10}-13$	$1.95187\ 17160_{10}-\ 4$	$1.96740_{10}-14$
128	$5.17660\ 42895_{10}-\ 4$	$1.14874_{10}-13$	$5.17660\ 42895_{10}-\ 4$	$5.21779_{10}-14$
256	$2.40474\ 09004_{10}-\ 4$	$2.19807_{10}-14$	$2.40474\ 09004_{10}-\ 4$	$2.42387_{10}-14$
512	$6.59821\ 72615_{10}-\ 6$	$5.51868_{10}-16$	$6.59821\ 72615_{10}-\ 6$	$6.65072_{10}-16$

Example 3.11.5

Degree 9. Knots -2^{10-j}, $j=0,1,\ldots,10$ (Table 3.11.5).

This example is identical to Example 3.11.4, except that the knots have been reflected about the origin. The stable method gives identical results in each case. The conventional method again gives its most inaccurate results for the smaller (i.e. more negative) values of x. However, these values not only have large relative errors, but they also possess large absolute errors.

Table 3.11.5

Degree 9. Knots -1024, -512, -256, -128, -64, -32, -16, -8, -4, -2, -1

x	Conventional method		Stable method	
	Value	Error bound	Value	Error bound
-512	$-1.96549\ 35593_{10}-4$	$2.03640_{10}-2$	$6.59821\ 72615_{10}-6$	$6.65072_{10}-16$
-256	$2.41473\ 72045_{10}-4$	$3.80279_{10}-5$	$2.40474\ 09004_{10}-4$	$2.42387_{10}-14$
-128	$5.17659\ 22318_{10}-4$	$6.81279_{10}-8$	$5.17660\ 42895_{10}-4$	$5.21779_{10}-14$
-64	$1.95187\ 16988_{10}-4$	$1.13289_{10}-10$	$1.95187\ 17160_{10}-4$	$1.96740_{10}-14$
-32	$1.72139\ 97255_{10}-5$	$1.67259_{10}-13$	$1.72139\ 97251_{10}-5$	$1.73510_{10}-15$
-16	$3.81224\ 94452_{10}-7$	$2.18970_{10}-16$	$3.81224\ 94453_{10}-7$	$3.84258_{10}-17$
-8	$1.97471\ 78228_{10}-9$	$2.63580_{10}-19$	$1.97471\ 78228_{10}-9$	$1.99043_{10}-19$
-4	$1.79167\ 15893_{10}-12$	$1.54897_{10}-22$	$1.79167\ 15893_{10}-12$	$1.80593_{10}-22$
-2	$9.60166\ 98247_{10}-17$	$8.03074_{10}-27$	$9.60166\ 98247_{10}-17$	$9.67807_{10}-27$

In all the above examples the error bounds for the stable method are realistic. For the conventional method they are somewhat pessimistic, but are considerably better than could be achieved using some form of a priori error analysis (but see the special case $x_{i-1} \leq x < x_i$ considered below).

In every case the accuracy of the conventional method falls off as x ranges from x_i to x_{i-n}. However, for values of x sufficiently close to x_i, and always for values of x between x_{i-1} and x_i, values comparable in accuracy to those given by the stable method are produced. This result is easily explained by means of a running error analysis along the lines of Section 3.8 followed by an inductive proof similar in nature to that in Section 3.9. In fact the a priori bound,

$$|F_{ni}| \leq 1.179(4n-1)2^{-t}\bar{D}_{ni} \quad (x_{i-1} \leq x < x_i)\ , \qquad (3.11.1)$$

is readily derived. The computed value of $M_{ni}(x)$, viz $\bar{D}_{ni}$, is positive and can be shown to have a maximum relative error of $1.337(4n-1)2^{-t}$. (The slightly improved bound with the term $4n-1$ replaced by $3n-1$ can be obtained if further advantage is taken of the zeros in the D_{rj} array in this special case). Note that this bound is even better for $n \geqslant 3$ than (3.9.11) for the stable method. However, this bound applies only for $x_{i-1} \leqslant x < x_i$, whereas the bound for the stable method applies for $x_{i-n} < x < x_i$.

For B-splines of low degree with relatively uniform knot spacing the conventional method is adequate for certain purposes in that the absolute errors in the computed values are usually small (as in Examples 3.11.1 and 3.11.4). However, in a common situation to be examined in Section 3.12 it is seen that the stable method is faster in that fewer arithmetic operations are required and hence should be preferred on grounds of both accuracy and speed.

3.12 The evaluation, for a prescribed argument, of all non-zero B-splines of order n

In many applications, including interpolation (Chapter 6), least-squares approximation (Chapter 7) and constrained spline fitting (Chapter 8), it is necessary to evaluate for any prescribed argument x not an isolated value of $M_{ni}(x)$ (or $N_{ni}(x)$) but all those values of $M_{ni}(x)$ that are non-zero. Since at most n values of $M_{ni}(x)$ are non-zero for any particular x, it follows that n applications of either Algorithm 3.3.2 or Algorithm 3.4.2, for example, enable the required values to be computed. However, such an approach entails considerable repetition of computation since common elements in overlapping trapezoidal arrays (of the type illustrated in Fig. 3.3.2), in the case of Algorithm 3.3.2, or those in overlapping rhomboidal arrays (of the type illustrated in Fig. 3.4.2), in the case of Algorithm 3.4.2, are formed. It is easily verified that with such approaches the amount of arithmetic is proportional to n^3. It is far less expensive, taking an amount of arithmetic proportional to n^2, to compute a <u>single</u> array of elements which contains <u>all</u> the results required. Specifically, let l be the unique integer such that $x_{l-1} \leqslant x < x_l$. Then the non-zero B-splines of order n are $M_{nl}(x)$, $M_{n,l+1}(x)$, ..., $M_{n,l+n-1}(x)$. These n values can be computed using the following algorithm, based on the method of convex combinations, which is written to use minimal storage requirements and also economizes somewhat further on the number of arithmetic operations.

<u>Algorithm 3.12.1</u>: The efficient evaluation of all non-zero B-splines for a given argument using convex combinations.

Comment: Find the interval containing x.

Step 1. Determine the unique integer l such that $x_{l-1} \leqslant x < x_l$.

Comment: Initial conditions are set in Steps 2-3.

Step 2. Set $e_1 = x-x_{l-1}$, $e_2 = x_l-x$ and $v_1 = (x_l-x_{l-1})^{-1}$.

Step 3. For $j = 2, 3, \ldots, n$ set $v_j = e_1 v_{j-1}/(x_{l-1+j}-x_{l-1})$.

Comment: B-splines are computed by recurrence in Steps 4-6.

Step 4. For $j = 1, 2, \ldots, n-1$ execute Steps 5-6.

Step 5. Set $e_3 = x-x_{l-1-j}$ and replace v_1 by $e_2 v_1/(x_l-x_{l-1-j})$.

Step 6. For $r = 2, 3, \ldots, n-j$ replace v_r by

$$\frac{e_3 v_{r-1} + (x_{l-1+r}-x)v_r}{x_{l-1+r} - x_{l-1-j}} .$$

Step 7. For $j = 1, 2, \ldots, n$ set $M_{n,l-1+j}(x) = v_j$.

An illustration of the scheme in the case $n = 4$ is given in Fig. 3.12.1.

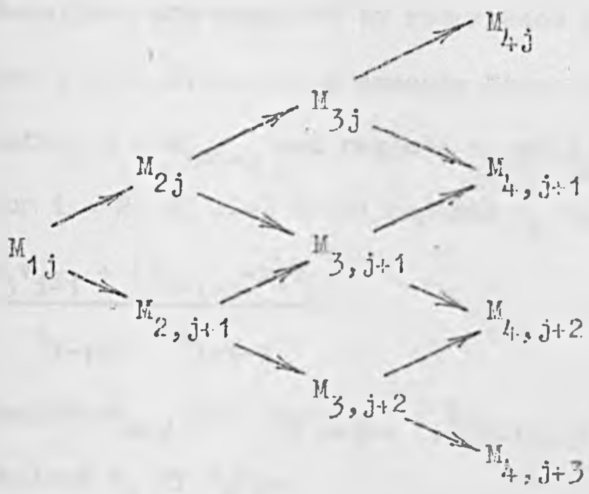

Fig. 3.12.1 Illustration of a scheme using convex combinations for the evaluation of all non-zero B-splines.

In Algorithm 3.12.1 the elements of the $M_{rj}(x)$ array are formed along successive downward-sloping diagonals. As with the algorithms for computing a single value of $M_{ni}(x)$ or $N_{ni}(x)$, minor variants may be constructed which form elements along successive upward-sloping diagonals or in successive columns.

Algorithm 3.12.2 below is the counterpart of Algorithm 3.12.1 in the case of <u>normalized</u> B-splines.

<u>Algorithm 3.12.2</u>: Efficient evaluation using convex combinations of all normalized B-splines that are non-zero for a given argument.

Comment: Find the interval containing x.

Step 1. Determine the unique integer l such that $x_{l-1} \leqslant x < x_l$.

Comment: The case n = 1 is treated separately.

Step 2. If n = 1 set $v_1 = 1$ and advance to Step 11.

Comment: Initial conditions are set in Steps 3-5.

Step 3. Set $e_1 = x-x_{l-1}$, $e_2 = x_l-x$ and $v_1 = 1/(x_l-x_{l-1})$.

Step 4. For j = 2, 3, ..., n-1 set $v_j = e_1 v_{j-1}/(x_{l-1+j}-x_{l-1})$.

Step 5. Set $v_n = e_1 v_{n-1}$.

Comment: B-splines are computed by recurrence in Steps 6-10.

Step 6. For $j = 1, 2, \ldots, n-2$ execute Steps 7-9.

Step 7. Set $e_3 = x - x_{l-1-j}$ and replace v_1 by $e_2 v_1 / (x_l - x_{l-1-j})$.

Step 8. For $i = 2, 3, \ldots, n-j-1$ replace v_i by

$$\frac{e_3 v_{i-1} + (x_{l-1+i} - x) v_i}{x_{l-1+i} - x_{l-1-j}}.$$

Step 9. Replace v_{n-j} by $e_3 v_{n-j-1} + (x_{l-1+n-j} - x) v_{n-j}$.

Step 10. Replace v_1 by $e_2 v_1$.

Step 11. For $j = 1, 2, \ldots, n$ set $N_{n,l-1+j}(x) = v_j$.

Note that with this scheme, as with Algorithm 3.4.2, if the values of the nth-order B-splines are required for a number of arguments x in the interval $x_{l-1} \leqslant x < x_l$, all denominators can be pre-computed, with a consequent saving in arithmetic. Such a strategy is particularly worthwhile in the context of data-fitting by splines (see Chapters 7 and 8) in which there are frequently many data points between an adjacent pair of knots.

A scheme based on divided differences for evaluating all non-zero B-splines can also be derived and this is illustrated for the case $n = 4$ in Fig. 3.12.2.

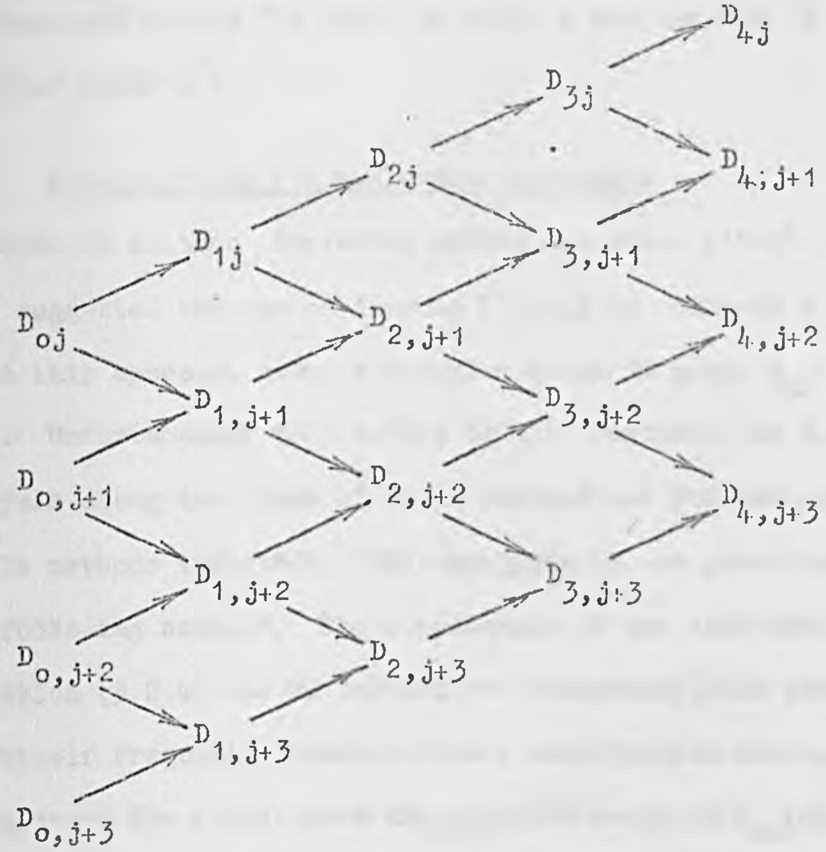

Fig. 3.12.2 Illustration of a scheme using divided differences for the evaluation of all non-zero B-splines

For the case n=4 the conventional method requires 32 additions, 8 multiplications and 16 divisions to compute the required values; the stable method requires 25 additions, 12 multiplications and 10 divisions (the number of additions for the stable method can be reduced to 19 by careful programming; it does not appear possible to achieve a corresponding reduction for the method of divided differences). Since on any computing machine multiplication is at least as fast as division it is evident that the stable method is faster for cubic B-splines. In fact this result applies in general; for a B-spline of degree n-1 ($n > 2$) the number of additions, multiplications and divisions are $2n^2$, $n(n-2)$ and n^2, respectively, if the conventional method is used, whereas for the stable method these numbers are $2n^2-2n+1$, $n(n-1)$ and $\frac{1}{2}n(n+1)$, respectively. For large n the stable method has a saving of $\frac{1}{2}n^2$ divisions over the conventional method (if terms of order n are ignored in comparison with those of order n^2).

3.13 Other methods for evaluating B-splines

A number of authors including LaFata and Rosen (1970) and Powell (1970) have suggested the use of formula (3.2.4) to evaluate $M_{ni}(x)$. At first sight this approach seems attractive since it gives $M_{ni}(x)$ in explicit form. Unfortunately this method is also unstable, as a running error analysis along the lines of those carried out for the conventional and stable methods indicates. This analysis is not given here, but we make the following comment. The computation of the individual terms of the summation (3.2.4) can be carried out accurately, but the evaluation of the sum itself frequently involves heavy cancellation between the individual terms, with the result that the computed value of $M_{ni}(x)$ may suffer from appreciable loss of accuracy. In our experience this loss of accuracy is comparable to that incurred using divided differences.

Segethova (1970, 1972) has considered a very different approach for the evaluation of B-splines. He expresses $M_{ni}(x)$ in terms of its n constituent polynomial arcs:

$$M_{ni}(x) = \begin{cases} p_{n,i-n+1}(x) & (x_{i-n} \leqslant x < x_{i-n+1}) \\ p_{n,i-n+2}(x) & (x_{i-n+1} \leqslant x < x_{i-n+2}) \\ \cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots & \\ p_{n,i-1}(x) & (x_{i-2} \leqslant x < x_{i-1}) \\ p_{ni}(x) & (x_{i-1} \leqslant x \leqslant x_i) \ . \end{cases} \tag{3.13.1}$$

Then, by representing each of these arcs as a Legendre series of degree $n-1$ in a normalized variable, ie

$$\left.\begin{aligned} p_{nj}(x) &= \sum_{u=0}^{n-1} a_{nju} P_u(X), \\ X &= (x-x_{j-1})/(x_j-x_{j-1}) \end{aligned}\right\} \quad (j = i-n+1, i-n+2, \ldots, i) \ , \tag{3.13.2}$$

where $P_u(X)$ denotes the Legendre polynomial of the first kind of degree u in X, he obtains recurrence relations from which the coefficients a_{nju} may be evaluated. Having computed these coefficients he uses the three-term recurrence relation for the Legendre polynomials to evaluate $P_u(X)$ ($u = 0, 1, \ldots, n-1$) and hence $M_{ni}(x)$, for any given value of x, having first, of course, determined an interval containing x.

Segethova's approach suffers a number of disadvantages compared with the method of convex combinations. Firstly, the method, as developed, applies only to the case of equally-spaced knots, although a generalization to the unequally-spaced case could presumably be made. Secondly, the determination of the Legendre coefficients proves to be somewhat unstable numerically. Segethova carries out computations in both single- and double-precision floating-point arithmetic, using the differences between the results to obtain estimates of the accuracy of his values. He finds on an IBM 7094 computer that for $n=21$ some of the coefficients have relative errors as large as 10^{-3}. Note that errors of this order will then inevitably be propagated to the computed value of $M_{ni}(x)$, even if the resulting Legendre series are evaluated exactly. Also note that Segethova's actual relative errors are as large as 10^{-3}, whereas our relative error bound is, using (3.9.11), $(1.337)(102)2^{-t}$. Thirdly, the arithmetic work to evaluate the set of non-zero $M_{ni}(x)$ values, for a prescribed value of x, even assuming the Legendre coefficients have been pre-computed, is about $4n^2$ additions and $5n^2$ multiplications, which also compares unfavourably with the method of convex combinations. Fourthly, a Fortran subroutine (Segethova, 1970) for evaluating the Legendre coefficients of the B-splines up to order 21 requires over 18,000 words of store!

An approach similar to that of Segethova's was developed at an early stage of this work, in which Chebyshev rather than Legendre polynomials were utilized. However, a corresponding loss of precision for large n was observed, although the resulting computational procedure proved to be more efficient in storage and speed.

CHAPTER 4

DIFFERENTIATION AND INTEGRATION OF B-SPLINES

In this chapter we examine methods for differentiating and integrating B-splines. In particular, we develop a number of recurrence relations for performing these operations. The results of this chapter are used in Chapter 5 to express the derivatives and integrals of arbitrary splines in their B-spline form, in Chapter 8 to impose convexity and concavity constraints in spline fitting problems, and in Chapter 9 where a general class of constraint conditions in spline approximation problems is examined.

In Section 4.1 we derive two recurrence relations for B-spline derivatives. In Section 4.2 the important case of derivatives at the range end points is examined and a proof is given that these derivatives can be obtained in a stable manner. In Section 4.3 it is established that the derivatives required in fitting with convexity and concavity constraints can also be evaluated stably. Algorithms for evaluating B-spline derivatives in the general case are discussed in Section 4.4. Finally, in Section 4.5, stable methods for evaluating indefinite integrals of B-splines are presented.

4.1 Recurrence relations for the derivatives of B-splines

We state and prove in this section two recurrence relations for the derivatives of B-splines; the first is due to de Boor (1972) and the second is believed to be new.

Theorem 4.1.1

The derivatives of B-splines satisfy the relation

$$N_{ni}^{(l)}(x) = (x_i - x_{i-n}) M_{ni}^{(l)}(x) = (n-1)\left\{ M_{n-1,i-1}^{(l-1)}(x) - M_{n-1,i}^{(l-1)}(x) \right\}$$

$$= (n-1)\left\{ \frac{N_{n-1,i-1}^{(l-1)}(x)}{x_{i-1} - x_{i-n}} - \frac{N_{n-1,i}^{(l-1)}(x)}{x_i - x_{i-n+1}} \right\} . \qquad (4.1.1)$$

Proof

Recalling that $M_{ni}(x) = M_n(x_{i-n}, x_{i-n+1}, \ldots, x_i; x)$ (Section 3.2) and using the properties of divided differences, we have

$$M'_{ni}(x) = \frac{d}{dx}\left\{M_n(x_{i-n}, x_{i-n+1}, \ldots, x_i; x)\right\} \tag{4.1.2}$$

$$= \frac{d}{dx}\left\{\frac{M_n(x_{i-n+1}, x_{i-n+2}, \ldots, x_i;x)-M_n(x_{i-n}, x_{i-n+1}, \ldots, x_{i-1};x)}{x_i - x_{i-n}}\right\} \tag{4.1.3}$$

$$= (n-1)\left\{-M_{n-1,i}(x) + M_{n-1,i-1}(x)\right\}/(x_i-x_{i-n}). \tag{4.1.4}$$

Differentiation of (4.1.4) l-1 times with respect to x, together with the use of (3.2.6), proves the theorem. □

Theorem 4.1.2

The derivatives of B-splines satisfy the relation

$$M_{ni}^{(l)}(x) = \left(\frac{n-1}{n-l-1}\right)\left\{\frac{(x-x_{i-n})M_{n-1,i-1}^{(l)}(x)+(x_i-x)M_{n-1,i}^{(l)}(x)}{x_i - x_{i-n}}\right\} \tag{4.1.5}$$

$$(l = 0, 1, \ldots, n-2).$$

Proof

Assume the theorem to be true for $l = s \geqslant 0$. Differentiation of (4.1.5) after replacing l by s then gives

$$\left(\frac{n-s-1}{n-1}\right)M_{ni}^{(s+1)}(x) = \frac{(x-x_{i-n})M_{n-1,i-1}^{(s+1)}(x)+(x_i-x)M_{n-1,i}^{(s+1)}(x)}{x_i - x_{i-n}}$$

$$+ \frac{M_{n-1,i-1}^{(s)}(x) - M_{n-1,i}^{(s)}(x)}{x_i - x_{i-n}}. \tag{4.1.6}$$

But from (4.1.1),

$$\frac{M^{(s)}_{n-1,i-1}(x) - M^{(s)}_{n-1,i}(x)}{x_i - x_{i-n}} = \frac{M^{(s+1)}_{ni}(x)}{n-1} \quad . \tag{4.1.7}$$

Thus

$$M^{(s+1)}_{ni}(x) = \left(\frac{n-1}{n-s-2}\right) \frac{(x-x_{i-n})M^{(s+1)}_{n-1,i-1}(x)+(x_i-x)M^{(s+1)}_{n-1,i}(x)}{x_i - x_{i-n}} \quad . \tag{4.1.8}$$

So the theorem is true for $l = s+1$. But the theorem is true for $l = 0$ by virtue of (3.4.1). Hence, by induction, it is true for $l = 0, 1, \ldots, n-2$. $\square$

Define the <u>reduced derivative</u> $m^{(l)}_{ni}(x)$ by

$$m^{(l)}_{ni}(x) = \frac{(n-l-1)!}{(n-1)!} M^{(l)}_{ni}(x). \tag{4.1.9}$$

Note that

$$m^{(o)}_{ni}(x) = M_{ni}(x) = m_{ni}(x), \tag{4.1.10}$$

say. In terms of reduced derivatives, (4.1.1) becomes

$$m^{(l)}_{ni}(x) = \frac{m^{(l-1)}_{n-1,i-1}(x) - m^{(l-1)}_{n-1,i}(x)}{x_i - x_{i-n}} \tag{4.1.11}$$

and (4.1.5) becomes

$$m^{(l)}_{ni}(x) = \frac{(x-x_{i-n})m^{(l)}_{n-1,i-1}(x)+(x_i-x)m^{(l)}_{n-1,i}(x)}{x_i - x_{i-n}} \quad . \tag{4.1.12}$$

It is usually more convenient to work in terms of the reduced derivatives $m^{(l)}_{ni}(x)$ rather than the derivatives $M^{(l)}_{ni}(x)$, because of the above simplifications in the basic recurrence relations. If the values of the $M^{(l)}_{ni}(x)$ or the $N^{(l)}_{ni}(x)$ are required, it is of course a trivial matter to obtain the former by multiplying $m^{(l)}_{ni}(x)$ by the factor $(n-1)!/(n-l-1)!$ and then by x_i-x_{i-n} to obtain the latter. These multiplications introduce negligible relative errors of the order of at most a small multiple of the relative machine precision.

Note that relation (4.1.12), like (3.4.1), involves the computation of convex combinations, a process that is numerically stable (cf Sections 3.9 and 5.3). Unlike the $M_{ni}(x)$, however, the $M_{ni}^{(l)}(x)$ for $l > 0$ may be positive, negative or zero. We expect therefore that a priori relative error bounds are not obtainable (except in special cases - see Sections 4.2 and 4.3) for either of these relations. However, Butterfield (1975) has recently suggested a class of algorithms for B-spline derivatives based upon both (4.1.11) and (4.1.12) and has presented some convincing arguments which suggest that a certain member of this class is the best possible choice (see Section 4.4).

Observe that (4.1.12) is in fact an interesting generalization of the fundamental relation (3.4.1) for B-splines.

4.2 The derivatives of the B-splines at the ends of the range

In many dealings with splines it is necessary to treat derivative boundary conditions (cf Chapter 9). If B-splines are used as the basis, their derivatives at the ends of the range need to be evaluated. We derive in this section some very satisfactory results relating to the numerical evaluation of these derivatives in the case where the B-splines are defined upon a standard knot set with coincident end knots.

Theorem 4.2.1

At the range endpoints $x = a$ and $x = b$ the rth derivatives $(0 \leq r < n)$ of the B-splines possess the properties

$$\text{sign}\left\{m_{ni}^{(r)}(a)\right\} = \begin{cases}(-1)^{r-i+1} & (i \leq r+1)\\ 0 & (i > r+1)\end{cases} \tag{4.2.1}$$

and

$$\text{sign}\left\{m_{ni}^{(r)}(b)\right\} = \begin{cases}(-1)^{N+n-1-i} & (i \geqslant N+n-1-r)\\ 0 & (i < N+n-1-r)\ ,\end{cases} \tag{4.2.2}$$

where

$$\text{sign}\ (u) = \begin{cases}-1 & (u < 0)\\ \ \ 0 & (u = 0)\\ +1 & (u > 0)\ .\end{cases} \tag{4.2.3}$$

Proof

We give the proof for the case $x = a$; that for $x = b$ is similar. From (4.1.11),

$$\text{sign}\left\{m_{ni}^{(l)}(a)\right\} = \text{sign}\left\{m_{n-1,i-1}^{(l-1)}(a) - m_{n-1,i}^{(l-1)}(a)\right\}\ . \tag{4.2.4}$$

Now suppose the theorem to be true for $r = l-1 \geqslant 0$. The use of (4.2.1) then enables (4.2.4) to be reduced to

$$\text{sign}\left\{m_{ni}^{(l)}(a)\right\} = \begin{cases}(-1)^{l-i+1} & (i \leq l+1)\\ 0 & (i > l+1)\ .\end{cases} \tag{4.2.5}$$

Thus the theorem is true for $r = l$. But from (3.5.1) it is true for $r = 0$. Hence, by induction it is true for $0 \leq r < n$. □

Theorem 4.2.2

Even in the presence of rounding errors the values of $m_{ni}^{(r)}(x)$ at the range endpoints $x = a$ and $x = b$ computed from (4.1.11) satisfy the relations (4.2.1) and (4.2.2).

Proof

Only the proof for $x = a$ is given since that for $x = b$ is similar.

Let $\bar{m}_{ni}^{(r)}(a)$ denote the value of $m_{ni}^{(r)}(a)$ computed from (4.1.11). A straightforward floating-point error analysis of this relation gives

$$\bar{m}_{ni}^{(r)}(x) = fl\left(\frac{\bar{m}_{n-1,i-1}^{(r-1)}(x) - \bar{m}_{n-1,i}^{(r-1)}(x)}{x_i - x_{i-n}}\right) \tag{4.2.6}$$

$$= \left(\frac{\bar{m}_{n-1,i-1}^{(r-1)}(x) - \bar{m}_{n-1,i}^{(r-1)}(x)}{x_i - x_{i-n}}\right)(1+\varepsilon)^3 \tag{4.2.7}$$

$$= \left(\frac{\bar{m}_{n-1,i-1}^{(r-1)}(x) - \bar{m}_{n-1,i}^{(r-1)}(x)}{x_i - x_{i-n}}\right)(1+3e) \quad , \tag{4.2.8}$$

where

$$|\varepsilon| \leq 2^{-t} \; , \quad |e| < 2^{-t_1} \quad . \tag{4.2.9}$$

After setting $x = x_{i-n} = a$ in (4.2.8), the remainder of the proof then follows closely that of Theorem 4.2.1 with (4.2.8) used in place of (4.1.11). The only difference is the factor $1+3e$ in (4.2.8) which, since $3\,|e| < 1$, has no influence on the sign of the term it multiplies. Thus the theorem is proved. □

Theorem 4.2.3

Let $\bar{m}_{ni}^{(r)}(a)$ be the value of $m_{ni}^{(r)}(a)$ computed from (4.1.11) for $r > 0$ and from (3.5.1) for $r = 0$. Let

$$\delta m_{ni}^{(r)}(a) = \bar{m}_{ni}^{(r)}(a) - m_{ni}^{(r)}(a) \; . \tag{4.2.10}$$

Then $\delta m_{ni}^{(r)}(a)$ satisfies the a posteriori absolute error bound

$$\left|\delta m_{ni}^{(r)}(a)\right| \leq (1-2^{-t})^{-2-3r}(3r+2)2^{-t_1}\left|\bar{m}_{ni}^{(r)}(a)\right| \tag{4.2.11}$$

$$\leq 1.179\,(3r+2)2^{-t}\left|\bar{m}_{ni}^{(r)}(a)\right| \tag{4.2.12}$$

and the a priori relative error bound

$$\left|\delta m_{ni}^{(r)}(a)\right| \leqslant 1.337(3r+2)2^{-t}\left|m_{ni}^{(r)}(a)\right| . \tag{4.2.13}$$

The results (4.2.11), (4.2.12) and (4.2.13) also hold with a replaced by b.

Proof

Only the proof for $x = a$ is given since that for $x = b$ is similar.

Firstly we note that (4.2.12) follows immediately from (4.2.11) upon using (1.1.9) and (1.1.12). Also (4.2.13) follows from (4.2.12), since the latter gives

$$\left|\delta m_{ni}^{(r)}(a)\right| \leqslant 1.179(3r+2)2^{-t}\left\{\left|\delta m_{ni}^{(r)}(a)\right| + \left|m_{ni}^{(r)}(a)\right|\right\} \tag{4.2.14}$$

and hence

$$\left|\delta m_{ni}^{(r)}(a)\right| \leqslant \frac{1.179(3r+2)2^{-t}\left|m_{ni}^{(r)}(a)\right|}{1 - 1.179(3r+2)2^{-t}} . \tag{4.2.15}$$

But, using (1.1.7), the denominator in (4.2.15) is bounded from below by $1 - 0.1179 = 0.8821$; relation (4.2.13) then follows. It remains to prove (4.2.11).

We consider initially the case $r = 0$. From (3.5.3) and (4.1.10),

$$m_{ni}(a) = \begin{cases} 1/(x_i-a) & (i = 1) \\ 0 & (i > 1) \end{cases} \tag{4.2.16}$$

and so $\bar{m}_{ni}(a) = m_{ni}(a) = 0$ if $i > 1$, which proves the theorem in the trivial case $r = 0$ and $i > 1$. For $r = 0$ and $i = 1$, (4.2.16) gives

$$\begin{aligned} \bar{m}_{n1}(a) &= fl\left\{1/(x_1-a)\right\} = (1+\varepsilon)^2/(x_1-a) \\ &= (1+\varepsilon)^2 m_{n1}(a) = (1+2e)m_{n1}(a) , \end{aligned} \tag{4.2.17}$$

where $|\varepsilon| \leqslant 2^{-t_1}$ and $|e| < 2^{-t_1}$. Thus

$$\delta m_{n1}(a) = \bar{m}_{n1}(a) - m_{n1}(a) = 2em_{n1}(a). \tag{4.2.18}$$

So

$$\left|\delta m_{n1}(a)\right| \leqslant (2)2^{-t_1} m_{n1}(a) . \tag{4.2.19}$$

But from (4.2.17),

$$m_{n1}(a) \leqslant (1-2^{-t})^{-2} \bar{m}_{n1}(a) . \tag{4.2.20}$$

Thus

$$\left|\delta m_{n1}(a)\right| \leqslant 2(1-2^{-t})^{-2} 2^{-t_1} \bar{m}_{n1}(a) , \tag{4.2.21}$$

which proves the theorem for the case $r = 0$ and $i = 1$.

We now assume the theorem to be true for $r = l-1$ $(l \geqslant 1)$ and show by induction that this implies it is true for $r = l$. From (4.2.8) and (4.2.10)

$$m_{ni}^{(l)}(a) + \delta m_{ni}^{(l)}(a) = \left(\frac{1}{x_i - a}\right) \left\{ m_{n-1,i-1}^{(l-1)}(a) - m_{n-1,i}^{(l-1)}(a) + \delta m_{n-1,i-1}^{(l-1)}(a) - \delta m_{n-1,i}^{(l-1)}(a) \right\} (1+3e). \tag{4.2.22}$$

The use of (4.1.11) reduces (4.2.22) to

$$\delta m_{ni}^{(l)}(a) = \left(\frac{1}{x_i - a}\right) \left[\delta m_{n-1,i-1}^{(l-1)}(a) - \delta m_{n-1,i}^{(l-1)}(a) + 3e \left\{ \bar{m}_{n-1,i-1}^{(l-1)}(a) - \bar{m}_{n-1,i}^{(l-1)}(a) \right\} \right]. \tag{4.2.23}$$

Then, using (4.2.11) with $r = l-1$, (4.2.23) yields

$$\left|\delta m_{ni}^{(l)}(a)\right| \leqslant \left(\frac{1}{x_i - a}\right) \left\{ (1-2^{-t})^{1-3l}(3l-1)+3 \right\} 2^{-t_1} \left\{ \left|\bar{m}_{n-1,i-1}^{(l-1)}(a)\right| + \left|\bar{m}_{n-1,i}^{(l-1)}(a)\right| \right\} . \tag{4.2.24}$$

As a consequence of Theorem 4.2.2,

$$\left|\bar{m}^{(l-1)}_{n-1,i-1}(a)\right| + \left|\bar{m}^{(l-1)}_{n-1,i}(a)\right| = \left|\bar{m}^{(l-1)}_{n-1,i-1}(a) - \bar{m}^{(l-1)}_{n-1,i}(a)\right| . \qquad (4.2.25)$$

Moreover, from (4.2.7),

$$\left|\bar{m}^{(l-1)}_{n-1,i-1}(a) - \bar{m}^{(l-1)}_{n-1,i}(a)\right| \leqslant (1-2^{-t})^{-3}(x_i-a)\left|\bar{m}^{(l)}_{ni}(a)\right| . \qquad (4.2.26)$$

Relations (4.2.24), (4.2.25) and (4.2.26) together yield

$$\left|\delta\, m^{(l)}_{ni}(a)\right| \leqslant \left\{(1-2^{-t})^{1-3l}(3l-1)+3\right\}(1-2^{-t})^{-3}2^{-t_1}\left|\bar{m}^{(l)}_{ni}(a)\right| \qquad (4.2.27)$$

$$\leqslant (1-2^{-t})^{-2-3l}(3l+2)2^{-t_1}\left|\bar{m}^{(l)}_{ni}(a)\right| . \qquad (4.2.28)$$

Thus the theorem is true for $r = 1$. But we have already proved it true for $r = 0$. Hence by induction it is true for $r \geqslant 0$. □

The numerical evaluation at the range endpoints of the derivatives of the B-splines defined upon a standard knot set with coincident end knots is thus unconditionally stable.

4.3 The derivatives of B-splines at the knots

In this section we prove firstly (Theorem 4.3.1) an interesting result relating to the signs of certain B-spline derivatives at the knots and then show (Theorem 4.3.2) that this result still holds in floating-point computation. Finally we prove (Theorem 4.3.3) that when these derivatives are evaluated using relation (4.1.11) the computed values satisfy excellent *a posteriori* absolute and *a priori* relative error bounds. These results are of particular relevance to the algorithm derived in Chapter 8 for spline fitting with convexity and concavity constraints.

Theorem 4.3.1

For $n \geqslant 2$ and all i the value of $m^{(n-2)}_{ni}(x)$ at the interior knot x_j

$(j = i-n+1,\ i-n+2,\ \dots\ i-1)$ is strictly positive or negative according to whether $i+j+n$ is respectively odd or even.

Proof

Suppose the theorem is true for $n = r-1 \geqslant 2$. It then follows immediately from relation (4.1.11) that the theorem is true for $n = r$. But the theorem is evidently true for $n = 2$, since $m_{2i}^{(0)}(x_{i-1}) = M_{2i}(x_{i-1}) > 0$ by virtue of Theorem 3.4.2. Hence by induction the theorem is true for all $n \geqslant 2$. $\square$

As a consequence of Theorem 4.3.1, $m_{ni}^{(n-2)}(x)$ is a first degree spline (ie a piecewise-linear function) with values at the interior knots which alternate in sign.

Theorem 4.3.2

Even in the presence of rounding errors, for all $n \geqslant 2$ and all i, the value of $m_{ni}^{(n-2)}(x)$ at the interior knot x_j $(j = i-n+1,\ i-n+2,\ \dots,\ i-1)$, when computed in floating-point arithmetic from (4.1.11) if $n > 2$ or from (3.2.8) and (3.4.1) if $n = 2$, is strictly positive or negative according to whether $i+j+n$ is respectively odd or even.

Proof

We merely have to show that $m_{ni}^{(n-2)}(x_j)$ and the computed value $\bar{m}_{ni}^{(n-2)}(x_j)$ have the same sign. The result will then follow immediately from Theorem 4.3.1.

Suppose the theorem is true for $n = r-1$, ie that for all i and for $j = i-r+2,\ i-r+3,\ \dots,\ i-1$, $\bar{m}_{r-1,i}^{(r-3)}(x_j)$ and $m_{r-1,i}^{(r-3)}(x_j)$ have the same sign. Then, using (4.2.8) (which holds independently of the assumption in Section 4.2 that the end knots are coincident) it follows immediately that for $j = i-r+1,\ i-r+2,\ \dots,\ i-1$, $\bar{m}_{ri}^{(r-2)}(x_j)$ and $m_{ri}^{(r-2)}(x_j)$ have the same sign, since the factor $1+3e > 0$. Hence the theorem is true for $n = r$.

But from Theorem 3.8.2, $\bar{m}_{2i}(x_j)$ and $m_{2i}(x_j)$ certainly have the same sign. Hence the theorem is true for $n = 2$ and therefore, by induction, for all $n \geq 2$. $\square$

Theorem 4.3.3

Let i be any integer, r be any integer ≥ 2, and j take any one of the values $i-r+1, i-r+2, \ldots, i-1$. Let $\bar{m}_{ri}^{(r-2)}(x_j)$ denote the value of $m_{ri}^{(r-2)}(x_j)$ computed from (4.1.11) if $r > 2$ or from

$$m_{2i}^{(0)}(x_{i-1}) = m_{2i}(x_{i-1}) = (x_i - x_{i-2})^{-1} \tag{4.3.1}$$

if $r = 2$. Let

$$\delta m_{ri}^{(r-2)}(x_j) = \bar{m}_{ri}^{(r-2)}(x_j) - m_{ri}^{(r-2)}(x_j). \tag{4.3.2}$$

Then $\delta m_{ri}^{(r-2)}(x_j)$ satisfies the a posteriori absolute error bound

$$\left|\delta m_{ri}^{(r-2)}(x_j)\right| \leq (1-2^{-t})^{4-3r}(3r-4)2^{-t_1}\left|\bar{m}_{ri}^{(r-2)}(x_j)\right| \tag{4.3.3}$$

$$\leq 1.179\ (3r-4)2^{-t}\left|\bar{m}_{ri}^{(r-2)}(x_j)\right| \tag{4.3.4}$$

and the a priori relative error bound

$$\left|\delta m_{ri}^{(r-2)}(x_j)/m_{ri}^{(r-2)}(x_j)\right| \leq 1.337\ (3r-4)2^{-t}\ . \tag{4.3.5}$$

Proof

Now (4.3.4) and (4.3.5) follow from (4.3.3) in essentially the same way that (4.2.12) and (4.2.13) follow from (4.2.11). Hence we only prove (4.3.3). The initial stages of the proof are very similar to those of Theorem 4.2.3. From (4.2.8), (4.3.2) and (4.1.11) we obtain

$$\left|\delta\, m_{ri}^{(r-2)}(x_j)\right| \leqslant \left(\frac{1}{x_i - x_{i-r}}\right)\left[\left|\delta\, m_{r-1,i-1}^{(r-3)}(x_j)\right| + \left|\delta\, m_{r-1,i}^{(r-3)}(x_j)\right| + (3)2^{-t_1}\left|\bar{m}_{r-1,i-1}^{(r-3)}(x_j) - \bar{m}_{r-1,i}^{(r-3)}(x_j)\right|\right]. \tag{4.3.6}$$

Now assume the theorem to be true for $r = s-1 \geqslant 2$, ie that

$$\left|\delta\, m_{s-1,i}^{(s-3)}(x_j)\right| \leqslant (1-2^{-t})^{7-3s}(3s-7)2^{-t_1}\left|\bar{m}_{s-1,i}^{(s-3)}(x_j)\right|. \tag{4.3.7}$$

Putting $r = s$ in (4.3.6) and using (4.3.7) then gives

$$\left|\delta\, m_{si}^{(s-2)}(x_j)\right| \leqslant \left(\frac{1}{x_i - x_{i-s}}\right)\left[(1-2^{-t})^{7-3s}(3s-7)2^{-t_1}\left\{\left|\bar{m}_{s-1,i-1}^{(s-3)}(x_j)\right| + \left|\bar{m}_{s-1,i}^{(s-3)}(x_j)\right|\right\} + (3)2^{-t_1}\left|\bar{m}_{s-1,i-1}^{(s-3)}(x_j) - \bar{m}_{s-1,i}^{(s-3)}(x_j)\right|\right] \tag{4.3.8}$$

But it follows from Theorem 4.3.2 and from (4.2.8) that

$$\left|\bar{m}_{s-1,i-1}^{(s-3)}(x_j)\right| + \left|\bar{m}_{s-1,i}^{(s-3)}(x_j)\right| = \left|\bar{m}_{s-1,i-1}^{(s-3)}(x_j) - \bar{m}_{s-1,i}^{(s-3)}(x_j)\right| \tag{4.3.9}$$

$$\leqslant (1-2^{-t})^{-3}(x_i - x_{i-s})\left|\bar{m}_{si}^{(s-2)}(x_j)\right|. \tag{4.3.10}$$

The use of (4.3.9) and (4.3.10) reduces (4.3.8) to

$$\left|\delta\, m_{si}^{(s-2)}(x_j)\right| \leqslant \left\{(1-2^{-t})^{7-3s}(3s-7)+3\right\}(1-2^{-t})^{-3}2^{-t_1}\left|\bar{m}_{si}^{(s-2)}(x_j)\right| \tag{4.3.11}$$

$$\leqslant (1-2^{-t})^{4-3s}(3s-4)2^{-t_1}\left|\bar{m}_{si}^{(s-2)}(x_j)\right|. \tag{4.3.12}$$

So the theorem is true for $r = s$. But it is very easily verified that the theorem is true for $r = 2$. Hence, by induction, it is true for all $r \geqslant 2$. □

We conclude from Theorem 4.3.3 that the computation of $m_{ri}^{(r-2)}(x_j)$ from the recurrence relation (4.1.11) is unconditionally stable.

4.4 Algorithms for the evaluation of B-spline derivatives

Two recurrence relations for the derivatives of B-splines have been established. One, (4.1.11), relates the lth derivative of a B-spline of order n to the $(l-1)$ st derivatives of B-splines of order $n-1$. The other, (4.1.12), relates the lth derivative of a B-spline of order n to derivatives of the same order of B-splines of order $n-1$.

These two relations, when used in conjunction with the fundamental recurrence (3.4.1) suggest (at least) two computational schemes for the numerical evaluation of $m_{ni}^{(l)}(x)$, for any prescribed value of x.

One such scheme (Scheme A) involves initially the use of (3.2.8) and (3.4.1), as in Algorithm 3.12.1 for example, to compute for all relevant i the values of $m_{n-l,i}(x)$. Then (4.1.11) is employed to compute successively the values of $m'_{n-l+1,i}(x)$, $m''_{n-l+2,i}(x)$, ..., $m_{ni}^{(l)}(x)$, for all appropriate values of i.

A second scheme (Scheme B) involves initially the use of (3.2.8) and (3.4.1) to compute the values of $m_{2i}(x)$, followed by the use of (4.1.11) to compute successively the values of $m'_{3i}(x)$, $m''_{4i}(x)$, ..., $m_{l+2,i}^{(l)}(x)$ for all appropriate i. This is followed by the application of (4.1.12) to compute successively the values of $m_{l+3,i}^{(l)}(x)$, $m_{l+4,i}^{(l)}(x)$, ..., $m_{ni}^{(l)}(x)$, again for all relevant values of i.

Butterfield (1975) has carried out a detailed analysis of a set of schemes, which includes Schemes A and B as special cases, for computing B-spline derivatives. A tentative result of his work is that Scheme A can be expected to have superior stability properties to all the other schemes in the set. Numerical evidence is accumulating to support this result. Algorithm 4.4.1 below implements Scheme A.

Algorithm 4.4.1: The evaluation of the lth reduced derivative of all non-zero B-splines for a given argument x $(x_{k-1} \leqslant x < x_k)$ using Scheme A.

Comment: The B-splines of order n-l are computed by convex combinations in Step 1.

Step 1. Employ Algorithm 3.12.1 to obtain the values of

$v_j = M_{n-l,k-1+j}(x)$ $(j = 1, 2, \ldots, n-l)$.

Comment: The required derivatives are computed by recurrence in Steps 2-5.

Step 2. For $r = n-l+1, n-l+2, \ldots, n$ execute Steps 3-5.

Step 3. Set $v_r = v_{r-1}/(x_{k+r-1}-x_{k-1})$.

Step 4. For $i = k+r-2, k+r-3, \ldots, k+1$ replace v_{i-k+1} by $(v_{i-k}-v_{i-k+1})/(x_i-x_{i-r})$.

Step 5. Replace v_1 by $-v_1/(x_k-x_{j-r})$.

Step 6. For $j = 1, 2, \ldots, n$ set $m^{(l)}_{n,k-1+j}(x) = v_j$.

4.5 The definite and indefinite integrals of B-splines

In this section some results relating to the integration of B-splines are established.

Theorem 4.5.1

The indefinite integral of a B-spline is given by

$$\int_{-\infty}^{x} M_{ni}(t)dt = \begin{cases} 0 & (x \leqslant x_{i-n}) \\ \dfrac{1}{n} \displaystyle\sum_{j=i+1}^{i+n} N_{n+1,j}(x) & (x_{i-n} \leqslant x \leqslant x_i) \\ \dfrac{1}{n} & (x_i \leqslant x) \end{cases} \tag{4.5.1}$$

and by

$$\int_{-\infty}^{x} M_{ni}(t)dt = \frac{1}{n} \sum_{j=i+1}^{k+n} N_{n+1,j}(x) \quad (x_{k-1} \leqslant x < x_k;\ i-n < k \leqslant i). \tag{4.5.2}$$

Proof

Use of the relationship (4.1.1) yields

$$M_{nj}(x) = M_{n,j-1}(x) - \frac{1}{n} N'_{n+1,j}(x). \tag{4.5.3}$$

Summing (4.5.3) over all values of j from i+1 to i+n gives

$$M_{n,i+n}(x) = M_{ni}(x) - \frac{1}{n} \sum_{j=i+1}^{i+n} N'_{n+1,j}(x) . \tag{4.5.4}$$

We concern ourselves with the interval $x_{i-n} \leqslant x \leqslant x_i$, since $M_{ni}(x) \equiv 0$ outside this interval. Thus, replacing x by t in (4.5.4), integrating with respect to t between the limits $-\infty$ and x and observing that $M_{n,i+n}(x) \equiv 0$ for $x < x_i$, we obtain

$$\int_{-\infty}^{x} M_{ni}(t)dt = \frac{1}{n} \sum_{j=i+1}^{i+n} \Big[N_{n+1,j}(t) \Big]_{-\infty}^{x} \tag{4.5.5}$$

$$= \frac{1}{n} \sum_{j=i+1}^{i+n} N_{n+1,j}(x) . \tag{4.5.6}$$

Putting $x = x_i$ in (4.5.6) and using the compact support of the B-splines gives

$$\int_{-\infty}^{\infty} M_{ni}(t)dt = \frac{1}{n} \sum_{j=i+1}^{i+n} N_{n+1,j}(x_i) . \tag{4.5.7}$$

Since $N_{n+1,j}(x_i) = 0$ for $j \leqslant i$ and $j > i+n$, the right-hand side of (4.5.7) is equal to $\frac{1}{n} \sum_j N_{n+1,j}(x_i)$, which by virtue of (3.6.1) then yields

$$\int_{-\infty}^{\infty} M_{ni}(t)dt = \frac{1}{n} \tag{4.5.8}$$

(cf Section 3.2). The results (4.5.6) and (4.5.8), together with the fact that $M_{ni}(x) \equiv 0$ for $x < x_{i-n}$ and $x > x_i$, prove (4.5.1).

Finally, if $x_{k-1} \leqslant x < x_k$, where $i-n < k \leqslant i$, then $N_{n+1,j}(x) \equiv 0$ for $j > k+n$, from which (4.5.2) follows. $\square$

We note that the lower limit in the integrals in (4.5.1) and (4.5.2) may be replaced by any value not exceeding x_{i-n} without affecting the results.

Moreover, it follows from (4.5.1) and (3.2.6) that the definite integral of $N_{ni}(x)$ is given by

$$\int_{-\infty}^{\infty} N_{ni}(x)dx = (x_i - x_{i-n})/n \ . \tag{4.5.9}$$

Finally, if the knots are uniformly spaced one unit apart, (4.5.9) reduces to

$$\int_{-\infty}^{\infty} N_{ni}(x)dx = 1. \tag{4.5.10}$$

If the values of $N_{n+1,j}(x)$ in (4.5.1) or (4.5.2) are computed using the unconditionally stable Algorithm 3.12.2, the value of $\int_{x_{i-n}}^{x} M_{ni}(t)dt$ has a very small relative error. Alternatively, one of the schemes discussed in Chapter 5 for evaluating a linear combination of B-splines can be used; the results of that chapter can be used to establish that the value of the integral computed in this way also has a very small relative error.

The following theorem shows that the integral of $M_{ni}(t)$ can also be computed from a reduction formula.

<u>Theorem 4.5.2</u>

If $x_{i-n} \leqslant x \leqslant x_i$ then

$$\int_{-\infty}^{x} M_{ni}(t)dt = \frac{1}{n}(x-x_{i-n})M_{ni}(x) + \frac{n-1}{n}\int_{-\infty}^{x} M_{n-1,i}(t)dt \ . \tag{4.5.11}$$

<u>Proof</u>

From (4.5.1), (3.4.1) and (3.2.6),

$$\int_{-\infty}^{x} M_{ni}(t)dt = \frac{1}{n}\sum_{j=i+1}^{i+n}\left\{(x-x_{j-n-1})M_{n,j-1}(x)+(x_j-x)M_{nj}(x)\right\}$$

$$= \frac{1}{n}\left\{\sum_{j=i}^{i+n-1}(x-x_{j-n})M_{nj}(x)+\sum_{j=i+1}^{i+n}(x_j-x)M_{nj}(x)\right\}$$

$$= \frac{1}{n}\left\{(x-x_{i-n})M_{ni}(x)+\sum_{j=i+1}^{i+n-1}(x_j-x_{j-n})M_{nj}(x)\right.$$

$$\left. + (x_{i+n}-x)M_{n,i+n}(x)\right\}$$

$$= \frac{1}{n}\left\{(x-x_{i-n})M_{ni}(x)+\sum_{j=i+1}^{i+n-1}N_{nj}(x)\right\}, \qquad (4.5.12)$$

since $M_{n,i+n}(x) \equiv 0$ for $x_{i-n} \leqslant x \leqslant x_i$. But, by replacing n by n-1 in (4.5.1),

$$\sum_{j=i+1}^{i+n-1} N_{nj}(x) = (n-1)\int_{-\infty}^{x} M_{n-1,i}(t)dt \quad (x_{i-n+1} \leqslant x \leqslant x_i). \qquad (4.5.13)$$

Now (4.5.13) also holds trivially for $x_{i-n} \leqslant x \leqslant x_{i-n+1}$. Hence (4.5.13) holds for $x_{i-n} \leqslant x \leqslant x_i$. Substitution of (4.5.13) into (4.5.12) then yields (4.5.11). □

The following theorem establishes another useful form for $\int_{-\infty}^{x} M_{ni}(t)dt$.

Theorem 4.5.3

For $x_{i-n} \leqslant x \leqslant x_i$,

$$\int_{-\infty}^{x} M_{ni}(t)dt = \frac{1}{n}\sum_{r=1}^{n}(x-x_{i-r})M_{ri}(x). \qquad (4.5.14)$$

Proof

The repeated application of (4.5.11) yields

$$\int_{-\infty}^{x} M_{ni}(t)dt = \frac{1}{n}\left\{(x-x_{i-n})M_{ni}(x)+(x-x_{i-n+1})M_{n-1,i}(x)+(n-2)\int_{-\infty}^{x} M_{n-2,i}(t)dt\right\}$$

$$= \frac{1}{n}\left\{(x-x_{i-n})M_{ni}(x)+(x-x_{i-n+1})M_{n-1,i}(x)+(x-x_{i-n+2})M_{n-2,i}(x)\right.$$
$$\left. +(n-3)\int_{-\infty}^{x} M_{n-3,i}(t)dt\right\}$$

$$= \frac{1}{n}\left\{(x-x_{i-n})M_{ni}(x)+(x-x_{i-n+1})M_{n-1,i}(x)+ \dots +(x-x_{i-2})M_{2i}(x)\right.$$
$$\left. + \int_{-\infty}^{x} M_{1i}(t)dt\right\}$$

$$= \frac{1}{n}\sum_{r=2}^{n}(x-x_{i-r})M_{ri}(x)+\frac{1}{n}\int_{-\infty}^{x} M_{1i}(t)dt. \tag{4.5.15}$$

But it is readily established that

$$\int_{-\infty}^{x} M_{1i}(t)dt = \begin{cases} 0 & (x \leqslant x_{i-1}) \\ (x-x_{i-1})/(x_i-x_{i-1}) & (x_{i-1} \leqslant x \leqslant x_i) \end{cases}$$

$$= (x-x_{i-1})M_{1i}(x) \qquad (x \leqslant x_i)\ . \tag{4.5.16}$$

The result (4.5.14) then follows from (4.5.15) and (4.5.16). $\square$

The reduction formula (4.5.11) and the explicit form (4.5.14) have recently been discovered independently by Gaffney (1974). The value of $\int_{-\infty}^{x} M_{ni}(t)dt$ can be calculated particularly efficiently from (4.5.14) since a single application of Algorithm 3.12.1 yields as by-products all the required values of $M_{ri}(x)$ $(r = 1, 2, \dots, n)$.

We may also express (4.5.1), in the case $x_{i-r} \leqslant x \leqslant x_i$, as

$$\int^{x} M_{ni}(t)dt = \frac{1}{n}\sum_{j=i+1}^{i+n} N_{n+1,j}(x) + C , \qquad (4.5.17)$$

where C is a constant whose value depends on the lower limit of integration. The r-fold indefinite integral of $M_{ni}(t)$ can similarly be represented as a linear sum of B-splines of order n+r plus an additional term of the form

$$C_1x^{r-1}+C_2x^{r-2}+ \dots + C_r. \qquad (4.5.18)$$

We discuss in detail in Chapter 5 the representation of polynomials of degree < n in terms of B-splines of order n. Hence the complete expression for the r-fold indefinite integral of $M_{ni}(x)$ can be represented solely in terms of B-splines of order n+r.

Finally, we state and prove an interesting result due to Butterfield (1973). The result is in fact an even broader generalization than (4.1.5) of the fundamental recurrence relation (3.4.1).

Theorem 4.5.4

Let

$$M_{ni}^{(l)}(x) = \int_{-\infty}^{x} M_{ni}^{(l+1)}(t)dt \qquad (l < 0) , \qquad (4.5.19)$$

with

$$M_{ni}^{(0)}(x) = M_{ni}(x). \qquad (4.5.20)$$

Then the result (4.1.5) also holds for $l < 0$.

Proof

Suppose the theorem is true for $l = -1, -2, \dots, r$ $(r < 0)$. Then (4.1.5) gives

$$M_{ni}^{(r)}(x) = \left(\frac{n-1}{n-r-1}\right)\left\{\frac{(x-x_{i-n})M_{n-1,i-1}^{(r)}(x)+(x_i-x)M_{n-1,i}^{(r)}(x)}{x_i - x_{i-n}}\right\}, \qquad (4.5.21)$$

the integration of which (by parts) yields

$$M_{ni}^{(r-1)}(x) = \left(\frac{n-1}{n-r-1}\right) \left\{ (x-x_{i-n})M_{n-1,i-1}^{(r-1)}(x)+(x_i-x)M_{n-1,i}^{(r-1)}(x) \right.$$

$$\left. -M_{n-1,i-1}^{(r-2)}(x)+M_{n-1,i}^{(r-2)}(x) \right\} \Big/ (x_i-x_{i-n}). \qquad (4.5.22)$$

But since (4.1.1) evidently holds with $l < 0$ we have

$$(n-1) \frac{\left\{M_{n-1,i-1}^{(r-2)}(x)-M_{n-1,i}^{(r-2)}(x)\right\}}{x_i - x_{i-n}} = M_{ni}^{(r-1)}(x) . \qquad (4.5.23)$$

The insertion of (4.5.23) into (4.5.22) reduces the latter to

$$M_{ni}^{(r-1)}(x) = \left(\frac{n-1}{n-r}\right)\left\{\frac{(x-x_{i-n})M_{n-1,i-1}^{(r-1)}(x)+(x_i-x)M_{n-1,i}^{(r-1)}(x)}{x_i - x_{i-n}}\right\}. \qquad (4.5.24)$$

Thus the result is true for $l = r-1$. By using integration by parts it is easily proved true for $l = -1$. Hence, by induction, the theorem is true for all $l < 0$. $\square$

CHAPTER 5

THE B-SPLINE REPRESENTATION OF SPLINES AND POLYNOMIALS

In this chapter we consider the representation of splines and polynomials in terms of B-splines. There are at least three reasons why such representations are useful. Firstly, in problems of interpolation and data-fitting by splines, B-spline representations usually prove to be well-conditioned in that the coefficients in the representations are relatively insensitive to changes in the data. Secondly, as we show in Section 5.3, the numerical evaluation of the B-spline representation itself can be carried out in an unconditionally stable manner. Thirdly, it is convenient to be able to represent polynomials in terms of splines in order that repeated indefinite integrations of arbitrary splines can be accomplished readily, in order to impose fairly general forms of line constraint in least-squares bivariate spline approximation, and also to provide an "interface" between mathematical software employing polynomials with that employing splines.

In Section 5.1 we present a particularly useful result due to de Boor (1972) which expresses a linear combination of B-splines in terms of B-splines of lower order with certain polynomial coefficients. The result is then used to establish a new proof that the B-splines form a linearly independent set of basis functions in terms of which an arbitrary spline $s(x)$ can be expressed, and to establish local lower and upper bounds for $s(x)$. In Section 5.2 two schemes for the evaluation of $s(x)$ are presented and in Section 5.3 rigorous floating-point error analyses of these schemes are given. In Section 5.4 the effects of errors in the B-spline coefficients are examined. In Section 5.5 the problem of representing powers in terms of B-splines is addressed and in Section 5.6 algorithms for obtaining these representations are presented. The extension of the algorithms of

Section 5.6 to cover finite power series is treated in Section 5.7, where an interesting result relating the absolute convergence of a Taylor series representation to the boundedness of the coefficients in a related B-spline representation is established. Error analyses of the algorithms of Sections 5.6 and 5.7 are given Section 5.8. Sections 5.9 and 5.10 discuss methods for representing in their B-spline form the derivatives and indefinite integrals of $s(x)$.

Section 5.11 is exceptional in that it treats the conversion of the B-spline representation of splines (or, indirectly, of powers or polynomials) into their equivalent piecewise-Chebyshev-series representations. The latter representations require considerably more store than the B-spline form, but they have the advantage that they are quicker to evaluate.

5.1 The B-spline representation of splines

Given an n-extended partition $\{x_i\}$ and a set of coefficients $\{c_i\}$, let

$$s(x) = \sum_i c_i N_{ni}(x) , \tag{5.1.1}$$

where the $\{N_{ni}(x)\}$ are the B-splines of order n defined upon the knots $\{x_i\}$. Evidently, any linear combination of the form (5.1.1) defines a spline with knots $\{x_i\}$.

We now establish a result due to de Boor (1972) of which we make considerable use in this chapter. Using (3.4.2), (5.1.1) becomes

$$s(x) = \sum_i c_i \left\{ \left(\frac{x-x_{i-n}}{x_{i-1}-x_{i-n}} \right) N_{n-1,i-1}(x) + \left(\frac{x_i-x}{x_i-x_{i-n+1}} \right) N_{n-1,i}(x) \right\} . \tag{5.1.2}$$

Thus

$$s(x) = \sum_i c_i^{[1]}(x) N_{n-1,i}(x) , \tag{5.1.3}$$

where the reduced coefficients $c_i^{[1]}(x)$ are given by

$$c_i^{[1]}(x) = \frac{(x-x_{i-n+1})c_{i+1}+(x_i-x)c_i}{x_i - x_{i-n+1}} \quad . \tag{5.1.4}$$

Clearly this reduction process may be repeated; we obtain

$$s(x) = \sum_i c_i^{[l]}(x)N_{n-l,i}(x) \ , \tag{5.1.5}$$

where

$$c_i^{[l]}(x) = \begin{cases} c_i & (l=0) \\ \dfrac{(x-x_{i-n+l})c_{i+1}^{[l-1]}(x)+(x_i-x)c_i^{[l-1]}(x)}{x_i - x_{i-n+l}} & (l>0). \end{cases} \tag{5.1.6}$$

In particular, because of (3.2.9),

$$s(x) = c_i^{[n-1]}(x) \qquad (x_{i-1} \leqslant x < x_i) \ . \tag{5.1.7}$$

Note that, as a consequence of (5.1.6), for any value of x such that $x_{i-1} \leqslant x < x_i$, $c_i^{[l]}(x)$ is a convex combination of the values of c_i, c_{i+1}, ..., c_{i+l}. In particular, $s(x)$ is a convex combination of the values of c_i, c_{i+1}, ..., c_{i+n-1}.

Curry and Schoenberg (1966) give a lengthy proof that the B-splines $\{N_{ni}(x)\}$ defined upon an n-extended partition $\{x_i\}$ are linearly independent and form a basis for splines of order n with knots $\{x_i\}$. We present simpler proofs, which we believe to be new, of these results.

Theorem 5.1.1

The B-splines $\{N_{ni}(x)\}$ defined upon an n-extended partition $\{x_i\}$ are linearly independent.

Proof

If the $N_{ni}(x)$ are linearly independent then no non-trivial linear combination of the $N_{ni}(x)$ can be identically zero. Assume, therefore, that there exist values c_i, not all zero, such that for all x,

$$s(x) = \sum_i c_i N_{ni}(x) = 0 . \tag{5.1.8}$$

We shall show that such an assumption leads to a contradiction.

Consider values of x in the interval $x_{j-1} \leqslant x < x_j$. Then, using (5.1.7),

$$s(x) = c_j^{[n-1]}(x) . \tag{5.1.9}$$

Now, by virtue of (5.1.6), $c_j^{[n-1]}(x)$ can be identically zero only if $c_j^{[n-2]}(x)$ and $c_{j+1}^{[n-2]}(x)$ are both identically zero. In turn, $c_j^{[n-2]}(x)$ and $c_{j+1}^{[n-2]}(x)$ can be identically zero only if $c_j^{[n-3]}(x)$, $c_{j+1}^{[n-3]}(x)$ and $c_{j+2}^{[n-3]}(x)$ are all identically zero. Continuation of this argument leads to the result that $c_j^{[n-1]}(x)$ can be identically zero only if the values of c_i $(i = j, j+1, \ldots, j+n-1)$ are all zero, which is the required contradiction, at least for values of x such that $x_{j-1} \leqslant x < x_j$. Consideration of such intervals for all j leads to the required contradiction for all x. $\square$

Theorem 5.1.2

An arbitrary spline s(x) of order n defined upon a standard knot set can be represented uniquely as

$$s(x) = \sum_{i=1}^{N+n-1} c_i N_{ni}(x) \qquad (a \leqslant x \leqslant b) . \tag{5.1.10}$$

Proof

Since the B-splines $N_{ni}(x)$ $(i = 1, 2, \ldots, N+n-1)$ defined upon a standard knot set are linearly independent (by virtue of Theorem 5.1.1) and since an arbitrary spline of order n can be described in terms of $N+n-1$ linear parameters (Section 3.1), it follows that $s(x)$ takes the form (5.1.10). $\square$

We shall sometimes make use of the equivalent representation involving the un-normalized B-splines, viz

$$s(x) = \sum_{i=1}^{N+n-1} c_i^* M_{ni}(x) \qquad (a \leqslant x \leqslant b). \tag{5.1.11}$$

As a consequence of (3.2.6), the coefficients in the two representations are related by

$$c_i^* = (x_i - x_{i-n}) c_i \qquad (i = 1, 2, \ldots, N+n-1). \tag{5.1.12}$$

The next theorem establishes lower and upper bounds on a spline in terms of the coefficients of its B-spline representation.

<u>Theorem 5.1.3</u>

If $s(x)$ has the B-spline representation (5.1.1), then for $x_{j-1} \leqslant x < x_j$,

$$\min_{j \leqslant i < j+n} c_i \leqslant s(x) \leqslant \max_{j \leqslant i < j+n} c_i. \tag{5.1.13}$$

<u>Proof</u>

The proof follows immediately from the observation made earlier in this section that for $x_{j-1} \leqslant x < x_j$, $s(x)$ is a convex combination of the values of $c_j, c_{j+1}, \ldots, c_{j+n-1}$. $\square$

5.2 The numerical evaluation of a spline from its B-spline representation

We give in this section algorithmic presentations of two schemes, both based upon the use of convex combinations, for evaluating a spline $s(x)$ of order n from its representation as a linear combination of B-splines.

Given a standard knot set and a set of coefficients c_i $(i = 1, 2, \ldots, N+n-1)$, we wish to evaluate (5.1.10) for a prescribed value of x $(a \leq x \leq b)$. For either scheme let j be the unique integer satisfying $x_{j-1} \leq x < x_j$ (in the exceptional case $x = b$, set $j = N$). The value of j may be found either by sequential search or, if N is large, more efficiently by binary search. As a consequence of the compact support property of the B-splines, the sum (5.1.10) reduces to

$$s(x) = \sum_{i=j}^{j+n-1} c_i N_{ni}(x) \qquad (x_{j-1} \leq x < x_j) \ . \tag{5.2.1}$$

In the first scheme (Scheme A) we use relation (5.1.6) to form the triangular array $c_i^{[l]}$ $(i = j, j+1, \ldots, j+n-1-l;\ l = 0, 1, \ldots, n-1)$, typified here by the case $n = 4$:

$$\begin{array}{cccc}
c_j^{[0]} & & & \\
 & c_j^{[1]} & & \\
c_{j+1}^{[0]} & & c_j^{[2]} & \\
 & c_{j+1}^{[1]} & & c_j^{[3]} \\
c_{j+2}^{[0]} & & c_{j+1}^{[2]} & \\
 & c_{j+2}^{[1]} & & \\
c_{j+3}^{[0]} & & &
\end{array} \tag{5.2.2}$$

It is convenient to form this array column by column, the single entry in the last column being the required value of $s(x)$. Evidently, the value of $c_i^{[l]}(x)$, once computed, may overwrite the value of $c_i^{[l-1]}(x)$, since the latter is then no longer required. Thus only n storage locations are required by Scheme A, an algorithmic presentation of which is given below.

Algorithm 5.2.1: The evaluation of $s(x)$ from its normalized B-spline representation using Scheme A.

Step 1. Determine j such that $x_{j-1} \leq x < x_j$ using sequential or binary search.

Comment: Set the initial conditions.

Step 2. For $i = j, j+1, \ldots, j+n-1$ set $d_i = c_i$.

Comment: The value of $s(x)$ is computed by convex combinations in Steps 3-5.

Step 3. For $l = 1, 2, \ldots, n-1$ execute Step 4.

Step 4. For $i = j, j+1, \ldots, j+n-l-1$ replace d_i by

$$\frac{(x-x_{i-n+l})d_{i+1}+(x_i-x)d_i}{x_i - x_{i-n+l}} .$$

Step 5. Set $s(x) = d_j$.

It has been observed empirically by de Boor (1972) that Scheme A is stable even for orders as high as 80. In Section 5.3 we prove rigorously that de Boor's observation is in fact a property of the method for arbitrary coefficients and knots.

The second scheme (Scheme B) is more appropriate if two or more splines with the same knots are to be evaluated from their respective B-spline coefficients (for an important application see Chapter 10). Scheme B is based upon the initial generation of the non-zero values of the nth-order B-splines, ie the values of $v_i = N_{ni}(x)$ $(i = j, j+1, \ldots, j+n-1)$, from Algorithm 3.12.2, followed by the direct evaluation of

$$s(x) = \sum_{i=j}^{j+n-1} c_i v_i . \tag{5.2.3}$$

An algorithm for this scheme is given below. Again only n storage locations are required.

Algorithm 5.2.2: The evaluation of $s(x)$ from its normalized B-spline representation using Scheme B.

Step 1. Determine j such that $x_{j-1} \leqslant x < x_j$ using sequential or binary search.

Step 2. Use Algorithm 3.12.2 to evaluate $v_i = N_{ni}(x)$ for $i = j, j+1, \ldots, j+n-1$.

Step 3. Form $s(x) = \sum_{i=j}^{j+n-1} c_i v_i$.

Either scheme takes $\frac{3}{2}n^2 + O(n)$ long operations.

5.3 Error analyses of algorithms for evaluating a spline from its B-spline representation

To carry out a floating-point error analysis of Algorithm 5.2.1 (Scheme A), let $\bar{c}_i^{[l]}(x)$ denote the computed value of $c_i^{[l]}(x)$ and

$$\delta c_i^{[l]}(x) = \bar{c}_i^{[l]}(x) - c_i^{[l]}(x). \tag{5.3.1}$$

In accordance with (5.1.6) we set the initial conditions

$$\bar{c}_i^{[0]}(x) = c_i^{[0]}(x) = c_i \, , \quad \delta c_i^{[0]}(x) = 0. \tag{5.3.2}$$

For $l > 0$, the floating-point equivalent of (5.1.6) is

$$\bar{c}_i^{[l]}(x) = fl \left\{ \frac{(x-x_{i-n+l})\bar{c}_{i+1}^{[l-1]}(x) + (x_i - x)\bar{c}_i^{[l-1]}(x)}{x_i - x_{i-n+l}} \right\}. \tag{5.3.3}$$

Relation (5.3.3) is similar in form to (3.8.5), and the method of analysis of the latter may be applied to some extent to the former. However, the $c_i^{[l]}(x)$ and $\bar{c}_i^{[l]}(x)$ may be positive, negative or zero, whereas the $\bar{M}_{rj}(x)$ in (3.8.5) are always non-negative (Theorem 3.8.2). By analogy with (3.8.8)

$$\delta c_i^{[l]}(x) = \Big[(x-x_{i-n+l})\Big\{\delta c_{i+1}^{[l-1]}(x)+5e_1\bar{c}_{i+1}^{[l-1]}(x)\Big\}$$

$$+(x_i-x)\Big\{\delta c_i^{[l-1]}(x)+5e_2\bar{c}_i^{[l-1]}(x)\Big\}\Big]/(x_i-x_{i-n+l}), \tag{5.3.4}$$

where $|e_1|$, $|e_2| < 2^{-t_1}$ and e_1 and e_2 depend on i and l. Using (5.3.1), (5.3.4) becomes

$$\delta c_i^{[l]}(x) = \Big[(x-x_{i-n+l})\Big\{\delta c_{i+1}^{[l-1]}(x)(1+5e_1)+5e_1c_{i+1}^{[l-1]}(x)\Big\}$$

$$+(x_i-x)\Big\{\delta c_i^{[l-1]}(x)(1+5e_2)+5e_2c_i^{[l-1]}(x)\Big\}\Big]/(x_i-x_{i-n+l}). \tag{5.3.5}$$

<u>Theorem 5.3.1</u>

If $x_{j-1} \leq x < x_j$ and the array $c_i^{[l]}(x)$ (i = j, j+1, ..., j+n-1-l; l = 0, 1, ..., n-1) is formed using relation (5.1.6), then the values $\bar{c}_i^{[l]}(x)$ actually computed are such that the errors $\delta c_i^{[l]}(x)$ satisfy

$$\left|\delta c_i^{[l]}(x)\right| \leq 5.862l2^{-t} \max_{i \leq k \leq i+l} |c_k|. \tag{5.3.6}$$

<u>Proof</u>

The slightly stronger result

$$\left|\delta c_i^{[l]}(x)\right| \leq 5l(1+2^{-t})^{5.3l}2^{-t_1} \max_{i\leq k \leq i+l} |c_k| \tag{5.3.7}$$

is derived, from which (5.3.6) follows by virtue of (1.1.9) and (1.1.11).

Assume the theorem to be true for l = r-1 $\geq$ 0, ie that

$$\left|\delta c_i^{[r-1]}(x)\right| \leq 5(r-1)(1+2^{-t})^{5.3(r-1)}2^{-t_1} \max_{i\leq k \leq i+r-1} |c_k|. \tag{5.3.8}$$

Then (5.3.5) yields

$$\left|\delta c_i^{[r]}(x)\right| \leqslant \left(\frac{x-x_{i-n+r}}{x_i-x_{i-n+r}}\right)\left\{5(r-1)(1+2^{-t})^{5.3r}2^{-t_1} \max_{i+1\leqslant k\leqslant i+r}\left|c_k\right| + (5)2^{-t_1}\left|c_{i+1}^{[r-1]}(x)\right|\right\}$$

$$+\left(\frac{x_i-x}{x_i-x_{i-n+r}}\right)\left\{5(r-1)(1+2^{-t})^{5.3r}2^{-t_1} \max_{i\leqslant k\leqslant i+r-1}\left|c_k\right| + (5)2^{-t_1}\left|c_i^{[r-1]}(x)\right|\right\}. \tag{5.3.9}$$

But, since $c_i^{[r-1]}(x)$ is a convex combination of the values of c_k $(k = i, i+1, \ldots, i+r-1)$ (Section 5.1),

$$\left|c_i^{[r-1]}(x)\right| \leqslant \max_{i\leqslant k\leqslant i+r-1}\left|c_k\right|. \tag{5.3.10}$$

Thus the expression in the second set of braces in (5.3.9) reduces to

$$\left\{5(r-1)(1+2^{-t})^{5.3r}+5\right\}2^{-t_1} \max_{i\leqslant k\leqslant i+r-1}\left|c_k\right|, \tag{5.3.11}$$

which is bounded by

$$5r(1+2^{-t})^{5.3r}2^{-t_1} \max_{i\leqslant k\leqslant i+r-1}\left|c_k\right|. \tag{5.3.12}$$

Similarly, the expression in the first set of braces in (5.3.9) is bounded by a quantity that is identical to (5.3.12), but with i replaced by i+1. But from (5.3.9), $\left|\delta c_i^{[r]}(x)\right|$ is bounded by a convex combination of (5.3.12) and its counterpart with i replaced by i+1. Thus

$$\left|\delta c_i^{[r]}(x)\right| \leqslant 5r(1+2^{-t})^{5.3r}2^{-t_1} \max_{i\leqslant k\leqslant i+r}\left|c_k\right|. \tag{5.3.13}$$

Thus (5.3.7) is true for $l = r$. But (5.3.7) is evidently true for $l = 0$. Hence, by induction, it is true for $l = 0, 1, \ldots, n-1$. $\square$

Corollary 5.3.1

If $c_i \geqslant 0$ $(i = j, j+1, \ldots, j+n-1)$ the elements $\bar{c}_i^{[l]}(x)$ generated by Scheme A have errors $\delta c_i^{[l]}(x) = \bar{c}_i^{[l]}(x) - c_i^{[l]}(x)$ satisfying the relative error bound

$$\left|\delta c_i^{[l]}(x)\right| \leqslant 6.54912^{-t} c_i^{[l]}(x) . \tag{5.3.14}$$

In particular, the error $\delta s(x) = \bar{s}(x) - s(x)$ satisfies the relative error bound

$$\left|\delta s(x)\right| \leqslant 6.549(n-1)2^{-t} s(x) . \tag{5.3.15}$$

Proof

Firstly, the a posteriori bound

$$\left|\delta c_i^{[l]}(x)\right| \leqslant 5l(1-2^{-t})^{-10.3l} 2^{-t_1} \bar{c}_i^{[l]}(x) \tag{5.3.16}$$

is established.

Suppose that (5.3.16) is true for $l = 0, 1, \ldots, r-1 \geqslant 0$. Then (5.3.5) gives

$$\left|\delta c_i^{[r]}(x)\right| \leqslant \left\{\frac{(x-x_{i-n+r})\bar{c}_{i+1}^{[r-1]}(x)+(x_i-x)\bar{c}_i^{[r-1]}(x)}{x_i - x_{i-n+r}}\right\} \times$$

$$\times \left[5(r-1)(1-2^{-t})^{-10.3(r-1)} 2^{-t_1}\left\{1+(5)2^{-t_1}\right\} + (5)2^{-t_1}\right] . \tag{5.3.17}$$

Now $1+(5)2^{-t_1} = 1+(5.3)2^{-t} < (1+2^{-t})^{5.3} < (1-2^{-t})^{-5.3}$. Hence the term in square brackets in (5.3.17) is less than $5\left\{(r-1)(1-2^{-t})^{5-10.3r}+1\right\}2^{-t_1} < 5r(1-2^{-t})^{5-10.3r}2^{-t_1}$. Now the non-negativity of the $\bar{c}_i^{[l]}(x)$ follows from the non-negativity of the c_i (cf Theorem 3.8.2) and, as a consequence, $\bar{c}_i^{[r]}(x)(1-2^{-t})^{-5}$ is a bound for the first term in braces in (5.3.17). (cf (3.9.4)). Thus

$$\left|\delta c_i^{[r]}(x)\right| \leqslant 5r(1-2^{-t})^{-10.3r}2^{-t}\bar{c}_i^{[r]}(x) \; . \qquad (5.3.18)$$

Thus (5.3.16) is true for $l = r$. But it is evidently true for $l = 0$ and hence by induction for $l \geqslant 0$. Since from (1.1.12), $(1-2^{-t})^{-10.31} < 1.112$, (5.3.16) may be simplified to

$$\left|\delta c_i^{[l]}(x)\right| \leqslant 5.8941\,2^{-t}\bar{c}_i^{[l]}(x) \; , \qquad (5.3.19)$$

from which the relative error bounds (5.3.14) and (5.3.15) follow readily. □

We now analyze Algorithm 5.2.2, presenting our main result as a theorem.

<u>Theorem 5.3.2</u>

The value of $\bar{s}(x)$ generated by Algorithm 5.2.2 (Scheme B) has an error $\delta s(x) = \bar{s}(x) - s(x)$ satisfying the bound

$$\left|\delta s(x)\right| \leqslant 7.745n2^{-t} \max_{j \leqslant k < j+n} \left|c_k\right| \; . \qquad (5.3.20)$$

<u>Proof</u>

Summation of the series (5.2.3) yields

$$\bar{s}(x) = \sum_{i=j}^{j+n-1} c_j\bar{v}_i(1+\varepsilon_i)^n \; , \qquad (5.3.21)$$

where

$$\left|\varepsilon_i\right| \leqslant 2^{-t} \quad (i = j, j+1, \ldots, j+n-1) \qquad (5.3.22)$$

and $\bar{v}_i$ denotes the computed value of $v_i = N_{ni}(x)$. The term $(1+\varepsilon_i)^n$ in (5.3.21) can in fact be replaced (in the case of forward summation) by $(1+\varepsilon_j)^n$ if $i = j$ and by $(1+\varepsilon_i)^{n+1+j-i}$ if $i = j+1, j+2, \ldots, j+n-1$ (Wilkinson, 1965: 114), but we need only the weaker result here. Now, from (3.9.13),

$$\bar{v}_i = \bar{N}_{ni}(x) = N_{ni}(x)(1+E_{ni}) \; , \qquad (5.3.23)$$

where

$$\left|E_{ni}\right| \leqslant 6.685(n-1)2^{-t} . \tag{5.3.24}$$

Hence

$$\delta s(x) = \sum_{i=j}^{j+n-1} c_i N_{ni}(x) \left\{ (1+E_{ni})(1+\varepsilon_i)^n - 1 \right\} . \tag{5.3.25}$$

Now the term in braces in (5.3.25) is bounded in modulus by

$$\begin{aligned}
&\left\{1+6.685(n-1)2^{-t}\right\}(1+2^{-t})^n-1 \\
&< \left\{1+6.685(n-1)2^{-t}\right\}(1+1.06n2^{-t})-1 \\
&= \left\{7.745n-6.685+(6.685)(1.06)n(n-1)2^{-t}\right\} 2^{-t} \\
&< (7.745n-6.685+0.1)2^{-t} = (7.745n-6.585)2^{-t} \\
&< 7.745n2^{-t} .
\end{aligned} \tag{5.3.26}$$

Hence

$$\left|\delta s(x)\right| \leqslant 7.745n2^{-t} \sum_{i=j}^{j+n-1} \left|c_i\right| \left|N_{ni}(x)\right| . \tag{5.3.27}$$

The result (5.3.20) then follows from (5.3.27) since $N_{ni}(x) \geqslant 0$ and $\sum_{i=j}^{j+n-1} N_{ni}(x) = 1$. $\square$

Corollary 5.3.2

If $c_i \geqslant 0$ $(i = j, j+1, \ldots, j+n-1)$, then the value of $\bar{s}(x)$ generated by Scheme B has an error $\delta s(x)$ satisfying the relative error bound

$$\left|\delta s(x)\right| \leqslant 7.745n2^{-t}s(x) . \tag{5.3.28}$$

Proof

Since $c_i \geqslant 0$ and $N_{ni}(x) \geqslant 0$, the sum in (5.3.27) can be replaced by $\sum_{i=j}^{j+n-1} c_i N_{ni}(x) = s(x)$ from (5.2.1), which then establishes (5.3.28). $\square$

Note that we may also interpret our result in the sense of a backward error analysis. For Scheme B, from (5.3.21) and (5.3.23),

$$\bar{s}(x) = \sum_{i=j}^{j+n-1} \bar{c}_i N_{ni}(x) , \qquad (5.3.29)$$

where

$$\bar{c}_i = c_i(1+G_{ni}) = c_i(1+E_{ni})(1+\varepsilon_i)^n . \qquad (5.3.30)$$

It is easily established that

$$\left|G_{ni}\right| < 7.745n2^{-t} . \qquad (5.3.31)$$

Hence the computed value $\bar{s}(x)$ is the value that would be obtained using exact computation upon a set of coefficients $\bar{c}_i$ perturbed slightly (in a relative sense) from the c_i. Similar results may be established for Scheme A, the only difference being the magnitude of the numerical constant in (5.3.31).

5.4 The effect of errors in the B-spline coefficients on the computed value of the spline

We now consider the numerical evaluation of $s(x)$ when the coefficients $\{c_i\}$ are subject to uncertainty. This would be the case if the c_i were the results of a previous computation, as they would be in the determination of spline approximations and spline interpolants (Chapters 6 and 7) and also in the representation of polynomials as splines (Section 5.7). Specifically, suppose that perturbed coefficients $\{\bar{c}_i\}$ are known and that a bound δc such that

$$\max_i \left|\bar{c}_i - c_i\right| \leq \delta c \qquad (5.4.1)$$

is available. Let

$$\bar{s}(x) = fl\left\{\sum_i \bar{c}_i N_{ni}(x)\right\} . \qquad (5.4.2)$$

We require a bound for $\left|\bar{s}(x)-s(x)\right|$ in terms of the known quantities $\{\bar{c}_i\}$ and δc.

For Schemes A and B of Section 5.2, the use of (5.3.6) and (5.3.20) gives

$$\bar{s}(x) = \sum_i \bar{c}_i N_{ni}(x) + E , \tag{5.4.3}$$

where

$$\left| E \right| \leqslant K2^{-t} \max_i \left| \bar{c}_i \right| \tag{5.4.4}$$

and

$$K = \begin{cases} 5.9(n-1) & \text{(Scheme A)} \\ 7.8n & \text{(Scheme B)} . \end{cases} \tag{5.4.5}$$

Thus

$$\bar{s}(x) = s(x) + \sum_i (\bar{c}_i - c_i) N_{ni}(x) + E \tag{5.4.6}$$

and hence

$$\left|\delta s(x)\right| = \left|\bar{s}(x) - s(x)\right| \leqslant \max_i \left|\bar{c}_i - c_i\right| + K2^{-t} \max_i \left|\bar{c}_i\right| \tag{5.4.7}$$

$$\leqslant \delta c + K2^{-t} \max_i \left|\bar{c}_i\right| , \tag{5.4.8}$$

which demonstrates that the bulk of the effect on the computed value of any errors in the $\{c_i\}$ is at most equal to the largest of these errors. There is a further very mild contribution to the error from the term $\max_i \left|\bar{c}_i\right|$ in (5.4.8). From (5.4.8),

$$\left|\delta s(x)\right| \leqslant \delta c + K2^{-t} \max_i \left|c_i\right| + K2^{-t} \max_i \left|\bar{c}_i - c_i\right| \tag{5.4.9}$$

$$\leqslant 1.1 \delta c + K2^{-t} \max_i \left|c_i\right| , \tag{5.4.10}$$

under the very weak assumption (in accordance with (1.1.7)) that $K2^{-t} < 0.1$.

5.5 The B-spline representation of powers

Since a polynomial of degree less than n is a special case of a spline of order n, it follows from Theorem 5.1.2 that any such polynomial has a unique representation on (a,b) as a linear combination of the B-splines $N_{ni}(x)$ ($i = 1, 2, \ldots, N+n-1$). This result will therefore apply in particular to the "polynomial" x^r ($r = 0, 1, \ldots, n-1$).

Marsden (1970) gives a result (see (5.5.1) below), which enables certain powers of x to be represented explicitly in terms of the $N_{ni}(x)$. Marsden's original (unpublished) proof of (5.5.1) was in fact rather complicated, so in his 1970 paper he gave a more elegant proof communicated privately to him by T N E Greville. A far neater proof, however, is due to de Boor (1972) and is based upon the use of identity (3.4.2) which was unknown of course to the above authors at the time of their work.

We show that the Marsden-Greville result, which in fact gives representations of x^r in terms of the $N_{ni}(x)$ for $r = 0$, 1 and 2, may be generalized to the case of all $r < n$. We establish a simple recurrence relation, which enables the coefficients in these representations to be computed efficiently and accurately. These results are applied in Section 5.7 to the problem of representing an arbitrary polynomial given in its power-series form in terms of B-splines.

Finally we prove that the coefficients in the B-spline representation of a function f(x) are bounded if the Taylor series expansion of f(x) ($|x| \leq 1$) converges absolutely.

Theorem 5.5.1

If p and q are integers such that

$$x_{p-1} < x_{q-n+1} \ ,$$

then the relation

$$(t-x)^{n-1} = \sum_{i=p}^{q} (t-x_{i-n+1})(t-x_{i-n+2}) \dots (t-x_{i-1})N_{ni}(x) \qquad (5.5.1)$$

is valid for all t and for $x_{p-1} < x < x_{q-n+1}$.

Proof

The proof is by induction. Assume the theorem to be true for $n = r \geqslant 1$, ie that

$$(t-x)^{r-1} = \sum_{i=p}^{q} (t-x_{i-r+1})(t-x_{i-r+2}) \dots (t-x_{i-1})N_{ri}(x) , \qquad (5.5.2)$$

where

$$x_{p-1} < x < x_{q-r+1} . \qquad (5.5.3)$$

Now consider the expression

$$E = \sum_{i=p}^{q} (t-x_{i-r})(t-x_{i-r+1}) \dots (t-x_{i-1})N_{r+1,i}(x). \qquad (5.5.4)$$

We wish to show that $E = (t-x)^r$ if $x_{p-1} < x < x_{q-r}$. Now, by making use of the recurrence (3.4.2), and the limited support of the B-splines, we obtain

$$E = \sum_{i=p}^{q} (t-x_{i-r})(t-x_{i-r+1}) \dots (t-x_{i-1}) \left\{ \left(\frac{x-x_{i-r-1}}{x_{i-1}-x_{i-r-1}} \right) N_{r,i-1}(x) + \left(\frac{x_i-x}{x_i-x_{i-r}} \right) N_{ri}(x) \right\}$$

$$= \sum_{i=p-1}^{q-1} (t-x_{i-r+1})(t-x_{i-r+2}) \dots (t-x_i) \left(\frac{x-x_{i-r}}{x_i-x_{i-r}} \right) N_{ri}(x)$$

$$+ \sum_{i=p}^{q} (t-x_{i-r})(t-x_{i-r+1}) \dots (t-x_{i-1}) \left(\frac{x_i-x}{x_i-x_{i-r}} \right) N_{ri}(x)$$

$$= \sum_{i=p}^{q-1} (t-x_{i-r+1})(t-x_{i-r+2}) \dots (t-x_{i-1}) \left\{ \frac{(t-x_i)(x-x_{i-r})+(t-x_{i-r})(x_i-x)}{x_i - x_{i-r}} \right\} N_{ri}(x)$$

(5.5.5)

Simplification of (5.5.5) yields

$$E = (t-x)\sum_{i=p}^{q-1}(t-x_{i-r+1})(t-x_{i-r+2})\ \cdots\ (t-x_{i-1})N_{ri}(x), \tag{5.5.6}$$

which by virtue of (5.5.2), (5.5.3), and the limited support of the $N_{ri}(x)$, is equal to $(t-x)^r$ if $x_{p-1} < x < x_{q-r}$. Thus the theorem is true for $n = r+1$. But for $n = 1$ the right-hand side of (5.5.1) is simply $\sum_{i=p}^{q} N_{1i}(x)$, which for $x_{p-1} < x < x_q$ is equal to unity, by virtue of (5.6.1); the left-hand side of (5.5.1) equals $(t-x)^0 = 1$, also. Hence the theorem is true for $n = 1$ and therefore, by induction, for all $n \geqslant 1$. $\square$

Now let $p = 1$ and $q = N+n-1$. Then by equating coefficients of powers of t in (5.5.1), and once again utilizing the finite support of the $N_{ni}(x)$, we obtain for $x_0 < x < x_N$,

$$1 = \sum_{i=1}^{N+n-1} \xi_{ni}^{(0)}\, N_{ni}(x) = \sum_{i=1}^{N+n-1} N_{ni}(x)\ , \tag{5.5.7}$$

$$x = \sum_{i=1}^{N+n-1} \xi_{ni}^{(1)}\, N_{ni}(x)\ , \tag{5.5.8}$$

$$x^2 = \sum_{i=1}^{N+n-1} \xi_{ni}^{(2)}\, N_{ni}(x) \tag{5.5.9}$$

and, in general, for $0 \leqslant r < n$, by

$$x^r = \sum_{i=1}^{N+n-1} \xi_{ni}^{(r)}\, N_{ni}(x). \tag{5.5.10}$$

The coefficients $\xi_{ni}^{(r)}$ are obtained by equating like powers of t in (5.5.1) and hence are defined by

$$\xi_{ni}^{(0)} = 1\ , \tag{5.5.11}$$

$$\xi_{ni}^{(1)} = \sum_{k=i-n+1}^{i-1} x_k/(n-1), \tag{5.5.12}$$

$$\xi_{ni}^{(2)} = \sum_{\substack{k,l=i-n+1 \\ k<l}}^{i-1} x_k x_l / {}^{n-1}C_2, \tag{5.5.13}$$

and, in general, for $0 \leq r < n$, by

$$\xi_{ni}^{(r)} = \begin{cases} 1 & (r = 0) \\ \sum\limits_{i-n<k_1<k_2<\ldots<k_r<i} x_{k_1} x_{k_2} \ldots x_{k_r} / {}^{n-1}C_r & (r > 0). \end{cases} \tag{5.5.14}$$

5.6 Algorithms for computing the B-spline coefficients

Define

$$S_{ni}^{(r)} = {}^{n-1}C_r \, \xi_{ni}^{(r)} . \tag{5.6.1}$$

Theorem 5.6.1

$$S_{ni}^{(r)} = S_{n-1,i-1}^{(r)} + x_{i-1} S_{n-1,i-1}^{(r-1)} . \tag{5.6.2}$$

Proof

From (5.6.1) and (5.5.14),

$$S_{n-1,i-1}^{(r-1)} = \sum_{i-n<k_1<k_2<\ldots<k_{r-1}<i-1} x_{k_1} x_{k_2} \ldots x_{k_{r-1}} . \tag{5.6.3}$$

and

$$S_{n-1,i-1}^{(r)} = \sum_{i-n<k_1<k_2<\ldots<k_r<i-1} x_{k_1} x_{k_2} \ldots x_{k_r} . \tag{5.6.4}$$

Hence

$$S^{(r)}_{n-1,i-1} + x_{i-1}S^{(r-1)}_{n-1,i-1} =$$

$$\sum_{\substack{i-n<k_1<k_2 \ldots <k_r<i \\ k_r \neq i-1}} x_{k_1}x_{k_2}\ldots x_{k_r} + \sum_{\substack{i-n<k_1<k_2<\ldots<k_r<i \\ k_r = i-1}} x_{k_1}x_{k_2}\ldots x_{k_r}$$

$$= \sum_{i-n<k_1<k_2<\ldots<k_r<i} x_{k_1}x_{k_2} \ldots x_{k_r} = S^{(r)}_{ni} . \qquad \square \qquad (5.6.5)$$

It is easily verified that the elements in the first diagonal and the first row of the array (see Figs 5.6.1 and 5.6.2 below) are given by

$$S^{(0)}_{j,i-n+j} = 1 \qquad (j = 1, 2, \ldots, n-r) \qquad (5.6.6)$$

and those in the first row by

$$S^{(0)}_{1,i-n+1} = 1 \qquad (5.6.7)$$

and

$$S^{(j)}_{j+1,i-n+j+1} = x_{i-n+j}S^{(j-1)}_{j,i-n+j} \qquad (j = 1, 2, \ldots, r). \qquad (5.6.8)$$

Relations (5.6.6) or relations (5.6.7) and (5.6.8) provide a set of starting values for recurrence (5.6.2). Values in the array may then be generated diagonal by diagonal, row by row or column by column.

It follows from Theorem 5.6.1 that a particular value of $S^{(r)}_{ni}$ may be generated by computing the elements of a rhomboidal array, typified here by the case $n = 7$, $i = 9$, $r = 4$:

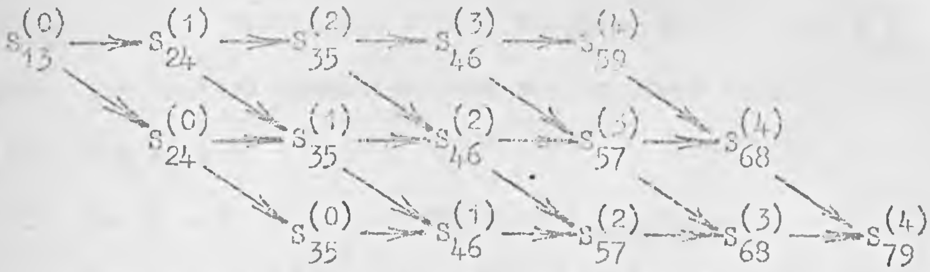

Fig 5.6.1. Schematic illustration of the computation of $S_{79}^{(4)}$.

The arrows in Fig 5.6.1 indicate the dependence of an element in the array upon its immediate neighbours (predecessors). Thus, for example, $S_{57}^{(3)}$ is computed from $S_{46}^{(2)}$ and $S_{46}^{(3)}$.

In the general case the array takes the form:

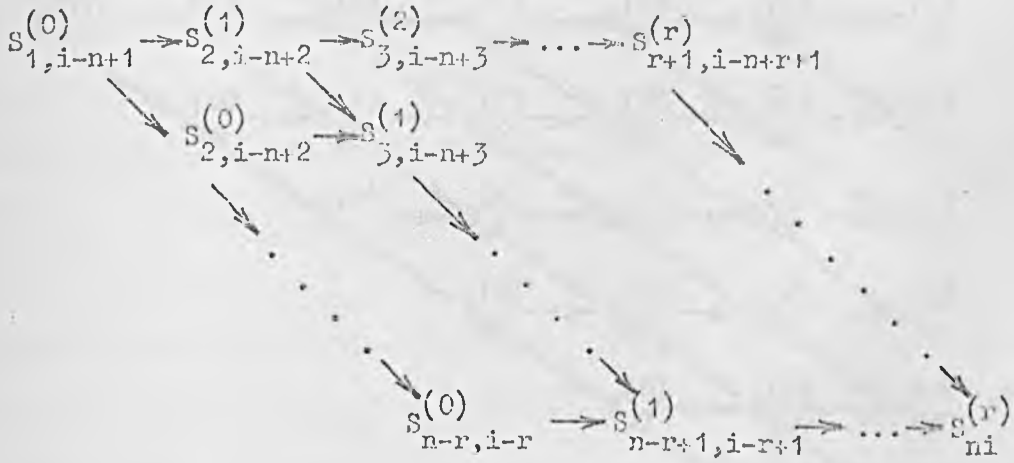

Fig 5.6.2. Schematic illustration of the computation of the general value of $S_{ni}^{(r)}$.

In practice it is unnecessary to store the complete array since a new row (column or diagonal) may overwrite the previous row (column or diagonal), the latter being no longer required once the former has been computed. In Algorithm 5.6.1 below for evaluating $\xi_{ni}^{(r)}$, the elements in the array are formed row by row in the vector $\left\{ v_0, v_1, \ldots, v_r \right\}$.

Algorithm 5.6.1: Evaluation of the B-spline coefficient $\xi_{ni}^{(r)}$.

Comment: The initial conditions are set in Steps 1-2.

Step 1. Set $v_0 = 1$.

Step 2. For $k = 1, 2, \ldots, r$ set $v_k = x_{i-n+k} v_{k-1}$.

Comment: The value of $S_{ni}^{(r)}$ is computed by recurrence in Steps 3-4.

Step 3. For $k = 2, 3, \ldots, n-r$ execute Step 4.

Step 4. For $j = 1, 2, \ldots, r$ replace v_j by $v_j + x_{i-n+k-1+j} v_{j-1}$.

Step 5. Set $\xi_{ni}^{(r)} = v_r / {}^{n-1}C_r$.

A simple extension of the array enables the values of $S_{ni}^{(j)}$ (and hence $\xi_{ni}^{(j)}$) to be computed for all values of j from 0 to n-1. The extended array for the above example in which n = 7 and i = 9 becomes:

$$\begin{tikzcd}
S^{(0)}_{13} \arrow[r]\arrow[dr] & S^{(1)}_{24} \arrow[r]\arrow[dr] & S^{(2)}_{35} \arrow[r]\arrow[dr] & S^{(3)}_{46} \arrow[r]\arrow[dr] & S^{(4)}_{57} \arrow[r]\arrow[dr] & S^{(5)}_{68} \arrow[r]\arrow[dr] & S^{(6)}_{79} \\
 & S^{(0)}_{24} \arrow[r]\arrow[dr] & S^{(1)}_{35} \arrow[r]\arrow[dr] & S^{(2)}_{46} \arrow[r]\arrow[dr] & S^{(3)}_{57} \arrow[r]\arrow[dr] & S^{(4)}_{68} \arrow[r]\arrow[dr] & S^{(5)}_{79} \\
 & & S^{(0)}_{35} \arrow[r]\arrow[dr] & S^{(1)}_{46} \arrow[r]\arrow[dr] & S^{(2)}_{57} \arrow[r]\arrow[dr] & S^{(3)}_{68} \arrow[r]\arrow[dr] & S^{(4)}_{79} \\
 & & & S^{(0)}_{46} \arrow[r]\arrow[dr] & S^{(1)}_{57} \arrow[r]\arrow[dr] & S^{(2)}_{68} \arrow[r]\arrow[dr] & S^{(3)}_{79} \\
 & & & & S^{(0)}_{57} \arrow[r]\arrow[dr] & S^{(1)}_{68} \arrow[r]\arrow[dr] & S^{(2)}_{79} \\
 & & & & & S^{(0)}_{68} \arrow[r]\arrow[dr] & S^{(1)}_{79} \\
 & & & & & & S^{(0)}_{79}
\end{tikzcd}$$

Fig 5.6.3. Schematic illustration of the computation of the values of $S_{79}^{(j)}$ $(j = 0, 1, \ldots, 6)$.

The final column of such a triangular array then yields the required values. In general this triangular array assumes the form:

$$\begin{tikzcd}
S^{(0)}_{1,i-n+1} \arrow[r] \arrow[dr] & S^{(1)}_{2,i-n+2} \arrow[r] \arrow[dr] & S^{(2)}_{3,i-n+3} \arrow[r] \arrow[dr] & S^{(3)}_{4,i-n+4} \arrow[r] \arrow[dr] & \cdots \arrow[r] \arrow[dr] & S^{(n-2)}_{n-1,i-1} \arrow[r] \arrow[dr] & S^{(n-1)}_{ni} \\
& S^{(0)}_{2,i-n+2} \arrow[r] \arrow[dr] & S^{(1)}_{3,i-n+3} \arrow[r] \arrow[dr] & S^{(2)}_{4,i-n+4} \arrow[r] & \cdots \arrow[r] \arrow[dr] & S^{(n-3)}_{n-1,i-1} \arrow[r] \arrow[dr] & S^{(n-2)}_{ni} \\
& & S^{(0)}_{3,i-n+3} \arrow[r] \arrow[dr] & S^{(1)}_{4,i-n+4} \arrow[r] & \cdots \arrow[r] & S^{(n-4)}_{n-1,i-1} \arrow[r] & S^{(n-3)}_{ni} \\
& & & S^{(0)}_{4,i-n+4} \arrow[r, dotted, dash] \arrow[dr] & \cdots \arrow[r, dotted, dash] \arrow[dr] & \cdots \arrow[r, dotted, dash] \arrow[dr] & \cdots \\
& & & & \ddots \arrow[dr] & S^{(1)}_{n-1,i-1} \arrow[r] \arrow[dr] & S^{(2)}_{ni} \\
& & & & & S^{(0)}_{n-1,i-1} \arrow[r] \arrow[dr] & S^{(1)}_{ni} \\
& & & & & & S^{(0)}_{ni}
\end{tikzcd}$$

Fig 5.6.4. Schematic illustration of the computation of the values of $S^{(j)}_{ni}$ $(j = 0, 1, \ldots, n-1)$.

In Algorithm 5.6.2 below the elements are again generated row by row in the vector $\{v_0, v_1, \ldots, v_{n-1}\}$, the final values of v_k $(k = 0, 1, \ldots, n-1)$ holding the values of $\xi^{(k)}_{ni}$.

<u>Algorithm 5.6.2</u>: Evaluation of the B-spline coefficients $\xi^{(k)}_{ni}$ for $k = 0, 1, \ldots, n-1$.

Comment: Initial conditions are set in Steps 1-2.

Step 1. Set $v_0 = 1$.

Step 2. For $k = 1, 2, \ldots, n-1$ set $v_k = x_{i-n+k}v_{k-1}$.

Comment: The values of $S^{(k)}_{ni}$ are computed by recurrence in Steps 3-4.

Step 3. For $k = 2, 3, \ldots, n-1$ execute Step 4.

Step 4. For $j = 1, 2, \ldots, n-k$ replace v_j by $v_j + x_{i-n+k-1+j}v_{j-1}$.

Step 5. For $k = 0, 1, \ldots, n-1$ set $\xi^{(k)}_{ni} = v_k / {}^{n-1}C_k$.

In Step 5 of Algorithm 5.6.1 it is necessary to divide the final value of $v_r(=S_{ni}^{(r)})$ by ${}^{n-1}C_r$ to obtain $\xi_{ni}^{(r)}$. Similar remarks apply to Algorithm 5.6.2. A few words regarding the evaluation of ${}^{n-1}C_r$ are in order here. For a prescribed value of n, let

$$u_k = {}^{n-1}C_k. \tag{5.6.9}$$

Then it is readily verified that

$$u_k = \begin{cases} 1 & (k = 0) \\ u_{k-1}(n-k)/k & (k > 0). \end{cases} \tag{5.6.10}$$

We recommend that (5.6.10) be used recursively to form $u_0, u_1, \ldots, u_r = {}^{n-1}C_r$. In Algorithm 5.6.2 and also in the methods considered in Section 5.7 it is necessary to form $\xi_{ni}^{(k)}$ for $k = 0, 1, \ldots, n-1$ and in such cases it is efficient to form the required values of u_k in the above manner. Since u_k is an integer (it being the number of ways of choosing k objects from n-1) then, unless integer overflow occurs, integer arithmetic can be used throughout to compute <u>exactly</u> the values of u_k $(k = 0, 1, \ldots, n-1)$. Note that the precise order of operations is important here, it being necessary to form $u_k = \{u_{k-1}(n-k)\}/k$ rather than $u_{k-1}\{(n-k)/k\}$, since the former expression in braces is always integral whereas the latter may well not be so. If integer overflow is likely to occur, floating-point arithmetic must be employed, in which case a straightforward error analysis of (5.6.10) shows that the computed value $\bar{u}_k$ satisfies

$$\bar{u}_k = u_k(1+\varepsilon_k)^{2k-1} \qquad (k > 0), \tag{5.6.11}$$

where $|\varepsilon_k| \leq 2^{-t}$, ie $\bar{u}_k$ has a relative error bounded in modulus by $(2k-1)2^{-t_1} = 1.06(2k-1)2^{-t}$. We assume, for the sake of complete rigour in our subsequent analyses, that the computed values of u_k do indeed satisfy (5.6.11) rather than being exact. However, for most practically useful values of n (eg for n less than about 50 on KDF9), $\bar{u}_k$ will indeed be equal to u_k for all $k = 0, 1, \ldots, n-1$.

5.7 The B-spline representation of polynomials

We now consider the representation of a polynomial of degree less than n (expressed explicitly in its power-series form) as a series of B-splines of order n defined on a standard knot set. That is, we wish to determine the coefficients c_i ($i = 1, 2, \ldots, N+n-1$) in the B-spline representation of

$$p(x) = \sum_{r=0}^{n-1} b_r x^r, \tag{5.7.1}$$

where the coefficients b_r ($r = 0, 1, \ldots, n-1$) are prescribed.

Using (5.5.10),

$$p(x) = \sum_{r=0}^{n-1} b_r \sum_{i=1}^{N+n-1} \xi_{ni}^{(r)} N_{ni}(x)$$

$$= \sum_{i=1}^{N+n-1} c_i N_{ni}(x), \tag{5.7.2}$$

where

$$c_i = \sum_{r=0}^{n-1} b_r \xi_{ni}^{(r)} \qquad (i = 1, 2, \ldots, N+n-1). \tag{5.7.3}$$

To determine each coefficient c_i it is merely necessary to invoke Algorithm 5.6.2 for the values of $\xi_{ni}^{(r)}$ ($r = 0, 1, \ldots, n-1$), multiply by the respective values of b_r and sum. A slightly more efficient approach, given as Algorithm 5.7.1 below, is to form <u>scaled</u> coefficients

$$d_r = b_r / {}^{n-1}C_r \quad (r = 0, 1, \ldots, n-1), \tag{5.7.4}$$

evaluate $s_{ni}^{(r)}$ ($r = 0, 1, \ldots, n-1$) from Steps 1-4 of Algorithm 5.6.2 and finally to form the values of c_i from

$$c_i = \sum_{r=0}^{n-1} d_r S_{ni}^{(r)} \quad (i = 1, 2, \ldots, N+n-1). \tag{5.7.5}$$

<u>Algorithm 5.7.1:</u> Conversion of a polynomial power series into its equivalent B-spline representation.

Comment: The values of $d_i = b_i / {}^{n-1}C_i$ are determined in Steps 1-4.

Step 1. Set $p = 1$ and $d_0 = b_0$.

Step 2. For $i = 1, 2, \ldots, n-1$ execute Steps 3-4.

Step 3. Replace p by $p(n-i)/i$.

Step 4. Set $d_i = b_i/p$.

Comment: The values of c_i are formed in Steps 5-7.

Step 5. For $i = 1, 2, \ldots, N+n-1$ execute Steps 6-7.

Step 6. Use Steps 1-4 of Algorithm 5.6.2 to evaluate $S_{ni}^{(k)} = v_k$ $(k = 0, 1, \ldots, n-1)$.

Step 7. Form $c_i = \sum_{k=0}^{n-1} d_k v_k$.

Now suppose that the x_i form a standard knot set with coincident end knots and that

$$a = -1 \ , \ b = +1 \ . \tag{5.7.6}$$

There is little loss of generality in this assumption, since any finite interval can, under a linear transformation, be mapped into the interval $-1 \leqslant x \leqslant 1$ (for details on the way in which such a mapping should be carried out see Section 1.2). Then, using (5.5.14) and (5.7.6) (recall that $a = \ldots = x_{-1} = x_0 < x_1 \leqslant \ldots \leqslant x_{N-1} < x_N = x_{N+1} = \ldots = b$),

$$\begin{aligned} \left| \xi_{ni}^{(r)} \right| &\leqslant \sum_{i-n<k_1<k_2<\ldots<k_r<i} \left| x_{k_1} x_{k_2} \ldots x_{k_r} \right| \Big/ {}^{n-1}C_r \\ &\leqslant \sum_{i-n<k_1<k_2<\ldots<k_r<i} 1 \Big/ {}^{n-1}C_r \\ &= {}^{n-1}C_r \Big/ {}^{n-1}C_r = 1. \end{aligned} \tag{5.7.7}$$

Thus, using (5.7.3) and (5.7.7),

$$\left|c_i\right| \leqslant \sum_{r=0}^{n-1} \left|b_r\right| \left|\xi_{ni}^{(r)}\right| \leqslant \sum_{r=0}^{n-1} \left|b_r\right| \quad (i = 1, 2, \ldots, N+n-1).$$

(5.7.8)

This result is interesting in that if the b_r $(r = 0, 1, \ldots)$ denote the coefficients in the Taylor expansion about the origin of a function $f(x)$ $(\left|x\right| \leqslant 1)$, then the absolute convergence of the Taylor series implies the boundedness of the B-spline coefficients for any order n. Indeed, as long as the x_i form a standard knot set with coincident end knots, the bound (5.7.8) is <u>independent of the number of knots and of their positions.</u> The bound is sharp in the sense that it can be attained arbitrarily closely (see the example below) for certain functions $f(x)$.

As a simple example consider the series expansion

$$e^x = \sum_{r=0}^{\infty} x^r/r! \; . \tag{5.7.9}$$

We have $b_r = 1/r!$ and therefore, for any standard knot set with coincident end knots $a = -1$, $b = +1$,

$$\left|c_i\right| \leqslant \sum_{r=0}^{\infty} 1/r! = e \tag{5.7.10}$$

(cf Example 6.8.3 in Chapter 6). For the exponential function the bound (5.7.8) may be approached arbitrarily closely for sufficiently large n (see the following numerical example).

As an illustration of the remarkable numerical stability of the processes described by Algorithms 5.6.1, 5.6.2 and 5.7.1, consider the computation of the values of c_i corresponding to the truncated Taylor expansion of e^x about $x = 0$ for the interval $\left|x\right| \leqslant 1$. The error in truncating this expansion after n terms is

$$R_n = \sum_{r=n}^{\infty} t^r/r! \tag{5.7.11}$$

for some t in (-1, 1). Thus

$$\left|R_n\right| \leq \sum_{r=n}^{\infty} 1/r! < \frac{1}{n!} \sum_{r=0}^{n} (n+1)^{-r} = \frac{n+1}{nn!} . \tag{5.7.12}$$

Now for $n > 14$, $\left|R_n\right| < 10^{-12} < 2^{-39}$, the relative machine precision of KDF9. Thus the B-spline representation with $n > 14$ of the truncated Taylor series should (at least in the absence of rounding errors) provide full machine accuracy on KDF9. The computed values of c_i in Table 5.7.1 correspond to the choice $n = 15$, $N = 1$, $x_i = -1$ $(i \leq 0)$, $x_i = 1 (i > 0)$ and $b_i = 1/i!$ The resulting B-spline series was computed from these values using Algorithm 5.2.1 for $x = -1(0.1)1$. These values are given in Column 2 of Table 5.7.2. In Column 3 of Table 5.7.2 are the differences between these values and the corresponding values of e^x as computed by the library exponential function on KDF9. In Column 4 are the differences between the values of the power series $p(x) = \sum_{i=0}^{14} x^i/i!$ computed by nesting and those of e^x.

Note that over the 21 points of evaluation the maximum departure of the computed B-spline series from the value of e^x is 2×10^{-11}, which is merely twice that of the maximum departure of $p(x)$ from e^x. This excellent agreement occurs in spite of the fact that there are three main sources of rounding error contributing to the values in Column 2 of Table 5.7.2: the rounding errors in the computed Taylor coefficients, the evaluation of the c_i from these coefficients and the evaluation of the B-spline series from these values of c_i.

i	c_i
1	0.36787 94411 7
2	0.42043 36470 5
3	0.48107 31153 8
4	0.55114 53898 8
5	0.63224 30222 8
6	0.72625 25737 7
7	0.83541 45060 0
8	0.96239 66833 3
9	1.11038 49869 6
10	1.28319 55777 7
11	1.48541 47454 2
12	1.72257 41989 9
13	2.00137 23352 4
14	2.32995 58529 7
15	2.71828 18284 6

Table 5.7.1 Values of the coefficients in the B-spline representation of e^x.

x	$s(x)$	$10^{11}\left\{s(x)-e^{x}\right\}$	$10^{11}\left\{p(x)-e^{x}\right\}$
-1.0	0.36787 94411 7	0	0
-0.9	0.40656 96597 4	0	0
-0.8	0.44932 89641 2	0	0
-0.7	0.49658 53037 9	0	0
-0.6	0.54881 16361 0	0	0
-0.5	0.60653 06597 2	+1	0
-0.4	0.67032 00460 5	+1	0
-0.3	0.74081 82206 9	+1	0
-0.2	0.81873 07530 9	+1	0
-0.1	0.90483 74180 6	+2	0
0.0	1.00000 00000 1	+1	0
0.1	1.10517 09180 8	0	0
0.2	1.22140 27581 7	+1	0
0.3	1.34985 88075 9	+2	0
0.4	1.49182 46976 6	+2	0
0.5	1.64872 12707 1	+1	0
0.6	1.82211 88003 9	0	0
0.7	2.01375 27074 8	+1	+1
0.8	2.22554 09284 9	0	0
0.9	2.45960 31111 6	0	+1
1.0	2.71828 18284 6	0	0

Table 5.7.2 Tabulation of the values of the computed B-spline representation of e^x and a comparison of their departures from e^x with the departures of the equivalent truncated Taylor series from e^x.

5.8 Error analyses of the algorithms for computing B-spline coefficients

We establish in this section some results relating to the stability of the processes described by Algorithms 5.6.1, 5.6.2 and 5.7.1.

Let $\bar{S}_{ni}^{(r)}$ denote the computed value of $S_{ni}^{(r)}$ and

$$\delta S_{ni}^{(r)} = \bar{S}_{ni}^{(r)} - S_{ni}^{(r)} . \tag{5.8.1}$$

Then, from (5.6.2),

$$\bar{S}_{ni}^{(r)} = fl\,(\bar{S}_{n-1,i-1}^{(r)} + x_{i-1}\bar{S}_{n-1,i-1}^{(r-1)})$$

$$= \left\{\bar{S}_{n-1,i-1}^{(r)} + x_{i-1}\bar{S}_{n-1,i-1}^{(r-1)}(1+\varepsilon_1)\right\}(1+\varepsilon_2) \tag{5.8.2}$$

$$= \bar{S}_{n-1,i-1}^{(r)}(1+2e_1) + x_{i-1}\bar{S}_{n-1,i-1}^{(r-1)}(1+2e_2), \tag{5.8.3}$$

where

$$|\varepsilon_1| \;,\; |\varepsilon_2| \leqslant 2^{-t} , \tag{5.8.4}$$

$$|e_1| \;,\; |e_2| < 2^{-t_1}. \tag{5.8.5}$$

(Actually, $|e_1| \leqslant (\frac{1}{2})2^{-t}$, but we make no use of this fact here). The expansion of (5.8.3), with the use of (5.8.1), gives

$$S_{ni}^{(r)} + \delta S_{ni}^{(r)} = S_{n-1,i-1}^{(r)} + \delta S_{n-1,i-1}^{(r)} + 2e_1\bar{S}_{n-1,i-1}^{(r)}$$

$$+ x_{i-1}\,(S_{n-1,i-1}^{(r-1)} + \delta S_{n-1,i-1}^{(r-1)} + 2e_2\bar{S}_{n-1,i-1}^{(r-1)}), \tag{5.8.6}$$

which, using (5.6.2), reduces to

$$\delta S_{ni}^{(r)} = \delta S_{n-1,i-1}^{(r)} + x_{i-1}\delta S_{n-1,i-1}^{(r-1)} + 2e_1\bar{S}_{n-1,i-1}^{(r)} + 2e_2 x_{i-1}\bar{S}_{n-1,i-1}^{(r-1)}. \tag{5.8.7}$$

It follows from (5.8.7) that

$$\left|\delta S_{ni}^{(r)}\right| \leqslant 2^{-t_1} F_{ni}^{(r)} , \tag{5.8.8}$$

where $F_{ni}^{(r)}$ is defined recursively by

$$F_{ni}^{(r)} = F_{n-1,i-1}^{(r)} + \left|x_{i-1}\right| F_{n-1,i-1}^{(r-1)} + 2\left|\bar{S}_{n-1,i-1}^{(r)}\right| + 2\left|x_{i-1}\bar{S}_{n-1,i-1}^{(r-1)}\right| . \tag{5.8.9}$$

Relation (5.8.9) can be used in a running error analysis with appropriate starting values obtained from (5.6.6) or from (5.6.7) and (5.6.8) to provide bounds on the errors in the values of the $\bar{S}_{ni}^{(r)}$ as they are computed. However, in at least two important cases we show that (5.8.9) can be solved explicitly to yield sharp <u>a priori</u> bounds for $\delta S_{ni}^{(r)}$.

The first case we consider is where the x_i are non-negative.

<u>Theorem 5.8.1</u>

If the x_i are non-negative then $\delta S_{ni}^{(r)}$ and $\delta\bar{S}_{ni}^{(r)}$ satisfy the <u>a priori</u> relative error bounds

$$\left|\delta S_{ni}^{(r)}\right| \leqslant 2.620(n-1)2^{-t} S_{ni}^{(r)} , \tag{5.8.10}$$

$$\left|\delta\bar{S}_{ni}^{(r)}\right| \leqslant 5.106(n-1)2^{-t}\bar{S}_{ni}^{(r)} . \tag{5.8.11}$$

<u>Proof</u>

We first establish by induction the result

$$F_{ni}^{(r)} \leqslant 2(n-1)(1-2^{-t})^{2-2n}\bar{S}_{ni}^{(r)} . \tag{5.8.12}$$

The non-negativity of the x_i implies from (5.6.2) the non-negativity of the $S_{ni}^{(r)}$ and from (5.8.3) of the $\bar{S}_{ni}^{(r)}$. Hence (5.8.9) becomes

$$F_{ni}^{(r)} = F_{n-1,i-1}^{(r)} + x_{i-1}F_{n-1,i-1}^{(r-1)} + 2\bar{S}_{n-1,i-1}^{(r)} + 2x_{i-1}\bar{S}_{n-1,i-1}^{(r-1)}. \tag{5.8.13}$$

Now assume that (5.8.12) holds for $n = p-1 \geqslant 1$. Then (5.8.13) yields

$$\begin{aligned} F_{pi}^{(r)} &\leqslant \left\{2(p-2)(1-2^{-t})^{4-2p} + 2\right\}(\bar{S}_{p-1,i-1}^{(r)} + x_{i-1}\bar{S}_{p-1,i-1}^{(r-1)}) \\ &\leqslant 2(p-1)(1-2^{-t})^{4-2p}(\bar{S}_{p-1,i-1}^{(r)} + x_{i-1}\bar{S}_{p-1,i-1}^{(r-1)}). \end{aligned} \tag{5.8.14}$$

But from (5.8.2),

$$\bar{S}_{p-1,i-1}^{(r)} + x_{i-1}\bar{S}_{p-1,i-1}^{(r-1)} \leqslant (1-2^{-t})^{-2}\bar{S}_{pi}^{(r)} . \tag{5.8.15}$$

Thus

$$F_{pi}^{(r)} \leqslant 2(p-1)(1-2^{-t})^{2-2p}\bar{S}_{pi}^{(r)} . \tag{5.8.16}$$

Hence (5.8.12) holds for $n = p$. But (5.8.12) is trivially true for $n = 1$. Hence by induction it is true for all $n \geqslant 1$.

It follows from (5.8.1), (5.8.8) and (5.8.12) that

$$\left|\delta S_{ni}^{(r)}\right| \leqslant K2^{-t_1}\bar{S}_{ni}^{(r)} \leqslant K2^{-t_1}(S_{ni}^{(r)} + \left|\delta S_{ni}^{(r)}\right|), \tag{5.8.17}$$

where

$$K = 2(n-1)(1-2^{-t})^{2-2n} . \tag{5.8.18}$$

Thus

$$\left|\delta S_{ni}^{(r)}\right| \leqslant \frac{K2^{-t_1}}{1-K2^{-t_1}} S_{ni}^{(r)} . \tag{5.8.19}$$

It is readily verified that the application of the results of Section 1.1 to (5.8.19) then yields (5.8.10).

Finally, the computed value of $\xi_{ni}^{(r)}$ is given by

$$\bar{\xi}_{ni}^{(r)} = \frac{\bar{S}_{ni}^{(r)}}{\bar{u}_r}(1+\varepsilon') , \tag{5.8.20}$$

where $|\varepsilon'| \leqslant 2^{-t}$. Thus, using (5.6.11), (1.1.11) and (1.1.12),

$$\bar{\xi}_{ni}^{(r)} = \frac{(S_{ni}^{(r)}+\delta S_{ni}^{(r)})(1+\varepsilon')}{u_r(1+\varepsilon)^{2r-1}} \tag{5.8.21}$$

$$= (S_{ni}^{(r)} + \delta S_{ni}^{(r)})(1+E)/u_r , \tag{5.8.22}$$

where

$$|E| \leqslant 2.224r2^{-t} . \tag{5.8.23}$$

Hence

$$\bar{\xi}_{ni}^{(r)} = \xi_{ni}^{(r)} + \frac{\delta S_{ni}^{(r)}}{u_r} + \frac{ES_{ni}^{(r)}}{u_r} , \tag{5.8.24}$$

from which

$$\delta\xi_{ni}^{(r)} = \bar{\xi}_{ni}^{(r)} - \xi_{ni}^{(r)} = \frac{\delta S_{ni}^{(r)}}{u_r}(1+E) + \frac{ES_{ni}^{(r)}}{u_r} , \tag{5.8.25}$$

giving

$$\left|\delta\xi_{ni}^{(r)}\right| \leqslant 2.620(n-1)2^{-t}\frac{S_{ni}^{(r)}}{u_r}(1+0.1) + 2.224r2^{-t}\frac{S_{ni}^{(r)}}{u_r} \tag{5.8.26}$$

$$= \left\{2.882(n-1) + 2.224r\right\}2^{-t}\xi_{ni}^{(r)} , \tag{5.8.27}$$

upon using (5.8.10), (5.8.23) and (1.1.7). The bound (5.8.11) then follows from (5.8.27) after setting r to its maximum value of n-1. □

Note that roughly half of the contribution to the error bound (5.8.11) comes from the rounding errors made while using the recurrence (5.6.2) and half from the formation of and multiplication by u_r.

We now examine the case where the x_i form a standard knot set with coincident end knots and (5.7.6) applies.

It is then apparent from (5.5.14), (5.6.1) and (5.7.7) that

$$\left|\xi_{ni}^{(r)}\right| \leqslant 1 \tag{5.8.28}$$

and

$$\left|s_{ni}^{(r)}\right| \leqslant {}^{n-1}C_r\ , \tag{5.8.29}$$

the bounds being attained for $i = 1$, $i = N+n-1$ and $r = 0$.

We consider the growth of the numbers $F_{ni}^{(r)}$ for these knot sets. If we replace computed quantities in (5.8.9) by their exact counterparts (subsequently we remove the assumption implied in this replacement), we obtain, upon using (5.8.29),

$$\begin{aligned} F_{ni}^{(r)} &\leqslant F_{n-1,i-1}^{(r)} + F_{n-1,i-1}^{(r-1)} + 2\ {}^{n-2}C_r + 2\ {}^{n-2}C_{r-1} \\ &= F_{n-1,i-1}^{(r)} + F_{n-1,i-1}^{(r-1)} + 2\ {}^{n-1}C_r\ . \end{aligned} \tag{5.8.30}$$

If we now define quantities $G_{ni}^{(r)}$ by the recursion

$$G_{ni}^{(r)} = G_{n-1,i-1}^{(r)} + G_{n-1,i-1}^{(r-1)} + 2\ {}^{n-1}C_r\ , \tag{5.8.31}$$

then

$$F_{ni}^{(r)} \leqslant G_{ni}^{(r)}\ . \tag{5.8.32}$$

Some values of $G_{ni}^{(r)}$ computed from (5.8.31) are given in the array in Fig 5.8.1.

r \ n	1	2	3	4	5	6	7
0	0	2	4	6	8	10	12
1		2	8	18	32	50	72
2			4	18	48	100	180
3				6	32	100	240
4					8	50	180
5						10	72
6							12

Fig 5.8.1. Some values of the bound $G_{ni}^{(r)}$ computed from (5.8.31).

Note that, at least for the values of n and r in Fig 5.8.1,

$$G_{ni}^{(r)} = 2(n-1)\ {}^{n-1}C_r\ , \tag{5.8.33}$$

and hence from (5.8.8), (5.8.32) and (1.1.9) that

$$\left|\delta S_{ni}^{(r)}\right| \leqslant 2.12(n-1)\ {}^{n-1}C_r 2^{-t}\ . \tag{5.8.34}$$

It is now proved rigorously in Theorem 5.8.2 that (5.8.34) is qualitatively correct and it is also shown that the sole effect of there being computed rather than exact values in (5.8.9) is to inflate the factor of 2.12 in (5.8.34) to 2.346.

Theorem 5.8.2

For a standard knot set with coincident end knots and a = -1 and b = +1, $\delta S_{ni}^{(r)}$ and $\delta \bar{S}_{ni}^{(r)}$ satisfy the a priori bounds

$$\left|\delta S_{ni}^{(r)}\right| \leqslant 2.346(n-1)\ {}^{n-1}C_r 2^{-t} , \tag{5.8.35}$$

$$\left|\delta \bar{S}_{ni}^{(r)}\right| \leqslant 4.821(n-1)2^{-t} . \tag{5.8.36}$$

Proof

We first establish the result

$$F_{ni}^{(r)} \leqslant 2 \left\{ 1+(2)2^{-t_1} \right\}^{n-1}(n-1)\ {}^{n-1}C_r , \tag{5.8.37}$$

from which (5.8.35) follows upon using (5.8.8), (1.1.9) and (1.1.11). From (5.8.9) and (5.8.1),

$$F_{pi}^{(r)} \leqslant F_{p-1,i-1}^{(r)} + 2\left|S_{p-1,i-1}^{(r)}\right| + 2\left|\delta S_{p-1,i-1}^{(r)}\right|$$

$$+ F_{p-1,i-1}^{(r-1)} + 2\left|S_{p-1,i-1}^{(r-1)}\right| + 2\left|\delta S_{p-1,i-1}^{(r-1)}\right| , \tag{5.8.38}$$

since $\left|x_{i-1}\right| \leqslant 1$. Thus, using (5.8.8),

$$F_{pi}^{(r)} \leqslant \left\{1+(2)2^{-t_1}\right\}(F_{p-1,i-1}^{(r)}+F_{p-1,i-1}^{(r-1)}) + 2\left|S_{p-1,i-1}^{(r)}\right| + 2\left|S_{p-1,i-1}^{(r-1)}\right| \tag{5.8.39}$$

$$\leqslant \left\{1+(2)2^{-t_1}\right\}(F_{p-1,i-1}^{(r)}+F_{p-1,i-1}^{(r-1)} + 2\left|S_{p-1,i-1}^{(r)}\right| + 2\left|S_{p-1,i-1}^{(r-1)}\right|). \tag{5.8.40}$$

But, using (5.8.29),

$$\left|S_{p-1,i-1}^{(r)}\right| + \left|S_{p-1,i-1}^{(r-1)}\right| \leqslant {}^{p-2}C_r + {}^{p-2}C_{r-1} = {}^{p-1}C_r \tag{5.8.41}$$

and therefore (5.8.40) yields

$$F_{pi}^{(r)} \leqslant \left\{1+(2)2^{-t_1}\right\}\left(F_{p-1,i-1}^{(r)} + F_{p-1,i-1}^{(r-1)} + 2\ {}^{p-1}C_r\right). \tag{5.8.42}$$

Now suppose that (5.8.37) is true for $n = p-1$, where $p > 1$. Then (5.8.42) gives

$$F_{pi}^{(r)} \leqslant \left\{1+(2)2^{-t_1}\right\}\left[2\left\{1+(2)2^{-t_1}\right\}^{p-2}(p-2)\ {}^{p-2}C_r\right.$$

$$\left. + 2\left\{1+(2)2^{-t_1}\right\}^{p-2}(p-2)\ {}^{p-2}C_{r-1} + 2\ {}^{p-1}C_r\right]$$

$$\leqslant \left\{1+(2)2^{-t_1}\right\}^{p-1}\left\{2(p-2)\ {}^{p-1}C_r + 2\ {}^{p-1}C_r\right\}$$

$$= 2\left\{1+(2)2^{-t_1}\right\}^{p-1}(p-1)\ {}^{p-1}C_r. \tag{5.8.43}$$

Thus (5.8.37) is true for $n = p$. But (5.8.37) is trivially true for $n = 1$ and hence by induction it is true for all $n \geqslant 1$.

The remainder of the proof follows closely the latter part of Theorem 5.8.1. It is readily established that (5.8.25) and (5.8.23) hold, from which, using (5.8.35),

$$\left|\delta\xi_{ni}^{(r)}\right| \leqslant 2.346(n-1)2^{-t}\left(1+2.24r2^{-t}\right) + 2.24r2^{-t}$$

$$\leqslant 2.346(n-1)2^{-t}(1+0.1) + 2.24r2^{-t} \tag{5.8.44}$$

which, since $r \leqslant n-1$, leads to (5.8.36). $\square$

Note, finally, that for values of n and r for which $u_r = {}^nC_r$ can be computed <u>exactly</u> using integer arithmetic, the numerical constants in (5.8.11) and (5.8.36) are reduced by factors of about one half.

5.9 The derivatives of a spline represented in B-spline form

In this section we concern ourselves with the determination in its B-spline form of the rth derivative $(0 < r < n)$ of an arbitrary spline $s(x)$ of order n defined upon a set of knots which, apart from one restriction, form a standard knot set. The restriction is that the interior knots x_i $(i = 1, 2, \ldots, N-1)$ must form an (n-r)-extended partition of (a,b). The reason for the restriction is simply that the rth derivative of $s(x)$, viz $s^{(r)}(x)$, is evidently a spline of order n-r and hence can be meaningfully defined only upon an (n-r)-extended partition.

The first derivative of (5.1.10) with respect to x gives, for $a \leq x \leq b$,

$$s'(x) = \sum_{i=1}^{N+n-1} c_i N'_{ni}(x) \tag{5.9.1}$$

which, upon applying (4.1.i) and using the restricted support of the B-splines, becomes

$$s'(x) = (n-1)\left[-c_1 \frac{N_{n-1,1}(x)}{x_1 - x_{2-n}} + \sum_{i=2}^{N+n-1} c_i \left\{\frac{N_{n-1,i-1}(x)}{x_{i-1}-x_{i-n}} - \frac{N_{n-1,i}(x)}{x_i - x_{i-n+1}}\right\}\right.$$

$$\left. + \, c_{N+n-1} \frac{N_{n-1,N+n-2}(x)}{x_{N+n-2} - x_{N-1}}\right]$$

$$= (n-1) \sum_{i=1}^{N+n-2} \left(\frac{c_{i+1} - c_i}{x_i - x_{i-n+1}}\right) N_{n-1,i}(x). \tag{5.9.2}$$

Thus

$$s'(x) = \sum_{i=1}^{N+n-2} c_i^{(1)} N_{n-1,i}(x) \qquad (a \leq x \leq b) \ , \tag{5.9.3}$$

where

$$c_i^{(1)} = \frac{(n-1)(c_{i+1}-c_i)}{x_i-x_{i-n+1}} \quad (i = 1, 2, \ldots, N+n-2). \tag{5.9.4}$$

Evidently higher derivatives of $s(x)$ may be obtained by repeated application of this process. We state, without proof, this result as a theorem.

Theorem 5.9.1

Let $s(x)$ be an arbitrary spline of order n defined upon a set of knots which form a standard knot set with the exception that the interior knots form an $(n-r)$-extended partition of (a,b). Then, for $0 \leqslant r < n$,

$$s^{(r)}(x) = \sum_{i=1}^{N+n-1-r} c_i^{(r)} N_{n-r,i}(x) \quad (a \leqslant x \leqslant b), \tag{5.9.5}$$

where the coefficients $c_i^{(r)}$ are defined recursively by

$$c_i^{(r)} = \begin{cases} c_i & (r = 0) \\ \dfrac{(n-r)(c_{i+1}^{(r-1)}-c_i^{(r-1)})}{x_i - x_{i-n+r}} & (0 < r < n). \end{cases} \tag{5.9.6}$$

Having obtained the representation (5.9.5), the $(n-r)$th-order spline $s^{(r)}(x)$ can then be evaluated as required using either Algorithm 5.2.1 or Algorithm 5.2.2.

It may sometimes be appropriate to define modified coefficients $\tilde{c}_i^{(r)}$ by

$$s^{(r)}(x) = B_{nr} \sum_{i=1}^{N+n-1-r} \tilde{c}_i^{(r)} N_{n-r,i}(x), \tag{5.9.7}$$

where

$$B_{nr} = (n-1)(n-2) \ldots (n-r). \tag{5.9.8}$$

Then, using (5.9.5) and (5.9.6),

$$c_i^{(r)} = B_{nr}\tilde{c}_i^{(r)} \tag{5.9.9}$$

and

$$\tilde{c}_i^{(r)} = \begin{cases} c_i & (r = 0) \\ \dfrac{\tilde{c}_{i+1}^{(r-1)} - \tilde{c}_i^{(r-1)}}{x_i - x_{i-n+r}} & (0 < r < n) \end{cases} \tag{5.9.10}$$

Since the factor B_{nr} can be formed exactly, at least for reasonably small values of n, the latter form has the advantage that smaller rounding errors can be expected in the computation of the $\tilde{c}_i^{(r)}$ from (5.9.10).

To conclude this section we make some observations relating to bounds on the derivatives of a spline in the case of equispaced knots. Consider the interval $x_{j-1} \leqslant x < x_j$. By analogy with (5.1.13),

$$\min_{j \leqslant i \leqslant j+n-2} c_i^{(1)} \leqslant s'(x) \leqslant \max_{j \leqslant i \leqslant j+n-2} c_i^{(1)} \tag{5.9.11}$$

which, using (5.9.4), gives

$$\min_{j \leqslant i \leqslant j+n-2} \frac{(n-1)(c_{i+1}-c_i)}{x_i - x_{i-n+1}} \leqslant s'(x) \leqslant \max_{j \leqslant i \leqslant j+n-2} \frac{(n-1)(c_{i+1}-c_i)}{x_i - x_{i-n+1}} . \tag{5.9.12}$$

For knots with constant spacing h,

$$\min_{j \leqslant i \leqslant j+n-2} (c_{i+1}-c_i) \leqslant hs'(x) \leqslant \max_{j \leqslant i \leqslant j+n-2} (c_{i+1}-c_i). \tag{5.9.13}$$

Evidently, this approach can be extended to higher derivatives. We obtain

$$\min_{j \leqslant i \leqslant j+n-3} (c_{i+2}-2c_{i+1}+c_i) \leqslant h^2 s''(x) \leqslant \max_{j \leqslant i \leqslant j+n-3} (c_{i+2}-2c_{i+1}+c_i) \tag{5.9.14}$$

and, in general, for $0 \leqslant r < n$,

$$\min_{j \leqslant i \leqslant j+n-r-1} \Delta^r c_i \leqslant h^r s^{(r)}(x) \leqslant \max_{j \leqslant i \leqslant j+n-r-1} \Delta^r c_i \qquad (5.9.15)$$

where Δ denotes the usual forward difference operator.

We note that the B-spline coefficients have an analogy with function values, since their differences (or, in the case of non-uniformly-spaced knots, derived quantities similar to divided differences) give us knowledge related to the derivatives of $s(x)$. In one respect, this information is superior to that obtained from the differences of an arbitrary function, since in that case, without further a priori or computable knowledge, no useful bounds on the derivatives can be obtained from the differences.

We do not carry out an error analysis of the recurrence (5.9.6), but content ourselves with a simple but informative numerical experiment.

Consider firstly the conversion of the polynomial power series (5.7.1) into its equivalent B-spline representation (5.1.10). For any given standard knot set this conversion can be carried out using Algorithm 5.7.1. Now suppose that for some value of $r(0 < r < n)$ the recurrence (5.9.6) is used to obtain the coefficients $c_i^{(r)}$ in the B-spline representation (5.9.5) of $s^{(r)}(x)$. The computed coefficients $\bar{c}_i^{(r)}$ will be contaminated to some extent by the inevitable floating-point arithmetical errors made. The bulk of the contribution to the error in $\bar{c}_i^{(r)}$, if $r > 1$, will be due to loss of significance when forming the differences of previously computed B-spline coefficients.

We can obtain values for the coefficients $c_i^{(r)}$ in another way that is relatively more accurate by using the explicit form of $p(x)$ in (5.7.1). We formally differentiate the power series r times to obtain

$$p^{(r)}(x) = \sum_{i=0}^{n-r-1} b_i^{(r)} x^i , \tag{5.9.16}$$

where the coefficients $b_i^{(r)}$ are defined recursively by

$$b_i^{(r)} = \begin{cases} b_i & (r = 0) \\ (i+1)b_{i+1}^{(r-1)} & (0 < r < n). \end{cases} \tag{5.9.17}$$

Evidently the values of $b_i^{(r)}$ computed from (5.9.17) will have very small relative errors. The B-spline coefficients $c_i^{(r)}$ can then be formed from these values of $b_i^{(r)}$ using Algorithm 5.7.1. Let $\hat{c}_i^{(r)}$ denote the value of $c_i^{(r)}$ computed by this latter process. We shall assume that $\hat{c}_i^{(r)}$ is a relatively good estimate of the true value $c_i^{(r)}$. It is readily established using the error analyses of Section 5.8 that this assumption is well justified.

Let

$$\bar{s}^{(r)}(x) = \sum_{i=1}^{N+n-1-r} \bar{c}_i^{(r)} N_{n-r,i}(x) \tag{5.9.18}$$

and

$$\hat{s}^{(r)}(x) = \sum_{i=1}^{N+n-1-r} \hat{c}_i^{(r)} N_{n-r,i}(x) . \tag{5.9.19}$$

Then the error $\delta s^{(r)}(x)$ in the rth derivative of $s(x)$ due to using the inaccurate coefficients $\bar{c}_i^{(r)}$ is

$$\delta s^{(r)}(x) = \bar{s}^{(r)}(x)-s^{(r)}(x) = \left\{\bar{s}^{(r)}(x)-\hat{s}^{(r)}(x)\right\} + \left\{\hat{s}^{(r)}(x)-s^{(r)}(x)\right\}. \tag{5.9.20}$$

In accordance with the above assumption we ignore the term $\hat{s}^{(r)}(x)-s^{(r)}(x)$ and obtain

$$\delta s^{(r)}(x) = \bar{s}^{(r)}(x) - \hat{s}^{(r)}(x)$$

$$= \sum_{i=1}^{N+n-1-r} (\bar{c}_i^{(r)} - \hat{c}_i^{(r)}) N_{n-r,i}(x) \tag{5.9.21}$$

and hence, as a consequence of (5.1.13),

$$\left|\delta s^{(r)}(x)\right| \leqslant \max_i \left|\bar{c}_i^{(r)} - \hat{c}_i^{(r)}\right| . \tag{5.9.22}$$

Thus, by evaluating the $\bar{c}_i^{(r)}$ and $\hat{c}_i^{(r)}$ as described and using (5.9.22) the required error bound is obtained.

Although a bound obtained in this way is of considerable interest, it is more valuable to compare its value with a bound for a well-established process. We therefore consider a process analagous to the above in which we employ Chebyshev polynomials instead of B-splines.

Firstly we convert the representation (5.7.1) into its equivalent Chebyshev-series form

$$p(x) = \sum_{i=0}^{n-1}{}' a_i T_i(x), \tag{5.9.23}$$

where $T_i(x)$ is the Chebyshev polynomial of the first kind of degree i in x, the prime on the summation symbol denotes that the constant term in (5.9.23) is to be taken with weight one-half and, for simplicity, it is assumed that the range over which $p(x)$ and $s(x)$ are defined is $[-1, +1]$. This conversion can be carried in an extremely stable manner (see Cox, 1974, and also Section 5.11), with expected errors of similar magnitude to those in obtaining the B-spline form. The coefficients $a_i^{(r)}$ in the representation

$$p^{(r)}(x) = \sum_{i=0}^{n-r-1}{}' a_i^{(r)} T_i(x) \tag{5.9.24}$$

can be determined by the recursion

$$a_i^{(r)} = \begin{cases} a_i & (r = 0) \\ a_{i+2}^{(r)} + 2(i+1)a_{i+1}^{(r-1)} & (0 < r < n) \end{cases}, \qquad (5.9.25)$$

given by Clenshaw (1962). All undefined terms in (5.9.25) are to be regarded as zero. Let the values of $a_i^{(r)}$ computed in this manner be denoted by $\bar{a}_i^{(r)}$.

As with the B-spline coefficients we can obtain good values of the $a_i^{(r)}$ by evaluating the Chebyshev coefficients directly from the form (5.9.16). We denote the computed values of these coefficients by $\hat{a}_i^{(r)}$. Now let

$$\bar{p}^{(r)}(x) = \sum_{i=0}^{n-r-1}{}' \bar{a}_i^{(r)} T_i(x) \qquad (5.9.26)$$

and

$$\hat{p}^{(r)}(x) = \sum_{i=0}^{n-r-1}{}' \hat{a}_i^{(r)} T_i(x). \qquad (5.9.27)$$

Then we define

$$\delta p^{(r)}(x) = \bar{p}^{(r)}(x) - p^{(r)}(x) = \left\{\bar{p}^{(r)}(x) - \hat{p}^{(r)}(x)\right\} + \left\{\hat{p}^{(r)}(x) - p^{(r)}(x)\right\}.$$

Again we can readily justify that the term $\hat{p}^{(r)}(x) - p^{(r)}(x)$ may safely be ignored and hence essentially

$$\begin{aligned} \delta p^{(r)}(x) &= \bar{p}^{(r)}(x) - \hat{p}^{(r)}(x) \\ &= \sum_{i=0}^{n-r-1}{}' (\bar{a}_i^{(r)} - \hat{a}_i^{(r)}) T_i(x). \end{aligned} \qquad (5.9.28)$$

Since $|T_i(x)| \leq 1$ for $|x| \leq 1$, we obtain

$$\left|\delta p^{(r)}(x)\right| \leq \sum_{i=0}^{n-r-1}{}' \left|\bar{a}_i^{(r)} - \hat{a}_i^{(r)}\right|. \qquad (5.9.29)$$

We now give some numerical results. We set $b_i = 1/i!$ for $i = 0, 1, \ldots, n-1 = 13$. Thus $p(x)$ is the Taylor expansion of e^x about the origin, truncated after 14 terms. This choice of coefficients b_i has the advantage that $b_i^{(r)} = b_i$ and hence has no further error. We also set $N = 1$, $x_i = -1$ $(i \leqslant 0)$ and $x_i = +1$ $(i > 0)$. For $r = 1, 2, \ldots, 6$ the values of $\bar{c}_i^{(r)}$, $\hat{c}_i^{(r)}$, $\bar{a}_i^{(r)}$ and $\hat{a}_i^{(r)}$ were computed as described above and the bounds (5.9.22) and (5.9.29) formed. The bounds are given in Table 5.9.1.

	Polynomial representation	
	B-spline form	Chebyshev series
r	The bound (5.9.22)	The bound (5.9.29)
1	$1.27329_{10}{-11}$	$1.38372_{10}{-10}$
2	$1.47338_{10}{-10}$	$8.31460_{10}{-9}$
3	$1.45246_{10}{-9}$	$2.56133_{10}{-7}$
4	$1.25247_{10}{-8}$	$5.22095_{10}{-6}$
5	$1.08547_{10}{-7}$	$7.84445_{10}{-5}$
6	$7.58009_{10}{-7}$	$9.10534_{10}{-4}$

Table 5.9.1 A comparison of bounds for the errors in the rth derivative of a polynomial expressed in its Chebyshev series form and in its B-spline form.

It is seen that the bound (5.9.29) compares favourably with (5.9.22). Other experiments were also carried out, but Table 5.9.1 typifies the results obtained.

5.10 The indefinite integral of a spline represented in B-spline form

We consider the inverse process of that of Section 5.9, viz the determination in its B-spline form of the r-fold indefinite integral ($r > 0$) of an arbitrary spline $s(x)$ of order n defined upon a standard knot set. There is no further restriction on the knot set, as in Section 5.9, however.

Theorem 5.10.1

Let $s(x)$ be an arbitrary spline of order n defined upon a standard knot set and have the representation (5.1.10). Then the r-fold indefinite integral $s^{(-r)}(x)$ of $s(x)$ is given by

$$s^{(-r)}(x) = \underbrace{\int\int\cdots\int}_{r} s(x)\underbrace{dx\,dx\ldots dx}_{r} = \sum_{i=1}^{N+n-1+r} c_i^{(-r)} M_{n+r,i}(x) + \sum_{j=1}^{r} \frac{k_j}{(r-j)!}\, x^{r-j} , \tag{5.10.1}$$

where $c_i^{(-r)}$ is defined recursively by

$$(n+r-1)c_i^{(-r)} = \begin{cases} 0 & (i \leq r) \\ c_{i-1}^{(-r)} + (x_{i-1} - x_{i-n-r})c_{i-1}^{(1-r)} & (i > r) \end{cases} \tag{5.10.2}$$

and k_j $(j = 1, 2, \ldots, r)$ are arbitrary constants.

Proof

It is sufficient to establish that, for arbitrary $r > 0$, a single differentiation of the right-most expression in (5.10.1), after the inclusion of an arbitrary additive constant, yields an expression of similar form with r replaced by r-1. Accordingly, upon employing Theorem 5.9.1, we obtain, as the first derivative of this expression,

$$(n+r-1)\sum_{i=1}^{N+n-2+r}\frac{(c_{i+1}^{(-r)}-c_i^{(-r)})}{x_i-x_{i-n+1-r}}N_{n+r-1,i}(x)+\sum_{j=1}^{r-1}\frac{k_j}{(r-j-1)!}x^{r-j-1} \qquad (5.10.3)$$

which, using (5.10.2), reduces to

$$\sum_{i=1}^{N+n-2+r}c_i^{(1-r)}N_{n+r-1,i}(x)+\sum_{j=1}^{r-1}\frac{k_j}{(r-j-1)!}x^{r-j-1}\;. \qquad \square \qquad (5.10.4)$$

It may be somewhat inconvenient in some applications to work with the expression (5.10.1) since it involves both B-splines and powers of x. However, for any particular choice of the constants k_j ($j = 1, 2, \ldots, r$), the power-series part of (5.10.1) may be converted into a linear combination of the B-splines $N_{n+r,i}(x)$ ($i = 1, 2, \ldots, N+n-1+r$) using the method of Section 5.7 (and, in particular, Algorithm 5.7.1). Thus we obtain the representation

$$s^{(-r)}(x)=\sum_{i=1}^{N+n-1+r}c_i^{(-r)}N_{n+r,i}(x)\;, \qquad (5.10.5)$$

where the $c_i^{(-r)}$ now include the contributions from the power-series terms. In common with most integration processes involving summation, it can be expected that the use of recurrence (5.10.2), in which the difference $x_{i-1} - x_{i-n-r}$ can be formed with a very small relative error (cf Section 1.1), will give rise to a stable algorithm for forming the coefficients $c_i^{(-r)}$.

We conclude this section with an expression for the definite integral of s(x) over the range (a, b). If s(x) is a spline of order n defined upon a standard knot set with coincident end knots then each of the B-splines $N_{ni}(x)$ ($i = 1, 2, \ldots, N+n-1$) is identically zero outside of the interval $[a, b]$. Consequently,

$$\int_a^b s(x)dx = \sum_{i=1}^{N+n-1} c_i \int_a^b N_{ni}(x)dx$$

$$= \sum_{i=1}^{N+n-1} c_i \int_{-\infty}^{\infty} N_{ni}(x)dx . \qquad (5.10.6)$$

Hence, using (4.5.9),

$$\int_a^b s(x)dx = \frac{1}{n} \sum_{i=1}^{N+n-1} (x_i - x_{i-n})c_i \qquad (5.10.7)$$

or, in terms of the coefficients of the un-normalized B-spline representation (5.1.11),

$$\int_a^b s(x)dx = \frac{1}{n} \sum_{i=1}^{N+n-1} c_i^* . \qquad (5.10.8)$$

Thus, having obtained the coefficients in a B-spline representation of $s(x)$, it is a very simple matter to determine the value of the definite integral. For example, $s(x)$ may be an interpolatory or a least-squares approximation to data representative of a function $f(x)$ (Chapters 6, 7 and 8), in which case (5.10.7) or (5.10.8) will then provide an estimate of $\int_a^b f(x)dx$.

5.11 Representation in piecewise-Chebyshev-series form

The representation (5.1.10) is satisfactory for many purposes in that only $N+n-1$ coefficients (the smallest possible number in general) are required in its definition, and about $\frac{3}{2}n^2$ long operations in its evaluation for a prescribed argument x. If an increased number of linear coefficients, viz Nn, to define the spline, can be tolerated then, at the expense of some pre-computation, its subsequent evaluation may be carried out for a given argument in about n long operations. In order to obtain such a representation the following approach is recommended.

In each interval $x_{j-1} \leqslant x \leqslant x_j$ ($j = 1, 2, \ldots, N$) for which $x_{j-1} < x_j$, $s(x)$ is a polynomial of degree $n-1$ and hence may be expressed in the Chebyshev-series form,

$$s(x) \equiv s_j(X) = \sum_{i=0}^{n-1}{}'' \alpha_{ji} T_i(X), \tag{5.11.1}$$

where

$$X = (2x - x_{j-1} - x_j)/(x_j - x_{j-1}). \tag{5.11.2}$$

In (5.11.1), $T_i(X)$ is the Chebyshev polynomial of the first kind of degree i in X, and the double prime indicates that in the summation the first and last terms are to be halved. The linear transformation (5.11.2) maps each interval $x_{j-1} \leqslant x \leqslant x_j$ into the interval $-1 \leqslant X \leqslant 1$. This representation has also been used in the allied context of curve fitting with piecewise polynomials (Cox, 1971). For completeness, in the case $x_{j-1} = x_j$, we define $\alpha_{jo} = 2s(x_j-)$ and $\alpha_{ji} = 0$ ($i = 1, 2, \ldots, n-1$).

In order to obtain the values of the coefficients α_{ji} in (5.11.1) we may utilize the fact that $s_j(X)$ is a polynomial of degree $n-1$ in X, and hence (Clenshaw, 1962),

$$\alpha_{ji} = \frac{2}{n-1} \sum_{k=0}^{n-1}{}'' \cos\left(\frac{\pi ik}{n-1}\right) s_j\left(\cos\left(\frac{\pi k}{n-1}\right)\right) \quad (i = 0, 1, \ldots, n-1) \tag{5.11.3}$$

The values of $s_j\left(\cos\left(\frac{\pi k}{n-1}\right)\right)$ required in (5.11.3) are conveniently calculated using Algorithm 5.2.1 or Algorithm 5.2.2 for computing a spline from its B-spline representation.

The evaluation of the α_{ji}, as well as the subsequent evaluation of $s_j(X)$ for a prescribed argument X, can be carried out using the scheme for summing a Chebyshev series (very slightly modified to accomodate the halving of the last term in both (5.11.1) and (5.11.3)) due to Clenshaw (1962).

Greater stability in these computations can be achieved if the Reinsch-Gentleman modification of Clenshaw's scheme (see Gentleman, 1969) is used or, alternatively, plane rotations are employed. For full details see Cox (1974).

An important aspect of the computation is the linear transformation (5.11.2). In Section 1.2 it was shown that if X were computed from an unsatisfactory representation the error in X depends upon the value of $|x_j|/(x_j-x_{j-1})$. For highly non-uniformly spaced knots, or if both x_{j-1} and x_j are far-removed from the origin (compared with the interval length x_j-x_{j-1}), this ratio may well be very large. Of course, if N is large, there will inevitably be values of j for which $|x_j| \gg x_j-x_{j-1}$. It follows that one of the stable forms, eg

$$X = \left\{(x-x_{j-1}) - (x_j-x)\right\}/(x_j-x_{j-1}) \tag{5.11.4}$$

should be employed.

CHAPTER 6

SPLINE INTERPOLATION

In many problems involving computations with splines the choice of representation of the spline is of the utmost importance; the spline-interpolation problem is no exception. Evidently there are many possible sets of basis functions in terms of which the spline can be expressed. For any particular set there are three main stages in the spline interpolation problem: (i) the formation of the system of linear equations defining the coefficients of the basis functions, (ii), the solution of this linear system, and (iii) the numerical evaluation of the interpolating spline at various values of the argument. In stages (i) and (iii) it is of course necessary to compute accurately the values of the basis functions at various points. There are in existence excellent methods for stage (ii) (see, for example, Wilkinson, 1965 and Wilkinson and Reinsch, 1971), although for maximum efficiency these methods need to be tailored to take full advantage of the structure of the linear systems. The accuracy to which these methods can deliver the desired coefficients depends upon the numerical conditioning of the system, and therefore upon the choice of basis functions.

In Section 6.1 the spline interpolation problem is defined. In Section 6.2 a method of forming the linear system defining the coefficients is given, and the solution of this system is discussed in Section 6.3. In Section 6.4 algorithms for the solution of the problem are presented. In Section 6.5 it is shown that one of these algorithms yields a solution that is an exact interpolant for data function values close to those given, and computable a posteriori measures of this closeness are derived. In Section 6.6 a brief discussion of the method when applied to splines with multiple knots is given. In Sections 6.7 and 6.8 the choices of exterior and interior knots are discussed and in Section 6.9 some numerical examples are presented. Parts of this chapter appear also in Cox (1975).

6.1 The spline interpolation problem

The problem of concern may be stated as follows:

Interpolate function values $f(x)$ at the points $x = t_1, t_2, \ldots, t_m$ by a spline $s(x)$ of order n (degree $n-1$) with prescribed (interior) knots $x_1, x_2, \ldots, x_{N-1}$.

Let $a = t_1$ and $b = t_m$. It is assumed that

$$t_1 < t_2 < \ldots < t_m \qquad (6.1.1)$$

and that the interior knots form an n-extended partition of (a,b). We choose additional knots $x_i = a$ $(i \leqslant 0)$ and $x_i = b$ $(i \geqslant N)$ in accordance with the definition of a standard knot set with coincident end knots (Section 3.1). The $N+n-1$ free linear parameters of $s(x)$ are to be determined such that the conditions

$$s(t_j) = f_j \quad (j = 1, 2, \ldots, m), \qquad (6.1.2)$$

where $f_j = f(t_j)$, are satisfied. To guarantee the possibility of a unique interpolation for arbitrarily-prescribed function values, it is manifestly required that

$$m = N+n-1 . \qquad (6.1.3)$$

Schoenberg and Whitney (1953) have shown that a unique solution exists if and only if the inequalities

$$\left.\begin{array}{l} t_1 < x_1 < t_{1+n} , \\ t_2 < x_2 < t_{2+n} , \\ \cdots\cdots\cdots\cdots\cdots \\ t_{N-1} < x_{N-1} < t_m \end{array}\right\} \qquad (6.1.4)$$

are satisfied. It is therefore assumed henceforth that conditions (6.1.3) and (6.1.4) hold. We refer to (6.1.4) subsequently as the Schoenberg-Whitney conditions.

The main interest in this interpolation problem is that no additional information relating to end derivatives, such as with a natural interpolating spline, is required. The spline is treated simply as a conventional interpolating function (such as a polynomial, a rational function or a trigonometric series), but advantage is taken of the particular structure of the problem in order to yield an efficient algorithm. Schumaker (1969) remarks that this particular interpolation problem has been all but neglected.

This approach to spline interpolation has the additional advantage that it has considerable approximating power since it enables arbitrary polynomials of degree n-1 to be reproduced exactly. This property is not shared by natural splines which, if of order n = 2k, can reproduce only polynomials of degree less than k (Greville, 1969). Neither is the property shared by spline interpolation with derivative end conditions, unless it is possible to provide the exact values of the required derivatives.

6.2 The linear system: formation

For any given set of knots, s(x) can be expressed in the form

$$s(x) = \sum_{i=1}^{m} c_i \phi_i(x) , \qquad (6.2.1)$$

where the $\phi_i(x)$ $(i = 1, 2, \ldots, m)$ form a linearly independent set of basis functions, each of which is itself a spline function of order n with interior knots $x_1, x_2, \ldots, x_{N-1}$. $s(x)$ cannot in general be expressed in terms of fewer than m such functions.

Having selected appropriate basis functions $\phi_i(x)$, the coefficients c_i are given by the solution of the following system of linear algebraic equations,

$$\sum_{i=1}^{m} c_i \phi_i(t_j) = f_j \quad (j = 1, 2, \ldots, m). \tag{6.2.2}$$

The linear independence of the functions $\phi_i(x)$, together with the satisfaction of the Schoenberg-Whitney conditions (6.1.4), ensures the existence of a unique solution to (6.2.2). The system can be expressed in an obvious matrix notation as

$$\underset{\sim}{A}\,\underset{\sim}{c} = \underset{\sim}{f}\,, \tag{6.2.3}$$

where the element in position (i,j) of the m by m matrix $\underset{\sim}{A}$ is $a_{ij} = \phi_j(t_i)$.

The B-splines, which we intend to employ as a basis, are particularly advantageous in that the linear system defining the spline coefficients can be formed and solved extremely efficiently and, moreover, in a numerically stable manner. We give some details of the arithmetic work required to set up and solve the system at the end of this section and in Section 6.4. A discussion of the numerical stability is given in Section 6.5.

As shown in Section 5.1, the spline can be expressed in the form (5.1.10) or (5.1.11). Consequences of the choice of coincident end knots are that the full set of m B-splines is identically zero outside the range $a \leq x \leq b$ and that the definite integral of $s(x)$ over the data range (a,b) can be computed very easily and used as an estimate of $\int_a^b f(x)dx$ once the B-spline coefficients have been determined (cf Section 5.10).

By putting

$$\phi_i(x) \equiv N_{ni}(x) , \tag{6.2.4}$$

the system (6.2.2) becomes

$$\sum_{i=1}^{m} c_i N_{ni}(t_j) = f_j \quad (j = 1, 2, \ldots, m). \tag{6.2.5}$$

Because of the restricted support property of the B-splines it follows that the coefficient matrix defined by equations (6.2.5) contains at most n non-zero elements in each row, and that the column position of the first non-zero element in each row is a non-decreasing function of the row number. Thus $\underset{\sim}{A}$ is a stepped-banded matrix (Section 2.11) of bandwidth n. For instance, if N=6 and n=4, the matrix $\underset{\sim}{A}$ obtained from data and knots satisfying the conditions,

$$a = t_1 < t_2 < x_1 < t_3 < x_2 < x_3 < t_4 < t_5$$
$$< t_6 < x_4 < t_7 < t_8 < x_5 < t_9 = b, \tag{6.2.6}$$

takes the form

$$A = \begin{bmatrix} X & & & & & & & 0 \\ X & X & X & X & & & & \\ & X & X & X & X & & & \\ & & X & X & X & X & & \\ & & X & X & X & X & & \\ & & X & X & X & X & & \\ & & & X & X & X & X & \\ & & & X & X & X & X & \\ 0 & & & & & & & X \end{bmatrix} . \tag{6.2.7}$$

In (6.2.7) non-zero entries are denoted by X. It is straightforward to verify that data and knots disposed as in (6.2.6) satisfy the Schoenberg-Whitney conditions (6.1.4). The presence of only one non-zero element in the first and last rows of $\underset{\sim}{A}$ is a further consequence of the choice of coincident end knots. As a result of this choice and of (3.6.1) the respective values of c_1 and c_m (= c_9 in this example) are simply f_1 and f_m. A further feature of the matrix $\underset{\sim}{A}$ is that it is nicely balanced for computational purposes in the following sense. The maximum element in each row is bounded from above by unity, as a consequence of relation (3.6.1), and from below by $1/n$, as a result of Theorem 3.6.2.

6.3 The linear system: solution

The solution of the system (6.2.5) can be achieved efficiently if advantage is taken of the stepped-banded structure of $\underset{\sim}{A}$. For instance, either Algorithm 2.12.1 based upon Gaussian elimination or Algorithm 2.13.1 which uses elementary transformations may be used. Alternatively, since $\underset{\sim}{A}$ can be regarded as a band matrix with n-1 super-diagonals and n-1 sub-diagonals, a result which is a consequence of the Schoenberg-Whitney conditions (6.1.4), a standard algorithm (eg Martin and Wilkinson, 1967) for solving band systems can be employed. With the last-mentioned approach some loss of efficiency can be expected, since advantage is not taken of zero elements within the band. In fact at least $(m-n)(n-1)$ of the total number of $(2m-n)(n-1)+m$ elements in the band are zero.

6.4 Algorithms for the spline interpolation problem

Algorithms 6.4.1 and 6.4.2 are implementations of the method described in the earlier sections of this chapter. The algorithms allow either coincident or non-coincident end knots to be chosen. The former choice is usually to be preferred for the reasons given in Section 6.2, as well as for the stability considerations discussed in Section 6.7. However, the latter choice may well be more appropriate if the knots x_i ($i = 0, 1, \ldots, N$) are at a constant spacing, h, say, since, if the exterior knots are chosen such that the complete set of knots is at the spacing h, the B-splines so defined are simply linear translations of each other.

The first five steps of either algorithm constitute checks on the data, the computation being terminated if any of the five checks is violated. (For simplicity of presentation there is an element of redundancy in these checks). In Algorithm 6.4.1, the stepped-banded system (6.2.3) is formed using Algorithm 3.12.2 to compute the values of the normalized B-splines for each of the m data points. In accordance with the requirements of Algorithm 2.12.1, which is then used to solve the system using Gaussian elimination, the matrix $\underset{\sim}{A}$ is stored in condensed form as an m by n array with the vector $\underset{\sim}{p}$ holding the row numbers that terminate each block. In Algorithm 6.4.2, which makes direct use of Algorithm 2.13.1 for solving stepped-banded systems by elementary transformations, the ith row ($i, 1, 2, \ldots, m$) of $\underset{\sim}{A}$ is formed as required by the latter algorithm, using Algorithm 3.12.2 to evaluate the non-zero B-splines at $x = t_i$. There is little to choose between Algorithm 6.4.1 and Algorithm 6.4.2 in terms of speed or storage requirements and, in our experience, in terms of stability. An error analysis of Algorithm 6.4.1 is given in Section 6.5.

It is assumed that values of m and n, data points (t_i, f_i) ($i = 1, 2, \ldots, m$) and knots x_i ($i = 1-n, 2-n, \ldots, m = N+n-1$) are supplied to the algorithms. The last data point is always chosen to lie within the interval x_{N-1} to x_N. This choice, together with the following minor modification to Algorithm 3.12.2 for computing the normalized B-splines, is necessary to ensure that the appropriate B-spline values are properly defined in the case $t_m = x_N$: Replace Step 1 of Algorithm 3.12.2 by

Step 1. If $x < x_N$ determine the unique integer l such that $x_{l-1} \leq x < x_l$; otherwise set $l = N$.

Algorithm 6.4.1: Data interpolation by a spline of order n using normalized B-splines and Gaussian elimination.

Comment: Check whether there are sufficient data points for the prescribed order of the spline.

Step 1. Finish if the inequality $m \geqslant n$ is violated.

Comment: Check whether the complete set of knots is ordered.

Step 2. Finish if the inequalities $x_{1-n} \leqslant x_{2-n} \leqslant \dots \leqslant x_m$ are not all satisfied.

Comment: Check whether the complete set of knots forms an n-extended partition.

Step 3. Finish if the inequalities $x_{i-n} < x_i$ $(i = 1, 2, \dots, m)$ are not all satisfied.

Comment: Check whether the data abscissae are strictly ordered and lie within the range $[a, b] = [x_0, x_N]$.

Step 4. Finish if the inequalities $x_0 \leqslant t_1 < t_2 < \dots < t_{m-1} < t_m \leqslant x_N$ are not all satisfied.

Comment: Check whether the Schoenberg-Whitney conditions are satisfied.

Step 5. Finish if the inequalities $t_i < x_i < t_{i+n}$ $(i = 1, 2, \dots, N-1)$ are not all satisfied.

Comment: l denotes the number of the current interval.

Step 6. Set $l = 0$ and $p_N = m$.

Comment: The ith data point is processed in Steps 8-12.

Step 7. For $i = 1, 2, \dots, m$ execute Steps 8-12.

Comment: The interval containing t_i is located in Steps 8-10.

Step 8. If $t_i < x_l$ or $l = N$ advance to Step 11.

Step 9. Set $p_l = i-1$.

Step 10. Replace l by l+1 and return to Step 8.

Step 11. Use Algorithm 3.12.2 with $x = t_i$ to form the values of $N_{nj}(t_i)$ $(j = l, l+1, \dots, l+n-1)$.

Comment: Store the B-spline values for $x = t_i$ in row i of A.

Step 12. For $j = l, l+1, \ldots, l+n-1$ set $a_{i,j-l+1} = N_{nj}(t_i)$.

Comment: The B-spline coefficients are computed.

Step 13. Use Algorithm 2.12.1 to solve the stepped-banded system $\underset{\sim}{A}\underset{\sim}{c} = \underset{\sim}{f}$.

<u>Algorithm 6.4.2:</u> Data interpolation by a spline of order n using normalized B-splines and elementary transformations.

Comment: Check the data as in Algorithm 6.4.1.

Step 1. As Steps 1-5 of Algorithm 6.4.1.

Comment: k is the interval number as well as the number of the block currently being processed.

Step 1.1 Set $k = 1$.

Comment: Initialize $\underset{\sim}{R}$ and $\underset{\sim}{Q}$ to zero.

Steps 2-4. As Steps 2-4 of Algorithm 2.13.1 (with n interpreted as m and q as n).

Comment: Computations involving the ith data point are described by Steps 6-31.

Step 5. For $i = 1, 2, \ldots, m$ execute Steps 6-31.

Comment: The interval containing t_i is located in Steps 6-7.

Step 6. If $t_i < x_k$ or $k = N$ advance to Step 8.

Step 7. Replace k by k+1 and return to Step 6.

Comment: The ith row of $(\underset{\sim}{A}|\underset{\sim}{b})$, as required by Algorithm 2.13.1, is formed in Steps 8-8.2.

Step 8. Use Algorithm 3.12.2 with $x = t_i$ to form the values of $N_{nj}(t_i)$ $(j = k, k+1, \ldots, k+n-1)$.

Step 8.1. For $j = 1, 2, \ldots, n$ set $v_j = N_{n,k+j-1}(t_i)$.

Step 8.2. Set $u = f_i$.

Comment: Elementary transformations to annihilate the elements in row i of $\underset{\sim}{A}$ are applied in Steps 9-31.

Steps 9-31. As Steps 9-31 of Algorithm 2.13.1 (with n interpreted as m and q as n).

Step 32. Use Algorithm 2.1.4 to solve $\underset{\sim}{R}\underset{\sim}{c} = \underset{\sim}{\theta}$.

The computational work in Algorithms 6.4.1 and 6.4.2 is dominated by the formation of $\underset{\sim}{A}$, which takes about $\frac{3}{2}mn^2$ long operations, and the solution of $\underset{\sim}{A}\underset{\sim}{c} = \underset{\sim}{f}$, which takes about $\frac{1}{2}mn^2$ long operations (or, if $\underset{\sim}{A}$ is regarded as a uniformly banded matrix, about $2mn^2$ long operations). Consequently, the complete process takes about $2mn^2$ (or $\frac{7}{2}mn^2$) long operations. In particular, for a given order of spline, the computational work is directly proportional to m, the number of points of interpolation. Note that, if the basis functions $\phi_i(x)$ in (6.2.1) are not of compact support, the number of long operations required to solve the linear system alone is proportional to m^3.

As regards the subsequent numerical evaluation of the interpolating spline, the use of Algorithm 5.2.1 or Algorithm 5.2.2 enables s(x) to be evaluated for any particular value of x in about $\frac{3}{2}n^2$ long operations. However, if a representation of s(x) possessing a greater number of defining parameters is acceptable then, at the expense of some pre-computation, this number of operations can be reduced to about n by using the equivalent piecewise Chebyshev-series representation (Section 5.11).

6.5 Error analysis

We now give an error analysis of the formation and solution of the system (6.2.5) in the case where the matrix is regarded as a band of width 2n-1 centred upon the main diagonal and Gaussian elimination with partial pivoting is employed. It is assumed that all computations are carried out in single-length floating-point arithmetic with a mantissa of t binary digits and that the rounding rules (1.1.2), (1.1.3) and (1.1.4) apply.

Following Wilkinson (1963: p107) the solution of

$$\underset{\sim}{A}\underset{\sim}{c} = \underset{\sim}{f} \tag{6.5.1}$$

is reduced, using Gaussian elimination with partial pivoting, to that of

$$\underset{\sim}{L}\underset{\sim}{U}\underset{\sim}{c} = \underset{\sim}{f}. \tag{6.5.2}$$

where the computed $\underset{\sim}{L}$ and $\underset{\sim}{U}$ satisfy

$$\underset{\sim}{L}\underset{\sim}{U} = \underset{\sim}{A} + \underset{\sim}{E}. \tag{6.5.3}$$

The computed solution is then obtained by solving two triangular sets of equations and in practice we obtain $\underset{\sim}{d}$ and $\underset{\sim}{c}$ defined by

$$(\underset{\sim}{L} + \delta\underset{\sim}{L})\, \underset{\sim}{d} = \underset{\sim}{f}\ , \tag{6.5.4}$$

$$(\underset{\sim}{U} + \delta\underset{\sim}{U})\, \underset{\sim}{c} = \underset{\sim}{d}\ . \tag{6.5.5}$$

Hence $\underset{\sim}{c}$ satisfies

$$\begin{aligned}(\underset{\sim}{L} + \delta\underset{\sim}{L})\, \underset{\sim}{d} &= (\underset{\sim}{L} + \delta\underset{\sim}{L})\,(\underset{\sim}{U} + \delta\underset{\sim}{U})\, \underset{\sim}{c} \\ &= (\underset{\sim}{A} + \underset{\sim}{E} + \underset{\sim}{L}\delta\underset{\sim}{U} + \delta\underset{\sim}{L}\underset{\sim}{U} + \delta\underset{\sim}{L}\,\delta\underset{\sim}{U})\, \underset{\sim}{c} = \underset{\sim}{f}.\end{aligned} \tag{6.5.6}$$

Note that here and elsewhere in this section $\underset{\sim}{c}$ is used to denote the <u>computed</u> B-spline coefficients. For the case where $\underset{\sim}{A}$ is a full n by n matrix, bounds for $\|\underset{\sim}{E}\|_\infty$, $\|\delta\underset{\sim}{L}\|_\infty$ and $\|\delta\underset{\sim}{U}\|_\infty$ have been given by Wilkinson (1963: p 108). For the case where $\underset{\sim}{A}$ is a banded matrix with n-1 super-diagonals and n-1 sub-diagonals, Martin and Wilkinson (1967) give the bound*

$$\|\underset{\sim}{E}\|_\infty \leqslant g(2n-1)2^{-t}, \tag{6.5.7}$$

---

* Since the preparation of this work I have learned in discussion with Dr J H Wilkinson that the bound (6.5.7) is not an upper (ie rigorous) bound as stated in Martin and Wilkinson (1967). Rather, the "bound" is such that it is <u>unlikely</u> to be exceeded in practice. The remainder of this section should be read with this qualification in mind.

where g is the largest element (disregarding sign) that arises in $\underset{\sim}{A}$ at any stage of its reduction to $\underset{\sim}{LU}$ form. The analysis of Wilkinson (1963: p 99 et seq) for the solution of triangular systems is easily extended to band triangular systems. We merely quote the relevant results here. $\underset{\sim}{L}$ is lower band triangular of bandwidth n with all elements bounded in modulus by unity, as a result of the partial pivoting strategy. $\underset{\sim}{U}$ is upper band triangular of bandwidth 2n-1 as a consequence of the row interchanges during the reduction. The elements of $\underset{\sim}{U}$ are bounded in modulus by g. It is easily established that

$$\|\underset{\sim}{L}\|_\infty \leq n, \tag{6.5.8}$$

$$\|\underset{\sim}{U}\|_\infty \leq g(2n-1), \tag{6.5.9}$$

$$\|\delta\underset{\sim}{L}\|_\infty \leq \tfrac{1}{2}n(n+1)2^{-t_1}, \tag{6.5.10}$$

$$\|\delta\underset{\sim}{U}\|_\infty \leq gn(2n-1)2^{-t_1}, \tag{6.5.11}$$

where 2^{-t_1} is defined by (1.1.9).

Hence, writing (6.5.6) in the form

$$(\underset{\sim}{A} + \underset{\sim}{K})\, \underset{\sim}{c} = \underset{\sim}{f}\ , \tag{6.5.12}$$

we have

$$\|\underset{\sim}{K}\|_\infty \leq \|\underset{\sim}{E}\|_\infty + \|\underset{\sim}{L}\|_\infty \|\delta\underset{\sim}{U}\|_\infty + \|\delta\underset{\sim}{L}\|_\infty \|\underset{\sim}{U}\|_\infty + \|\delta\underset{\sim}{L}\|_\infty \|\delta\underset{\sim}{U}\|_\infty$$

$$\leq g\left\{(2n-1) + n^2(2n-1) + \tfrac{1}{2}n(n+1)(2n-1) + \tfrac{1}{2}n^2(n+1)(2n-1)2^{-t_1}\right\}2^{-t_1}, \tag{6.5.13}$$

Again making use of (1.1.9) and bounding the term $\frac{1}{2}n^2(n+1)(2n-1)(1.06)2^{-t}$ by 0.106 in accordance with (1.1.10) yields

$$\|\underset{\sim}{K}\|_\infty \leq g(3n^3 - \tfrac{1}{2}n^2 + \tfrac{3}{2}n - 0.894)2^{-t_1}\ . \tag{6.5.14}$$

However, $\underset{\sim}{A}$ is not known exactly, since its elements are the computed values of B-splines. Instead we have the computed matrix

$$\bar{\underset{\sim}{A}} = \underset{\sim}{A} + \underset{\sim}{H} , \tag{6.5.15}$$

where the elements of $\underset{\sim}{H}$ certainly satisfy (see (3.9.13)),

$$|h_{ij}| \leq 7(n-1)2^{-t} a_{ij} , \tag{6.5.16}$$

if Algorithm 3.12.2 has been used to generate the B-spline values. Thus

$$\begin{aligned} \|H\|_\infty &\leq 7(n-1)2^{-t} \max_i \sum_j a_{ij} \\ &= 7(n-1)2^{-t} \max_i \sum_j N_{nj}(x_i) \\ &= 7(n-1)2^{-t} , \end{aligned} \tag{6.5.17}$$

as a consequence of (3.6.1). So to complete the analysis we absorb $\underset{\sim}{H}$ into the matrix $\underset{\sim}{K}$ in (6.5.12) which yields

$$\begin{aligned} \|K\|_\infty &\leq g(3n^3 - \tfrac{1}{2}n^2 + \tfrac{17}{2}n - 7.894)2^{-t_1} \\ &< 3g(n+1)^3 2^{-t_1} . \end{aligned} \tag{6.5.18}$$

Equation (6.5.12) may be put in the form

$$\underset{\sim}{A}\,\underset{\sim}{c} = \bar{\underset{\sim}{f}} , \tag{6.5.19}$$

where

$$\bar{\underset{\sim}{f}} = \underset{\sim}{f} + \delta\underset{\sim}{f} , \tag{6.5.20}$$

$$\|\delta\underset{\sim}{f}\|_\infty \leq \|\underset{\sim}{K}\|_\infty \|\underset{\sim}{c}\|_\infty . \tag{6.5.21}$$

Consequently our computed solution has the property that it corresponds to the exact interpolation of the data points $(t_i, \bar{f}_i)$ $(i = 1, 2, \ldots, m)$ by a spline with interior knots x_j $(j = 1, 2, \ldots, N-1)$, where

$$\|\bar{f} - f\|_\infty \leq 3g(n+1)^3 \|c\|_\infty 2^{-t_1}. \tag{6.5.22}$$

According to Martin and Wilkinson (1967) g is seldom greater than $\max_{i,j} |a_{ij}|$ in the original matrix. But $a_{ij} = N_{nj}(x_i)$ and $0 \leq N_{nj}(x) \leq 1$. Thus $\max_{i,j} |a_{ij}| \leq 1$. Consequently, the result

$$\|\bar{f} - f\|_\infty \leq 3(n+1)^3 \|c\|_\infty 2^{-t_1} \tag{6.5.23}$$

will usually hold.

It is our experience that in most practical situations the ratio $\|c\|_\infty / \|f\|_\infty$ proves to be close to unity. In such cases (6.5.23) can be replaced by the approximate <u>relative</u> error bound

$$\frac{\|\bar{f} - f\|_\infty}{\|f\|_\infty} \leq 3(n+1)^3 2^{-t_1}. \tag{6.5.24}$$

The above bounds are very satisfactory in that they depend only upon the order n of the spline and are therefore independent of the number of data points m. Note that if a spline basis not haveing a compact support property such as that of the B-splines were employed then A would be a full matrix and consequently the resulting error bounds would contain a term in m^3 (cf Wilkinson, 1963: p 108) rather than in $(n+1)^3$.

The bounds (6.5.23) and (6.5.24) are quoted here since we believe they would hold in most circumstances. It is always possible to derive relatively pathological examples in which $\|c\|_\infty \gg \|f\|_\infty$ and in these circumstances (6.5.24) will provide an optimistic estimate of the accuracy of the results. Such cases usually correspond to data which comes close in some sense to violating the Schoenberg-Whitney conditions (6.1.4) and hence could not be considered well-posed interpolation problems. It is of course trivial in practice to verify whether $\|c\|_\infty$ is indeed of the order of $\|f\|_\infty$.

Only the bound (6.5.22) is rigorous, however, and the cautious user may prefer always to use it in practice. Its evaluation requires a value for g which in turn calls for the monitoring of the growth of the elements as $\underset{\sim}{A}$ is reduced to $\underset{\sim}{LU}$ form. Such a monitoring can be carried out efficiently using a method described by Businger (1971).

An analysis, similar to the above, can be carried out of elimination algorithms such as Algorithm 2.12.1 that utilize the specific structure of $\underset{\sim}{A}$. Unfortunately, the error bound now depends upon the precise nature of the stepped-banded structure. However, the bound (6.5.22) certainly holds. For a band centred roughly on the main diagonal, the right-hand side of (6.5.22) would be reduced by a factor of approximately 8. A somewhat weaker bound can be obtained for Algorithm 2.13.1 based on elementary transformations.

Analogous error bounds can be obtained for the methods that employ unitary transformations (eg Algorithm 2.14.1). These bounds are somewhat more satisfactory in that no factor g is present, there being no possibility of error growth since the 2-norm of each column remains essentially constant during the reduction (Wilkinson, 1965: p 246). We have found, at least on the basis of some 20-30 problems considered to date, that Wilkinson's contention that g is almost invariably of order unity, when using Gaussian elimination with partial pivoting, certainly seems to hold for the linear systems arising from spline interpolation problems. Consequently, because of the slightly simpler programming and faster computation of Gaussian elimination methods, it appears that elimination methods offer some advantages over methods employing unitary transformations. Moreover, in the cases studied, Gaussian elimination with partial pivoting has never given poorer results than unitary transformations (classical plane rotations). In a number of cases the maximum error was about half that for Givens rotations. We also observed comparable behaviour when stabilized elementary transformations were employed.

6.6 Multiple knots

The fact that the algorithm described in this chapter can be used for multiple knots is essentially implicit in our description. However, it may be emphasized that Algorithms 6.4.1 and 6.4.2 can be used just as efficiently to determine interpolating splines of a lower continuity class. For example, in order to interpolate by a spline of degree 5 with continuity up to and including the second derivative, triple knots in place of simple knots are employed. It may sometimes be advantageous to relax continuity at a single point. For instance, the function $|x|^{n-1}$ may be represented exactly by a spline of order n having a single knot of multiplicity n-1 at x = 0.

6.7 The choice of exterior knots

The condition number κ of the matrix $\underset{\sim}{A}$ is dependent on the choice of additional knots. We conjecture that, as regards obtaining a relatively small value for κ, a good choice of knots is that already suggested, viz knots of multiplicity n at the range end-points x = a and x = b. To support this conjecture and to investigate the possible extent of this dependence we give a class of simple numerical examples.

Consider the interpolation of the data points (t_i, f_i) (i = 1, 2, ..., m = N+3) by a cubic spline with interior knots $x_i = t_{i+2}$ (i = 1, 2, ..., N-1) (cf Section 6.8). We have used the singular value decomposition (Section 2.15) to determine the spectral condition number κ_2 for three choices of the exterior knots. We set

$$x_i = \begin{cases} t_1 - ih_1 & (i \leq 0) \\ t_m + ih_2 & (i \geq N), \end{cases} \tag{6.7.1}$$

where

$$h_1 = h_2 = 0 \tag{6.7.2}$$

(coincident end knots), or

$$h_1 = x_1 - t_1, \quad h_2 = t_m - x_{N-1} \tag{6.7.3}$$

(left-hand and right-hand end knots at spacings respectively equal to the first and last interval length), or

$$h_1 = h_2 = (t_m - t_1)/N \tag{6.7.4}$$

(end knots at a spacing equal to the average interval length).

The values of $\varkappa_2$ for equi-spaced data $t_i = i$ ($i = 1, 2, \ldots, m$) and values of $m = 4, 5, \ldots, 20$ for these three choices of exterior knots are given in Table 6.7.1. To compute the singular values and thus the spectral condition number of $\underset{\sim}{A}$ we first used plane rotations to reduce $\underset{\sim}{A}$ to upper band-triangular form. Then the published procedure 'minfit'*, which is one of the Algol 60 realizations given by Golub and Reinsch (1970) of the singular value decomposition, was employed to diagonalize the band triangle.

In Table 6.7.1 are the values of $\varkappa_2$, column 2 containing the values corresponding to coincident end knots and columns 3 and 4 containing

---

* In fact procedure 'minfit' failed, because of floating-point overflow, in attempting the case $m = 19$ for the second choice of knots. This failure was attributed to a division by zero, which resulted from underflow in attempting to compute the rotation parameters. After replacing this aspect of the computation by the modified process recommended in Section 2.9, 'minfit' then worked satisfactorily in all cases. In cases where the un-modified 'minfit' produced results, the singular values agreed, apart from a few units in the least significant place, with those produced by the modified version.

respectively the values corresponding to (6.7.3) and (6.7.4). Tests carried out with a variety of unequally spaced data as well as with splines of other orders generally reinforce the conclusion that the choice of coincident end knots seems to be a good one.

m	Values of κ_2: Coincident end knots	Distinct end knots (1)	Distinct end knots (2)
4	5.0541	58.193	58.193
5	5.0626	33.596	33.596
6	4.6119	31.175	23.559
7	4.4838	27.772	18.222
8	4.2120	25.826	15.456
9	4.0715	25.066	14.068
10	3.9935	24.757	13.249
11	3.9498	24.632	12.708
12	3.9252	24.580	12.320
13	3.9112	24.557	12.026
14	3.9032	24.548	11.793
15	3.8986	24.543	11.604
16	3.8959	24.541	11.447
17	3.8943	24.540	11.314
18	3.8933	24.540	11.200
19	3.8928	24.540	11.101
20	3.8924	24.540	11.014

Table 6.7.1. Values of the spectral condition number κ_2 of $\underset{\sim}{A}$ for m equi-spaced data and three choices for the end knots.

6.8 A conjecture relating to the choice of interior knots and comments on the "well-posedness" of the spline interpolation problem

We make the following conjecture. From the viewpoint of inherent stability (ie sensitivity of the spline coefficients with respect to the data), a good choice of interior knots in the case of even-order splines, ie $n = 2k$, is

$$x_i = t_{k+i} \quad (i = 1, 2, \ldots, N-1). \tag{6.8.1}$$

For the choice (6.8.1) the interpolating spline of order 2k is composed of polynomial arcs of degree 2k-1, each of which spans one interval between adjacent data points, except the first and last arcs, each of which spans k adjacent intervals. Later in this section we investigate the dependence of the conditioning of the cubic spline interpolation problem for the choice (6.8.1) upon the value of m. Firstly, however, we consider in detail what proves to be a very poor choice of knots and subsequently compare it with the above choice.

The second choice of knots emphasises an important observation: the satisfaction of the Schoenberg-Whitney conditions (6.1.4) is no guarantee in itself that the coefficients of the interpolating spline are well defined. This remark is true even if the conditions are "well-satisfied", ie even if the data and knots are such that there exist appreciable perturbations in their values which are such that (6.1.4) remains satisfied. Consider the following example. Interpolate data points (t_i, f_i) $(i = 1, 2, \ldots, m)$ by a cubic spline with knots $x_j = t_1$ $(j \leq 0)$, $x_j = t_{j+3}$ $(j = 1, 2, \ldots, N-1)$, $x_j = t_m$ $(j \geq N)$. Here $N = m-3$. The system of equations defining the B-spline coefficients is

$$\underset{\sim}{A}\underset{\sim}{c} = \underset{\sim}{f}, \tag{6.8.2}$$

where A takes the form, illustrated here for the case $m = 10$:

$$\underset{\sim}{A} = \begin{bmatrix} X & & & & & & & & & \underset{\sim}{0} \\ X & X & X & X & & & & & & \\ X & X & X & X & & & & & & \\ & X & X & X & & & & & & \\ & & X & X & X & & & & & \\ & & & X & X & X & & & & \\ & & & & X & X & X & & & \\ & & & & & X & X & X & & \\ & & & & & & X & X & X & \\ \underset{\sim}{0} & & & & & & & & & X \end{bmatrix} . \qquad (6.8.3)$$

Three stabilized elementary transformations, involving only the second, third and fourth rows, enable $\underset{\sim}{A}$ to be converted to the lower band-triangular form

$$\underset{\sim}{L} = \begin{bmatrix} X & & & & & & & & & \underset{\sim}{0} \\ X & X & & & & & & & & \\ X & X & X & & & & & & & \\ X & X & X & X & & & & & & \\ & & X & X & X & & & & & \\ & & & X & X & X & & & & \\ & & & & X & X & X & & & \\ & & & & & X & X & X & & \\ & & & & & & X & X & X & \\ \underset{\sim}{0} & & & & & & & & & X \end{bmatrix} . \qquad (6.8.4)$$

If the same transformations are applied to $\underset{\sim}{f}$ to produce a vector $\underset{\sim}{g}$, the solution of the system

$$\underset{\sim}{L}\underset{\sim}{c} = \underset{\sim}{g} \qquad (6.8.5)$$

yields the required coefficient vector.

It is readily verified that, for $i = 7, 8, \ldots, m-1$,

$$l_{i,i-2} = l_{ii} = \frac{1}{6}, \quad l_{i,i-1} = \frac{2}{3}. \tag{6.8.6}$$

Now consider the solution of (6.8.5) using forward substitution. After c_i ($i = 1, 2, \ldots, 6$) have been determined then, for $i = 7, 8, \ldots, m-1$,

$$c_i = 6f_i - (4c_{i-1} + c_{i-2}). \tag{6.8.7}$$

If $\bar{c}_i$ denotes the computed value of c_i, in floating-point arithmetic, (6.8.7) becomes

$$\bar{c}_i = \frac{6f_i(1+\varepsilon_{1,i}) - (4\bar{c}_{i-1}+\bar{c}_{i-2})(1+\varepsilon_{2,i})}{1+\varepsilon_{3,i}}, \tag{6.8.8}$$

where

$$|\varepsilon_{1,i}|, \; |\varepsilon_{2,i}|, \; |\varepsilon_{3,i}| \leqslant 2^{-t}. \tag{6.8.9}$$

Suppose that no rounding errors at all are committed, ie that $\varepsilon_{1,i} = \varepsilon_{2,i} = \varepsilon_{3,i} = 0$. Then, after setting $\bar{c}_i = c_i + \delta c_i$, we obtain from (6.8.8) and (6.8.7),

$$\delta c_i = -4\delta c_{i-1} - \delta c_{i-2}. \tag{6.8.10}$$

Now the solution of the <u>difference equation</u> (6.8.10) is

$$\delta c_i = A(-2-3^{\frac{1}{2}})^i + B(-2+3^{\frac{1}{2}})^i, \tag{6.8.11}$$

where A and B are constants that depend on the initial conditions. The term $(-2+3^{\frac{1}{2}})^i$ is oscillatory and <u>damped</u>, whereas the term $(-2-3^{\frac{1}{2}})^i$ is oscillatory and <u>undamped</u>. For sufficiently large values of i, the damped term is negligible and for all practical purposes

$$\delta c_i = -(2+3^{\frac{1}{2}})\delta c_{i-1}. \tag{6.8.12}$$

Thus, although there will be a slight contamination of this result because of the presence of the rounding errors, it may be expected that the errors in the c_i will eventually grow exponentially with i. In particular, replacing m by m+1 implies that δc_{m-1} will grow by a factor of $2+3^{\frac{1}{2}}$. Any sensible condition number associated with this problem should therefore grow by such a factor as the number of data points is increased by one. We see shortly that if $\|A\| \|A^{-1}\|$ is taken as the measure of condition, where $\|\cdot\|$ denotes the spectral norm, such growth is indeed observed.

It should be emphasised that the reason for this rapid growth in condition number is not that the forward-substitution process is itself unstable, but that the particular choice of knots gives rise to an ill-posed problem. That the problem is ill-posed can be seen heuristically as follows. The clue is given by the fact that in order to reduce A to triangular form, operations on only the first four rows are required. The interpretation of this observation in the context of the actual interpolation problem is that the data points (t_j, f_j) $(j = 1, 2, 3, 4)$ all lie within the interval spanned by the knots x_0 and x_1 and hence the cubic arc spanning this interval is defined uniquely by these four points. Because of the continuity of a cubic spline, the cubic arc spanning the interval (x_1, x_2) must take at $x=x_1$ the value and first and second derivatives of the first cubic arc constructed. Also it must pass through the point (t_5, f_5). These four pieces of information fully define the second cubic arc. In a similar way all remaining cubic arcs and hence the complete cubic spline may be constructed.

Evidently, any errors made in constructing a particular cubic arc will be propagated in some way to the subsequent cubic arc. Errors in the values of the first and second derivatives will have particularly detrimental effects. The situation is somewhat akin to the solution of an initial-value problem, where the solution is sometimes much more sensitive to the effects of build-up of errors than the solution of a corresponding boundary-value problem. Despite these comments, as a consequence of the analysis of Section 6.5, the resulting computed spline is nevertheless the exact interpolant of a set of data points close to those prescribed. However, the spline so constructed and the "true" spline may be very different, coinciding only at or near the data points.

If the knots of the spline are those in (6.8.1) then, in the case n=4, $\underset{\sim}{A}$ takes the form (cf 6.8.3)

$$\underset{\sim}{A} = \begin{bmatrix} X & & & & & & & & & \underset{\sim}{0} \\ X & X & X & X & & & & & & \\ & X & X & X & & & & & & \\ & & X & X & X & & & & & \\ & & & X & X & X & & & & \\ & & & & X & X & X & & & \\ & & & & & X & X & X & & \\ & & & & & & X & X & X & \\ & & & & & & X & X & X & X \\ \underset{\sim}{0} & & & & & & & & & X \end{bmatrix} \qquad (6.8.13)$$

By applying six stabilized elementary transformations the corresponding system may be converted to triple-diagonal form. The solution then proves relatively insensitive to data perturbations, the problem now being considerably better-posed. In fact $\underset{\sim}{A}$ may well be diagonally dominant.

The use of the singular value decomposition (Section 2.15) also displays very clearly the relative conditioning of the problems associated with the two choices of knots. The following test was carried out. For each value of m from 4 to 20 we set $t_i = i$ $(i = 1, 2, \ldots, m)$ and made the poor choice of knots

$$x_i = \begin{cases} 1 & (i \leqslant 0) \\ i+3 & (i = 1, 2, \ldots, m-4) \ . \\ m & (i \geqslant m-3) \end{cases} \tag{6.8.14}$$

The matrix $\underset{\sim}{A}$ based on this data, which takes the form (6.8.3), was then computed. The singular value decomposition was used to compute the spectral norm (spectral condition number) A. $\kappa_2(m)$ of $\underset{\sim}{A}$. In Table 6.8.1 we give in Column 2 for each value of m the value of $\kappa_2(m)$, as well as the ratios of successive values of $\kappa_2(m)$ in Column 3. The exercise was repeated but with the good choice of knots

$$x_i = \begin{cases} 1 & (i \leqslant 0) \\ i+2 & (i = 1, 2, \ldots, m-4), \\ m & (i \geqslant m-3) \end{cases} \tag{6.8.15}$$

which gives rise to a matrix $\underset{\sim}{A}$ of the form (6.8.13). The corresponding values of $\kappa_2(m)$ are given in Column 4 of Table 6.8.1 and their successive ratios in Column 5. It is to be noticed that for the better choice of knots the ratio $\kappa_2(m)/\kappa_2(m-1)$ tends to unity, whereas that for the poorer choice tends rapidly to a value approximating $2+3^{\frac{1}{2}}$, which was derived from other considerations earlier in this section.

m	Interior knots $x_i = t_{i+3}$: $\varkappa_2(m)$	$\varkappa_2(m)/\varkappa_2(m-1)$	Interior knots $x_i = t_{i+2}$: $\varkappa_2(m)$	$\varkappa_2(m)/\varkappa_2(m-1)$
4	5.0540		5.0540	
5	8.0252	1.588	5.0627	1.002
6	$1.9254_{10}1$	2.399	4.6119	0.911
7	$7.0453_{10}1$	3.659	4.4838	0.972
8	$2.6356_{10}2$	3.741	4.2120	0.939
9	$9.8437_{10}2$	3.735	4.0715	0.967
10	$3.6742_{10}3$	3.733	3.9935	0.980
11	$1.3713_{10}4$	3.732	3.9498	0.989
12	$5.1177_{10}4$	3.732	3.9252	0.994
13	$1.9099_{10}5$	3.732	3.9112	0.996
14	$7.1280_{10}5$	3.732	3.9032	0.998
15	$2.6602_{10}6$	3.732	3.8986	0.999
16	$9.9280_{10}6$	3.732	3.8959	0.999
17	$3.7052_{10}7$	3.732	3.8943	1.000
18	$1.3828_{10}8$	3.732	3.8933	1.000
19	$5.1607_{10}8$	3.732	3.8928	1.000
20	$1.9260_{10}9$	3.732	3.8924	1.000

Table 6.8.1 Values of the spectral condition number $\varkappa_2$ of $\underset{\sim}{A}$ for m equi-spaced data and two choices of the interior knots

6.9 Numerical examples

All numerical examples were carried out on the English Electric KDF9 computer, which has a floating-point word with 39 binary bits (between 11 and 12 decimals) in the mantissa. The results quoted correspond to the use of Algorithm 6.4.2. Virtually identical results were obtained with Algorithm 6.4.1. Coincident end knots at the first and last data points were chosen in all cases.

The first three examples have been chosen to illustrate the numerical stability of the method, rather than to demonstrate the approximating power of splines. The remaining example arose in a study relating to the decay of β-particles.

Example 6.9.1 ($m=16$, $n=8$)

A spline $s(x)$ of order 8 was defined by the arbitrarily-chosen interior knots x_i and B-spline coefficients $\underset{\sim}{c}$ given in columns 3 and 4 of Table 6.9.1. Function values were computed using Algorithm 5.2.1 from the representation (5.1.10) for the values $x = t_i$ ($i = 1, 2, \ldots, 16$) given in column 2 of Table 6.9.1. Algorithm 6.4.1 was used to interpolate these values; the differences between the resulting coefficients $\bar{\underset{\sim}{c}}$ and the values of $\underset{\sim}{c}$ are given in column 5 of Table 6.9.1.

Values of $s(x)$ at $x = t_j$ ($j = 1, 2, \ldots, 16$) and at the half-way points $x = \frac{1}{2}(t_{j-1} + t_j)$ ($j = 2, 3, \ldots, 16$) were computed using Algorithm 5.2.1 from the B-spline representation (5.1.10) for the given coefficients $\underset{\sim}{c}$ and for the computed coefficients $\bar{\underset{\sim}{c}}$, and from the piecewise-Chebyshev-series representation (5.11.1). The maximum discrepancy between the given and computed B-spline representations over these 31 points was 1×10^{-9}, and that between the given B-spline representation and the Chebyshev-series form was also 1×10^{-9}.

i	t_i	x_i	c_i	$(\bar{c}_i - c_i) \times 10^8$
1	0	8	51	0.00
2	3	10	35	- 0.03
3	5	13	21	+ 0.03
4	7	15	13	+ 0.02
5	8	16	14	- 0.10
6	10	17	22	+ 0.17
7	13	18	37	- 0.14
8	15	20	60	+ 0.08
9	16		76	- 0.05
10	17		77	+ 0.14
11	18		66	- 0.47
12	20		54	+ 1.01
13	22		44	- 1.37
14	25		40	+ 1.29
15	28		41	- 0.52
16	30		44	0.00

Table 6.9.1 Prescribed and computed B-spline coefficients for Example 6.9.1.

Example 6.9.2 $(m=11,\ n=6)$

In order to illustrate the performance of the Algorithm 6.A.2 upon an example with multiple knots, the following case was considered. Values of the function $f(x) = |x + x^5|$ were computed for the values $x = t_i$ (deliberately chosen not to lie symmetrically disposed about $x = 0$) given in column 2 of Table 6.9.2. $f(x)$ is essentially a spline of order 6 with a knot of multiplicity 5 at the origin. Accordingly, the interior knots given in column 3 of Table 6.9.2 were chosen. The algorithm produced values $\bar{c}$ which differ from the true values $\underset{\sim}{c}$ (given in column 4 of Table 6.9.2) by the quantities given in column 5.

The value of $\displaystyle\int_{-1}^{1} s(x)dx$ computed from (5.10.7) was 1.33333 33333, which agrees to 11 significant figures with $\displaystyle\int_{-1}^{1} \left|x+x^5\right| dx = \frac{4}{3}$.

Values of $s(x)$ were computed at $x = t_j$ ($j = 1, 2, \ldots, 11$) and at $x = \frac{1}{2}(t_{j-1}+t_j)$ ($j = 2, 3, \ldots, 11$) from the B-spline representation for the computed coefficients, from the piecewise-Chebyshev-series representation, and from $f(x)$. The maximum discrepancy over these 21 points between the B-spline representation and $f(x)$ was 3×10^{-12}, and that between the Chebyshev-series form and $f(x)$ was 1×10^{-11}.

i	t_i	x_i	c_i	$(\bar{c}_i - c_i) \times 10^{11}$
1	– 1.0	0	2.0	0.00
2	– 0.8	0	0.8	+ 0.36
3	– 0.6	0	0.6	– 1.82
4	– 0.4	0	0.4	+ 2.18
5	– 0.2	0	0.2	– 0.91
6	0.1		0	– 0.01
7	0.3		0.2	+ 0.14
8	0.5		0.4	– 0.55
9	0.7		0.6	+ 1.09
10	0.9		0.8	– 0.36
11	1.0		2.0	0.00

Table 6.9.2 True and computed B-spline coefficients for Example 6.9.2

Example 6.9.3 (m=11, n=11)

By way of a special test case, this example illustrates the interpolation of 11 equally-spaced values of e^x in the interval $-1 \leq x \leq 1$, by a spline of order 11 with no interior knots. In other words the spline degenerates into a single polynomial of degree 10. The function e^x over the range $-1 \leq x \leq 1$ can in fact be approximated to 10 decimals by such a polynomial Clenshaw, 1962). The computed Chebyshev coefficients (Table 6.9.4) differ by at most 1×10^{-10} from those given by Clenshaw.

Note that the computed B-spline coefficients (Table 6.9.3) are all positive and display a very systematic behaviour. We also observe that to 11 significant figures $\bar{c}_1 = e^{-1}$ and $\bar{c}_{11} = e$ (as a consequence of the choice of coincident end knots). The integral of the spline between -1 and +1 was computed from (5.10.7) as 2.35040 23873. This value agrees to 11 significant figures with that of $\int_{-1}^{1} e^x dx = e - e^{-1}$.

It is of interest to observe that the functions $N_{ni}(x)$ (or $M_{ni}(x)$) in the case N = 1, when translated to the range $0 \leq x \leq 1$, are simply multiples of the basis functions $x^{i-1}(1-x)^{n-i}$ $(i = 1, 2, \ldots, n)$ of the Bernstein polynomials (Davis, 1963).

i	$\bar{c}_i$
1	0.36787 94411 7
2	0.44145 53279 4
3	0.53138 14217 7
4	0.64174 52279 4
5	0.77730 22904 5
6	0.94636 49130 0
7	1.15634 83957
8	1.41954 72052
9	1.75178 16123
10	2.17462 54652
11	2.71828 18285

Table 6.9.3 Computed B-spline coefficients for Example 6.9.3

i	α_{1i}
0	2.53213 17555
1	1.13031 82081
2	0.27149 53395 6
3	0.04433 68499 0
4	0.00547 42404 5
5	0.00054 29263 1
6	0.00004 49773 2
7	31983 8
8	1992 0
9	109 9
10	10 9

Table 6.9.4 Computed Chebyshev-series coefficients for Example 6.9.3

Example 6.9.4 ($m=24$, $n=4$ and 6)

This example is concerned with one aspect of a problem that originated in the Division of Radiation Science of the National Physical Laboratory. I am indebted to this division for permission to include their data and the results of some of the computations upon it.

The 24 data points (t_i, f_i) ($i = 1, 2, \ldots, 24$) in Columns 2 and 3 of Table 6.9.5 represent the theoretical number of electrons in the β-decay of a radioactive isotope (dependent variable) for various values of momentum (independent variable). The determination of each value of the dependent variable involved the numerical evaluation of an extremely complicated integral; it is believed that the value is correct to the number of figures quoted. It was required to interpolate these data points by a smooth function that would facilitate subsequent rapid evaluation of a good approximation to the number of electrons for any value of momentum in the prescribed range. An estimate of the definite integral over the range of the data was also required. In the absence of any further information it was decided to interpolate the data by a cubic spline $s_4(x)$ with knots chosen in accordance with (6.8.1). A further interpolation was carried out with a quintic spline $s_6(x)$, again choosing knots in accordance with (6.8.1). B-spline coefficients c_i of the interpolating splines obtained using Algorithm 6.4.2, and the integrals formed from (5.10.7) are given in Columns 4 and 5 of Table 6.9.5. Note that the values of c_i, particularly in the cubic case, mimic quite closely the values of f_i.

i	t_i	f_i	c_j n=4	n=6
1	0.1	5.5613	5.56130	5.56130
2	0.2	5.6200	5.58855	5.58526
3	0.3	5.7159	5.66430	5.64054
4	0.4	5.8516	5.84435	5.74222
5	0.5	6.0300	6.02289	5.93048
6	0.6	6.2502	6.24411	6.24109
7	0.7	6.5069	6.50189	6.49939
8	0.8	6.7938	6.78974	6.78771
9	0.9	7.1052	7.10197	7.16291
10	1.0	7.4361	7.54413	7.63436
11	1.2	8.1407	8.13453	8.20618
12	1.4	8.8837	8.88005	8.87827
13	1.6	9.6496	9.64748	9.64660
14	1.8	10.429	10.42765	10.42669
15	2.0	11.216	11.21593	11.21633
16	2.2	12.005	12.00465	12.00406
17	2.4	12.795	12.79548	12.79605
18	2.6	13.583	13.58345	13.58344
19	2.8	14.368	14.36874	14.36931
20	3.0	15.149	15.14959	15.46189
21	3.2	15.926	15.92690	16.39353
22	3.4	16.698	16.95612	17.15985
23	3.6	17.465	17.72069	17.77176
24	3.8	18.227	18.22700	18.22700
Estimates of definite integral			41.46130	41.46131

Table 6.9.5 Data and computed B-spline coefficients of orders 4 and 6 for Example 6.9.4

The first use made of the interpolating splines was to evaluate them throughout a particularly important part of the range of the independent variable, viz from $x = 0.10$ to $x = 3.22$ at in interval of 0.02 in x. The resulting table is too bulky to reproduce in full; we give part in Table 6.9.6.

It is reassuring to see a strong measure of agreement between the values of $s_4(x)$ and $s_6(x)$ that is almost consistent with the supposed accuracy of the data. However, without further knowledge or assumptions, this agreement tells us nothing about the closeness of either $s_4(x)$ or $s_6(x)$ to $f(x)$.

x	$s_4(x)$	$s_6(x)$	$10^5\left\{s_4(x)-s_6(x)\right\}$
0.10	5.56130	5.56130	0
0.12	5.57018	5.57006	+12
0.14	5.58046	5.58033	+13
0.16	5.59219	5.59208	+11
0.18	5.60536	5.60531	+5
0.20	5.62000	5.62000	0
0.22	5.63613	5.63616	−3
0.24	5.65377	5.65381	−4
0.26	5.67293	5.67297	−4
0.28	5.69363	5.69365	−2
0.30	5.71590	5.71590	0
1.50	9.26448	9.26452	−4
1.52	9.34120	9.34123	−3
1.54	9.41807	9.41810	−3
1.56	9.49511	9.49513	−2
1.58	9.57229	9.57230	−1
1.60	9.64960	9.64960	0
1.62	9.72704	9.72704	0
1.64	9.80461	9.80460	+1
1.66	9.88229	9.88227	+2
1.68	9.96009	9.96007	+2
1.70	10.03800	10.03797	+3
3.00	15.14900	15.14900	0
3.02	15.22689	15.22689	0
3.04	15.30474	15.30474	0
3.06	15.38256	15.38255	+1
3.08	15.46033	15.46032	+1
3.10	15.53806	15.53804	+2
3.12	15.61574	15.61573	+1
3.14	15.69338	15.69337	+1
3.16	15.77097	15.77096	+1
3.18	15.84851	15.84851	0
3.20	15.92600	15.92600	0
3.22	16.00343	16.00344	−1

Table 6.9.6 Part of the tabulation of the interpolating splines of orders 4 and 6 for Example 6.9.4

CHAPTER 7

LEAST-SQUARES SPLINE APPROXIMATION

In this chapter we consider the least-squares approximation of discrete data sets and of functions by polynomial splines. We pay considerably more attention to the discrete problem, where the data is usually empirical in nature, since we consider it to be of far greater practical importance than the continuous case. In fact, if a spline approximation of a mathematical function <u>is</u> required, it is usually more appropriate to seek a <u>minimax</u> approximation, ie one that minimizes the maximum error of the approximation (see, for example, Rice, 1969: 145-154). Methods for the determination of such approximations are outside the scope of this work. However, for completeness, we show briefly in the last section of this chapter that if a <u>least-squares</u> spline approximation <u>is</u> required, it can be computed efficiently if the spline is first expressed in its B-spline form.

The organization of the earlier sections of this chapter follows to some extent that of Chapter 6 on spline interpolation, since least-squares approximation by splines can be considered as a generalization of spline interpolation. After all, if the problem is properly posed (see Sections 6.2 and 7.1) and if the number of free linear parameters of the spline is the same as the number of data points (assumed distinct), then the least-squares approximation of this data set <u>interpolates</u> the points. Moreover, similar numerical methods (Chapter 2) can be applied to the resulting linear systems in both the interpolation and least-squares cases.

In Section 7.1 we introduce the least-squares spline-fitting problem and in Section 7.2 discuss a method of solution, using B-splines, in the case where the knots are prescribed. Also in Section 7.2 a simple algorithm is presented for testing whether a unique spline approximant exists in any given case. In Section 7.3 an algorithm for the least-squares spline-fitting problem is detailed and in Section 7.4 an error analysis of the algorithm is given. The sensitivity of the B-spline coefficients to perturbations in the data is discussed in Section 7.5. The important case of cubic splines is considered in Section 7.6 and in Section 7.7 methods of assessing the acceptability of a cubic spline approximant are discussed. The choice of knot positions is treated in Section 7.8 and numerical examples are given in Section 7.9. Previous work on the automatic placement of knots is surveyed in Section 7.10. Finally, in Section 7.11, a method is proposed for the least-squares spline approximation of a mathematical function.

7.1 The least-squares spline-fitting problem

The least-squares spline-fitting problem may be posed in the following manner.

Suppose a set of values $\underset{\sim}{t} = \{t_1, t_2, \ldots, t_m\}$ of an independent variable x and corresponding function values (ordinates) $\underset{\sim}{f} = \{f_1, f_2, \ldots, f_m\}$ are prescribed. These function values may be the computed values of a mathematical function; they may be the results of a previous computation; usually they will be values derived from an experimental situation and hence be contaminated to a greater or lesser extent by experimental error. We assume that the values of the independent variable are ordered such that

$$t_1 \leqslant t_2 \leqslant \cdots \leqslant t_m . \qquad (7.1.1)$$

Note that equalities are permitted in (7.1.1), corresponding in an experimental situation to the repetition, or replication, of measurements. Suppose also that a set of corresponding positive weighting factors $w_1, w_2, \ldots, w_m$ is prescribed. In many cases of interest all weighting factors shall be set equal to unity; however, the general case is considered here because of its importance in certain situations.

The problem is to compute the parameters of a spline function s(x) of order n (degree n-1) with interior knots $x_1, x_2, \ldots, x_{N-1}$ so as to minimize the residual sum of squares

$$\left\| \underset{\sim}{r} \right\|_2^2 = \left\| \underset{\sim}{W}^{\frac{1}{2}} \underset{\sim}{\varepsilon} \right\|_2^2 = \sum_{i=1}^{m} w_i \varepsilon_i^2 , \qquad (7.1.2)$$

where

$$\underset{\sim}{W} = \text{diag} \left\{ w_1, w_2, \ldots, w_m \right\} \qquad (7.1.3)$$

and

$$\varepsilon_i = s(t_i) - f_i \quad (i = 1, 2, \ldots, m) . \qquad (7.1.4)$$

We assume that the set of N-1 interior knots is prescribed and, as in the interpolation problem of Chapter 6, forms an n-extended partition of $(a, b) \equiv (t_1, t_m)$. Again, as in Chapter 6, we introduce additional knots so that the complete set forms a standard knot set with coincident end knots. Guidance relating to the choice of the number of knots and their locations is given in Section 7.8.

Let $\tilde{m}$ denote the number of distinct values of t_i $(i = 1, 2, \ldots, m)$ (for a given set of values of t_i, $\tilde{m}$ is one greater than the number of inequalities in (7.1.1) that can be replaced by strict inequalities). It is assumed henceforth that m, n and N satisfy the condition

$$\tilde{m} \geqslant N+n-1$$

(cf Section 6.1), otherwise there is no possibility in general of a unique solution to the problem as defined.

7.2 Method of solution

For similar reasons to those discussed in Section 6.2 we intend to employ the B-splines $N_{ni}(x)$ as a basis for $s(x)$. Then $s(x)$ may be expressed in the form

$$s(x) = \sum_{i=1}^{q} c_i N_{ni}(x) \quad (a \leqslant x \leqslant b), \tag{7.2.1}$$

where

$$q = N+n-1. \tag{7.2.2}$$

The least-squares problem is solved by determining $\underset{\sim}{c} = \{c_1, c_2, \ldots, c_q\}$ such that $\|\underset{\sim}{r}\|_2$ in (7.1.2) is minimized. Firstly, we observe that in the particular case $m=\tilde{m}=q$ there is a unique solution if and only if the Schoenberg-Whitney conditions (6.1.4) are satisfied (in which case the solution interpolates the given function values and $\|\underset{\sim}{r}\|_2 = 0$). In this case the parameters $\underset{\sim}{c}$ are defined uniquely by the system of linear algebraic equations

$$\underset{\sim}{A}\underset{\sim}{c} = \underset{\sim}{f} \tag{7.2.3}$$

(cf Section 6.2), where $\underset{\sim}{A}$ is the q by q stepped banded matrix of bandwidth n and rank q with $a_{ij} = N_{nj}(t_i)$.

Now consider the general case where $m \geqslant \tilde{m} \geqslant q$. The solution vector then minimizes $\| \underset{\sim}{W}^{\frac{1}{2}}(\underset{\sim}{A}\underset{\sim}{c}-\underset{\sim}{f}) \|_2$, where $\underset{\sim}{A}$ is the m by q stepped-banded matrix of bandwidth n with $a_{ij} = N_{nj}(t_i)$. Again, for $\underset{\sim}{c}$ to be unique, $\underset{\sim}{A}$ must have full rank q. For $\underset{\sim}{A}$ to have this rank there must be at least one set of q linearly independent rows of $\underset{\sim}{A}$. In other words, there must be at least one ordered subset $\hat{\underset{\sim}{t}} = \{t_{k_1}, t_{k_2}, \ldots, t_{k_q}\}$ of $\underset{\sim}{t}$, where

$$1 \leqslant k_1 < k_2 < \ldots < k_q \leqslant m, \tag{7.2.4}$$

for which the Schoenberg-Whitney conditions hold. It follows that for any given data set a unique least-squares spline approximation exists if and only if at least one ordered subset of the data satisfying the Schoenberg-Whitney conditions can be identified. We term the complete set of conditions

$$\left.\begin{array}{lllll} t_{k_1} & < & x_1 & < & t_{k_{1+n}} \\ t_{k_2} & < & x_2 & < & t_{k_{2+n}} \\ \multicolumn{5}{c}{\dots\dots\dots\dots\dots\dots\dots\dots\dots} \\ t_{k_{N-1}} & < & x_{N-1} & < & t_{k_q} \\ t_{k_1} & < & t_{k_2} < \cdots < & & t_{k_q} \\ 1 \leq k_1 & < & k_2 < \cdots < & & k_q \leq m \end{array}\right\} , \qquad (7.2.5)$$

which must hold for at least one choice of the integers $k_1, k_2, \ldots, k_q$ in order to guarantee the existence of a unique least-squares spline approximation, the generalized Schoenberg-Whitney conditions.

It is to be noted that nearly all "reasonable" data sets arising in practice will satisfy these conditions. Only if there are regions where there are too many knots compared with the number of data points are the conditions likely to be violated. We now show that a simple but efficient algorithm, taking $O(m)$ operations, can be constructed to scan any given data set to identify whether such a subset exists. We first re-write conditions (6.1.4) as the equivalent set of inequalities

$$\left.\begin{array}{ll} t_j < x_j & (j = 1, 2, \ldots, n), \\ x_{j-n} < t_j < x_j & (j = n+1, n+2, \ldots, N-1), \\ x_{j-n} < t_j & (j = N, N+1, \ldots, q). \end{array}\right\} \qquad (7.2.6)$$

In the interpolation case these inequalities may be interpreted thus: for each value of j from 1 to $q=N+n-1$ the jth value of the independent variable must lie strictly within the support of the jth B-spline. The sole exceptions to this rule are that in the case of coincident end knots,

$t_1 = x_{1-n}$ ($= x_0$) and $t_m = x_q$ ($= x_N$) are allowed. In the context of the data-fitting problem, there must be at least one subset of $N+n-1$ distinct values of the independent variable, the jth of which lies strictly within the support of the jth B-spline. Algorithm 7.2.1 below, based on this observation, is composed of q steps, the jth of which ($j = 1, 2, \ldots, q$) involves the determination of the first data point, ie the value of t_i with smallest i, distinct from previously-used points, that lies within the support of the jth B-spline. If, for any of these values of j, no such point can be found, the least-squares spline approximation of the data is not unique. Otherwise, the approximation is unique. It is assumed in Algorithm 7.2.1 that the data points are ordered according to (7.1.1), that the knots form a standard knot set with coincident end knots and that $x_0 = t_1$ and $x_N = t_m$. An Algol implementation of Algorithm 7.2.1 in the case of cubic splines ($n=4$) appears in Cox and Hayes (1973).

<u>Algorithm 7.2.1</u>: Determination of whether the generalized Schoenberg-Whitney conditions are satisfied (in which case I is set to zero) or violated (in which case I is set to unity).

Comment: i denotes the data point currently being examined.

Step 1. Set $i = 0$.

Comment: The first data point, distinct from previously-used points, within the support of the jth B-spline is determined in Steps 3-7.

Step 2. For $j = 1, 2, \ldots, N+n-1$ execute Steps 3-7.

Comment: The first data point, distinct from previously-used points, lying to the right of the left-most knot of the jth B-spline is found in Steps 3-6.

Step 3. Replace i by $i+1$.

Comment: If the test in Step 4 is violated the data points have been exhausted before all the conditions have been satisfied.

Step 4. If $i > m$ set $I = 1$ and finish.

Step 5. If $i > 1$ and $t_i = t_{i-1}$ return to Step 3.

Step 6. If $j > n$ and $t_i \leq x_{j-n}$ return to Step 3.

Comment: If the point so found does not lie to the left of the right-most knot a condition is violated.

Step 7. If $j < N$ and $t_i \geq x_j$ set $I = 1$ and finish.

Comment: All the conditions are satisfied if Step 8 is reached.

Step 8. Set $I = 0$.

The least-squares solution of the system

$$\underset{\sim}{W}^{\frac{1}{2}}\underset{\sim}{A}\underset{\sim}{c} = \underset{\sim}{W}^{\frac{1}{2}}\underset{\sim}{f} \tag{7.2.7}$$

may be solved efficiently using one of the algorithms for stepped-banded systems discussed in Chapter 2.

7.3 An algorithm for least-squares spline approximation

Algorithm 7.3.1 is an implementation of the method described in Section 7.2. As with Algorithms 6.4.1 and 6.4.2, either coincident or non-coincident end knots may be supplied. Again, coincident end knots are usually to be preferred. Steps 1.1 to 1.8 of the algorithm constitute checks on the data. As with the algorithms of Section 6.4 there is an element of redundancy in these checks. The algorithm employs Algorithm 7.2.1 to check whether the data satisfies the generalized Schoenberg-Whitney conditions, Algorithm 3.12.2 to compute the values of the normalized B-splines for each data point and Algorithm 2.14.1 to solve the resulting stepped-banded system using classical plane rotations. As with Algorithm 6.4.2 the complete matrix $\underset{\sim}{A}$ of this system is not formed initially, but rather each row is constructed as and when required by Algorithm 2.14.1.

It is assumed that values of m, n and N, data points (t_i, f_i) and corresponding weights w_i ($i = 1, 2, \ldots, m$) and knots x_i ($i = 1-n, 2-n, \ldots, N+n-1$) are supplied to the algorithm.

Algorithm 7.3.1: Data approximation in the least-squares norm by a spline of order n using normalized B-splines and classical plane rotations.

Comment: The number of distinct data points is determined in Steps 1.1 - 1.3.

Step 1.1. Set $\tilde{m} = 1$.

Step 1.2. For $i = 2, 3, \ldots, m$ execute Step 1.3.

Step 1.3. If $t_i > t_{i-1}$ replace $\tilde{m}$ by $\tilde{m}+1$.

Comment: Check whether there is a sufficient number of distinct data points consistent with the order of the spline and the number of knots.

Step 1.4. Finish if the inequality $\tilde{m} \geqslant N+n-1$ is violated.

Comment: Check whether the complete set of knots is ordered.

Step 1.5. Finish if the inequalities $x_{1-n} \leqslant x_{2-n} \leqslant \ldots \leqslant x_{N+n-1}$ are not all satisfied.

Comment: Check whether the complete set of knots forms an n-extended partition.

Step 1.6. Finish if the inequalities $x_{i-n} < x_i$ ($i = 1, 2, \ldots, N+n-1$) are not all satisfied.

Comment: Check whether the data abscissae are ordered and lie within the range $[a, b] \equiv [x_0, x_N]$.

Step 1.7. Finish if the inequalities $x_0 \leqslant t_1 \leqslant t_2 \leqslant \ldots \leqslant t_m \leqslant x_N$ are not all satisfied.

Comment: Check whether the generalized Schoenberg-Whitney conditions are satisfied.

Step 1.8. Use Algorithm 7.2.1 to determine the value of I. Finish if $I = 1$.

Comment: k is the interval number as well as the number of the block currently being processed. σ is the current value of the residual sum of squares.

Step 1.9. Set $k = 1$ and $\sigma = 0$.

Comment: Initialize $\underset{\sim}{R}$ and $\underset{\sim}{\theta}$ to zero.

Steps 2-4. As Steps 2-4 of Algorithm 2.14.1.

Comment: Computations involving the ith data point are described by Steps 6-30.

Step 5. For $i = 1, 2, \ldots, m$ execute Steps 6-30.

Comment: The interval containing t_i is located in Steps 6-7.

Step 6. If $t_i < x_k$ or $k = N$ advance to Step 8.

Step 7. Replace k by $k+1$ and return to Step 6.

Comment: The ith row of $(\underset{\sim}{A} \mid \underset{\sim}{b})$ and the corresponding weight, as required by Algorithm 2.14.1, are formed in Steps 8-8.2.

Step 8. Use Algorithm 3.12.2 (with the minor modification of Section 6.4) with $x = t_i$ to form the values of $N_{nj}(t_i)$ $(j = k, k+1, \ldots, k+n-1)$.

Step 8.1. For $j = 1, 2, \ldots, n$ set $v_j = N_{n,k+j-1}(t_i)$.

Step 8.2. Set $u = f_i$ and $w = v_i$.

Comment: Classical plane rotations to annihilate the elements in row i of $\underset{\sim}{W}^{\frac{1}{2}}\underset{\sim}{A}$ are applied in Steps 9-30.

Steps 9-30. As Steps 9-30 of Algorithm 2.14.1 (with q interpreted as n and n as $N+n-1$).

Step 31. Use Algorithm 2.1.4 to solve $\underset{\sim}{R}\underset{\sim}{c} = \underset{\sim}{\theta}$.

7.4 Error analysis

We give an error analysis of the formation and solution of the over-determined system of equations (7.2.7) in the case of unit weights, ie $\underset{\sim}{W} = \underset{\sim}{I}$. Our results will also hold approximately in cases where all the weights are of roughly the same magnitude. We cannot derive useful results in cases where the weights differ significantly in size.

With exact computation we form the m by q matrix $\underset{\sim}{A}$ with elements $a_{ij} = N_{nj}(t_i)$, followed by the factorization

$$(\underset{\sim}{A} \mid \underset{\sim}{f}) = \underset{\sim}{Q}(\underset{\sim}{R} \mid \underset{\sim}{g}) \; , \tag{7.4.1}$$

where $\underset{\sim}{Q}$ is orthogonal of order m by q, $\underset{\sim}{R}$ is upper triangular of order q and $\underset{\sim}{g}$ is the transformed ordinate vector. The B-spline coefficients $\underset{\sim}{c}$ are then defined by the triangular system

$$\underset{\sim}{R}\underset{\sim}{c} = \underset{\sim}{g} \; . \tag{7.4.2}$$

There are three sources of error in the practical realization of this process: in the formation of $\underset{\sim}{A}$, in the factorization of $(\bar{\underset{\sim}{A}} \mid \underset{\sim}{f})$, where $\bar{\underset{\sim}{A}}$ is the computed $\underset{\sim}{A}$, and in the back-substitution process to solve (7.4.2). Let $\bar{\underset{\sim}{A}} = \underset{\sim}{A} + \delta\underset{\sim}{A}$ be the matrix actually formed, and the <u>computed</u> $\underset{\sim}{R}$ and $\underset{\sim}{g}$, $\bar{\underset{\sim}{R}}$ and $\bar{\underset{\sim}{g}}$, say, the exact factors in a slightly modified system with $\bar{\underset{\sim}{A}}$ replaced by $\bar{\underset{\sim}{A}} + \underset{\sim}{E}$ and $\underset{\sim}{f}$ by $\underset{\sim}{f} + \underset{\sim}{k}$. Thus

$$(\bar{\underset{\sim}{A}} + \underset{\sim}{E} \mid \underset{\sim}{f} + \underset{\sim}{k}) = \bar{\underset{\sim}{Q}}(\bar{\underset{\sim}{R}} \mid \bar{\underset{\sim}{g}}) \; , \tag{7.4.3}$$

where $\bar{\underset{\sim}{Q}}$ is orthogonal. We shall make the realistic assumption that the errors in the back-substitution process are negligible (cf Gentleman, 1973).

Since the careful use of orthogonal transformations results in an exact factorization of a neighbouring system, any of the methods of Sections 2.6 to 2.9 ensures that, in a suitable norm,

$$\|\underset{\sim}{E}\| \leqslant K_1 \|\underset{\sim}{A}\| 2^{-t} \tag{7.4.4}$$

and

$$\|\underset{\sim}{k}\| \leqslant K_2 \|\underset{\sim}{f}\| 2^{-t} \; , \tag{7.4.5}$$

where K_1 and K_2 are "modest" functions of m and q (for precise forms of K_1 and K_2 in the case of dense rectangular matrices see eg Björck, 1967, Gentleman, 1973; Lawson and Hanson, 1974). For all practical values of m and q, $K_1 2^{-t}$ and $K_2 2^{-t} \ll 1$. If the values of the B-splines required in forming $\underset{\sim}{A}$ are computed using Algorithm 3.12.2, the very small error $\delta\underset{\sim}{A}$ (see Section 3.9) can conveniently be absorbed into the perturbation matrix $\underset{\sim}{E}$ (cf the error analysis in Section 6.5 of one of our algorithms for spline interpolation), the only effect being to inflate slightly the value of K_1.

In the error analysis of Section 6.5 it was convenient to interpret the computed solution as the exact solution of a perturbed system with $\underset{\sim}{f}$ replaced by $\bar{\underset{\sim}{f}} = \underset{\sim}{f}+\delta\underset{\sim}{f}$. Bounds for $\|\delta\underset{\sim}{f}\|$ were then derived, from which it was possible to state that the computed solution had the property that it corresponded to the exact interpolation of a set of data points with an ordinate vector slightly perturbed (usually in a relative sense) from that prescribed. In the main, the derivation of the bounds for $\|\delta\underset{\sim}{f}\|$ were straightforward and followed closely the conventional approach of backward error analysis of linear systems.

We believe it appropriate to seek a similar interpretation of our computed least-squares solution. That is, we wish to find bounds for $\|\delta\underset{\sim}{f}\|$ such that the computed least-squares solution of the rectangular system (7.2.7) is the exact least-squares solution of a similar system with a (hopefully slightly) perturbed right-hand side. The derivation of such a bound is somewhat harder than in the square case (interpolation) and I am indebted to Dr J H Wilkinson for suggesting the following method of approach, which we specialize to the circumstances of our particular problem.

Suppose that such a $\delta\underset{\sim}{f}$ exists. Then it satisfies the normal equations

$$\underset{\sim}{A}^T\underset{\sim}{A}\underset{\sim}{c} = \underset{\sim}{A}^T(\underset{\sim}{f}+\delta\underset{\sim}{f}) , \qquad (7.4.6)$$

where $\underset{\sim}{c}$ in (7.4.6) denotes the computed solution. But the same solution

c satisfies the equations

$$(A+E)^T(A+E)c = (A+E)^T(f+k) , \qquad (7.4.7)$$

where E and k satisfy (7.4.4) and (7.4.5). Subtraction of (7.4.6) from (7.4.7) yields

$$E^T(A+E)c + A^TEc = E^T(f+k) + A^Tk - A^T\delta f , \qquad (7.4.8)$$

from which

$$A^T\delta f = A^Tk - A^TEc + E^T(f+k) - E^T(A+E)c . \qquad (7.4.9)$$

In general (7.4.9) has an infinity of solutions for δf. We are interested, of course, in that which is smallest in some sense; accordingly we select that with minimum norm. Now the minimum-norm solution of the system

$$A^T\delta f = v, \qquad (7.4.10)$$

for any vector v, can be obtained as follows. Let

$$A = QR \qquad (7.4.11)$$

be the exact orthogonal triangularization of A. Then

$$A^T = R^TQ^T . \qquad (7.4.12)$$

We now associate A^T, R^T and Q^T of (7.4.12) with A, G and H, respectively, of (2.2.4). Then, using (2.2.13), the minimal least-squares solution of (7.4.10) is

$$\delta f = Q(Q^TQ)^{-1}(RR^T)^{-1}Rv , \qquad (7.4.13)$$

which, in the full-rank case, simplifies to

$$\delta f = QR^{-T}v . \qquad (7.4.14)$$

It follows that the minimal least-squares solution of (7.4.9) is

$$\underset{\sim}{\delta f} = \underset{\sim}{Q}\underset{\sim}{R}^{-T}\left\{\underset{\sim}{A}^T\underset{\sim}{k} - \underset{\sim}{A}^T\underset{\sim}{E}\underset{\sim}{c} + \underset{\sim}{E}^T(\underset{\sim}{f}+\underset{\sim}{k}) - \underset{\sim}{E}^T(\underset{\sim}{A}+\underset{\sim}{E})\underset{\sim}{c}\right\}$$

$$= \underset{\sim}{k} - \underset{\sim}{E}\underset{\sim}{c} + \underset{\sim}{Q}\underset{\sim}{R}^{-T}\underset{\sim}{E}^T(\underset{\sim}{k}-\underset{\sim}{E}\underset{\sim}{c}-\underset{\sim}{r}) , \tag{7.4.15}$$

using (7.4.12), where

$$\underset{\sim}{r} = \underset{\sim}{A}\underset{\sim}{c} - \underset{\sim}{f} \tag{7.4.16}$$

is the vector of residuals. Thus, using 2-norms,

$$\|\underset{\sim}{\delta f}\| \leqslant \|\underset{\sim}{k}-\underset{\sim}{E}\underset{\sim}{c}\| + \|\underset{\sim}{R}^{-1}\|\,\|\underset{\sim}{E}\|\left(\|\underset{\sim}{k}-\underset{\sim}{E}\underset{\sim}{c}\| + \|\underset{\sim}{r}\|\right)$$

$$\leqslant \|\underset{\sim}{k}-\underset{\sim}{E}\underset{\sim}{c}\| + \varkappa_2(\underset{\sim}{A})\frac{\|\underset{\sim}{E}\|}{\|\underset{\sim}{A}\|}\left(\|\underset{\sim}{k}-\underset{\sim}{E}\underset{\sim}{c}\| + \|\underset{\sim}{r}\|\right)$$

$$\leqslant \left(1+\varkappa_2(\underset{\sim}{A})\frac{\|\underset{\sim}{E}\|}{\|\underset{\sim}{A}\|}\right)\left(\|\underset{\sim}{k}\| + \|\underset{\sim}{E}\|\,\|\underset{\sim}{c}\|\right) + \varkappa_2(\underset{\sim}{A})\frac{\|\underset{\sim}{E}\|}{\|\underset{\sim}{A}\|}\|\underset{\sim}{r}\| . \tag{7.4.17}$$

The main difference between this and the corresponding result for the spline interpolation algorithm is that in the least-squares case the perturbation (or, at least, its bound) depends explicitly on the condition number and on the residual vector $\|\underset{\sim}{r}\|$.

Let

$$u = \|\underset{\sim}{c}\|/\|\underset{\sim}{f}\| \tag{7.4.18}$$

and

$$v = \varkappa_2(\underset{\sim}{A})\|\underset{\sim}{r}\|/\|\underset{\sim}{f}\| . \tag{7.4.19}$$

(The trivial case $\underset{\sim}{f} = 0$ can be ignored; it is easily verified that Algorithm 7.3.1 yields $\underset{\sim}{c} = 0$ in this case). Then, using (7.4.4) and (7.4.5), (7.4.17) yields

$$\frac{\|\underset{\sim}{\delta f}\|}{\|\underset{\sim}{f}\|} \leqslant \left\{1 + K_1\varkappa_2(\underset{\sim}{A})2^{-t}\right\}(K_1 u\|\underset{\sim}{A}\|+K_2)2^{-t} + K_1 v 2^{-t}$$

$$= \left\{1 + K_1\varkappa_2(\underset{\sim}{A})2^{-t}\right\}(K_2+K_3u)2^{-t} + K_1 v 2^{-t}, \tag{7.4.20}$$

where $K_3 (= K_1 \|\underset{\sim}{A}\|)$, like K_1 and K_2, is a "modest" function of m and q. (Note that $\|\underset{\sim}{A}\| \leq \|\underset{\sim}{A}\|_E \leq m^{\frac{1}{2}}$, since $a_{ij} \geqslant 0$ and $\sum_j a_{ij} = 1$).

We now make the following three assumptions:

(i) $u \sim 1$ (or smaller), (7.4.21)

(ii) $v \sim 1$ (or smaller), (7.4.22)

(iii) $K_1 \varkappa_2(\underset{\sim}{A}) 2^{-t} \ll 1$. (7.4.23)

Then (7.4.20) approximates to

$$\|\underset{\sim}{\delta f}\| / \|\underset{\sim}{f}\| \leq K_4 2^{-t} , \qquad (7.4.24)$$

where K_4 is a further "modest" function of m and q.

If the assumptions (7.4.21), (7.4.22) and (7.4.23) hold, the interpretation of (7.4.24) is that the computed coefficients are those of the exact least-squares spline approximation to a set of data whose ordinate vector differs only slightly in a relative sense from the actual ordinate vector.

In all practical spline-fitting problems considered to date (some 20 in all) it was found that all three assumptions were well satisfied. In particular, u was typically closer to a value of q/m than to unity, v was usually of order 10^{-1} or 10^{-2} since $\varkappa_2(\underset{\sim}{A})$ was always less than 10 and $\|\underset{\sim}{r}\| / \|\underset{\sim}{f}\|$ was of order 10^{-2} or 10^{-3}, and the value of $K_1 \varkappa_2(\underset{\sim}{A}) 2^{-t}$ was smaller than unity by several orders of magnitude. Cases in which the assumptions do not all hold appear to have to be constructed artificially and seem to occur only for badly-posed problems.

Although the three conditions cannot of course be guaranteed to hold in all practical circumstances, the first is easy to check once the solution has been obtained, and the remaining two likewise if the singular value decomposition has been employed or if $\varkappa_2(\underset{\sim}{A})$ can be estimated in some other manner. We believe the conditions <u>will</u> hold for all well-posed spline approximation problems. See Section 7.5 for some values of $\varkappa_2(A)$.

7.5 Sensitivity of the B-spline coefficients to perturbations in the data

The problem of estimating the effects of errors or perturbations in the data on the values of the B-spline coefficients and on the approximating spline itself is of considerable practical importance. We go some way towards determining such effects by employing the results of Section 2.16. In that section the bound (2.16.21) for the relative error in the computed solution of the over-determined system $\underset{\sim}{A}\underset{\sim}{x} = \underset{\sim}{b}$ in terms of bounds for the relative errors in $\underset{\sim}{A}$ and $\underset{\sim}{b}$ was established. For each of some 20 practical data sets (the bulk of which originated at the National Physical Laboratory and the British Standards Institution), very satisfactory approximations were obtained using cubic splines, and in every case the conditions assumed in establishing (2.16.21) were well satisfied. (Note that in using (2.16.21) we associate respectively $\underset{\sim}{f}$, $\underset{\sim}{c}$ and $\underset{\sim}{W}^{\frac{1}{2}}\underset{\sim}{\varepsilon}$ of this chapter with $\underset{\sim}{b}$, $\underset{\sim}{x}$ and $\underset{\sim}{r}$). In particular, (i) $\|\delta \underset{\sim}{f}\| / \|\underset{\sim}{f}\|$ and $\|\underset{\sim}{W}^{\frac{1}{2}}\underset{\sim}{\varepsilon}\| / \|\underset{\sim}{f}\|$ were of order 10^{-2} or 10^{-3}, (ii) $\mathcal{K}_2(A)$ was less than 10 in all cases and (iii)

$$\frac{\|\delta \underset{\sim}{A}\|_E}{\|\underset{\sim}{A}\|_E} = \frac{\sum_{i,j} \delta a_{ij}^2}{\sum_{i,j} a_{ij}^2} < 7(n-1)2^{-t} \tag{7.5.1}$$

(cf Section 6.5). It then follows from (2.16.21) that

$$\frac{\|\delta \underset{\sim}{c}\|_2}{\|\underset{\sim}{c}\|_2} < \frac{10}{9}\,\mathcal{K}_2(\underset{\sim}{A})\,\frac{\|\delta \underset{\sim}{f}\|_2}{\|\underset{\sim}{f}\|_2}\,, \tag{7.5.2}$$

the terms omitted being negligible for a machine such as KDF9 with t = 39. The interpretation of (7.5.2) is that a relative error bounded by μ, say, in the vector of data ordinates is amplified by a factor of about $\mathcal{K}_2(\underset{\sim}{A})$ to produce a relative error of at most $\mu\mathcal{K}_2(\underset{\sim}{A})$ in the vector of B-spline coefficients. Since the B-spline coefficients themselves provide bounds for the values of $s(x)$ (Theorem 5.1.3), we believe this result implies that (at least for cubic splines) our formulation of the problem of least-squares data approximation by splines is generally extremely well conditioned. Inequality (7.5.2) also follows from the results of Section 7.4.

In Column 3 of Table 7.5.1 we give the spectral condition numbers of $\underset{\sim}{A}$ for 10 practical cases. These 10 cases are representative of the 20-odd cases referred to earlier in this section, and include near-uniform distributions of interior knots, highly nonlinear knot distributions (such as interior knots at $x = 1, 10, 100, 1000, \ldots$) and cases of coincident interior knots. Coincident end knots were used in each case. Among these ten cases is the one with the largest condition number ($\kappa_2 = 7.7192$) yet observed. For comparison we give in

		Values of κ_2		
m	N	Coincident end knots	Distinct end knots (1)	(2)
17	5	4.7975	23.178	66.928
25	7	4.5702	26.576	18.242
26	9	4.9444	37.986	16.711
28	5	5.3270	33.505	74.305
28	5	7.7192	64.208	171.530
30	6	5.7847	28.387	40.341
31	3	5.2595	38.282	62.318
32	4	5.2592	33.628	80.670
36	10	7.5524	34.084	26.861
84	9	6.6674	54.847	33.242

Table 7.5.1 Values of the spectral condition number κ_2 of $\underset{\sim}{A}$ for a variety of data sets and three choices for the end knots

Columns 4 and 5 of Table 7.5.1 the values of $\varkappa_2$ corresponding to the choices (6.7.3) and (6.7.4) for the exterior knots. As with the use of B-splines for spline interpolation, the choice of coincident end knots is evidently to be preferred. The values of $\varkappa_2$ were obtained from the singular values of $\underset{\sim}{A}$ (Section 2.15), which were computed by reducing $\underset{\sim}{A}$ to band-triangular form using Algorithm 7.3.1, followed by the use of the Golub-Reinsch procedure 'minfit' (cf Section 6.7) to diagonalize the band triangle.

It is beyond the scope of this work to derive and to discuss in detail statistical estimates of the B-spline coefficients. However, if such estimates are required they can be obtained readily, under appropriate assumptions, as follows (cf Draper and Smith, 1968: 58 et seq). If it assumed that the values of t_i are exact, that the values of $w_i^{\frac{1}{2}}f_i$ have errors that are uncorrelated with zero mean and (generally unknown) variance σ^2, and that a spline function with the given knots is the correct model (or in practice a good approximation to the correct model), ie in statistical terms it does not suffer from lack of fit, then the following results hold:

(i) The values of c_i computed by our algorithm are unbiassed estimates of the true (unknown) coefficients.

(ii) The matrix

$$\underset{\sim}{H} = \sigma^2\underset{\sim}{G} = \sigma^2(\underset{\sim}{R}^T\underset{\sim}{R})^{-1} \tag{7.5.3}$$

provides the variances (diagonal elements) and covariances (off-diagonal elements) of the estimates.

(iii) An unbiassed estimate of σ^2 is $\|\underset{\sim}{W}^{\frac{1}{2}}\underset{\sim}{\varepsilon}\|_2^2/(m-q)$.

Note that the bulk of the computation involves the formation of the inverse $\underset{\sim}{G}$ of $\underset{\sim}{R}^T\underset{\sim}{R}$ $(=\underset{\sim}{A}^T\underset{\sim}{A})$, which can be computed efficiently by solving the two band-triangular systems

$$\underset{\sim}{R}^T \underset{\sim}{V} = \underset{\sim}{I} \tag{7.5.4}$$

and

$$\underset{\sim}{R}\underset{\sim}{G} = \underset{\sim}{V} \, . \tag{7.5.5}$$

7.6 The important case of cubic splines

There is little doubt that many practical data-fitting problems can be treated satisfactorily using splines as the approximating functions. Particularly useful are cubic splines (order $n=4$) which appear to have much to commend them. The choice of $n=4$ proves to be a good compromise between the efficient computation of the coefficients of the spline, the subsequent evaluation of $s(x)$, and a degree of approximation and smoothing power that seems to be acceptable in many circumstances.

For sufficiently accurate data we can expect, by analogy with the continuous case (Ahlberg, Nilson and Walsh, 1967: 19 et seq) that the departure of $s(x)$ from the data varies essentially as h_{max}^4, where

$$h_{max} = \max_{1 \leqslant i \leqslant N} h_i = \max_{1 \leqslant i \leqslant N} (x_i - x_{i-1}) \tag{7.6.1}$$

is the largest spacing between adjacent knots. Thus, a new approximation with additional knots inserted at points half-way between each adjacent pair of current knots, ie at $\frac{1}{2}(x_{i-1}+x_i)$ ($i = 1, 2, \ldots, N$), can be expected to have a maximum departure from the data of about 1/16 of the previous value. For many sets of practical experimental data, with typically 2 to 3 significant decimal digits in the ordinates, even if an initial approximation has barely any accuracy at all, the above insertion process carried out once or perhaps twice may well achieve an accuracy of approximation warranted by the data. In practice, the insertion of knots will not follow precisely the pattern suggested here. Because the behaviour of a spline approximation in the neighbourhood of any given argument tends to depend predominantly on data local to that argument, considerable improvements in the accuracy of the approximation in regions of poor fit can often be achieved simply by inserting additional knots in those regions. The nature of the new approximation in regions sufficiently removed from regions where knots have been inserted tends to be little changed. Of course, the discussion here has been concerned with accuracy rather than smoothness. In Section 7.8 we describe in outline a method of selecting knots that has worked successfully for many different types of data set, enabling both smooth and sufficiently accurate cubic spline approximations to be obtained.

While discussing smoothness we think it important to point out that a cubic spline (with simple knots) is the spline of lowest order that visually appears to be smooth. By this remark we mean that, in the graph of a function, most observers would be able to detect, by eye, discontinuities in value, in slope, and even in second derivative, but not in higher derivatives. The cubic spline (with simple knots), having continuity in value, first and second derivatives, is the spline of lowest order that is satisfactory from this point of view. Our belief is that the trained eye is sensitive to changes in curvature, which is of course dependent particularly on second as well as on first derivative. A spline of lower order, such as a quadratic spline, would have in general a visible change in curvature at each knot.

Although versions, in a high-level language, of Algorithm 7.3.1 for arbitrary values of n have been developed by the author, the case $n=4$ was considered sufficiently important that a code be made available specifically for it. For any particular value of n ($\leqslant 5$, say) it is possible to make various economies by tailoring Algorithm 7.3.1 specifically to the case in hand. Such a version for $n=4$, programmed in

Algol 60, appears in Cox and Hayes (1973). This version employs Gentleman's 3-multiplication rule (as in Algorithm 2.9.3), but is of course tailored (as is Algorithm 2.14.1) to the case of a stepped-banded system of bandwidth 4. This Algol code, together with code based on Algorithm 5.2.1 for evaluating s(x), accompanied by detailed documentation, are available as NPL Algorithms Library Documents E2/03/0/Algol 60/4/74 and E2/05/0/Algol 60/4/74. ANSI Standard Fortran IV versions are available as NPL Algorithms Library Documents E2/03/0/Fortran IV/11/74 and E2/05/0/Fortran IV/11/74.

It should not be inferred from the comments of this section that the cubic spline is satisfactory in all situations. We believe that it will be very suitable in the majority of practical data-fitting problems, but there will always be special circumstances in which splines of other orders are appropriate. For instance, first-degree splines (ie polygonal or piecewise-linear functions) are useful if the approximations are to be implemented on an analogue computer using diode function generators (see Cox, 1971, for a method for approximating convex functions by first-degree splines, with optimal knot selection; the method described there can be extended to the approximation of data having a convex hull). Moreover, splines of degree higher than cubic are required if certain derivatives of the approximating function are themselves to be smooth. A further consideration relates to the amount of information (ie the number of knots plus the number of B-spline coefficients) necessary to describe s(x). For example, Esch and Eastman (1969) show that for the approximation of data representative of a function in the neighbourhood of a singularity, a spline of low degree is to be preferred, whereas for a "very smooth" function such as exp(x), a spline of high degree is more economical.

7.7 Assessing the acceptability of a least-squares cubic-spline approximation

Suppose the set of data points (t_i, f_i) $(i = 1, 2, \ldots, m)$ has been approximated, using Algorithm 7.3.1, by a cubic spline $s(x)$ defined on a certain set of knots. It is important to consider whether $s(x)$ is acceptable from a number of points of view:

(i) Is the residual sum of squares $\sum_{i=1}^{m} w_i\varepsilon_i^2$ tolerably small?

(ii) Are the individual values of ε_i (or of $w_i^{\frac{1}{2}}\varepsilon_i$) tolerably small? It may well be, particularly if m is large, that a poor distribution of knots could give rise to some very small values of ε_i at the expense of others being unacceptably large (even if account is taken of the presence of the weights w_i), although the residual sum of squares is itself acceptable.

(iii) Is $s(x)$ sufficiently "smooth"?

Points (i) and (ii) are usually not difficult to answer, and to treat if necessary, since the insertion of extra knots or the re-distribution of existing knots can often result in an acceptable approximation. The "smoothness" of $s(x)$, however, is somewhat more difficult to assess and to correct. Mathematically $s(x)$ is smooth in the sense that it is twice continuously differentiable (assuming here that it is based upon distinct knots). However, a mathematically smooth function can of course exhibit oscillatory behaviour (even if it is infinitely continuously differentiable), whereas such behaviour may be absolutely unacceptable to those who require the approximation. It is important to be able to check quickly and with certainty whether any particular cubic spline approximation does indeed possess spurious oscillations or inflexions. Such oscillations and inflexions can arise, for instance, if the data has a high noise content and the chosen knot distribution is poor (see Example 7.9.3).

Now since the second derivative of the cubic spline $s(x)$ is piecewise linear, it follows that $s(x)$ has an inflexion between two (distinct) adjacent knots x_{j-1} and x_j if and only if the values of $s''(x_{j-1})$ and $s''(x_j)$ have unlike signes (cf Chapter 8, where properties of this type are used to impose conditions upon the approximating cubic spline). Thus we recommend that any algorithm for least-squares cubic-spline approximation should not only provide (or present results in such a form so as to be able to compute easily) values of quantities such as the residual sum of squares, the individual residuals and the B-spline coefficients, but also the values of $s''(x)$ at each knot x_j $(j = 0, 1, \ldots, N)$.

We now show that it is a trivial matter to compute the values of $s''(x_j)$ $(j = 0, 1, \ldots, N)$, once the B-spline coefficients c_j $(j = 1, 2, \ldots, N+3)$ have been determined. From (5.1.10),

$$s(x) = \sum_{i=1}^{N+3} c_i N_{4i}(x) \tag{7.7.1}$$

and thus

$$s''(x) = \sum_{i=1}^{N+3} c_i N_{4i}''(x) , \tag{7.7.2}$$

which, by virtue of (4.1.1) and (3.2.6), reduces to

$$s''(x) = 3 \sum_{i=1}^{N+3} c_i \left\{ M_{3,i-1}'(x) - M_{3i}'(x) \right\}$$

$$= 6 \sum_{i=1}^{N+3} c_i \left\{ \frac{M_{2,i-2}(x) - M_{2,i-1}(x)}{x_{i-1} - x_{i-4}} - \frac{M_{2,i-1}(x) - M_{2i}(x)}{x_i - x_{i-3}} \right\}. \tag{7.7.3}$$

Now, setting $x = x_j$ and noting from (3.4.1) and (3.2.8) that

$$M_{2i}(x_j) = \begin{cases} (x_{j+1}-x_{j-1})^{-1} & (i = j+1) \\ 0 & (\text{otherwise}) \end{cases}, \tag{7.7.4}$$

(7.7.3) becomes

$$s''(x_j) = \frac{6}{x_{j+1}-x_{j-1}} \left\{ \frac{c_{j+1}}{x_{j+1}-x_{j-2}} - c_{j+2}\left(\frac{1}{x_{j+1}-x_{j-2}} + \frac{1}{x_{j+2}-x_{j-1}}\right) + \frac{c_{j+3}}{x_{j+2}-x_{j-1}} \right\}$$

$$= \frac{6}{x_{j+1}-x_{j-1}} \left(\frac{c_{j+3}-c_{j+2}}{x_{j+2}-x_{j-1}} - \frac{c_{j+2}-c_{j+1}}{x_{j+1}-x_{j-2}} \right). \tag{7.7.5}$$

Note the similarity of the expression (7.7.5) to a second divided difference (cf Section 5.9). Also note that if the knots are equally spaced, viz $x_{i+1} = x_i+h$ (for all i), then (7.7.5) reduces to

$$h^2 s''(x_j) = c_{j+3} - 2c_{j+2} + c_{j+1},$$

which is identical in form to the familiar expression for a finite difference approximation to the second derivative of a function. However, here, instead of functional values, the B-spline coefficients themselves are employed (again cf Section 5.9).

Finally, it should be remarked that, at least for low-accuracy work, some form of graphical output (as in the examples of Section 7.9) is of considerable value.

7.8 The choice of knots

Sensible choices for the number and positions of the interior knots of s(x) may often be estimated in any particular instance by examining the shape of the required curve. In general, more knots will be required in regions where the behaviour of the curve is severe and fewer where it is relatively smooth. In the experience of the writer, a sensible strategy for obtaining an approximation is as follows:

Step 1. Position an initial set of knots in accordance with the above "criterion".

Step 2. Obtain the approximating spline based on these knots.

Step 3. Examine key parameters and other features of the approximation, such as the residual sum of squares or root mean square residual, the individual residuals, the values of the second derivatives at the knots, and the behaviour of the approximation in regions where there are few data points or in the neighbourhood of special features such as discontinuities (in function or derivatives), inflexion points, maxima and minima. A graphical form of output, in which the data points, the approximating spline and the knot positions are displayed, is particularly useful at this stage.

Step 4. In regions where the approximation is inadequate, introduce additional knots, perhaps after adjusting the positions of existing ones, and in regions where the approximation is "too good", ie where the approximation follows the data values so closely that the spline has oscillations with amplitudes of the order of the noise level of the data, or even greater, remove a number of knots, adjusting the positions of the remaining ones if necessary.

Step 5. Repeat as necessary from Step 2.

With a little experience in applying the above process, the writer contends that very many data sets can be approximated satisfactorily after having made typically two or perhaps three passes through the process. Some of the examples in Section 7.9 are intended to be illustrative of the approach.

If the required approximation is to have special features such as a discontinuity in slope or a very sharp peak, this knowledge in itself gives a good guide to the choice of at least some of the knots. For continuity in $s^{(r-1)}(x)$, but not in $s^{(r)}(x)$ $(0 \leqslant r < n)$ at $x = t$, a knot of multiplicity $n-r$ should be introduced at this point. Note that the case $r = 0$ corresponds to a discontinuity in $s(x)$ itself at $x = t$; such a case could be treated, but no more efficiently in fact, by computing separately spline approximations to the data to the left and to the right of $x = t$. Two of the examples in Section 7.9 illustrate the imposition of discontinuities.

7.9 Numerical examples

As for the spline interpolation algorithm of Chapter 6 we consider two types of numerical example for Algorithm 7.3.1. The first type (Example 7.9.1) is intended to demonstrate the ability of the algorithm to reproduce a cubic spline from data that itself is taken from a cubic spline, and therefore constitutes a partial test of the stability of the algorithm. The second type (Examples 7.9.2, 7.9.3 and 7.9.4) is intended to demonstrate the measure of success of cubic splines in providing approximations to data drawn from practical experimental situations. All examples were carried out on the KDF9 computer, for which $t = 39$.

Example 7.9.1

Data points (t_i, f_i) $(i = 1, 2, \ldots, m = 41)$ with $t_i = (i-1)/8$, $f_i = f(t_i)$, where

$$f(x) = 4 - (x-1)_+^0 + (x-2)_+ - 4(x-3)_+^2 + 16(x-4)_+^3 \quad , \qquad (7.9.1)$$

were selected. Since $f(x)$ is a cubic spline with knots of multiplicity 4, 3, 2 and 1 at $x = 1$, 2, 3 and 4, respectively, the data should be representable exactly, if the above knots are selected, by Algorithm 7.3.1. To measure the degree of success of the algorithm in reproducing a spline from such data, the B-spline coefficients were evaluated and compared with the exact values, which are readily verified to be those in Column 3 of Table 7.9.1. Note that the values of t_i, f_i and x_i can all be held exactly on the machine and hence any errors in the computed results are due solely to rounding errors.

Table 7.9.1 gives some of the results for this example. In particular, in Column 4 are the errors in the computed values $\bar{c}_j$ of the B-spline coefficients; in Column 5 are the true values of $s''(x_j)$ and in Column 6 the departures from these of the values computed from (7.7.5) using the computed values $\bar{c}_j$. Note that the maximum value of $\left|(\bar{c}_j - c_j)/c_j\right|$ was 10^{-11}, which is a factor of only about 6 greater than the possible relative error in rounding the true c_j-values to their machine-representable binary equivalents. The maximum departure over the 41 data points of the computed spline from the function $f(x)$ was 2.9×10^{-11}.

A graph of $s(x)$, together with the data points, is given in Fig 7.9.1. In this and subsequent figures, the knots are denoted by vertical lines and the data points thus: $\odot$.

In subsequent examples, c_j and $s''(x_j)$ denote the computed values of the B-spline coefficients and of the second derivative of the spline at the knots.

j	x_j	c_j	$10^{11}(\bar{c}_j - c_j)$	$s''(x_j)$	$10^{10}\{\bar{s}''(x_j) - s''(x_j)\}$
0	0			0	+1.3
1	1	4	-1	0	-3.9
2	1	4	0	-	-
3	1	4	+3	-	-
4	1	4	-1	0	-0.4
5	2	3	0	0	+3.5
6	2	3	-1	-	..
7	2	3	-2	0	+5.7
8	3	3	+2	0	-4.8
9	3	$3\frac{2}{3}$	-2	-8	-2.5
10	4	$4\frac{1}{3}$	+4	-8	-0.9
11	5	$5\frac{1}{3}$	-1	88	+2.0
12		$4\frac{1}{3}$	+3		
13		$-2\frac{1}{3}$	-2		
14		3	+3		

Table 7.9.1 Departures of the computed values $\bar{c}_j$ from the exact B-spline coefficients c_j and those of the computed values $\bar{s}''(x_j)$ from the exact values of $s''(x_j)$ for Example 7.9.1.

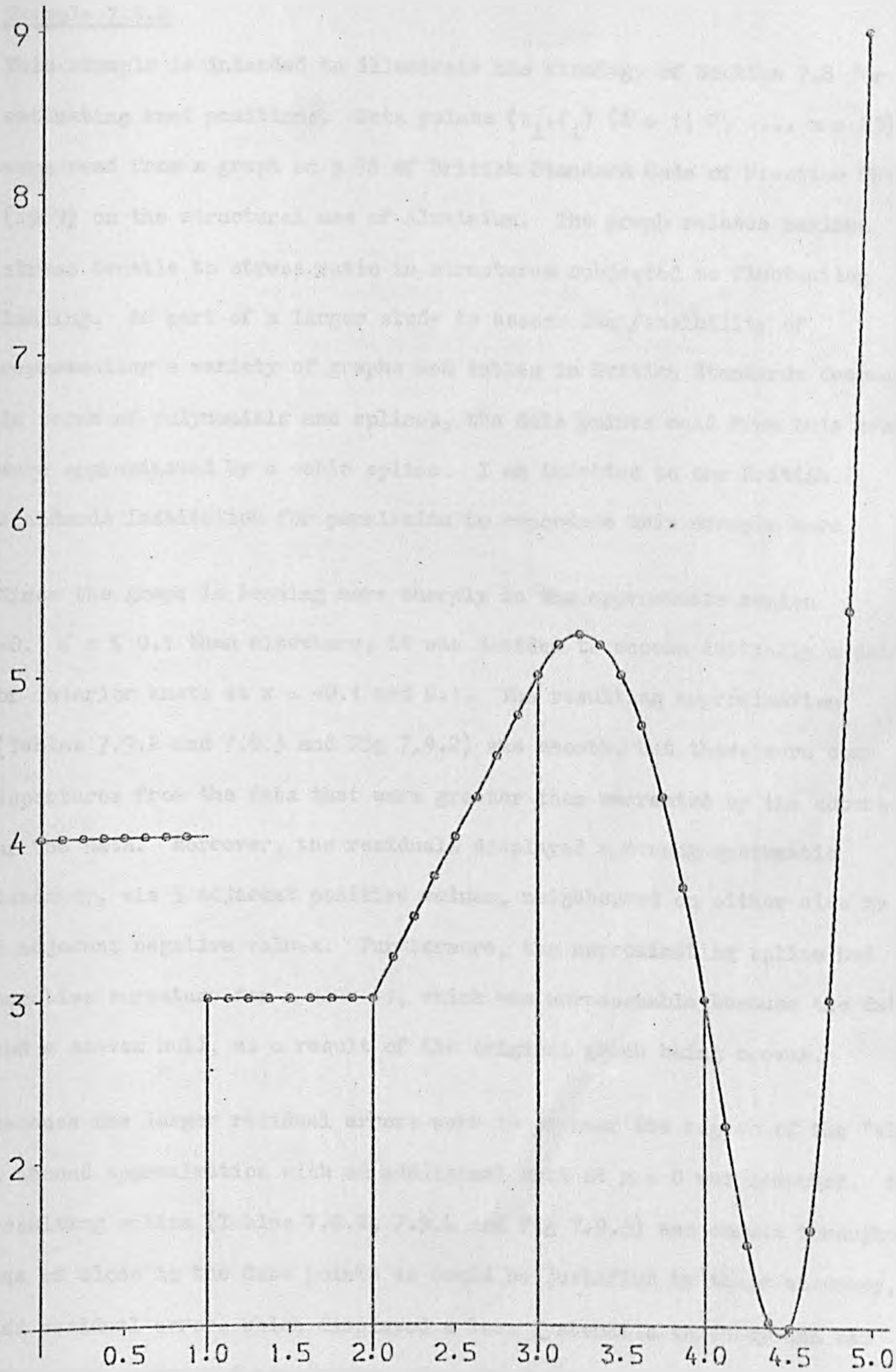

Fig. 7.9.1 Test example with knots of multiplicity 4, 3, 2 and 1

Example 7.9.2

This example is intended to illustrate the strategy of Section 7.8 for estimating knot positions. Data points (t_i, f_i) $(i = 1, 2, \ldots, m = 23)$ were read from a graph on p 78 of British Standard Code of Practice CP118 (1969) on the structural use of aluminium. The graph relates maximum stress tensile to stress ratio in structures subjected to fluctuating loading. As part of a larger study to assess the feasibility of representing a variety of graphs and tables in British Standards documents in terms of polynomials and splines, the data points read from this graph were approximated by a cubic spline. I am indebted to the British Standards Institution for permission to reproduce this example here.

Since the graph is bending more sharply in the approximate region $-0.1 \leqslant x \leqslant 0.1$ than elsewhere, it was decided to choose initially a pair of interior knots at $x = -0.1$ and 0.1. The resulting approximation (Tables 7.9.2 and 7.9.3 and Fig 7.9.2) was smooth, but there were some departures from the data that were greater than warranted by the accuracy of the data. Moreover, the residuals displayed a strong systematic tendancy, viz 5 adjacent positive values, neighboured on either side by 5 adjacent negative values. Furthermore, the approximating spline had negative curvature for x near -1, which was unreasonable because the data had a convex hull, as a result of the original graph being convex.

Because the larger residual errors were in or near the region of the "elbow", a second approximation with an additional knot at $x = 0$ was computed. The resulting spline (Tables 7.9.2, 7.9.4 and Fig 7.9.3) was convex throughout, was as close to the data points as could be justified by their accuracy, had residual errors which displayed a less systematic tendancy and was therefore considered acceptable. Note that although the residual errors for $x \geqslant 0.1$ are in the main greater than the remainder, this was considered acceptable because of the greater difficulty in reading accurately data values on the steeper parts of a graph than elsewhere. A useful refinement, not considered here, would be to incorporate weighting factors which are estimated to reflect this variable reading accuracy.

i	t_i	f_i	Values of $100\{s(t_i)-f_i\}$ for 2 knots	3 knots
1	-1.00	5.30	-5	-1
2	-0.90	5.44	+4	+1
3	-0.80	5.62	+5	0
4	-0.70	5.80	+4	0
5	-0.60	6.01	-1	-1
6	-0.50	6.20	-3	0
7	-0.40	6.42	-6	0
8	-0.30	6.67	-7	-1
9	-0.20	6.91	-2	+1
10	-0.15	7.05	+1	+1
11	-0.10	7.20	+5	0
12	-0.05	7.38	+9	-2
13	0.00	7.63	+9	-2
14	0.05	7.98	+5	+1
15	0.10	8.42	-1	+4
16	0.15	8.95	-8	+1
17	0.20	9.52	-9	-1
18	0.25	10.16	-8	-4
19	0.30	10.85	-1	-2
20	0.35	11.64	+6	+2
21	0.40	12.60	+9	+2
22	0.45	13.75	+4	+1
23	0.50	15.10	-7	-2

Table 7.9.2 Data points and values of two approximating splines for Example 7.9.2.

j	x_j	c_j	$s''(x_j)$
0	-1.0		-5.505
1	-0.1	5.247	8.806
2	0.1	6.014	34.543
3	0.5	6.043	53.476
4		8.505	
5		11.562	
6		15.026	
Residual sum of squares = 0.0804			

Table 7.9.3 B-spline coefficients and values of $s''(x)$ at the knots for the first approximation of Example 7.9.2.

j	x_j	c_j	$s''(x_j)$
0	-1.0		0.670
1	-0.1	5.292	2.307
2	0.0	5.764	64.108
3	0.1	6.390	7.371
4	0.5	7.501	86.617
5		9.390	
6		11.270	
7		15.085	
Residual sum of squares = 0.0061			

Table 7.9.4 B-spline coefficients and values of $s''(x)$ at the knots for the second approximation of Example 7.9.2.

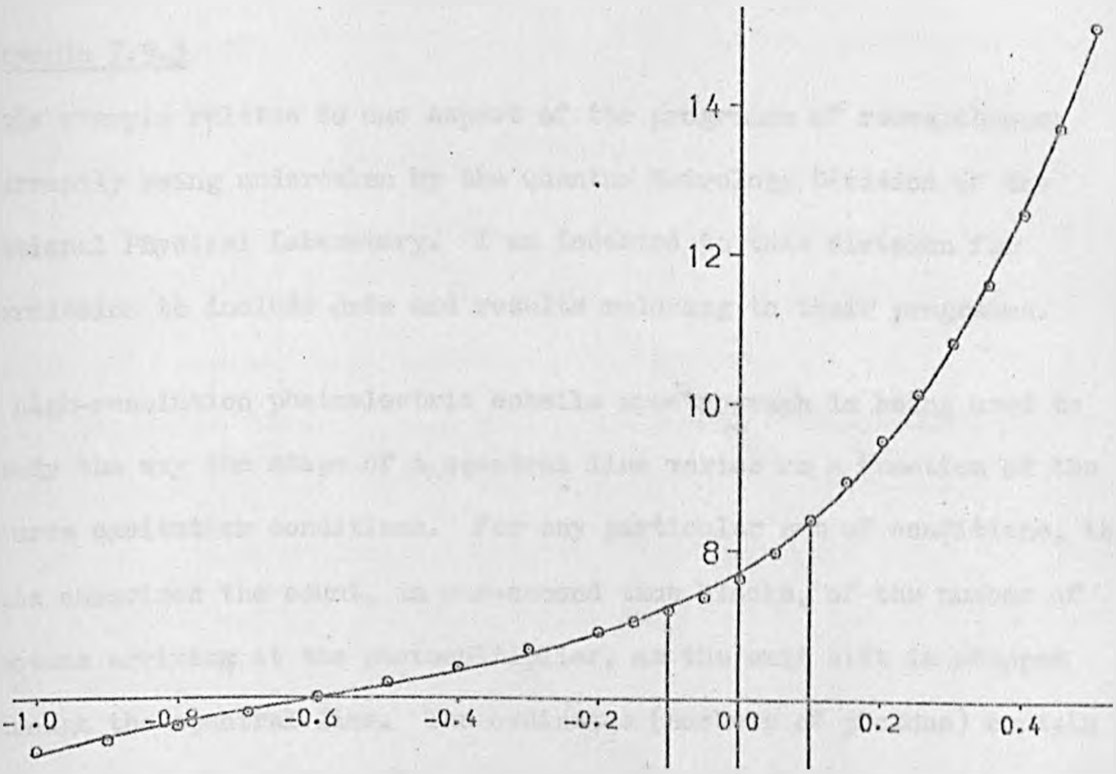

Fig 7.9.2 Maximum stress tensile distribution : 2 interior knots

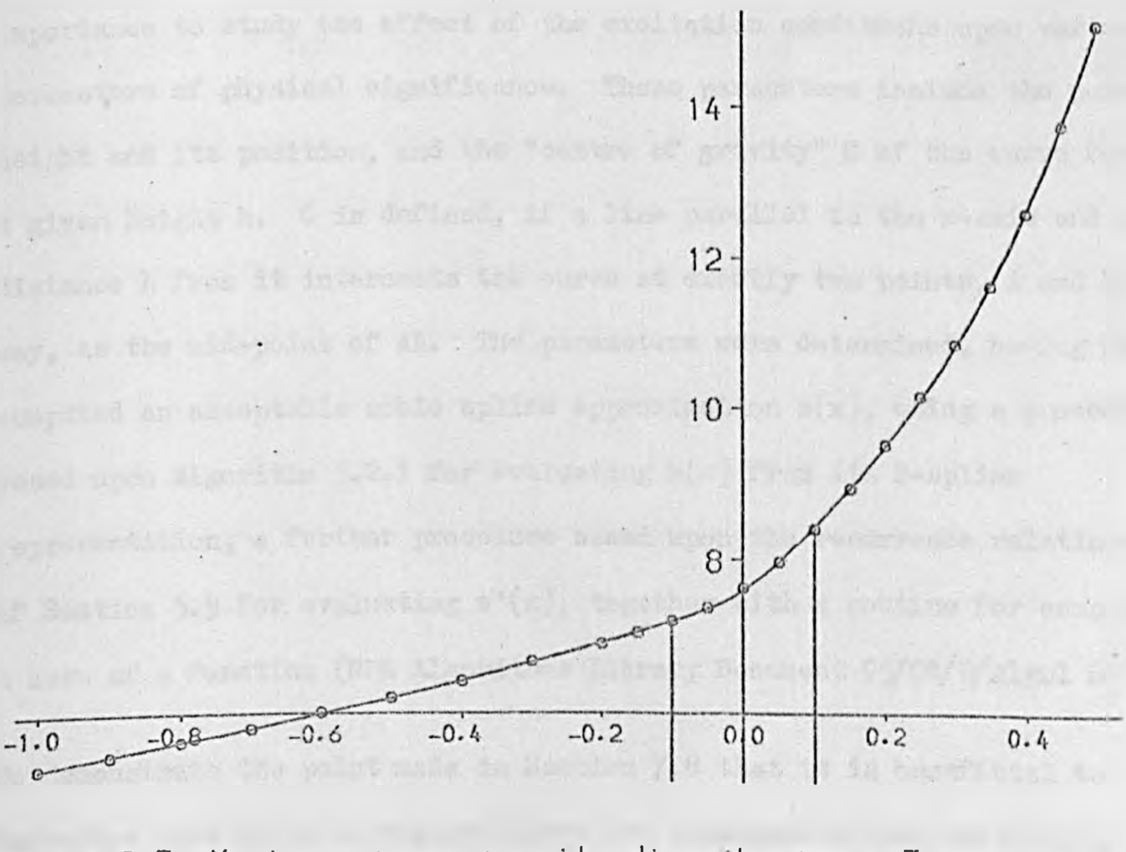

Fig 7.9.3 Maximum stress tensile distribution : 3 interior knots

Example 7.9.3

This example relates to one aspect of the programme of research work currently being undertaken by the Quantum Metrology Division of the National Physical Laboratory. I am indebted to this division for permission to include data and results relating to their programme.

A high-resolution photoelectric echelle spectrograph is being used to study the way the shape of a spectral line varies as a function of the source excitation conditions. For any particular set of conditions, the data comprises the count, in one-second time blocks, of the number of photons arriving at the photomultiplier, as the exit slit is stepped through the spectral line. The ordinates (numbers of photons) contain appreciable noise, physical considerations indicating that the probable error in an ordinate y is proportional to $y^{\frac{1}{2}}$.

The main requirement is to obtain a smooth unimodal approximation to the data for use in subsequent computations. In particular, it is of importance to study the effect of the excitation conditions upon various parameters of physical significance. These parameters include the peak height and its position, and the "centre of gravity" G of the curve for a given height h. G is defined, if a line parallel to the x-axis and a distance h from it intersects the curve at exactly two points, A and B, say, as the mid-point of AB. The parameters were determined, having first computed an acceptable cubic spline approximation $s(x)$, using a procedure based upon Algorithm 5.2.1 for evaluating $s(x)$ from its B-spline representation, a further procedure based upon the recurrence relations of Section 5.9 for evaluating $s'(x)$, together with a routine for computing a zero of a function (NPL Algorithms Library Document C5/01/0/Algol 60/1/73).

To demonstrate the point made in Section 7.8 that it is beneficial to introduce more knots in regions where the behaviour of the underlying function is severe than elsewhere, two cubic spline approximations to one of the data sets were computed. In both cases weights $w_i = y_i^{-\frac{1}{2}}$ were incorporated to reflect the knowledge of the errors in the values of y_i. The first approximation was based upon choosing 19 uniformly-spaced interior knots at $x = 10, 20, 30, \ldots, 190$ and the second upon the choice of 7 non-uniformly-spaced interior knots at $x = 30, 60, 80, 90, 100, 120, 150$. The second set of knots was chosen to cluster around the sharp peak of the curve and to be widely displaced in the tails. Summaries of the two approximations are given in Tables 7.9.5 and 7.9.6 and graphs depicting the data points, the approximating splines and the knot lines are presented as Figs 7.9.4 and 7.9.5.

The first approximation, although being adequate for the bulk of the range of the data, possesses spurious oscillations in both tails, due to the spline following the data too closely, as a result of there being an excessive number of knots in these regions. The oscillations in the left-hand tail are visually evident; that those in the right-hand tail exist follows from the sign changes in the second derivative (see Table 7.9.5).

Because of the better distribution of knots, the second approximation is satisfactory throughout the complete data range, despite the residual sum of squares being about 20% greater. Moreover, as a result of there being a smaller number of parameters as well as the knots being better placed, the approximation possesses no spurious oscillations. It could be argued that the second approximation is superior to the first in the neighbourhood of the physically-important peak (compare Figs 7.9.4 and 7.9.5).

j	x_j	c_j	$s''(x_j)$
0	1		-0.578
1	10	57.78	-0.232
2	20	67.41	0.568
3	30	71.27	-0.032
4	40	55.88	1.342
5	50	96.79	1.400
6	60	134.48	4.000
7	70	306.31	0.549
8	80	617.82	-3.256
9	90	1329.80	-8.943
10	100	2096.72	-3.966
11	110	2538.09	1.891
12	120	2085.19	3.641
13	130	1235.68	1.505
14	140	575.31	1.078
15	150	279.01	0.163
16	160	133.25	0.047
17	170	95.25	0.069
18	180	73.59	0.039
19	190	56.61	-0.030
20	200	46.57	0.095
21		40.41	
22		34.30	
23		32.82	
Residual sum of squares = 200.1			

Table 7.9.5 B-spline coefficients and values of $s''(x)$ at the knots for the first approximation of Example 7.9.3.

j	x_j	c_j	$s''(x_j)$
0	1		0.069
1	30	65.60	0.206
2	60	59.10	2.993
3	80	65.66	-1.592
4	90	234.61	-12.019
5	100	1859.31	-1.134
6	120	2624.14	3.040
7	150	1786.46	0.076
8	200	189.71	0.014
9		61.77	
10		40.98	
11		33.74	
Residual sum of squares = 238.4			

Table 7.9.6 B-spline coefficients and values of $s''(x)$ at the knots for the second approximation of Example 7.9.3.

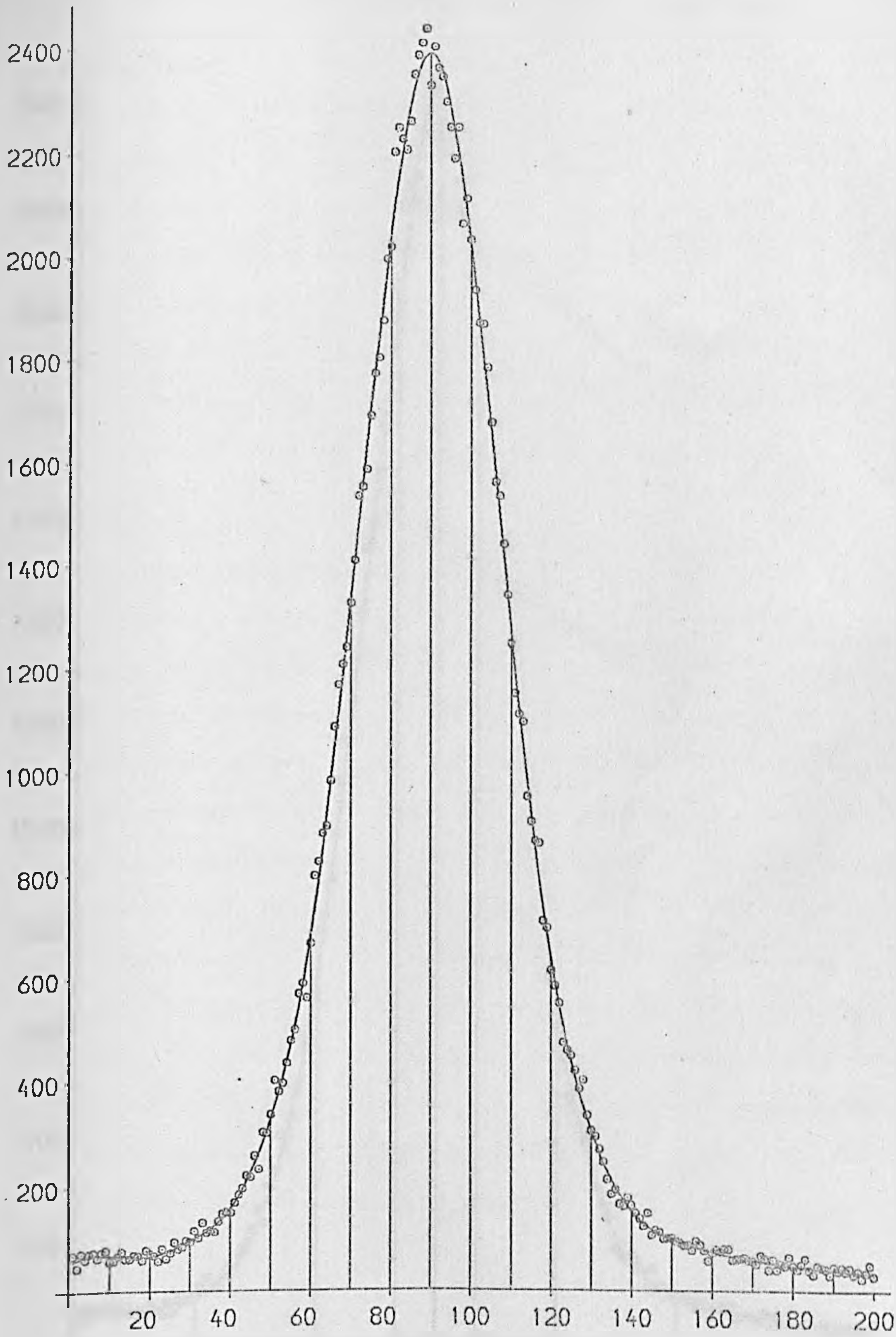

Fig 7.9.4 Photon count data : poor knot distribution

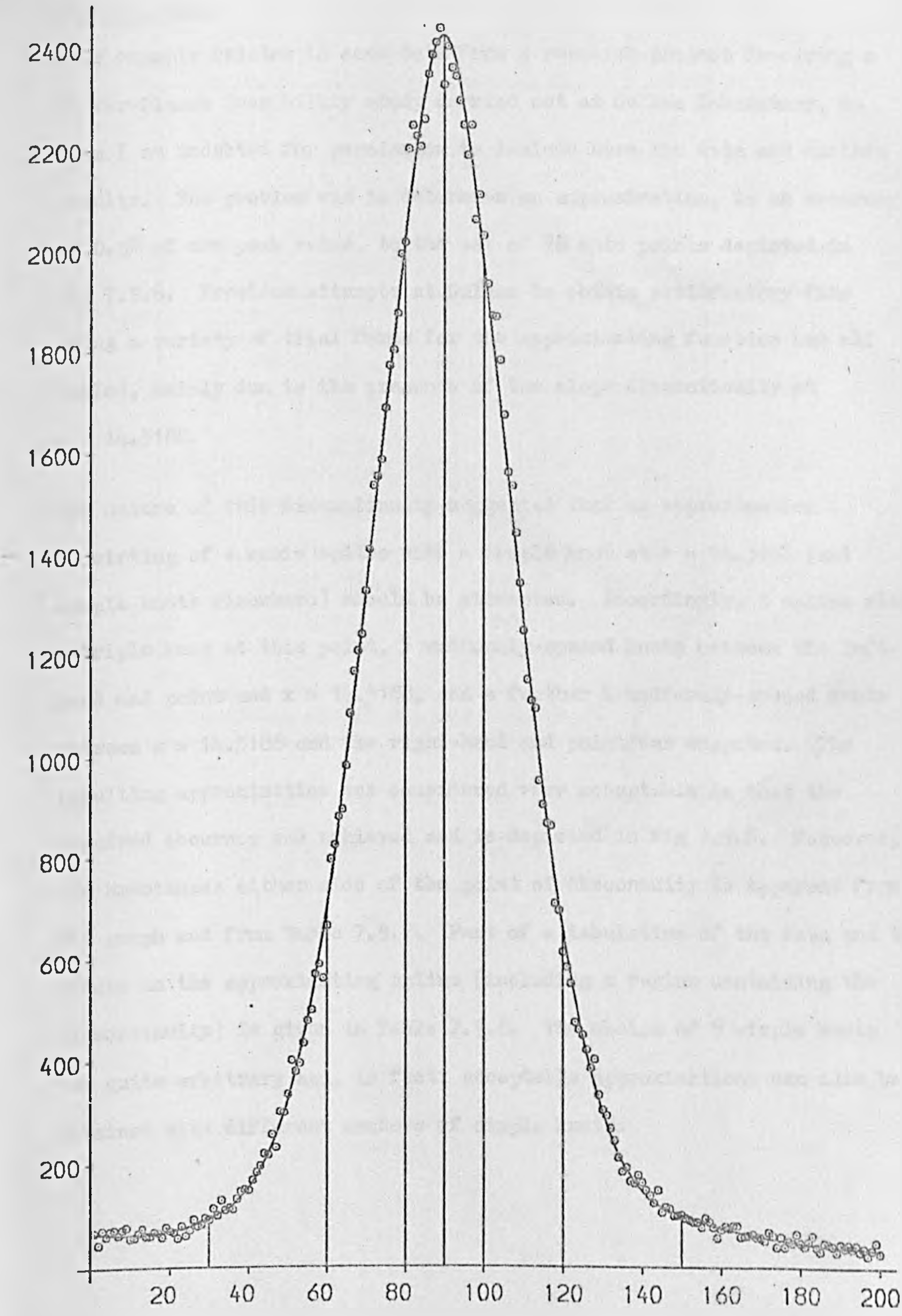

Fig 7.9.5 Photon count data : good knot distribution

Example 7.9.4

This example relates to some data from a research project involving a Fokker-Planck feasibility study carried out at Culham Laboratory, to whom I am indebted for permission to include here the data and certain results. The problem was to determine an approximation, to an accuracy of 0.5% of the peak value, to the set of 70 data points depicted in Fig 7.9.6. Previous attempts at Culham to obtain satisfactory fits using a variety of trial forms for the approximating function had all failed, mainly due to the presence of the slope discontinuity at $x = 14.3188$.

The nature of this discontinuity suggested that an approximation consisting of a cubic spline with a triple knot at $x = 14.3188$ (and simple knots elsewhere) should be attempted. Accordingly, a spline with a triple knot at this point, 5 uniformly-spaced knots between the left-hand end point and $x = 14.3188$, and a further 4 uniformly-spaced knots between $x = 14.3188$ and the right-hand end point was computed. The resulting approximation was considered very acceptable in that the required accuracy was achieved and is depicted in Fig 7.9.6. Moreover, its smoothness either side of the point of discontuity is apparent from the graph and from Table 7.9.7. Part of a tabulation of the data and the errors in the approximating spline (including a region containing the discontinuity) is given in Table 7.9.8. The choice of 9 simple knots was quite arbitrary and, in fact, acceptable approximations can also be obtained with different numbers of simple knots.

j	x_j	c_j	$s''(x_j)$
0	0.0000		0.0844
1	2.3865	0.0000	0.0916
2	4.7729	0.0007	0.0879
3	7.1594	0.1625	0.0533
4	9.5459	0.9268	-0.0195
5	11.9323	2.1918	-0.1035
6	14.3188	3.7601	-0.1618
7	14.3188	5.2175	-
8	14.3188	5.7962	0.0263
9	16.7053	5.9320	0.1434
10	19.0918	5.0095	0.1637
11	21.4782	3.2146	0.1076
12	23.8647	1.3388	0.0464
13	26.2512	0.3953	0.0035
14		0.0646	
15		0.0203	
Residual sum of squares = 8.7×10^{-5}			

Table 7.9.7 B-spline coefficients and values of $s''(x)$ at the knots for Example 7.9.4.

i	t_i	f_i	$10^4\{s(t_i)-f_i\}$
1	0.0000	0.0000	0
2	1.0696	0.0497	+1
3	1.5162	0.1002	0
4	1.8617	0.1514	-1
5	2.2887	0.2294	-2
6	2.8726	0.3627	+1
7	3.3672	0.5000	+1
8	3.8078	0.6409	+1
9	4.2111	0.7853	0
36	12.6542	5.4426	-1
37	13.0969	5.6104	-3
38	13.5362	5.7507	+1
39	13.9721	5.8631	+2
40	14.3188	5.9319	+1
41	14.5348	5.6826	-4
42	14.9643	5.1911	+1
43	15.3912	4.7133	+4
44	15.8161	4.2525	+2
62	23.2215	0.1723	+1
63	23.6241	0.1303	-2
64	24.0260	0.0963	-5
65	24.4274	0.0680	+7
66	24.8284	0.0465	+8
67	25.2287	0.0304	+3
68	25.6285	0.0183	-6
69	26.0279	0.0087	-18
70	26.2512	0.0000	+15

Table 7.9.8 Part of the tabulation of the data and the errors in the approximating spline for Example 7.9.4.

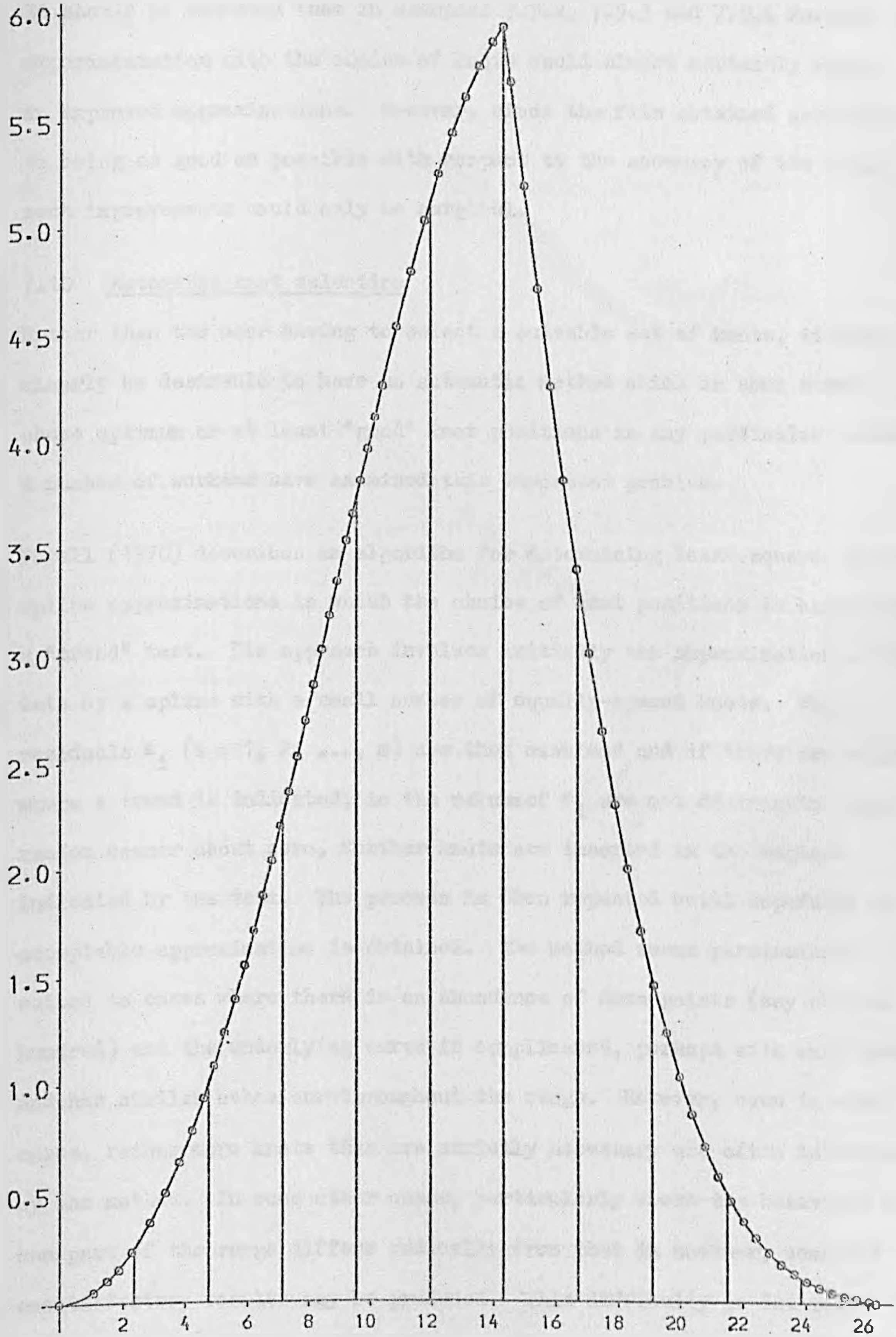

Fig 7.9.6 Fokker-Planck feasibility study data

It should be stressed that in Examples 7.9.2, 7.9.3 and 7.9.4 further experimentation with the choice of knots would almost certainly result in improved approximations. However, since the fits obtained were close to being as good as possible with respect to the accuracy of the data, such improvements could only be marginal.

7.10 Automatic knot selection

Rather than the user having to select a suitable set of knots, it would clearly be desirable to have an automatic method which in some sense chose optimum or at least "good" knot positions in any particular instance. A number of workers have examined this important problem.

Powell (1970) describes an algorithm for determining least-squares cubic-spline approximations in which the choice of knot positions is based upon a "trend" test. His approach involves initially the approximation of the data by a spline with a small number of equally-spaced knots. The residuals ε_i $(i = 1, 2, \ldots, m)$ are then examined and if there are regions where a trend is indicated, ie the values of ε_i are not distributed in a random manner about zero, further knots are inserted in the regions indicated by the test. The process is then repeated until hopefully an acceptable approximation is obtained. The method seems particularly suited to cases where there is an abundance of data points (say several hundred) and the underlying curve is complicated, perhaps with many peaks, and has similar behaviour throughout the range. However, even in such cases, rather more knots than are strictly necessary are often introduced by the method. In some other cases, particularly where the behaviour in one part of the range differs radically from that in another, somewhat unsatisfactory results may be produced. This difficulty is due mainly to the restriction the algorithm places on the rate of change of knot spacing throughout the range. It should be noted that Powell's algorithm is not based solely upon the least-squares criterion, but contains additional smoothing terms which tend to reduce the discontinuities in the third derivatives at the interior knots.

De Boor and Rice (1968) have developed an algorithm which attempts to determine a spline s(x) with as few knots as possible so that

$$\|\underset{\sim}{\varepsilon}\|_2 < \delta \tag{7.10.1}$$

where δ is a prescribed positive number. They attack this problem by solving successively for N = 1, 2, ... the least-squares <u>nonlinear</u> spline approximation problem: minimize $\|\underset{\sim}{\varepsilon}\|_2^2$ with respect to both the linear parameters <u>and</u> the interior knots of s(x). Since the value of $\|\underset{\sim}{\varepsilon}\|_2$ either decreases strictly with increasing N (Rice, 1969: 143) or, for some value of N, is equal to zero, it follows that in theory at least, N can be increased until condition (7.10.1) is satisfied.

The algorithm employed by de Boor and Rice is a method of descent. Given an initial set of N-1 interior knots (N fixed), they are improved cyclically to minimize $\|\underset{\sim}{\varepsilon}\|_2^2$. The cycle starts with the right-most knot and, working to the left, each knot is varied so as to reduce $\|\underset{\sim}{\varepsilon}\|_2^2$ as a function of this single knot. This cyclic process is continued until some criterion of convergence is met. Such a process can, of course, hope to find only <u>local</u> minima of $\|\underset{\sim}{\varepsilon}\|_2^2$. There may be many local minima (Cox, 1971) and, consequently, it is unlikely that the global minimum is obtained, unless the initial knots are sufficiently close to those corresponding to this minimum. Additional knots are introduced one at a time. A point is determined where the approximation for N-1 knots is poorest and the Nth knot is introduced midway between the two knots which bracket this point. The method has been implemented in Fortran by de Boor and Rice (1968) for the cubic-spline case n=4.

Although the method can sometimes yield very satisfactory approximations, the whole process is fraught with difficulties. Firstly, there is the problem of determining whether convergence has taken place. This problem appears on three levels, namely, for the whole algorithm, for the least-squares nonlinear spline approximation problem for any particular value of N and for the adjustment of knots within this latter problem. The decisions that convergence has taken place are made on the basis of rather delicate ad hoc numerical tests which are not infallible. Secondly, the resulting approximation may correspond to a local minimum of $\|\underset{\sim}{\varepsilon}\|_2$. As indicated in Cox (1971) there may be many local minima, many of which are far inferior to the global minimum. An example is given by Cox in which the global minimum is relatively infinitely superior to a local minimum in the sense that the former has a zero value of $\|\underset{\sim}{\varepsilon}\|_2$, whereas the latter takes a finite value. In fact the differences between the two approximations, when drawn to typical graphical accuracy, are easily discernible by eye. Thirdly, the method can easily consume enormous amounts of computation time. For instance, de Boor and Rice quote an example with a final value of N of about 30 which takes some 20 minutes computation time on the powerful IBM 7090 computer. This time is to be compared with a one of a fraction of a second for 30 fixed knots.

A somewhat different approach has been suggested more recently by de Boor (1973) (also see Dodson, 1972), in which initial estimates of the nth derivative of the function underlying the data are made. Then, using the fact that, at least for a mathematical function, the local error in an approximation by a spline of order n is proportional to the nth power of the local knot spacing and directly to the magnitude of the nth derivative of the function, he describes an algorithm for estimating "good" knot positions. He outlines a way of iterating the process in an attempt to improve further the approximation so obtained. The current writer has compared an implementation of this method with the approach discussed in Section 7.8; starting with the approximations produced by the process described in Section 7.8, in only two cases out of 20 did de Boor's approach produce a superior approximation, and even these two were only marginally better. However, de Boor's suggestion appears to be worth exploring further. It may be that certain refinements would enable a good algorithm to be developed. The main advantage compared with, say, the de Boor-Rice approach, is its speed. A major difficulty is the initial estimation of the nth derivative of the underlying function. After all, the nth derivative (even for a cubic spline, n=4) is surely much harder to estimate than the function itself, and the latter problem of course is essentially the one we wish to solve!

7.11 Least-squares spline-approximation of a mathematical function

This section is exceptional in that we consider the approximation by splines of functions rather than data. The main reason for incorporating this digression is to demonstrate that B-splines are a powerful tool in this area also, and that their use compares very favourably with other approaches (eg Bellman, Kashef and Vasudevan, 1974) that have been proposed recently.

Consider the problem of approximating in the least-squares norm the function $f(x)$ over the range $a \leqslant x \leqslant b$ by a spline $s(x)$. As usual we introduce a set of interior knots x_i $(i = 1, 2, \ldots, N-1)$ and augment this by additional knots at $x=a$ and $x=b$ so that the complete knot set $\{x_i\}$ forms a standard knot set with coincident end knots. Our approximation problem can be posed in the following way.

Determine coefficients c_i $(i = 1, 2, \ldots, q=N+n-1)$ which minimize

$$\int_a^b \left\{ \sum_{i=1}^{q} c_i N_{ni}(x) - f(x) \right\}^2 dx \,. \qquad (7.11.1)$$

The minimizing values of c_i are defined by the equations

$$\int_a^b N_{nj}(x)\left\{\sum_{i=1}^{q} c_i N_{ni}(x) - f(x)\right\} dx = 0 \quad (j = 1, 2, \ldots, q), \tag{7.11.2}$$

that is, by

$$\sum_{i=1}^{q} c_i \int_a^b N_{ni}(x) N_{nj}(x)dx = \int_a^b N_{nj}(x)f(x)dx \quad (j = 1, 2, \ldots, q). \tag{7.11.3}$$

Because of the compact support property of the B-splines, the equations (7.11.3) reduce to

$$\sum_{i=j-n+1}^{j+n-1} c_i \int_a^b N_{ni}(x)N_{nj}(x)dx = \int_a^b N_{nj}(x)f(x)dx \quad (j = 1, 2, \ldots, q). \tag{7.11.4}$$

Equations (7.11.4) constitute a system of q linear equations of bandwidth $2n-1$, symmetric about the main diagonal, which may be solved efficiently in $O(qn^2)$ operations. To determine the elements of the system it is necessary to compute the values of

$$a_{ij} = \int_a^b N_{ni}(x)N_{nj}(x)dx \qquad (|i-j| < n) \tag{7.11.5}$$

and

$$b_i = \int_a^b N_{ni}(x)f(x)dx . \tag{7.11.6}$$

Further use of the compact support of the B-splines enables (7.11.5) and (7.11.6) to be reduced to

$$a_{ij} = a_{ji} = \int_{x_{j-n}}^{x_i} N_{ni}(x)N_{nj}(x)dx \quad (i \leqslant j < i+n) \tag{7.11.7}$$

and

$$b_i = \int_{x_{i-n}}^{x_i} N_{ni}(x)f(x)dx . \tag{7.11.8}$$

There are many ways in which the integrals (7.11.7) and (7.11.8) may be evaluated. The following approach is recommended. Express (7.11.7) in the form

$$a_{ij} = \sum_{k=j-n+1}^{i} \int_{x_{k-1}}^{x_k} N_{ni}(x)N_{nj}(x)dx. \tag{7.11.9}$$

In each of the intervals (x_{k-1}, x_k) $(k = j-n+1, j-n+2, \ldots, i)$ the integrand in (7.11.9) is a polynomial of degree 2n-1 and hence the corresponding integral may be evaluated exactly by any quadrature rule that is exact for polynomials of degree 2n-1. The values of the integrand required by the quadrature rule are products of the values of B-splines of order n which may be calculated using Algorithm 3.12.2. It will not be possible in general to compute the values of b_i exactly. However, their values may be approximated by expressing b_i as

$$b_i = \sum_{k=i-n+1}^{i} \int_{x_{k-1}}^{x_k} N_{ni}(x)f(x)dx \tag{7.11.10}$$

and applying an appropriate quadrature rule to each of the integrals in (7.11.10).

In cases where the knots are equally spaced, explicit expressions for the a_{ij} in terms of B-splines of order 2n are available (see Schoenberg, 1969).

The approach we have outlined in this section is a natural use of the B-spline basis. Bellman, Kashef and Vasudevan (1974) have also considered what they term "mean square spline approximation" and have described an algorithm based on dynamic programming for the case n=4. Because they do not use a suitable basis and because they employ dynamic programming unnecessarily (in a situation where more direct methods suffice), their approach is relatively unwieldy. Moreover, we believe their approach is comparitively inefficient and also suffers from a considerable degree of ill-conditioning. It is necessary in their method to evaluate integrals of the form

$$\int_{x_{k-1}}^{x_k} x^i f(x)dx \quad (i = 0, 1, 2, 3) \tag{7.11.11}$$

and

$$\int_{x_{k-1}}^{x_k} f^2(x)dx. \tag{7.11.12}$$

CHAPTER 8

SPLINE FITTING WITH CONVEXITY AND CONCAVITY CONSTRAINTS

In this chapter a straightforward extension of some algorithms for solving unconstrained linear approximation problems in the L_1 and L_∞ norms is given. The extended algorithms allow lower or upper bounds to be placed on the parameters of the approximating function, whilst still retaining the computational efficiency of the unconstrained algorithm.

A representation of a cubic spline in terms of the values of its second derivatives at the knots and its values at the ends of the range is derived. By placing simple non-negativity or non-positivity constraints upon the values of these derivatives the spline can be forced to satisfy prescribed properties such as local convexity or concavity.

The extended linear approximation algorithms, when used in conjunction with this representation of a cubic spline, enable approximations to discrete data sets to be obtained which are free from undesirable inflexions or oscillations.

In Section 8.1 we discuss the need for constrained approximation and indicate how some important types of continuous constraints may be enforced by imposing upon a cubic spline a finite number of point constraints. In Section 8.2 we consider the formulation as linear programs of the general discrete linear L_1 and L_∞ approximation problems with simple constraints upon the parameters. In Section 8.3 we derive a representation of a cubic spline in terms of the values of its second derivatives at the knots and its values at the ends of the range. In Section 8.4 it is shown that the linear programs obtained in Section 8.2 can be used in conjunction with the representation derived in Section 8.3 to obtain cubic-spline approximations which satisfy local convexity and concavity constraints. Also in Section 8.4 we discuss briefly the numerical stability of the process. In Section 8.5 some numerical examples are given.

8.1 The need for constrained approximations

In problems of data approximation it is often important that the approximating function employed should reflect certain properties of the function underlying the data. For instance, if it is known that the underlying function is convex, then it is usually desirable that the approximating function is also convex.

In many circumstances it is found that cubic splines form good approximating functions (see, for instance, the examples in Chapter 7). Unfortunately, the algorithm discussed in Chapter 7 is not guaranteed to produce approximations that are free from spurious oscillations, although very frequently the approximations are indeed oscillation-free. However, if cubic splines are represented in an appropriate way, they can be forced to display desired local behaviour by the imposition of a finite number of very simple point constraints. In fact, many aspects of the local behaviour of a cubic spline depend upon the values of its second derivative at the knots. In particular, since the second derivative of a cubic spline $s(x)$ with simple knots is linear between any pair of adjacent knots x_{j-1} and x_j, the following types of behaviour can be forced:

(i) convexity in the interval $x_{j-1} \leq x \leq x_j$ is achieved by ensuring that both $s''(x_{j-1})$ and $s''(x_j)$ are non-negative.

(ii) concavity in the interval $x_{j-1} \leq x \leq x_j$ is achieved by ensuring that both $s''(x_{j-1})$ and $s''(x_j)$ are non-positive,

(iii) the requirements that $s(x)$ be convex for $x \leq x_{j-1}$, possess a single inflexion point in the interval $x_{j-1} \leq x \leq x_j$ and be concave for $x \geq x_j$ can be achieved by ensuring that $s''(x_i) \geq 0$ for $i \leq j-1$ and $s''(x_i) \leq 0$ for $i \geq j$.

8.2 A class of constrained linear approximation problems

Consider the approximation of the set of points (t_i, f_i) $(i = 1, 2, \ldots, m)$ by the function

$$F(\underset{\sim}{g}, x) = \sum_{j=1}^{q} g_j \phi_j(x) , \tag{8.2.1}$$

where the unknown coefficients $\underset{\sim}{g} = \{g_1, g_2, \ldots, g_q\}$ are to satisfy the constraints

$$\left.\begin{array}{ll} g_j \leqslant d_j & (j \in J_1) \\ g_j \geqslant d_j & (j \in J_2) \\ g_j \text{ unrestricted} & (j \in J_3) \end{array}\right\} . \tag{8.2.2}$$

In (8.2.2), J_1, J_2 and J_3 are distinct sets, the union of which contains precisely q elements. Define

$$e(\underset{\sim}{g}, t_i) = F(\underset{\sim}{g}, t_i) - f_i \quad (i = 1, 2, \ldots, m). \tag{8.2.3}$$

The L_1 approximation problem is to determine $\underset{\sim}{g}^*$ such that

$$\sum_{i=1}^{m} \left| e(\underset{\sim}{g}^*, t_i) \right| \leqslant \sum_{i=1}^{m} \left| e(\underset{\sim}{g}, t_i) \right| , \tag{8.2.4}$$

for all $\underset{\sim}{g}$, $\underset{\sim}{g}^*$ satisfying (8.2.2).

The L_∞ approximation problem is to determine $\underset{\sim}{g}^*$ such that

$$\max_{1 \leqslant i \leqslant m} \left| e(\underset{\sim}{g}^*, t_i) \right| \leqslant \max_{1 \leqslant i \leqslant m} \left| e(\underset{\sim}{g}, t_i) \right| \tag{8.2.5}$$

for all $\underset{\sim}{g}$, $\underset{\sim}{g}^*$ satisfying (8.2.2).

If we let

$$\alpha_j = \begin{cases} d_j - g_j & (j \in J_1) \\ g_j - d_j & (j \in J_2) \\ g_j + \alpha_{q+1} & (j \in J_3) , \end{cases} \tag{8.2.6}$$

with

$$\alpha_{q+1} = \max \left\{ 0, \max_{j \in J_3} (- g_j) \right\} , \tag{8.2.7}$$

the constraints (8.2.2) are equivalent to

$$\underset{\sim}{\alpha} \geqslant 0 , \tag{8.2.8}$$

where $\alpha = \left\{ \alpha_1, \alpha_2, \ldots, \alpha_{q+1} \right\}$. Then, putting

$$a_{ij} = \phi_j(x_i) \tag{8.2.9}$$

gives

$$e(\underset{\sim}{g}, t_i) \equiv \varepsilon(\underset{\sim}{\alpha}, t_i) = \sum_{j \in J_1} a_{ij} (d_j - \alpha_j)$$

$$+ \sum_{j \in J_2} a_{ij} (d_j + \alpha_j) + \sum_{j \in J_3} a_{ij} (\alpha_j - \alpha_{n+1}) - f_i \tag{8.2.10}$$

$$= \sum_{j \in J_1} (- a_{ij}) \alpha_j + \sum_{j \in J_2 \cup J_3} a_{ij} \alpha_j$$

$$+ \left(- \sum_{j \in J_3} a_{ij} \right) \alpha_{n+1} - \left(f_i - \sum_{j \in J_1 \cup J_2} a_{ij} d_j \right) \tag{8.2.11}$$

$$(i = 1, 2, \ldots, m).$$

We define, for $i = 1, 2, \ldots, m$,

$$\hat{a}_{ij} = \begin{cases} -a_{ij} & (j \in J_1) \\ a_{ij} & (j \in J_2 \cup J_3) \\ -\sum\limits_{k \in J_3} a_{ik} & (j = q+1) \end{cases} \tag{8.2.12}$$

and

$$\hat{f}_i = f_i - \sum_{j \in J_1 \cup J_2} a_{ij}\, d_j \, . \tag{8.2.13}$$

The L_1 approximation problem is then to determine $\underset{\sim}{\alpha}^* \geqslant 0$ such that

$$\sum_{i=1}^{m} \left| \sum_{j=1}^{q+1} \hat{a}_{ij}\, \alpha_j^* - \hat{f}_i \right| \leqslant \sum_{i=1}^{m} \left| \sum_{j=1}^{q+1} \hat{a}_{ij}\, \alpha_j - \hat{f}_i \right| \tag{8.2.14}$$

for all $\underset{\sim}{\alpha} \geqslant 0$.

The L_∞ approximation problem becomes the determination of $\underset{\sim}{\alpha}^* \geqslant 0$ such that

$$\max_{1 \leqslant i \leqslant m} \left| \sum_{j=1}^{q+1} \hat{a}_{ij}\, \alpha_j^* - \hat{f}_i \right| \leqslant \max_{1 \leqslant i \leqslant m} \left| \sum_{j=1}^{q+1} \hat{a}_{ij}\, \alpha_j - \hat{f}_i , \right| \tag{8.2.15}$$

for all $\underset{\sim}{\alpha} \geqslant 0$.

The problems are now in the form considered by Barrodale and Young (1966), except that in our formulation the parameters $\underset{\sim}{\alpha}$ are already non-negative (the first stage of Barrodale and Young's algorithm reduces the problem to contain just non-negative parameters). Barrodale and Young show that the problems can be reduced to linear programs as follows. For the L_1 approximation problem put

$$\varepsilon(\underset{\sim}{\alpha}, t_i) = u_i - v_i \, , \tag{8.2.16}$$

where $u_i, v_i \geqslant 0$, to give the following m equality constraints in non-negative variables,

$$\sum_{j=1}^{q+1} \hat{a}_{ij} \alpha_j - u_i + v_i = \hat{f}_i \quad (i = 1, 2, \ldots, m). \tag{8.2.17}$$

The problem is then solved by minimizing $\sum_{i=1}^{m} (u_i + v_i)$ subject to (8.2.17), $\underset{\sim}{\alpha} \geqslant 0$ and $u_i, v_i \geqslant 0$ $(i = 1, 2, \ldots, m)$.

For the L_∞ approximation problem put $u = \max_{1 \leqslant i \leqslant m} \left| s(\underset{\sim}{\alpha}, t_i) \right|$ to obtain the 2m constraints

$$\left.\begin{aligned} &\sum_{j=1}^{q+1} \hat{a}_{ij} \alpha_j - \hat{f}_i + u \geqslant 0 \\ &\sum_{j=1}^{q+1} \hat{a}_{ij} \alpha_j - \hat{f}_i - u \leqslant 0 \end{aligned}\right\} (i = 1, 2, \ldots, m). \tag{8.2.18}$$

This gives the linear programming problem of minimizing u subject to (8.2.18), $\underset{\sim}{\alpha} \geqslant 0$ and $u \geqslant 0$.

Efficient algorithms which exploit the specific structure of these formulations are given in Barrodale and Young (1966). Other versions of these algorithms are given by Barrodale (1967) and Barrodale and Roberts (1971).

8.3 A representation of cubic splines

We derive in this section an explicit representation of a cubic spline, which exhibits as parameters the values of the second derivative of the spline at the knots.

A cubic spline s(x) with strictly increasing knots $x_0, x_1, \ldots, x_N$ (if additional exterior knots are introduced in the usual way) can be expressed (Theorem 5.1.2) as

$$s(x) = \sum_{i=1}^{N+3} c_i N_{4i}(x) \tag{8.3.1}$$

for $x \in [a, b] \equiv [x_0, x_N]$. We intend to express the values of c_i $(i = 2, 3, \ldots, N+2)$ in terms of those of c_1, c_{N+3} and s''_j $(j = 0, 1, \ldots, N)$. Here s''_j denotes the values of $s''(x_j)$. Now

$$s''_j = \sum_{i=1}^{N+3} c_i N''_{4i}(x_j) \quad (j = 0, 1, \ldots, N) \tag{8.3.2}$$

which, because of the compact support property of the B-splines, reduces to

$$s''_j = \sum_{i=j+1}^{j+3} c_i N''_{4i}(x_j) \quad (j = 0, 1, \ldots, N). \tag{8.3.3}$$

We re-write equations (8.3.3) as follows:

$$\begin{bmatrix} N''_{42}(x_0) & N''_{43}(x_0) & & & & & \underset{\sim}{0} \\ N''_{42}(x_1) & N''_{43}(x_1) & N''_{44}(x_1) & & & & \\ & N''_{43}(x_2) & N''_{44}(x_2) & N''_{45}(x_2) & & & \\ & & \cdots\cdots & \cdots\cdots & & & \\ & & N''_{4,N-1}(x_{N-2}) & N''_{4,N}(x_{N-2}) & N''_{4,N+1}(x_{N-2}) & & \\ & & & N''_{4,N}(x_{N-1}) & N''_{4,N+1}(x_{N-1}) & N''_{4,N+2}(x_{N-1}) & \\ \underset{\sim}{0} & & & & N''_{4,N+1}(x_N) & N''_{4,N+2}(x_N) & \end{bmatrix} \begin{bmatrix} c_2 \\ c_3 \\ c_4 \\ \cdots \\ c_N \\ c_{N+1} \\ c_{N+2} \end{bmatrix} =$$

$$\begin{bmatrix} -N''_{41}(x_0) & 1 & & & & & & \underset{\sim}{0} & \\ & & 1 & & & & & & \\ & & & 1 & & & & & \\ & & & & \ddots & & & & \\ & & & & & 1 & & & \\ & & & & & & 1 & & \\ \underset{\sim}{0} & & & & & & & 1 & -N''_{4,N+3}(x_N) \end{bmatrix} \begin{bmatrix} c_1 \\ s''_0 \\ s''_1 \\ s''_2 \\ \cdots \\ s''_{N-2} \\ s''_{N-1} \\ s''_N \\ c_{N+3} \end{bmatrix} . \qquad (8.3.4)$$

The use of relations (4.1.1), (3.4.2) and (3.2.9) gives

$$\left.\begin{aligned} N''_{4,j+1}(x_j) &= 6(x_{j+1} - x_{j-2})^{-1}(x_{j+1} - x_{j-1})^{-1} \\ N''_{4,j+3}(x_j) &= 6(x_{j+2} - x_{j-1})^{-1}(x_{j+1} - x_{j-1})^{-1} \\ N''_{4,j+2}(x_j) &= -N''_{4,j+1}(x_j) - N''_{4,j+3}(x_j). \end{aligned}\right\} \qquad (8.3.5)$$

Insertion of the relations (8.3.5) in (8.3.4) and the multiplication of row j+1 ($j = 0, 1, \ldots, N$) by the factor $-\frac{1}{6}(x_{j+1} - x_{j-1})$ and putting

$k_j = (x_j - x_{j-3})^{-1}$ yields the system

$$\underset{\sim}{A}\underset{\sim}{c} = \underset{\sim}{B}\underset{\sim}{g} \ , \tag{8.3.6}$$

where

$$\underset{\sim}{c} = \left\{c_2, c_3, \ldots, c_{N+2}\right\} \tag{8.3.7}$$

and

$$\underset{\sim}{g} = \left\{c_1, s_0'', s_1'', \ldots, s_N'', c_{N+3}\right\} , \tag{8.3.8}$$

$\underset{\sim}{A}$ is the (N+1) by (N+1) symmetric triple-diagonal matrix with elements

$$\left.\begin{aligned} a_{j+1,j+1} &= k_{j+1} + k_{j+2} \quad (j = 0, 1, \ldots, N), \\ a_{j,j+1} &= a_{j+1,j} = - k_{j+1} \quad (j = 1, 2, \ldots, N) \end{aligned}\right\} \tag{8.3.9}$$

and $\underset{\sim}{B}$ is the (N+1) by (N+3) matrix whose only non-zero elements are

$$\left.\begin{aligned} & b_{11} = k_1 \ , \quad b_{N+1,N+3} = k_{N+2}, \\ & b_{j+1,j+2} = -\frac{1}{6}\,(x_{j+1} - x_{j-1}) \quad (j = 0, 1, \ldots, N). \end{aligned}\right\} \tag{8.3.10}$$

Since $\underset{\sim}{A}$ has a dominant main diagonal it is positive definite and of full rank. Hence

$$\underset{\sim}{c} = \underset{\sim}{H}\,\underset{\sim}{g} \ , \tag{8.3.11}$$

where $\underset{\sim}{H}$ is defined uniquely by

$$\underset{\sim}{A}\underset{\sim}{H} = \underset{\sim}{B} \ . \tag{8.3.12}$$

The (N+1) by (N+3) matrix $\underset{\sim}{H}$ can be computed in an efficient and numerically stable manner by forming the Cholesky factorization

$$\underset{\sim}{A} = \underset{\sim}{L}\underset{\sim}{D}\underset{\sim}{L}^T \ , \tag{8.3.13}$$

where $\underset{\sim}{L}$ is lower unit bi-diagonal and $\underset{\sim}{D}$ is diagonal, followed by the appropriate forward- and back-substitutions to form the columns of $\underset{\sim}{H}$ from the corresponding columns of $\underset{\sim}{B}$. It follows that

$$c_{j+2} = \sum_{r=1}^{N+3} h_{j+1,r}\, g_r \qquad (j = 0, 1, \ldots, N). \tag{8.3.14}$$

Thus

$$s(x) = c_1 N_{41}(x) + \sum_{i=2}^{N+2} \sum_{r=1}^{N+3} h_{i-1,r}\, g_r N_{4i}(x) + c_{N+3} N_{4,N+3}(x). \tag{8.3.15}$$

So, recalling that $c_1 = g_1$ and $c_{N+3} = g_{N+3}$, we obtain

$$s(x) = g_1 \left\{ N_{41}(x) + \sum_{i=2}^{N+2} h_{i-1,1} N_{4i}(x) \right\} + \sum_{r=2}^{N+2} g_r \left\{ \sum_{i=2}^{N+2} h_{i-1,r} N_{4i}(x) \right\}$$
$$+ g_{N+3} \left\{ \sum_{i=2}^{N+2} h_{i-1,N+3} N_{4i}(x) + N_{4,N+3}(x) \right\}. \tag{8.3.16}$$

Thus, defining,

$$\left. \begin{aligned} h_{o1} &= h_{N+2,N+3} = 1, \\ h_{o,r+1} &= h_{N+2,r} = 0 \quad (r = 1, 2, \ldots, N+2) , \end{aligned} \right\} \tag{8.3.17}$$

we have

$$s(x) = \sum_{r=1}^{N+3} g_r \phi_r(x) , \tag{8.3.18}$$

where

$$\phi_r(x) = \sum_{i=1}^{N+3} h_{i-1,r}\, N_{4i}(x) , \tag{8.3.19}$$

which is a representation of the cubic spline $s(x)$ in terms of the N+3 basis functions $\phi_r(x)$ $(r = 1, 2, \ldots, N+3)$.

In some previous work on this problem (Cox, 197.) a different approach was used, in which a representation of cubic splines in terms of a set of blended cubic arcs was employed. The jth such arc ($j = 1, 2, \ldots, N$) applied for $x_{j-1} \leqslant x \leqslant x_j$ and was defined in terms of the values of $s(x)$ and $s''(x)$ at $x = x_{j-1}$ and at $x = x_j$. Together with appropriate conditions to ensure the continuity of $s'(x)$, the use of this representation, which is given in Ahlberg, Nilson and Walsh (1967), also gave rise to a symmetric positive definite triple-diagonal system, but of order $N-1$ rather than $N+1$. The main reason for using the B-spline approach here is that its generality enables it to be extended more readily than other approaches to constrained spline-approximation problems of arbitrary degree. Specifically, a spline of order n, expressed in terms of B-splines, can be represented in a form which exhibits as parameters the values of its derivatives of order n-2 at the knots. Thus approximating functions can be constructed which enable conditions on particular derivatives to be imposed. For example, the use of a quadratic spline enables conditions to be placed on the first derivative (monotonicity); the use of a quartic spline enables conditions to be placed on the third derivative. Another important reason for using the B-spline basis relates to the evaluation of the derivatives in the generalization of equations (8.3.4) to the case of splines of order n. In this generalization all non-zero values of $N_{ni}^{(n-2)}(x_j)$ have to be evaluated; it was established in Section 4.3 (Theorem 4.3.3) that these values can be formed in an unconditionally stable manner.

8.4 Constrained cubic-spline approximation

We now return to the problem of approximating discrete data sets in the L_1 or L_∞ norms by cubic splines satisfying certain prescribed properties.

Suppose a set of data points (t_i, f_i) $(i = 1, 2, \ldots, m)$ is given, together with a strictly increasing set of knots $x_0, x_1, \ldots, x_N$, such that $x_0 \leqslant \min_i t_i$ and $x_N \geqslant \max_i t_i$. Additional exterior knots are added in the usual way such that the complete set forms a standard knot set. At the position of each knot x_j $(j = 0, 1, \ldots, N)$ the approximating spline $s(x)$ is to be

(i) locally convex (ie to possess a non-negative second derivative),

(ii) locally concave (ie to possess a non-positive second derivative),

or

(iii) unrestricted.

In terms of the representation (8.3.18) and (8.3.19) and recalling (8.3.8) this requirement is equivalent to

(i) $\varepsilon_{j+2} \geqslant 0$,

(ii) $\varepsilon_{j+2} \leqslant 0$, or

(iii) ε_{j+2} unrestricted.

But this formulation is just that discussed in Section 8.2, and hence can be solved by the method described there.

In some problems, it may be important that the value of the second derivative does not fall below (or above) a prescribed critical value. In such cases conditions (i) and (ii) are replaced by $\varepsilon_{j+2} \geqslant d_{j+2}$ or $\varepsilon_{j+2} \leqslant d_{j+2}$ as appropriate, where d_{j+2} denotes the critical value.

Other methods for finding constrained cubic-spline approacimations have been proposed by a number of authors including Rabinowitz (1968), Amos and Slater (1969) and LaFata and Rosen (1970). All these methods introduce additional equations to describe the constraints, rather than use an explicit representation of the spline which enables the constraints to be dealt with at virtually no extra cost, as we have suggested here. As a consequence their methods appear to be somewhat inefficient as regards both storage and computer time. Some of these methods also appear to suffer from a certain degree of ill-conditioning. Amos and Slater (1969) use the L_2 norm and solve the resulting quadratic program using the Theil-Van de Panne procedure, a method which is known to be very inefficient (Boot, 1964). Furthermore, they employ the representation

$$s(x) = \varepsilon_1 + \varepsilon_2 x + \varepsilon_3 x^2 + \varepsilon_4 x^3 + \sum_{j=5}^{q} \varepsilon_j (x - x_{j-4})_+^3 \qquad (8.4.1)$$

for the spline, which is a particularly poorly-conditioned form for numerical purposes (Carasso, 1966). Rabinowitz (1968) also suggests the use of this ill-conditioned representation in the solution of such problems in the L_∞ norm. LaFata and Rosen (1970) use the L_1 and L_∞ norms and, as a basis for $s(x)$, they employ B-splines, but consider only equally-spaced knots. Moreover they compute the required values of the B-splines from the unstable explicit formula (3.2.4), rather than from the numerically stable recurrence relation (3.4.1) or (3.4.2).

The method we employ appears to compare favourably with the above methods. It results in a relatively short computer code; for either the L_1 or the L_∞ norm, the complete procedure, including the code for the solution of the linear program, and for the monitoring of the growth factor (see below), contains only about 250 Algol statements.

It is now becoming widely recognized that, because the Gauss-Jordan elimination process without a pivotal strategy is employed, many of the existing linear programming codes (including those of Barrodale and Young (1966), Barrodale (1967) and Barrodale and Roberts (1971) for solving discrete linear L_1 and L_∞ approximation problems) are <u>potentially</u> unstable in that severe error growth (which in certain cases could swamp the true solution) may occur. It is to be expected, by analogy with Gaussian elimination (Wilkinson, 1965: 214; Reid, 1971), that a good indication of the loss of accuracy in such an implementation of the simplex method is given by the growth of the magnitudes of the elements in the tableau $\{a_{ij}\}$. A range of some fifteen examples, having from 10 to 100 data points, 2 to 10 knots and knot spacing ratios* from 1 to 20, were solved on an English Electric KDF9 computer. In each case the "growth factor" g, defined by

$$g = \max_{i,j,k} \left| a_{ij}^{(k)} \right| \Big/ \max_{i,j} \left| a_{ij}^{(0)} \right| , \tag{8.4.2}$$

was computed. In (8.4.2), $a_{ij}^{(k)}$ denotes the value of a_{ij} after k iterations of the simplex method. The largest value of g observed was approximately 10^4, indicating a loss of about four decimal figures (out of the 39 binary, or about 12 decimal, figures available on KDF9) in the computation. In many cases the value of g was less than 10, indicating a loss of at most one decimal digit; in some cases g was unity, indicating essentially no error growth at all. The size of g seemed to be unrelated to m, N or the knot spacing ratio. This reasonably encouraging evidence does not of course imply that we can preclude the possibility that, in some applications, completely unreliable results may be obtained. It is recommended therefore that the growth factor be computed and examined before the results are accepted. An efficient method for computing the growth factor has been given by Businger (1971).

In recent years, numerically stable algorithms for the simplex method, based upon triangular decomposition (Bartels and Golub, 1969) and upon

---

* The knot spacing ratio is defined by $\max_{1 \leq j \leq N} (x_j - x_{j-1}) \Big/ \min_{1 \leq j \leq N} (x_j - x_{j-1})$.

orthogonal decomposition (Gill and Murray, 1973), have appeared, which avoid the difficulties associated with the possibility of severe error growth. It is hoped that future variants of the spline-fitting algorithms discussed here will incorporate versions of one of these stable simplex methods, tailored to take advantage of the features of the approximation problem.

8.5 Numerical examples

We present two examples which, for the purpose of concise presentation, are small, but nevertheless illustrate some of the advantages of constrained approximation. The L_1 norm was considered appropriate in both cases. The results were obtained using the KDF9 computer, which has a floating-point word containing 39 binary digits in the mantissa.

For each example we give firstly an unconstrained approximation based on a prescribed set of knots, and secondly an approximation, based on the same set of knots, constrained to possess certain properties of the underlying function. The growth factors and the mean absolute residuals $\sum_{i=1}^{m} \left| s(t_i) - f_i \right| / m$ are also quoted.

Example 8.5.1

Data: Temperature distribution (Amos and Slater, 1969) - Table 8.5.1.

i	t_i	f_i
1	0.25	17.0
2	0.50	15.2
3	0.75	13.8
4	1.25	12.2
5	1.75	11.0
6	2.25	10.1
7	2.75	9.4
8	3.25	8.6
9	6.25	6.1
10	12.25	3.5

Table 8.5.1 Temperature distribution data

Property required: Convexity.

Interior knots: Those chosen by Amos and Slater, viz $x = 1.6, 2.5, 6.0$.

Approximation 1: Unconstrained - Tables 8.5.2 and 8.5.3 and Fig 8.5.1.

Growth factor: 104.

Comment: The approximation is unacceptable since $s(x)$ is concave for $8.08 \leqslant x \leqslant 12.25$ (Table 8.5.2 and Fig 8.5.1).

j	x_j	c_j	s_j''
0	0.25		8.53235
1	1.60	17.0000	0.62750
2	2.50	13.3075	0.34528
3	6.00	11.4728	0.31493
4	12.25	8.1373	-0.63193
5		4.6560	
6		6.4586	
7		3.5000	

Table 8.5.2 B-spline coefficients and values of the second derivative at the knots for the unconstrained spline approximation to the temperature distribution data of Example 8.5.1.

i	$s(t_i)$	$s(t_i) - f_i$
1	17.0000	0.0000
2	15.2000	0.0000
3	13.8418	0.0418
4	12.0847	-0.1153
5	11.0000	0.0000
6	10.1000	0.0000
7	9.3067	-0.0933
8	8.6000	0.0000
9	6.1000	0.0000
10	3.5000	0.0000
Mean absolute residual =		0.0250

Table 8.5.3 Unconstrained spline approximation to the temperature distribution data of Example 8.5.1.

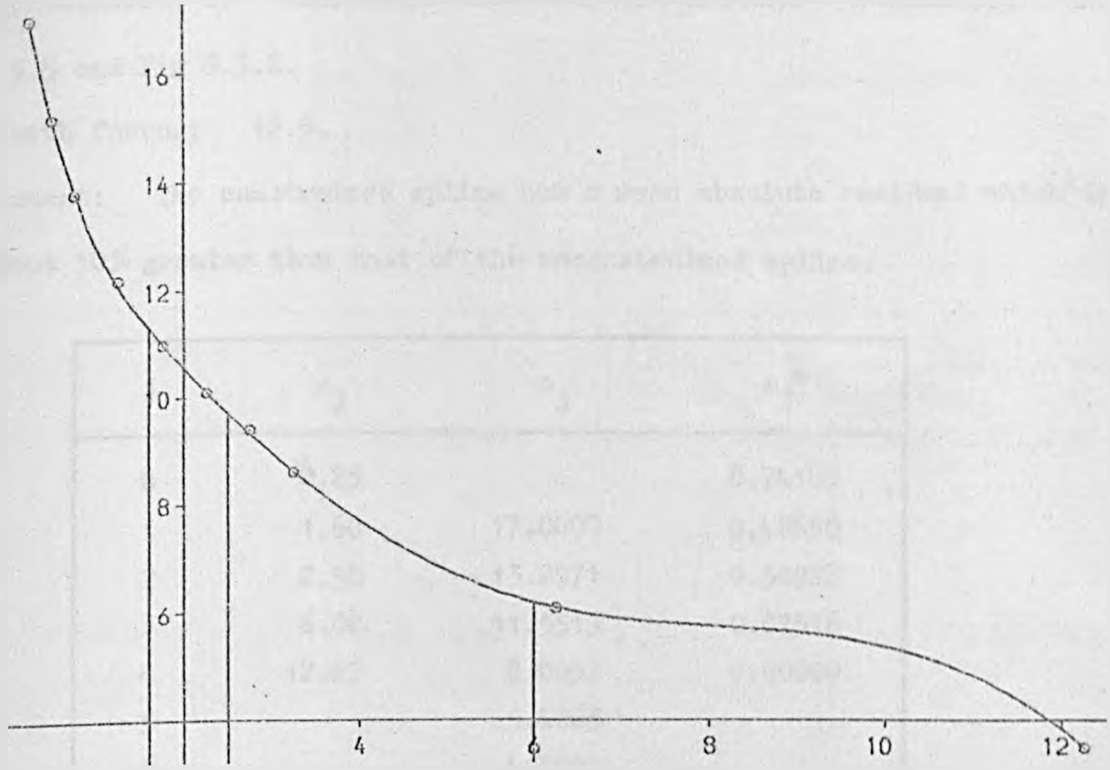

Fig 8.5.1 Temperature distribution : unconstrained spline approximation

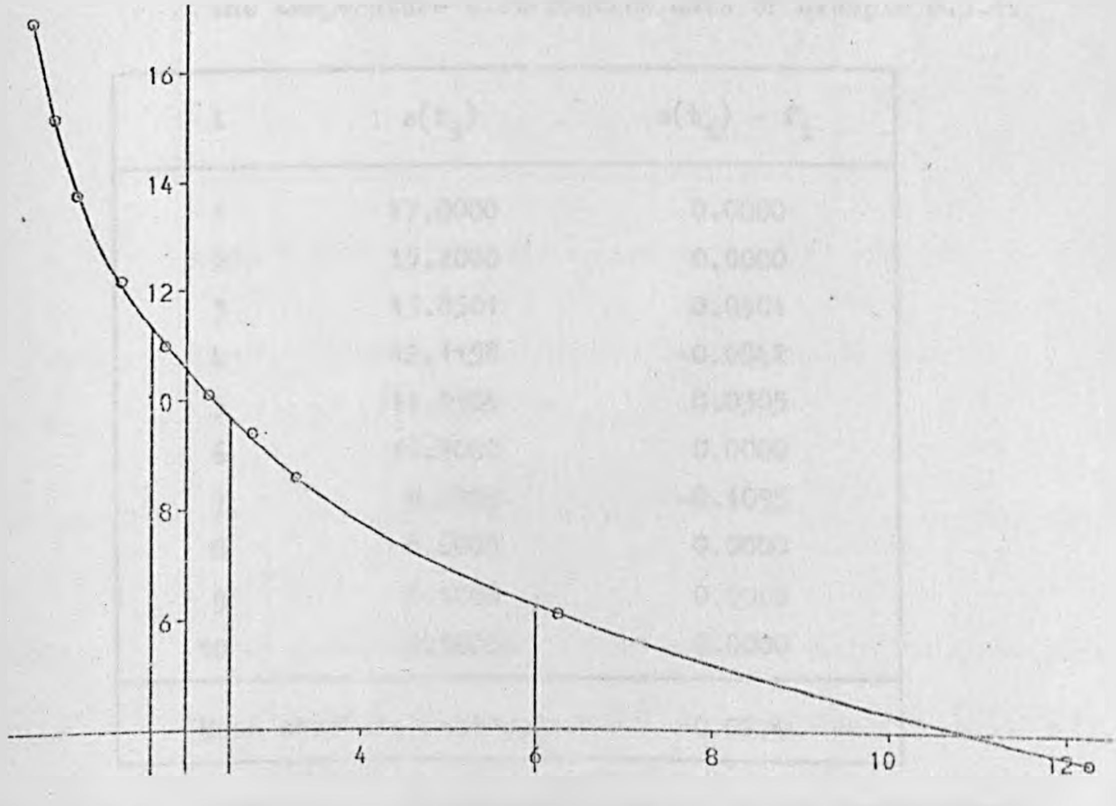

Fig 8.5.2 Temperature distribution : convex spline approximation

Approximation 2: Convexity constraint at each knot - Tables 8.5.4 and 8.5.5 and Fig 8.5.2.

Growth factor: 12.5.

Comment: The constrained spline has a mean absolute residual which is about 10% greater than that of the unconstrained spline.

j	x_j	c_j	s_j''
0	0.25		8.74185
1	1.60	17.0000	0.42650
2	2.50	13.2971	0.50928
3	6.00	11.5513	0.07516
4	12.25	8.0092	0.00000
5		5.4263	
6		4.2525	
7		3.5000	

Table 8.5.4 B-spline coefficients and values of the second derivative at the knots for the constrained spline approximation to the temperature distribution data of Example 8.5.1.

i	$s(t_i)$	$s(t_i) - f_i$
1	17.0000	0.0000
2	15.2000	0.0000
3	13.8501	0.0501
4	12.1158	-0.0842
5	11.0305	0.0305
6	10.1000	0.0000
7	9.2905	-0.1095
8	8.6000	0.0000
9	6.1000	0.0000
10	3.5000	0.0000
Mean absolute residual =		0.0274

Table 8.5.5 Constrained spline approximation to the temperature distribution data of Example 8.5.1.

Example 8.5.2

Data: Stress distribution for axially loaded aluminium struts (British Standard Code of Practice CP118, 1969) - Table 8.5.6.

i	t_i	f_i
1	1.05	19.9
2	1.1	17.8
3	1.2	14.8
4	1.3	13.0
5	1.4	12.1
6	1.5	11.4
7	1.6	10.7
8	1.7	10.0
9	1.8	9.4
10	1.9	8.6
11	2.0	7.9
12	2.1	7.1
13	2.2	6.4
14	2.3	5.6
15	2.4	4.3
16	2.5	2.3
17	2.588	0.0

Table 8.5.6 Stress distribution data

Property required: S-shaped (ie exactly one inflexion point).

Interior knots: $x = 1.2, 1.5, 2.1, 2.4$.

Approximation 1: Unconstrained - Tables 8.5.7 and 8.5.8 and Fig 8.5.3.

Growth factor: 75.4.

Comment: The approximation is unacceptable since $s(x)$ has three points of inflexion (Table 8.5.7 and Fig 8.5.3). The spurious oscillations can be seen clearly by sighting Fig 8.5.3 in the plane of the paper and looking along the curve.

j	x_j	c_j	s_j''
0	1.05		190.403
1	1.2	19.9000	121.997
2	1.5	17.5715	-7.858
3	2.1	12.7280	5.439
4	2.4	11.0338	-69.156
5	2.588	7.6831	-66.965
6		5.5328	
7		1.8235	
8		0.0000	

Table 8.5.7 B-spline coefficients and values of the second derivative at the knots for the unconstrained spline approximation to the stress distribution data of Example 8.5.2.

i	$s(t_i)$	$s(t_i) - f_i$
1	19.9000	0.0000
2	17.8000	0.0000
3	14.8000	0.0000
4	13.0238	0.0238
5	12.0348	-0.0652
6	11.4000	0.0000
7	10.7625	0.0625
8	10.0685	0.0685
9	9.3403	-0.0597
10	8.6000	0.0000
11	7.8698	-0.0302
12	7.1718	0.0718
13	6.4830	0.0830
14	5.6000	0.0000
15	4.2741	-0.0259
16	2.3000	0.0000
17	0.0000	0.0000
Mean absolute residual =		0.0289

Table 8.5.8 Unconstrained spline approximation to the stress distribution data of Example 8.5.2.

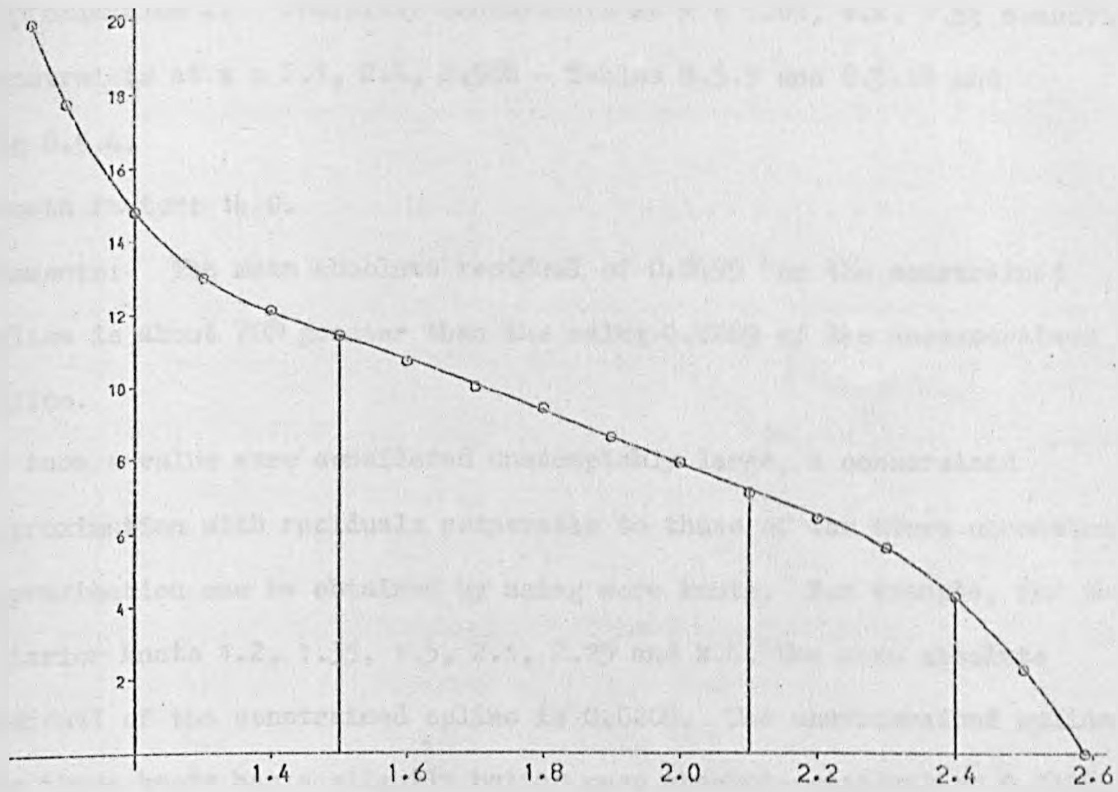

Fig 8.5.3 Stress distribution : unconstrained spline approximation

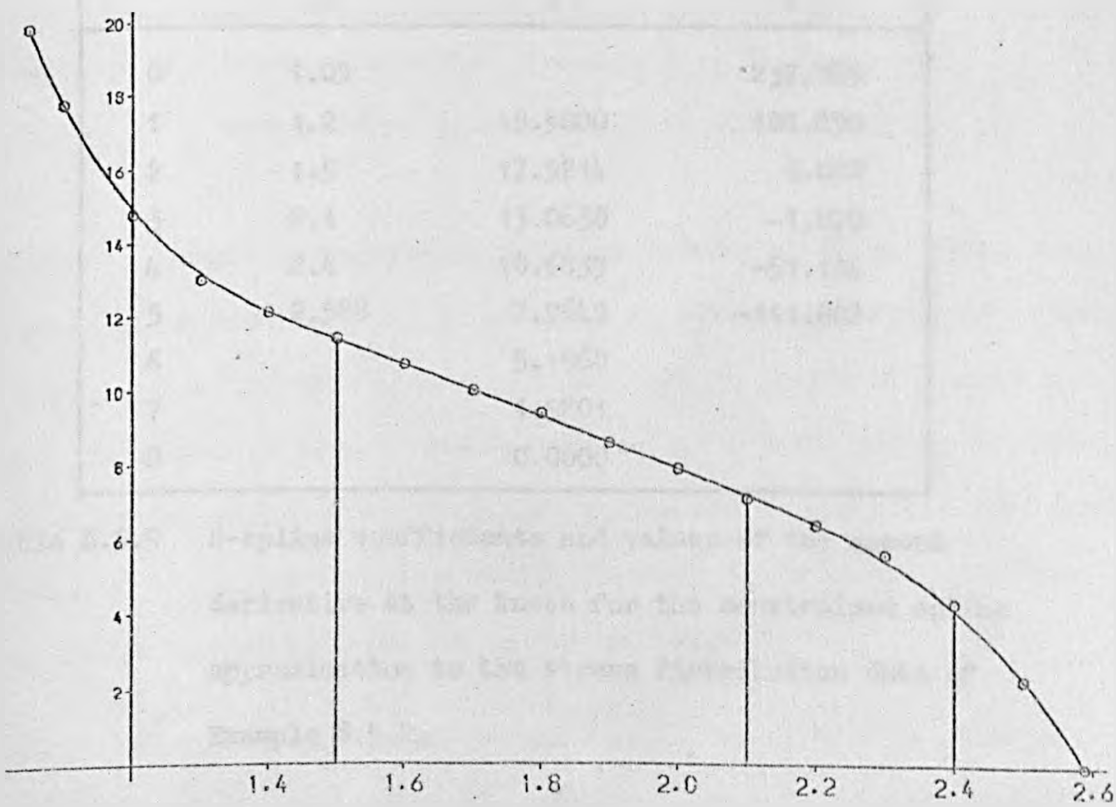

Fig 8.5.4 Stress distribution : S - shaped spline approximation

Approximation 2: Convexity constraints at $x = 1.05, 1.2, 1.5$; concavity constraints at $x = 2.1, 2.4, 2.588$ - Tables 8.5.9 and 8.5.10 and Fig 8.5.4.

Growth factor: 14.0.

Comments: The mean absolute residual of 0.0499 for the constrained spline is about 70% greater than the value 0.0289 of the unconstrained spline.

If such a value were considered unacceptably large, a constrained approximation with residuals comparable to those of the above unconstrained approximation can be obtained by using more knots. For example, for the interior knots 1.2, 1.35, 1.5, 2.1, 2.25 and 2.4, the mean absolute residual of the constrained spline is 0.0208. The unconstrained spline for these knots has a slightly better mean absolute residual of 0.0196, but again violates the requirement that the approximation is S-shaped.

j	x_j	c_j	s_j''
0	1.05		237.985
1	1.2	19.9000	101.890
2	1.5	17.5214	0.000
3	2.1	13.0630	-1.870
4	2.4	10.6839	-51.194
5	2.588	7.9649	-111.802
6		5.1960	
7		1.9201	
8		0.0000	

Table 8.5.9 B-spline coefficients and values of the second derivative at the knots for the constrained spline approximation to the stress distribution data of Example 8.5.2.

i	$s(t_i)$	$s(t_i) - f_i$
1	19.9000	0.0000
2	17.8000	0.0000
3	14.9312	0.1312
4	13.1760	0.1760
5	12.1000	0.0000
6	11.3636	-0.0364
7	10.6834	-0.0166
8	10.0000	0.0000
9	9.3104	-0.0896
10	8.6114	0.0114
11	7.9000	0.0000
12	7.1730	0.0730
13	6.4004	0.0004
14	5.4447	-0.1553
15	4.1415	-0.1585
16	2.3000	0.0000
17	0.0000	0.0000
Mean absolute residual =		0.0499

Table 8.5.10 Constrained spline approximation to the stress distribution data of Example 8.5.2.

Discrete linear L_1 approximation theory informs us (Rice, 1964) that, in the unconstrained case, the best L_1 approximation interpolates (at least) $N+3$ of the data points. We see that in the first example $N=4$ and the number of interpolated points is 7, as predicted by the theory. In the constrained case only 6 data points are interpolated, but one value of s''_j takes the value zero. In other words an interpolation condition has been traded for an active constraint.

Similar remarks apply to the second example in which $N=5$. The number of interpolated points is 8 in the unconstrained case, whereas in the constrained case the number of interpolated points is 7 and one value of s''_j takes the value zero.

CHAPTER 9

THE IMPOSITION OF BOUNDARY CONDITIONS AND OTHER EQUALITY CONSTRAINTS

It is sometimes necessary in problems of spline approximation to force the nth-order spline $s(x)$ to have the property that at the boundaries of, or within, the interval of interest, $s(x)$ or some of its derivatives are to take prescribed values. For instance, in spline interpolation it is often required that $s(x)$ satisfies given derivative conditions at the boundaries; in least-squares spline approximation it is sometimes required that either prescribed boundary conditions, as in the interpolation problem, are to be satisfies, or that $s(x)$ and possibly its derivatives are to take given values at certain interior points.

Because of their relative simplicity, we treat boundary conditions separately from the more general conditions. Thus, in Section 9.1 we discuss the imposition of a single derivative boundary condition. In Section 9.2 we treat the imposition of a set of derivative boundary conditions. Both of these types of conditions are incorporated by a simple change of basis. In Section 9.3 we consider simple point constraints and in Section 9.4 the most general type of linear equality point constraint. Finally, in Section 9.5 we outline algorithms for least-squares problems with linear constraints, and indicate how these algorithms can be applied to the general constrained spline approximation problem.

9.1 The imposition of a single derivative boundary condition

Let the nth-order spline $s(x)$ be expressed in its B-spline form (5.1.10), where the knots upon which $s(x)$ and the B-splines are defined form a standard knot set with coincident end knots. Suppose that in an interpolation or least-squares approximation problem it is required that $s(x)$ or one of its derivatives is to take a prescribed value at one or other of the range end-points a and b. In the case of interpolation the end condition would nearly always involve a derivative, since $s(x)$ is usually already required to take specific function values at a and b; moreover, in order that the number of free linear parameters of $s(x)$ matches the total number of conditions to be satisfied, the end condition would be traded for a conventional interpolation condition. For least-squares spline approximations, however, the end condition may involve either $s(x)$ or its derivatives; further, it will not usually be appropriate or necessary to trade the end condition for one of the data points.

We shall treat solely a condition at the left-hand end-point $x=a$, since the right-hand end-point $x=b$ is handled analogously.

Let r $(0 \leq r < n)$ and the value of $f^{(r)}(a)$ be prescribed. It is required that $s(x)$ satisfy

$$s^{(r)}(a) = f^{(r)}(a). \tag{9.1.1}$$

We shall show that condition (9.1.1) can be enforced by a simple modification of the data and of the basis functions.

We examine first the case $r = 1$. From (5.1.10), (9.1.1) and using Theorem 4.2.1 we obtain

$$c_1 N'_{n1}(a) + c_2 N'_{n2}(a) = f'(a) . \tag{9.1.2}$$

But from (4.2.1), $N'_{n1}(a) \neq 0$. Hence, by eliminating c_1 between (5.1.10) and (9.1.2) and setting $q = N+n-1$, we obtain

$$s(x) = \left(\frac{f'(a) - c_2 N'_{n2}(a)}{N'_{n1}(a)} \right) N_{n1}(x) + \sum_{i=2}^{q} c_i N_{ni}(x), \tag{9.1.3}$$

a simple re-arrangement of which yields

$$\tilde{s}(x) = s(x) - \frac{f'(a)}{N'_{n1}(a)} N_{n1}(x) = \sum_{i=2}^{q} c_i \tilde{N}_{ni}(x) , \tag{9.1.4}$$

where

$$\tilde{N}_{ni}(x) = \begin{cases} N_{n2}(x) - \dfrac{N'_{n2}(a)}{N'_{n1}(a)} N_{n1}(x) & (i = 2) \\ \\ N_{ni}(x) \quad (i = 3, 4, \ldots, q) . \end{cases} \tag{9.1.5}$$

But, by differentiating (3.6.1) and using (4.2.1), $N'_{n1}(a) + N'_{n2}(a) = 0$. Hence (9.1.5) simplifies to

$$\tilde{N}_{ni}(x) = \begin{cases} N_{n1}(x) + N_{n2}(x) & (i = 2) \\ \\ N_{ni}(x) \quad (i = 3, 4, \ldots, q) . \end{cases} \tag{9.1.6}$$

Consequently, if appropriate values of the expression $\left\{f'(a)/N'_{n1}(a)\right\} N_{n1}(x)$ are subtracted from the data to be approximated, the use of the modified representation $\tilde{s}(x)$ enables, in the case $r=1$, the condition (9.1.1) to be incorporated automatically. Note that, since $N_{n1}(x) = 0$ for $x \geqslant x_1$, the term $\left\{f'(a)/N'_{n1}(a)\right\} N_{n1}(x)$ involves modification only of data values in the interval $a \leqslant x < x_1$. Also observe that the function $\tilde{N}_{n2}(x)$ has the same support as $N_{n2}(x)$ and is non-negative. Moreover, $\tilde{N}_{n2}(x)$ is formed stably, since it is simply the sum of two non-negative quantities, each of which can be computed stably (Section 3.9).

We now consider the generalization of the above approach to the enforcement of the boundary condition (9.1.1) for a general value of $r (0 \leqslant r < n)$. Proceeding along lines similar to the above we obtain the modified representation

$$\tilde{s}(x) = s(x) - \frac{f^{(r)}(a)}{N^{(r)}_{n1}(a)} N_{n1}(x) = \sum_{i=2}^{q} c_i \tilde{N}_{ni}(x) , \tag{9.1.7}$$

where

$$\tilde{N}_{ni}(x) = \begin{cases} N_{ni}(x) - \dfrac{N^{(r)}_{ni}(a)}{N^{(r)}_{n1}(a)} N_{n1}(x) \\ \qquad\qquad (i = 2, 3, \ldots, r+1) \\ N_{ni}(x) \qquad (i = r+2, r+3, \ldots, q). \end{cases} \tag{9.1.8}$$

Unfortunately, no longer do all the basis functions $N_{ni}(x)$ have the property that they are formed as positive linear combinations of non-negative quantities, and hence there is no guarantee that the $N_{ni}(x)$ can be computed with small relative errors. However, their values will certainly possess small absolute errors compared with unity, the maximum possible value of $N_{ni}(x)$. In order to obtain basis functions which have small relative errors we proceed as follows.

Consider the representation (9.1.7) with the $\tilde{N}_{ni}(x)$ defined by

$$\tilde{N}_{ni}(x) = \begin{cases} N_{ni}(x) - \dfrac{N_{ni}^{(r)}(a)}{N_{n,i-1}^{(r)}(a)} \; N_{n,i-1}(x) & \\ & (i = 2, 3, \ldots, r+1) \\ N_{ni}(x) & (i = r+2, r+3, \ldots, q) , \end{cases} \tag{9.1.9}$$

rather than by (9.1.8). As with the representation (9.1.7) and (9.1.8) it is easily verified that $\tilde{s}^{(r)}(a) = 0$ and $s^{(r)}(a) = f^{(r)}(a)$, as required. Moreover, both representations enjoy the property that for $i = 2, 3, \ldots, q$, the basis functions $\tilde{N}_{ni}(x)$ have the same support as the functions $N_{ni}(x)$. However, the representation (9.1.7) and (9.1.9) has the distinct advantage that the factors $N_{ni}^{(r)}(a)/N_{n,i-1}^{(r)}(a)$ are all negative, by virtue of (4.2.1), and hence that the $\tilde{N}_{ni}(x)$ are formed as positive linear combinations of non-negative quantities, with the consequence that the computed values have small relative errors.

9.2 Imposition of a set of boundary conditions

In the previous section a method was given for forcing the nth-order spline $s(x)$ to have the property that $s^{(r)}(a)$ takes a prescribed value $f^{(r)}(a)$. We now consider the case where, for some k $(0 \leq k < n)$, the values of $s^{(r)}(a)$ are to take prescribed values $f^{(r)}(a)$ for $r = 0, 1, \ldots, k$. Thus the conditions

$$\sum_{i=1}^{q} c_i N_{ni}^{(r)}(a) = f^{(r)}(a) \tag{9.2.1}$$

are to be satisfied for $r = 0, 1, \ldots, k$. Because of (4.2.1), conditions (9.2.1) reduce to

$$\sum_{i=1}^{r+1} c_i N_{ni}^{(r)}(a) = f^{(r)}(a) \qquad (r = 0, 1, \ldots, k), \tag{9.2.2}$$

ie to

$$\underset{\sim}{L}\underset{\sim}{c}^{(o)} = \underset{\sim}{d}, \tag{9.2.3}$$

where $\underset{\sim}{L}$ is the lower-triangular matrix of order $k+1$ with non-zero elements $l_{ij} = N_{nj}^{(i-1)}(a)$, $\underset{\sim}{c}^{(o)} = \left\{c_1, c_2, \ldots, c_{k+1}\right\}$ and $\underset{\sim}{d} = \left\{f(a), f'(a), \ldots, f^{(k)}(a)\right\}$. The values of the B-spline derivatives required in $\underset{\sim}{L}$ are computed from Algorithm 4.4.1 in an unconditionally stable manner (Theorem 4.2.3). The triangular system possesses a unique solution since its diagonal elements $N_{n,r+1}^{(r)}(a)$ $(r = 0, 1, \ldots, k)$ are all non-zero, by virtue of (4.2.1). The system is easily solved by the usual process of forward substitution.

Having obtained the values of $c_1, c_2, \ldots, c_{k+1}$, we write

$$\tilde{s}(x) = s(x) - \sum_{i=1}^{k+1} c_i N_{ni}(x) = \sum_{i=k+2}^{q}{}' c_i N_{ni}(x) . \tag{9.2.4}$$

Then, for each data abscissa x we subtract from the corresponding ordinate the value of $\sum_{i=1}^{k+1} c_i N_{ni}(x)$. The modified data is then approximated by $\tilde{s}(x)$. Note that only data in the interval $a \leqslant x < x_{k+1}$ is affected by the subtraction. Boundary conditions at $x = b$ are treated in a similar fashion.

The method of this section has been used successfully in conjunction with a variant of Algorithm 7.3.1 in a number of applications. In particular, it has been applied to the fitting of various sets of data representative of modes of vibration of a clamped plate where, as a consequence of the clamping, values of $s^{(r)}(x)$ for $r = 0, 1, 2$ were prescribed at each end of the data range.

9.3 Simple point constraints

In some spline approximation problems it is necessary to impose restrictions on $s(x)$ or its derivatives at various points in the range of interest. We have already treated cases where a single boundary constraint or a certain set of boundary constraints is to be imposed. We now consider more general constraints. We deal in this section with simple point constraints, ie constraints involving a single value of the function or one of its derivatives, and in Section 9.4 with compound point constraints, which may involve the function value and the values of a number of derivatives.

Suppose $s^{(r)}(x)$ is to take the value $f^{(r)}(t_o)$ at $x = t_o$. Here $r(0 \leq r < n)$, $t_o(a \leq t_o \leq b)$ and $f^{(r)}(t_o)$ are prescribed. Thus we require $s(x)$ to satisfy

$$s^{(r)}(t_o) = \sum_{i=1}^{q} c_i N_{ni}^{(r)}(t_o) = f^{(r)}(t_o) \ . \qquad (9.3.1)$$

Relation (9.3.1) is evidently a linear equality in the B-spline coefficients c_i. In fact, because of the compact support of the B-splines, at most n of the values of the $N_{ni}^{(r)}(t_o)$ are non-zero. Moreover, (9.3.1) has precisely the same structure as the usual interpolation condition or "observational equation".

9.4 Compound point constraints

We now consider a more general form of linear equality constraint which we term a compound point constraint. Let L be a linear operator of the form

$$L = \sum_{r=0}^{n-1} e_r D^r , \qquad (9.4.1)$$

where D^r denotes r-fold differentiation with respect to x, and the e_r are prescribed constants, not all of which are zero. Let g be a given number (possibly zero). It is required that at a prescribed value of x, t_o say, $Ls(x)$ is to take the value g, ie

$$\left\{ L \sum_{i=1}^{q} c_i N_{ni}(x) \right\}_{x=t_o} = g. \qquad (9.4.2)$$

Thus

$$\sum_{i=1}^{q} c_i \left\{ LN_{ni}(x) \right\}_{x=t_o} = g, \qquad (9.4.3)$$

ie
$$\sum_{i=1}^{q} c_i \left\{ \sum_{r=0}^{n-1} e_r N_{ni}^{(r)}(t_o) \right\} = g. \qquad (9.4.4)$$

Again, (9.4.4) is a linear equality relationship in the coefficients c_i with the same structure as a relation of the form

$$\sum_{i=1}^{q} c_i N_{ni}(t_o) = g . \qquad (9.4.5)$$

In imposing constraints of the form (9.3.1) and (9.4.4) it is necessary to evaluate the appropriate values and derivatives of $N_{ni}(x)$. Such evaluations can be accomplished using Algorithms 3.12.2 and 4.4.1.

When used in least-squares data fitting by splines these constraints may be incorporated by the methods of Section 9.5.

9.5 Stable methods for the imposition of general linear constraints

It remains to discuss methods for imposing constraints of the form discussed in Sections 9.3 and 9.4. In the case of interpolation, by ordering the interpolation conditions and constraints (assumed consistent) appropriately, the resulting linear system, which is stepped-banded, can be solved by Algorithm 2.12.1 or Algorithm 2.13.1. In the case of least-squares approximation it is necessary to solve a problem of the form

$$\min_{\underset{\sim}{x}} \left\| \underset{\sim}{A}\underset{\sim}{x} - \underset{\sim}{b} \right\|_2 \tag{9.5.1}$$

subject to the equality constraints (assumed consistent)

$$\underset{\sim}{C}\underset{\sim}{x} = \underset{\sim}{g}, \tag{9.5.2}$$

where $\underset{\sim}{A}$ is an m by n matrix of (possibly unknown) rank k ($\leqslant$ n) and $\underset{\sim}{C}$ is a p by n matrix of (again possibly unknown) rank l ($\leqslant$ p). The notation used in this section is chosen to be similar to that of Chapter 2.

We first mention two numerically stable methods for solving the above problem. One of these methods is due to Golub (1965) and applies only in the case where both $\underset{\sim}{A}$ and $\underset{\sim}{C}$ have maximum rank. However, this method can be made very efficient for stepped-banded $\underset{\sim}{A}$ and $\underset{\sim}{C}$. The other method is given by Hayes and Halliday (1974) and allows either or both of $\underset{\sim}{A}$ and $\underset{\sim}{C}$ to be rank deficient. However, because of the need to carry out column interchanges in their method, little or no advantage can be taken of the structure of $\underset{\sim}{A}$ and $\underset{\sim}{C}$. Finally, we present an enhancement of Golub's method that allows cases of rank deficiency to be treated in a stable manner, whilst taking advantage of structure such as stepped-bandedness in $\underset{\sim}{A}$ and $\underset{\sim}{C}$.

Golub's method is based upon the use of Lagrange multipliers $\underset{\sim}{\lambda}$ to express the solution of (9.5.1) and (9.5.2) as that of the "augmented normal equations"

$$\begin{bmatrix} \underset{\sim}{A}^T\underset{\sim}{A} & \underset{\sim}{C}^T \\ \underset{\sim}{C} & \underset{\sim}{0} \end{bmatrix} \begin{bmatrix} \underset{\sim}{x} \\ \underset{\sim}{\lambda} \end{bmatrix} = \begin{bmatrix} \underset{\sim}{A}^T\underset{\sim}{b} \\ \underset{\sim}{g} \end{bmatrix}. \tag{9.5.3}$$

Householder transformations are applied to solve (9.5.3), without of course forming these equations explicitly and incurring the possible loss of information associated with such a formation (cf Section 2.3). In Golub's description, column interchanges are carried out but, as we indicated in Chapter 2, such interchanges are unnecessary. After setting $\underset{\sim}{x} = \underset{\sim}{u} + \underset{\sim}{\delta}$, where $\underset{\sim}{u}$ denotes the unconstrained solution (obtained in practice via the QR decomposition of $\underset{\sim}{A}$) satisfying $\underset{\sim}{A}^T\underset{\sim}{A}\underset{\sim}{u} = \underset{\sim}{A}^T\underset{\sim}{b}$, it is seen from (9.5.3) that $\underset{\sim}{\delta}$, the "correction term", satisfies

$$\underset{\sim}{A}^T\underset{\sim}{A}\underset{\sim}{\delta} + \underset{\sim}{C}^T\underset{\sim}{\lambda} = 0 \tag{9.5.4}$$

and

$$\underset{\sim}{C}(\underset{\sim}{u}+\underset{\sim}{\delta}) = \underset{\sim}{g}. \tag{9.5.5}$$

Eliminating $\underset{\sim}{\delta}$ from (9.5.4) and (9.5.5) yields

$$\underset{\sim}{C}(\underset{\sim}{A}^T\underset{\sim}{A})^{-1}\underset{\sim}{C}^T\underset{\sim}{\lambda} = \underset{\sim}{C}\underset{\sim}{u} - \underset{\sim}{g} \tag{9.5.6}$$

as an equation defining the Lagrange multipliers. Having solved (9.5.6) for $\underset{\sim}{\lambda}$, $\underset{\sim}{\delta}$ is found from (9.5.4) and then $\underset{\sim}{x} = \underset{\sim}{u} + \underset{\sim}{\delta}$. In actual computation advantage is taken of the factorization $\underset{\sim}{A} = \underset{\sim}{Q}\underset{\sim}{R}$ to simplify the process. In particular, (9.5.6) reduces to

$$\underset{\sim}{V}^T\underset{\sim}{V}\underset{\sim}{\lambda} = \underset{\sim}{C}\underset{\sim}{u} - \underset{\sim}{g}, \tag{9.5.7}$$

where $\underset{\sim}{V}$ is given by the triangular system $\underset{\sim}{R}^T\underset{\sim}{V} = \underset{\sim}{C}^T$. Equation (9.5.7) is solved by carrying out an orthogonal decomposition of $\underset{\sim}{V}$. Finally, $\underset{\sim}{\delta}$ is formed from (9.5.4) by taking further advantage of the already-factorized $\underset{\sim}{A}$.

Of course, plane rotations can be used in place of Householder transformations. Since the bulk of the work (assuming the usual case in which p is small compared with m) is involved in the factorization of $\underset{\sim}{A}$, and since advantage can be taken of the structure of $\underset{\sim}{A}$ during its QR decomposition, the complete process can be carried out in little more time than that taken by the computation of the unconstrained solution $\underset{\sim}{u}$.

The method of Hayes and Halliday provides essentially a means of eliminating in a stable manner l of the n unknowns and thus reducing the system to one of order $n-l$, rather than having to treat one of order $n+p$ as in (9.5.3). Specifically, their approach, which works with an orthogonal transformation $\underset{\sim}{y}$, say, of the solution vector, first reduces the constraint equations to a triangular system of order l, which is then solved for the first l components of $\underset{\sim}{y}$. They then show that the remaining $n-l$ components of $\underset{\sim}{y}$ can be found by solving an unconstrained least-squares problem. Finally, $\underset{\sim}{x}$ is recovered from an orthogonal transformation of $\underset{\sim}{y}$. We have described their approach only in very broad outline for two reasons. Firstly, it is given in considerable detail in their paper and, secondly, for stability, it is crucial to carry out column interchanges in their method; consequently, we can see no way of adapting their algorithm to solving stepped-banded systems efficiently without destroying structure.

We now propose an adaptation of Golub's method that permits rank deficiency in $\underset{\sim}{A}$ or $\underset{\sim}{C}$ or both and, moreover, allows considerable advantage to be taken of the structure of these matrices.

Firstly, we consider the constraint equations. Frequently, in practical spline-approximation problems, $\underset{\sim}{C}$ will be of full rank. However, whether or not this is true, we recommend the following approach. Carry out an orthogonal decomposition of $\underset{\sim}{C}^T$ using, say, plane rotations. The resulting upper-trapezoidal matrix, $\underset{\sim}{U}$, say, will have precisely $p-l$ zero diagonal elements (in the absence of errors in the elements of $\underset{\sim}{C}$ and in the arithmetic operations on $\underset{\sim}{C}$). In practice, $p-l$ diagonal elements will be "small" (relative to some norm of $\underset{\sim}{C}$), and a suitable threshold value should be selected to decide which diagonal elements are to be regarded as zero. By deleting the corresponding $p-l$ columns of $\underset{\sim}{C}^T$, ($p-l$ rows of $\underset{\sim}{C}$), the constraint equations are reduced to a total of l equations whose coefficient matrix is of full rank. Henceforth, we shall assume that the constraint equations have, if necessary, been so treated and let (9.5.2), with p-1 replacing p, denote the reduced system.

It remains to treat the rank deficiency, if any, in $\underset{\sim}{A}$. We observe that the solution to (9.5.1) and (9.5.2) is identical to that of

$$\min_{\underset{\sim}{x}} \left\| \underset{\sim}{A}'\underset{\sim}{x} - \underset{\sim}{b}' \right\|_2 \tag{9.5.8}$$

subject to (9.5.2), where

$$\underset{\sim}{A}' = \begin{pmatrix} \underset{\sim}{A} \\ \underset{\sim}{C} \end{pmatrix} \tag{9.5.9}$$

and

$$\underset{\sim}{b}' = \begin{pmatrix} \underset{\sim}{b} \\ \underset{\sim}{g} \end{pmatrix}. \tag{9.5.10}$$

In the case of stepped-banded $\underset{\sim}{A}$ and $\underset{\sim}{C}$ it is desirable to <u>interleave</u> the rows of $(\underset{\sim}{C} \mid \underset{\sim}{g})$ with those of $(\underset{\sim}{A} \mid \underset{\sim}{b})$ so that the resulting system is similarly stepped banded. It is advisable, if necessary, to introduce suitable scaling factors so that the rows of $\underset{\sim}{A}'$ have norms of similar magnitude. Such a scaling is particularly appropriate if, for instance, $\underset{\sim}{A}$ is a matrix of B-spline values and $\underset{\sim}{C}$ contains values of B-spline derivatives (perhaps of various orders).

It should now be apparent that if $\underset{\sim}{A}'$ is of rank n, Golub's method may be applied immediately to the solution of (9.5.8) and (9.5.2). If $\underset{\sim}{A}'$ is rank deficient we recommend that, after having computed the QR factors of $\underset{\sim}{A}'$, elements of the solution vector corresponding to columns of $\underset{\sim}{A}'$ containing "small" diagonal elements be made zero by using the "resolving constraint" concept due to Gentleman (1973). The resolving constraint is treated as an additional row of $(\underset{\sim}{A}' \mid \underset{\sim}{g}')$ and consists of the row vector $(0 \ldots 010 \ldots 0 \mid 0)$, where the non-zero element lies in the column containing the diagonal element to be regarded as zero. By rotating this row into the current triangular factor $(\underset{\sim}{R} \mid \underset{\sim}{Q})$, the rank of $\underset{\sim}{A}'$ is increased by one and the residual sum of squares is unaltered. All such diagonal elements are so treated. (This method of treating rank deficiency is also of considerable use in unconstrained problems).

A pilot computer program based upon the above ideas has been constructed and tested on cases containing rank deficiency in $\underset{\sim}{C}$ but not $\underset{\sim}{A}$, in $\underset{\sim}{A}$ but not $\underset{\sim}{C}$, and in both $\underset{\sim}{A}$ and $\underset{\sim}{C}$. Cases in which $\underset{\sim}{A}'$ was and was not rank deficient were also tested. The results achieved to date imply that the process appears to function extremely satisfactorily.

It should be noted that the tests for zero diagonal elements are not infallible since examples can be constructed (J H Wilkinson, private communication) for which a matrix is close to being rank deficient but for which the resulting diagonal elements in the triangular or trapezoidal factor are in no sense small even if arithmetic is carried out exactly. However, such examples are somewhat artifical and in practice are most unlikely to arise. In cases of doubt the singular value decomposition (Section 2.15) should be employed.

CHAPTER 10

MULTIVARIATE SPLINES

In this chapter we consider the extension to higher dimensions of the methods for one-dimensional interpolation and least-squares approximation by splines discussed in Chapters 6 and 7. In particular, we examine problems in two independent variables, a natural (but notationally complex) extension of which enables higher-dimensional problems to be treated.

Firstly, we consider in Sections 10.1 and 10.2 the interpolation and least-squares approximation to data given at all the vertices of a finite rectangular mesh by a tensor product of general univariate functions. This treatment is then specialized in Section 10.3 to the case where the univariate functions are B-splines. In Section 10.4 the important problem of least-squares spline approximation to arbitrarily-placed bivariate data is considered. The imposition of constraints is discussed briefly in Section 10.5. Finally, in Section 10.6, the evaluation of a multivariate spline from its B-spline representation is examined.

10.1 Interpolation of data on a rectangular mesh by a tensor product of univariate functions

Let

$$f(x,y) = \sum_{i=1}^{n_x} \sum_{j=1}^{n_y} c_{ij} g_i(x) h_j(y) \tag{10.1.1}$$

denote the space of functions obtained by taking the tensor product of the two linearly independent sets of basis functions

$$g_i(x) \quad (i = 1, 2, \ldots, n_x)\ , \quad h_j(y) \quad (j = 1, 2, \ldots, n_y). \tag{10.1.2}$$

Suppose data values z_{rs} are prescribed at all the vertices of the rectangular mesh defined by the lines $x = t_r$ $(r = 1, 2, \ldots, n_x)$ and

$y = u_s$ $(s = 1, 2, \ldots, n_y)$.

The problem is to determine the coefficients c_{ij} in (10.1.1) such that $f(x,y)$ interpolates the given data values, ie to compute values of c_{ij} which satisfy the $n_x n_y$ equations

$$z_{rs} = f(t_r, u_s) = \sum_{i=1}^{n_x} \sum_{j=1}^{n_y} c_{ij} g_i(t_r) h_j(u_s)$$

$$(r = 1, 2, \ldots, n_x \; ; \; s = 1, 2, \ldots, n_y). \tag{10.1.3}$$

The system (10.1.3), once formed, could be solved directly, since it is a square system of linear algebraic equations of order $n_x n_y$. For example, Gaussian elimination with partial pivoting could be used, in which case the solution would be obtained in about $n_x^3 n_y^3/3$ long operations. However, with such an approach no advantage, apart perhaps in the formation of the system, is taken of the tensor-product representation of the approximating function $f(x,y)$. For instance, for a problem of modest size in which $n_x = n_y = 30$, about 2×10^8 long operations are required. A second approach is therefore normally used (see eg Greville, 1961) and discussed briefly here in which full advantage is taken of the tensor-product form, with the consequence that only about $(n_x + n_y)^3/3$ long operations are necessary. For the case $n_x = n_y = 30$ this number is about 7×10^4. In the case of B-spline basis functions, further economies are achieved (Section 10.3).

The system (10.1.3) may be expressed in matrix form as

$$\underset{\sim}{G}\underset{\sim}{C}\underset{\sim}{H}^T = \underset{\sim}{Z} \tag{10.1.4}$$

or, equivalently,

$$\underset{\sim}{H}\underset{\sim}{C}^T\underset{\sim}{G}^T = \underset{\sim}{Z}^T , \tag{10.1.5}$$

where

$\underset{\sim}{G}$ is the n_x by n_x matrix with elements $g_{ij} = g_j(x_i)$,

$\underset{\sim}{H}$ is the n_y by n_y matrix with elements $h_{ij} = h_j(y_i)$,

$\underset{\sim}{C}$ is the n_x by n_y matrix of elements c_{ij}

and

$\underset{\sim}{Z}$ is the n_x by n_y matrix of elements z_{ij}.

So, defining

$$\underset{\sim}{E} = \underset{\sim}{C}\underset{\sim}{H}^T , \tag{10.1.6}$$

the matrix $\underset{\sim}{E}$ can be found by solving

$$\underset{\sim}{G}\underset{\sim}{E} = \underset{\sim}{Z} . \tag{10.1.7}$$

Then $\underset{\sim}{C}$ can be obtained by solving

$$\underset{\sim}{H}\underset{\sim}{C}^T = \underset{\sim}{E}^T . \tag{10.1.8}$$

Equations (10.1.7) involve an n_x by n_x matrix with n_y right-hand sides, the solution of which using Gaussian elimination with partial pivoting takes about $\frac{1}{3}n_x^3$ long operations for the decomposition plus about $n_x^2 n_y$ long operations for the solution of the resulting triangular systems. Equations (10.1.8) involve an n_y by n_y matrix with n_x right-hand sides, the solution of which requires about $\frac{1}{3}n_y^3 + n_x n_y^2$ long operations. Thus the total amount of work involved in the solutions of (10.1.7) and (10.1.8) is about $\frac{1}{3}n_x^3 + n_x^2 n_y + n_x n_y^2 + \frac{1}{3}n_y^3 = \frac{1}{3}(n_x+n_y)^3$ long operations. From the symmetry of this result it is immaterial from the point of view of computational effort whether we treat the system (10.1.4), as we have done here, or the system (10.1.5).

Equations (10.1.7) and (10.1.8) show that the problem degenerates into two sub-problems, each of which is essentially a set of univariate interpolation problems, and may be interpreted as follows. Along each mesh line $y = u_s$ ($s = 1, 2, \ldots, n_y$) determine the coefficients e_{rs} ($r = 1, 2, \ldots, n_x$) of the function $\sum_{r=1}^{n_x} e_{rs}g_r(x)$ which interpolates the data (t_r, z_{rs})

$(r = 1, 2, \ldots, n_x)$. Then for each value of $r = 1, 2, \ldots, n_x$ determine the coefficients c_{rs} $(s = 1, 2, \ldots, n_y)$ of the function $\sum_{s=1}^{n_y} c_{rs}h_s(y)$ which interpolates the data (u_s, e_{rs}) $(s = 1, 2, \ldots, n_y)$.

10.2 Least-squares approximation to data on a rectangular mesh by a tensor product of univariate functions

We treat in this section the extension of the interpolation problem considered in Section 10.1 to the case where the data values z_{rs} are prescribed at all the vertices of the rectangular mesh defined by the lines $x = t_r$ $(r = 1, 2, \ldots, m_x)$ and $y = u_s$ $(s = 1, 2, \ldots, m_y)$ and are to be approximated in the least-squares sense by a function of the form (10.1.1). Here $m_x \geqslant n_x$ and $m_y \geqslant n_y$ and it is required to determine the coefficients c_{ij} in (10.1.1) such that the residual sum of squares

$$\sum_{r=1}^{m_x} \sum_{s=1}^{m_y} \left\{ f(t_r, u_s) - z_{rs} \right\}^2 \tag{10.2.1}$$

is minimized.

Note that arbitrary weighting factors <u>cannot</u> be incorporated in (10.2.1) as they can in the one-dimensional case and, at the same time, full advantage taken of the tensor-product representation. For spline approximation, cases of unequal weight may be tackled using the more general but computationally relatively expensive method of Section 10.4.

Let $\underset{\sim}{G}$, $\underset{\sim}{H}$, $\underset{\sim}{C}$ and $\underset{\sim}{Z}$ be as defined in Section 10.1, except that now $\underset{\sim}{G}$ is m_x by n_x, $\underset{\sim}{H}$ is m_y by n_y and $\underset{\sim}{Z}$ is m_x by m_y ($\underset{\sim}{C}$ is n_x by n_y as before). Greville (1961) has shown that the solution to this least-squares problem is the natural extension of that for the interpolation problem discussed in Section 10.1. In fact, in place of the interpolatory solution

$$\underset{\sim}{C} = \underset{\sim}{G}^{-1}\underset{\sim}{Z}(\underset{\sim}{H}^{-1})^T , \tag{10.2.2}$$

obtained from (10.1.4), one uses

$$\underset{\sim}{C} = \underset{\sim}{G}^{\dagger}\underset{\sim}{Z}(\underset{\sim}{H}^{\dagger})^T , \tag{10.2.3}$$

where $\underset{\sim}{G}^{\dagger}$ and $\underset{\sim}{H}^{\dagger}$ are respectively the pseudo-inverses of $\underset{\sim}{G}$ and $\underset{\sim}{H}$. Of course it is unnecessary to compute explicitly these pseudo-inverses. Rather, by analogy with (10.1.7) and (10.1.8), $\underset{\sim}{C}$ may be formed by determining the least-squares solution of

$$\underset{\sim}{G}\underset{\sim}{E} = \underset{\sim}{Z} , \tag{10.2.4}$$

followed by that of

$$\underset{\sim}{H}\underset{\sim}{C}^T = \underset{\sim}{E}^T . \tag{10.2.5}$$

Assuming that one of the faster orthogonalization methods of Chapter 2 is employed and that $m_x, m_y \gg n_x, n_y$, an operation count reveals that the solution of (10.2.4) requires about $m_x n_x^2$ long operations for the decomposition of $\underset{\sim}{G}$ and about $m_x m_y n_x$ for the operations involving $\underset{\sim}{Z}$. Similarly, the count for the solution of (10.2.5) is about $m_y n_y^2 + m_y n_x n_y$. Thus the total amount of work in determining $\underset{\sim}{C}$ is dominated by the computations involving the multiple right-hand sides in the <u>first</u> least-squares system (10.2.4), and is approximately equal to $m_x m_y n_x$ long operations. Unlike that for the interpolation problem of Section 10.1, this count is not symmetric in its parameters. Note, therefore, that it may be cheaper to form $\underset{\sim}{C}$ from the transpose of (10.2.3), ie by computing the least-squares solutions of

$$\underset{\sim}{H}\underset{\sim}{F} = \underset{\sim}{Z}^T \tag{10.2.6}$$

and

$$\underset{\sim}{G}\underset{\sim}{C} = \underset{\sim}{F}^T , \tag{10.2.7}$$

rather than those of (10.2.4) and (10.2.5). The resulting operation count is then about $n_x m_y n_y$ long operations.

By analogy with our interpretation of equations (10.1.7) and (10.1.8) in the interpolation problem, we may interpret the least-squares solution of (10.2.4) and (10.2.5) as follows. Along each mesh line $y = u_s$ ($s = 1, 2, \ldots, m_y$) determine the coefficients e_{rs} ($r = 1, 2, \ldots, n_x$) of the function $\sum_{r=1}^{n_x} e_{rs} g_r(x)$ which provide the least-squares approximation to the data (t_r, z_{rs}) ($r = 1, 2, \ldots, m_x$). Then for each value of $r = 1, 2, \ldots, n_x$ determine the coefficients c_{rs} ($s = 1, 2, \ldots, n_y$) of the function $\sum_{s=1}^{n_y} c_{rs} h_s(y)$ which provide the least-squares approximation to the data (u_s, e_{rs}) ($s = 1, 2, \ldots, m_y$). Clenshaw and Hayes (1965) discuss the case where the $g_r(x)$ and $h_s(y)$ form polynomial bases.

10.3 Interpolation and least-squares approximation to data on a rectangular mesh by bivariate splines

We now specialize the approaches of Sections 10.1 and 10.2 to the case where in (10.1.1) the functions $g_i(x)$ are B-splines of order n in x (defined upon an appropriate set of x-knots) and the $h_j(y)$ are B-splines of the same order* in y (defined upon an appropriate set of y-knots), and we wish to interpolate or obtain least-squares approximations to data z_{rs} prescribed at all vertices of the rectangular mesh $x = t_r$ ($r = 1, 2, \ldots, m_x$), $y = u_s$ ($s = 1, 2, \ldots, m_y$). We shall assume that $t_1 \leqslant t_2 \leqslant \ldots \leqslant t_{m_x}$ and $u_1 \leqslant u_2 \leqslant \ldots \leqslant u_{m_y}$, that in the case of interpolation $m_x = n_x$ and $m_y = n_y$, and that in the least-squares case $m_x \geqslant n_x$ and $m_y \geqslant n_y$.

---

* The methods given in this and Section 10.4 may without difficulty be extended to the case where the B-splines are of different orders in x and in y.

In order to define our B-spline bases let x_i ($i = 1, 2, \ldots, N_x-1$) and y_j ($j = 1, 2, \ldots, N_y-1$), where $N_x = n_x - n + 1$ and $N_y = n_y - n + 1$, be two prescribed sets of interior knots which form respectively n-extended partitions (Section 3.1) of the x- and y-axes with $t_1 < x_1$, $x_{N_x-1} < t_{m_x}$ and $u_1 < y_1$, $y_{N_y-1} < u_{m_y}$. We introduce additional coincident end knots in the usual way by augmenting those prescribed by x-knots of multiplicity n at $x = t_1$ and at $x = t_{m_x}$, and y-knots of multiplicity n at $y = u_1$ and at $y = u_{m_y}$.

Let

$$a = x_0,\ b = x_{N_1},\ c = y_0,\ d = y_{N_2}. \tag{10.3.1}$$

The <u>knot-lines</u> $x = x_i$ ($i = 0, 1, \ldots, N_x$) and $y = y_j$ ($j = 0, 1, \ldots, N_y$) form a rectangular mesh, the boundary (formed by the lines $x = a$, $x = b$, $y = c$, $y = d$) of which contains all the data points. We define <u>panel</u> (i, j) as the rectangular region bounded by the x-knot lines $x = x_{i-1}$ and $x = x_i$ and the y-knot lines $y = y_{j-1}$ and $y = y_j$. A panel may be <u>null</u> in the sense that it has zero area, in which case $x_{i-1} = x_i$ or $y_{j-1} = y_j$. We say that a point (x, y) ($a \leqslant x < b$, $c \leqslant y < d$) lies in panel (i, j) if $x_{i-1} \leqslant x < x_i$ and $y_{j-1} \leqslant y < y_j$ (if $x = b$ we set $i = N_x$ and if $y = d$ we set $j = N_y$). Note that as a consequence of the above definitions, a null panel contains no points.

Upon the augmented set of x-knots we define the B-sline basis $N_{ni}(x)$ ($i = 1, 2, \ldots, n_x$) and upon the augmented set of y-knots the B-spline basis $P_{nj}(y)$ ($j = 1, 2, \ldots, n_y$). $P_{nj}(y)$ denotes the normalized B-spline of order n in y based on the knots $y_{j-n}, y_{j-n+1}, \ldots, y_j$. The tensor product

$$\left\{N_{n1}(x)\ N_{n2}(x)\ \ldots\ N_{nn_x}(x)\right\} \otimes \left\{P_{n1}(y)\ P_{n2}(y)\ \ldots\ P_{nn_y}(y)\right\} \tag{10.3.2}$$

forms a basis for the set of bivariate splines of order n in x and in y. Thus our representation of the bivariate spline s(x,y) is simply

$$s(x,y) = \sum_{i=1}^{n_x} \sum_{j=1}^{n_y} c_{ij} N_{ni}(x) P_{nj}(y), \tag{10.3.3}$$

in accordance with (10.1.1).

Evidently the interpolatory solution exists and is unique if and only if the Schoenberg-Whitney conditions (6.1.4) are satisfied for the x-knots x_i and the x-values t_r, and also for the y-knots y_j and the y-values u_s. In the least-squares case, the solution is unique if the conditions are satisfied for the x-knots x_i and at least one subset of the values of t_r, as well as for the y-knots y_j and at least one subset of the values of u_s.

It is apparent that the matrices $\underset{\sim}{G}$ and $\underset{\sim}{H}$ in (10.1.7), (10.1.8), (10.2.4) and (10.2.5) are all stepped-banded of bandwidth n, and that an obvious extension (to allow for multiple right-hand sides) of the methods developed in Chapter 2 for systems with such matrices can be applied.

Operation counts reveal that for interpolation about $n_x n_y n + (n_x + n_y)n^2$ long operations are required and that for least squares (again assuming that $m_x, m_y \gg n_x, n_y$) the dominant term is $m_x m_y n$. Thus for a fixed order of spline the computational effort is essentially proportional to the total number of data points (even taking into account the formation of $\underset{\sim}{G}$ and $\underset{\sim}{H}$), a result that holds also in one dimension (see Chapters 6 and 7). Such a desirable situation would fail to hold if a basis not having the compact support property were employed.

<u>Note</u> An excellent review (Hartley, 1976) of methods for tensor product approximations to data defined on rectangular meshes, is shortly to appear. Hartley also shows how the computations may be organised to

solve such problems in an arbitrary number of dimensions. Reference is made by Hartley to the gains in efficiency achieved by using B-splines as a basis in the case where the approximating function is a multivariate spline.

10.4 The general least-squares multivariate spline approximation problem

The approach considered here is a generalization of that of Chapter 7 to two independent variables. A further generalization to more than two independent variables is in principle straightforward but notationally complex and is not given here.

We consider only least-squares multivariate spline approximations since it is rarely of practical interest to interpolate multivariable data unless the data values are specially distributed such as at all vertices of a rectangular mesh (Section 10.3). However, if it is required to investigate whether a spline interpolant to an arbitrary set of data exists and is unique and, if so, to determine it, a simple extension of the method of this section can indeed be applied to such a problem.

Suppose values z_r of the dependent variable z are given at points (t_r, u_r) $(r = 1, 2, \ldots, m)$ in the (x,y)-plane. The problem is to determine a bivariate spline $s(x,y)$ of order n (degree n-1) in x and of the same order in y such that the residual sum of squares

$$\left\| \underset{\sim}{r} \right\|_2^2 = \left\| \underset{\sim}{W}^{\frac{1}{2}} \underset{\sim}{\varepsilon} \right\|_2^2 = \sum_{r=1}^{m} w_r \varepsilon_r^2 , \qquad (10.4.1)$$

where

$$\underset{\sim}{W} = \text{diag}\left\{ w_1, w_2, \ldots, w_m \right\} \qquad (10.4.2)$$

and

$$\varepsilon_r = s(t_r, u_r) - z_r \quad (r = 1, 2, \ldots, m), \qquad (10.4.3)$$

is minimized with respect to the free parameters of $s(x,y)$. It is assumed that interior x-knots x_i ($i = 1, 2, \ldots, N_x-1$) and interior y-knots y_j ($j = 1, 2, \ldots, N_y-1$) are prescribed. A detailed treatment of the case $n = 4$ is given by Hayes and Halliday (1974).

Just as in Section 10.3 we introduce additional end knots, define B-splines in x and in y, and employ the representation (10.3.3). Unfortunately, there is no analogue of the generalized Schoenberg-Whitney conditions (7.2.5) in the general multivariable situation (unless for instance, the data lies at all vertices of a rectangular mesh - cf Section 10.3). Thus it will not usually be possible to say, as the result of a simple test on the data and knots, whether the least-squares multivariate spline approximation problem has a unique solution. However, by analogy with the considerations of Section 7.2, we make the following conjecture.

Conjecture 10.4.1

In order for the least-squares bivariate spline approximation to be unique there must exist at least one subset of $n_x n_y$ distinct data points with the following property. It must be possible to find an "ordering" of these points such that the kth point ($k = 1, 2, \ldots, n_x n_y$) lies strictly within the support of the kth bivariate B-spline. (The kth bivariate B-spline is defined as the kth member of the tensor-product set (10.3.2), the support of $N_{ni}(x)P_{nj}(y)$ being the rectangle $x_{i-n} \leq x < x_i$, $y_{j-n} \leq y < y_j$).

A proof of this conjecture has not yet been attempted. Rather, efforts have been made to construct an algorithm (a bivariate counterpart of Algorithm 7.2.1) to test whether any given data and knot sets satisfy the property referred to in the conjecture. These efforts have so far proved unsuccessful for the following reason. In one dimension the data set has a natural ordering in the sense that it is possible to examine sequentially the relative positions of the points and the knots. In two (or more) dimensions, at least for arbitrarily-placed data, no such ordering exists. Accordingly, when an algorithm associates a particular point with the support of one of the bivariate B-splines, this decision affects subsequent decisions of the same nature. As a result, the algorithm may well conclude incorrectly that the conditions are not satisfied. Some form of back-tracking therefore seems to be required, but no satisfactory solution along these lines has yet been worked out.

The solution to the problem of minimizing (10.4.1) with respect to the c_{ij} is given by the least-squares solution of the system

$$\underset{\sim}{W}^{\frac{1}{2}}\underset{\sim}{A}\underset{\sim}{c} = \underset{\sim}{W}^{\frac{1}{2}}\underset{\sim}{z}\ , \tag{10.4.4}$$

where $\underset{\sim}{A}$ is of order m by $n_x n_y$. Normally, $m \gg n_x n_y$, but this need not be the case; indeed, least-squares solutions (though not necessarily unique) always exist (Peters and Wilkinson, 1970), even if $m < n_x n_y$. The rth row of $\underset{\sim}{A}$ contains the values

$$N_{n1}(t_r)P_{n1}(u_r),\ N_{n1}(t_r)P_{n2}(u_r),\ \ldots,\ N_{n1}(t_r)P_{nn_y}(u_r);$$

$$N_{n2}(t_r)P_{n1}(u_r),\ N_{n2}(t_r)P_{n2}(u_r),\ \ldots,\ N_{n2}(t_r)P_{nn_y}(u_r);$$

....

$$N_{nn_x}(t_r)P_{n1}(u_r),\ N_{nn_x}(t_r)P_{n2}(u_r),\ \ldots,\ N_{nn_x}(t_r)P_{nn_y}(u_r),$$

and the vector $\underset{\sim}{c}$ contains the coefficients

$$c_{11},\ c_{12},\ \ldots,\ c_{1n_y};\ c_{21},\ c_{22},\ \ldots,\ c_{2n_y};\ \ldots;$$

$$c_{n_x1},\ c_{n_x2},\ \ldots,\ c_{n_xn_y}.$$

As a result of the compact support of the B-splines, $\underset{\sim}{A}$ takes the <u>block stepped-banded</u> form

$$\underset{\sim}{A} = \begin{bmatrix} \underset{\sim}{A}_{11} & \underset{\sim}{A}_{12} & \cdots & \underset{\sim}{A}_{1n} & & & \underset{\sim}{0} \\ & \underset{\sim}{A}_{22} & \underset{\sim}{A}_{23} & \cdots & \underset{\sim}{A}_{2,n+1} & & \\ & & \cdots\cdots & \cdots\cdots & \cdots\cdots & \cdots\cdots & \\ 0 & & & \underset{\sim}{A}_{N_xN_x} & \underset{\sim}{A}_{N_x,N_x+1} & \cdots & \underset{\sim}{A}_{N_xn_x} \end{bmatrix} . \quad (10.4.5)$$

In order to achieve this form for $\underset{\sim}{A}$ it is necessary to order the values of the independent variable so that they lie in the successive panels (1,1), (1,2), ..., $(1,n_y)$; (2,1), (2,2), ..., $(2,n_y)$; ... ; $(n_x,1)$, $(n_x,2)$, ..., (n_x,n_y). We assume henceforth that such an ordering has been carried out.

Each sub-matrix $\underset{\sim}{A}_{ij}$ is itself a stepped-banded matrix of bandwidth n. The complete matrix $\underset{\sim}{A}$ is a stepped-banded matrix of bandwidth $(N_y-1)(n-1)+n^2$. Since the computational effort required to triangularize a stepped-banded matrix with m rows and bandwidth q is essentially proportional to mq^2 (Sections 2.12 - 2.14), it is more economical to interchange the roles of the independent variables x and y if $N_x < N_y$.

The computational effort to triangularize $\underset{\sim}{A}$ using one of the methods of Sections 2.12 - 2.14 is proportional to $m\left\{(N_y-1)(n-1)+n^2\right\}^2$. This number is to be compared with a value of $m(N_x+n-1)^2(N_y+n-1)^2$ if $\underset{\sim}{A}$ is regarded as full. Thus for a modest problem in which n = 4 (bicubic spline), $N_x = 8$ and $N_y = 5$, the above numbers are respectively about 800m and 8000m; consequently the algorithms that take advantage of the stepped-banded form are roughly an order of magnitude faster for this example.

Because of the remarks made earlier in this section relating to the difficulty of assessing in advance whether the least-squares solution is unique, and because we contend that many, probably most, practical data sets for which a multivariate spline approximation is required will give rise to a non-unique solution, the factorization method itself must be able to detect rank deficiency in $\underset{\sim}{A}$. The reason why we belive non-unique solutions are commonplace can be seen by the following illustration.

Suppose data covering a roughly elliptical region is prescribed. Then, if two sets of orthogonal knot lines are laid down over this data so as to contain it, there are very likely to be single panels void of data, or a number of adjacent panels with few data. In particular, a corner panel is likely to contain no data points, with the consequence that one of the basis functions will be zero at all data points, ie the corresponding column of $\underset{\sim}{A}$ will contain only zeros and hence $\underset{\sim}{A}$ will be rank deficient. Such a case is easily detected and remedied by setting the appropriate B-spline coefficient to zero and deleting the null column from the matrix before $\underset{\sim}{A}$ is triangularized. However, a less obvious form of deficiency may occur in which no columns of $\underset{\sim}{A}$ are identically zero. As a simple example, consider a case in which each panel contains precisely one data point. Then all columns of $\underset{\sim}{A}$ contain non-zeros, yet the rank of $\underset{\sim}{A}$ is at most equal to N_xN_y, the number of data points in this case, which is less than the number of columns of $\underset{\sim}{A}$ by $(n-1)(N_x+N_y+n-1)$. For a further informative discussion, see Hayes and Halliday (1974).

Any rank deficiency in $\underset{\sim}{A}$ is conveniently handled, after having computed the upper triangular factor, using the "resolving constraint" concept (Section 9.5).

10.5 The imposition of constraints

Many of the ideas of Sections 9.1 - 9.4 carry over to the multivariate case. We mention just two simple extensions. The first is the simple point constraint in which $s^{(r)}(x,y)$ is to take the value $f^{(r)}(t_o, u_o)$ at

$(x,y) = (t_0, u_0)$. Here r $(0 \leqslant r < n)$, t_0 $(a \leqslant t_0 \leqslant b)$, u_0 $(c \leqslant u_0 \leqslant d)$ and $f^{(r)}(t_0, u_0)$ are all prescribed. By analogy with the discussion of Section 9.3, such a constraint can be formed very readily and imposed using the methods of Section 9.5.

As an instance of a line constraint we consider the following example. Suppose $s(x,y)$ is to satisfy the line constraint

$$\left(\frac{\partial s}{\partial x}\right)_{x=a} = g(y), \tag{10.5.1}$$

where $g(y)$ is a prescribed function of y. Now

$$\left(\frac{\partial s}{\partial x}\right)_{x=a} = \sum_{i=1}^{n_x} \sum_{j=1}^{n_y} c_{ij} N'_{ni}(a) P_{nj}(y) \tag{10.5.2}$$

and hence

$$\sum_{j=1}^{n_y} d_j P_{nj}(y) = g(y) \ , \tag{10.5.3}$$

where

$$d_j = \sum_{i=1}^{n_x} c_{ij} N'_{ni}(a)$$

$$= \sum_{i=1}^{2} c_{ij} N'_{ni}(a) \ , \tag{10.5.4}$$

using the compact support property.

Evidently, such a constraint can be imposed exactly only if $g(y)$ is a spline of order n in y with the same y-knots as $s(x,y)$, or if $g(y)$ is a polynomial of degree less than n in y (which is of course a special case of a spline of order n). In the former case, if $g(y)$ is expressed in its normalized B-spline form its coefficients are simply the values of d_j.

In the latter case, if $g(y)$ is expressed in its power-series form, Algorithm 5.7.1 can be used to determine the d_j. If $g(y)$ falls into neither of these categories it is recommended that it is first approximated, perhaps by using one of the interpolation algorithms of Section 6.4, by a spline of order n having the same y-knots as $s(x,y)$.

In any case the d_j are usually readily found. The remaining step is the imposition of the n_y linear constraints (10.5.4), which may be carried out as in Section 9.5 or by a suitable modification of the basis as in Section 9.1.

10.6 Evaluation of a multivariate spline from its B-spline representation

Consider the evaluation of the bivariate spline (10.3.3) for given values of x and y ($a \leqslant x \leqslant b$, $c \leqslant y \leqslant d$). Since

$$s(x,y) = \sum_{j=1}^{n_y} d_j(x) P_{nj}(y) , \tag{10.6.1}$$

where

$$d_j(x) = \sum_{i=1}^{n_x} c_{ij} N_{ni}(x) \qquad (j = 1, 2, \ldots, n_y), \tag{10.6.2}$$

s can evidently be evaluated by forming each of the (at most) n values of $d_j(x)$ in (10.6.2) corresponding to the non-zero values of $P_{nj}(y)$, followed by the evaluation of (10.6.2). A total of $n+1$ spline evaluations is required and if either Algorithm 5.2.1 or Algorithm 5.2.2 were employed would take a total of $\frac{3}{2}n^3 + O(n^2)$ long operations. Note, however, that advantage can be taken of the fact that for n of these evaluations, the same knot set is employed. The following minor modification of Algorithm 5.2.2 accomplishes this improvement.

Algorithm 10.6.1 : The evaluation of $s(x,y)$ from its normalised B-spline representation.

Step 1. Determine k and l such that $x_{k-1} \leq x < x_k$ and $y_{l-1} \leq y < y_l$ using sequential or binary search.

Step 2. Use Algorithm 3.12.2 to evaluate $v_i = N_{ni}(x)$ for $i = k, k+1, \ldots, k+n-1$.

Step 3. For $j = l, l+1, \ldots, l+n-1$ form $d_j = \sum_{i=k}^{k+n-1} v_i c_{ij}$.

Step 4. Use either Algorithm 5.2.1 or Algorithm 5.2.2 to evaluate $\sum_{j=1}^{n_y} d_j P_{nj}(y)$.

The total number of long operations taken by Algorithm 10.6.1 is $4n^2+O(n)$.

An error analysis of Algorithm 10.6.1, carried out in a similar fashion to those of Algorithms 5.2.1 and 5.2.2 reveals that the computed value $\bar{s}(x,y)$ satisfies

$$\left|\bar{s}(x,y) - s(x,y)\right| \leq 15.59n2^{-t} \max_{k \leq i < k+n} \max_{l \leq j < j+n} \left|c_{ij}\right|. \qquad (10.6.3)$$

Algorithm 10.6.1 and the error analysis may be extended in an obvious way to multivariate splines in p dimensions. The form of the error bound is the natural extension of (10.6.3), the constant 15.59 being replaced by 7.745p.

REFERENCES

AHLBERG, J H, NILSON, E N and WALSH, J L. The Theory of Splines and Their Applications. New York: Academic Press, 1967.

AMOS, D E and SLATER, M L. Polynomial and spline approximation by quadratic programming. Commun. Assoc. Comput. Mach., 1969, 12, 379-381.

BAUER, F L. Elimination with weighted row combinations for solving linear equations and least squares problems. Numerische Math., 1965, 7, 338-352.

BARRODALE, I. Approximation in the L_1 and L_∞ norms by linear programming. University of Liverpool, PhD Thesis, 1967.

BARRODALE, I and ROBERTS, F D K. An improved algorithm for discrete l_1 linear approximation. University of Victoria, Department of Mathematics Report (un-numbered), British Columbia, Canada, 1971.

BARRODALE, I and YOUNG, A. Algorithms for best L_1 and L_∞ approximations on a discrete set. Numerische Math., 1966, 8, 295-306.

BARTELS, R H and GOLUB, G H. The simplex method of linear programming using LU decomposition. Commun. Assoc. Comput. Mach., 1969, 12, 266-268.

BELLMAN, R, KASHEF, B and VASUDEVAN, R. Mean square spline approximation. J. Math. Anal. Applics., 1974, 45, 47-53.

BJÖRCK, A. Solving linear least squares problems by Gram-Schmidt orthogonalization, BIT, 1967, 7, 1-21.

BOOR, C DE. On calculating with B-splines. J.Approximation Theory, 1972, 6, 50-62.

BOOR, C DE. Good approximation by splines with variable knots. In A Meir and A Sharma (Eds), Spline Functions and Approximation Theory, ISNM 21, 57-72. Basel: Birkhäuser Verlag, 1973.

BOOR, C DE and FIX, G J. Spline approximation by quasiinterpolants. J. Approximation Theory, 1973, 8, 19-45.

BOOR, C DE and RICE, J R. Least squares cubic spline approximation I - Fixed knots. Purdue University. Report CSD TR20, 1968.

BOOR, C DE and RICE, J R. Least squares cubic spline approximation II - Variable knots. Purdue University. Report CSD TR21, 1968.

BOOT, J C G. Quadratic Programming. Chicago: Rand McNally, 1964.

BRITISH STANDARD CODE OF PRACTICE. CP118. The Structural Use of Aluminium. London: British Standards Institution, 1969.

BUSINGER, P A. Monitoring the numerical stability of Gaussian elimination. Numerische Math., 1971, 16, 360-361.

BUSINGER, P and GOLUB, G H. 1965. Linear least squares solutions by Householder transformations. Numerische Math., 1965, 7, 269-276.

BUTTERFIELD, K R. A generalized matrix decomposition of the mth derivative of a B-spline basis of order k associated with numerical evaluations. To appear in J. Inst. Maths. Applics., 1975.

CARASSO, C. Méthodes numériques pour l'obtention de fonctions-spline. Thèse de 3ème Cycle, Université de Grenoble, 1966.

CARASSO, C and LAURENT, P J. On the numerical construction and practical use of interpolating spline-functions. In A J H Morrell (Ed.), Information Processing 68. Amsterdam: North-Holland, 1969.

CLENSHAW, C W. Chebyshev series for mathematical functions. National Physical Laboratory Mathematical Tables 5. London: Her Majesty's Stationery Office, 1962.

CLENSHAW, C W and HAYES, J G. Curve and surface fitting. J. Inst. Maths. Applics., 1965, 1, 164-183.

CLINE, A K. An elimination method for the solution of linear least squares problems. SIAM J. Numer. Anal., 1973, 10, 284-289.

COX, M G. An algorithm for approximating convex functions by means of first degree splines. Comput. J., 1971, 14, 272-275.

COX, M G. Curve fitting with piecewise polynomials. J. Inst. Maths. Applics., 1971, 8, 36-52.

COX, M G. The numerical evaluation of B-splines. J. Inst. Maths. Applics., 1972, 10, 134-149.

COX, M G. Cubic-spline fitting with convexity and concavity constraints. National Physical Laboratory, Teddington, Middlesex. Report NAC23, 1973.

COX, M G and LEHRIAN, D E. Real procedure 'zero': to determine a zero of a real continuous function, given lower and upper bounds on the zero. National Physical Laboratory, Teddington, Middlesex. NPL Algorithms Library Document C5/01/0/Algol 60/1/73, January 1973.

COX, M G. A data-fitting package for the non-specialist user. In D J Evans (Ed.), Software for Numerical Mathematics, pp 235-251. London: Academic Press, 1974.

COX, M G. Procedure 'cubic spline fit': to compute a weighted least-squares approximation to an arbitrary set of data points by a cubic spline with prescribed knots, and to perform cubic spline interpolation. National Physical Laboratory, Teddington, Middlex. NPL Algorithms Library Document E2/03/0/Algol 60/4/74, April 1974.

COX, M G. Procedure 'spdeg3': to evaluate a cubic spline from its B-spline representation. National Physical Laboratory, Teddington, Middlesex. NPL Algorithms Library Document E2/05/0/Algol 60/4/74, April 1974.

COX, M G. Numerical computations associated with Chebyshev polynomials. Presented at Royal Irish Academy Conference on Numerical Analysis, Dublin, August, 1974.

COX, M G. Subroutine 'SP3FIT': to compute a weighted least-squares approximation to an arbitrary set of data points by a cubic spline with prescribed knots, and to perform cubic spline interpolation. National Physical Laboratory, Teddington, Middlesex. NPL Algorithms Library Document E2/03/0/Fortran IV/11/74, November 1974.

COX, M G. Subroutine 'SPDEG3': to evaluate a cubic spline from its B-spline representation. National Physical Laboratory, Teddington, Middlesex. NPL Algorithms Library Document E2/05/0/Fortran IV/11/74, November 1974.

COX, M G. An algorithm for spline interpolation. J. Inst. Maths. Applics., 1975, 15, 95-108.

COX, M G and HAYES, J G. Curve fitting: a guide and suite of algorithms for the non-specialist user. National Physical Laboratory, Teddington, Middlesex.Report NAC 26, December 1973.

CURRY, H B and SCHOENBERG, I J. On Pólya frequency functions IV: the fundamental spline functions and their limits. J. Analyse Math., 1966, 17, 71-107.

DAVIS, P J. Interpolation and Approximation. New York, Blaisdell, 1963.

DODSON, D S. Optimal order approximation by polynomial spline functions. PhD Thesis. Computer Science Department, Purdue University, Lafayette, USA, 1972.

DRAPER, N R and SMITH, H. Applied Regression Analysis. New York: Wiley, 1968.

ESCH, R E and EASTMAN, W L. Computational methods for best spline function approximation. J. Approximation Theory, 1969, 2, 85-96.

FRANCIS, J G F. The QR transformation, Parts I and II. Comput. J., 1961/2, 4, 265-271, 332-345.

GAFFNEY, P W. The calculation of indefinite integrals of B-splines. Atomic Energy Research Establishment, Harwell. Report CSS11, 1974.

GENTLEMAN, W M. An error analysis of Goertzel's (Watt's) method for computing Fourier coefficients. Computer J., 1969, 12, 160-165.

GENTLEMAN, W M. Basic procedures for large, sparse, or weighted linear least squares problems. University of Waterloo, Ontario, Canada, Research Report CSRR-2068, 1972.

GENTLEMAN, W M. Least squares computations by Givens transformations without square roots. J. Inst. Maths. Applics., 1973, 12, 329-336.

GENTLEMAN, W M. Interface between numerical analysis and symbolic computation. Paper presented at the ACM/SIAM conference "Mathematical Software II", July 1974, Purdue University, Lafayette, Indiana.

GILL, P E and MURRAY, W. A numerically stable form of the simplex algorithm. Linear Algebra and it Applns., 1973, 7, 99-138.

GOLUB, G H. Numerical methods for solving linear least squares problems. Numerische Math., 1965, 7, 206-216.

GOLUB, G H and BUSINGER, P. Least-squares, singular values and matrix approximations. An ALGOL procedure for computing the singular value decomposition. Technical Report No. CS73, Stanford University, California, USA, 1967.

GOLUB, G H and KAHAN, W. Calculating the singular values and pseudo-inverse of a matrix. J. SIAM Numer. Anal., 1965, 2, 205-224.

GOLUB, G H and REINSCH, C. Singular value decomposition and least squares solutions. Numerische Math., 1970, 14, 403-420.

GOLUB, G H and WILKINSON, J H. Note on the iterative refinement of least squares solution. Numerische Math., 1966, 9, 139-148.

GREVILLE, T N E. Note on fitting of functions of several independent variables. J. Soc. Indust. Appl. Math., 1961, 9, 109-115.

GREVILLE, T N E. Introduction to spline functions. In T N E Greville (Ed), Theory and Application of Spline Functions, pp 1-35. New York: Academic Press, 1969.

HAMMARLING, S. A note on modifications to the Givens plane rotation. J. Inst. Maths. Applics., 1974, 13, 215-218.

HANSON, R J and LAWSON, C L. Extensions and applications of the Householder algorithm for solving linear least-squares problems. Math. Comput., 1969, 23, 787-812.

HARTLEY, P. Tensor product approximations to data defined on rectangular meshes in n-space. J. Inst. Maths. Applics., 1976. To appear.

HAYES, J G and HALLIDAY, J. The least-squares fitting of cubic spline surfaces to general data sets. J. Inst. Maths. Applics., 1974, 14, 89-103.

HOUSEHOLDER, A S. Unitary triangularization of a nonsymmetric matrix. J. Assoc. Comput. Mach., 1958, 5, 339-342.

LAFATA, P and ROSEN, J B. An interactive display for approximation by linear programming. Commun. Assoc. Comput. Mach., 1970, 13, 651-659.

LÄUCHLI, P. Jordan elimination und Ausgleichung nach kleinsten Quadraten. Numerische Math., 1961, 3, 226-240.

LAWSON, C L and HANSON, R J. Solving least squares problems. Englewood Cliffs, New Jersey: Prentice-Hall, Inc. 1974.

MARSDEN, M J. An identity for spline functions with applications to variation-diminishing spline approximation. J. Approximation Theory, 1970, 3, 7-49.

MARTIN, R S and WILKINSON, J H. Solution of symmetric and unsymmetric band equations and the calculation of the eigenvectors of band matrices. Numerische Math., 1967, 9, 279-301.

MOLER, C. Fast Givens. In informal proceedings of ACM/SIAM conference Mathematical Software II, pp 313-318. Purdue University, Lafayette, Indiana, May 1974.

NAUR, P. (Ed.). Revised report on the algorithmic language ALGOL 60. Comput. J., 1963, 5, 349-367.

PETERS, G and WILKINSON, J H. The least squares problem and pseudo-inverses. Comput. J., 1970, 13, 309-316.

PETERS, G and WILKINSON, J H. Practical problems arising in the solution of polynomial equations. J. Inst. Maths. Applics., 1971, 8, 16-35.

POWELL, M J D. Curve fitting by splines in one variable. In J G Hayes (Ed.), Numerical Approximation to Functions and Data, pp 65-83. London: Athlone Press, 1970.

RABINOWITZ, P. Applications of linear programming to numerical analysis. SIAM Rev., 1968, 10, 121-159.

REID, J K. A note on the least squares solution of a band system of linear equations by Householder reductions. Comput. J., 1967, 10, 188-189.

REID, J K. A note on the stability of Gaussian elimination. J. Inst. Maths. Applics., 1971, 8, 374-375.

RICE, J R. The Approximation of Functions, Vol 1. Reading, Massachusetts: Addison-Wesley, 1964.

RICE, J R. Experiments on Gram-Schmidt orthogonalization. Math. Comput., 1966, 20, 325-328.

RICE, J R. The Approximation of Functions, Vol 2. Reading, Massachusetts: Addison-Wesley, 1969.

ROOIJ, P L J VAN and SCHURER, F. A bibliography on spline functions II. Technological University Eindhoven, Netherlands. Report 73-WSK-01, 1973.

SCHOENBERG, I J. Contributions to the problem of approximation of equidistant data by analytic functions. Quart. Appl. Maths., 1946, 4, 45-99, 112-141.

SCHOENBERG, I J and WHITNEY, Anne. On Pólya frequency functions III. Trans. Amer. Math. Soc., 1953, 74, 246-259.

SCHUMAKER, L L. Some algorithms for the computation of interpolating and approximating spline functions. In T N E Greville (Ed.). Theory and Application of Spline Functions, pp 87-102. New York: Academic Press, 1969.

SCOWEN, R S. BABEL, a new programming language. National Physical Laboratory, Teddington, Middlesex. Report CCU 7, 1969.

SEGETHOVA, J. Numerical construction of the hill functions. Technical Report 70-110, NGL-21-002-008. University of Maryland.

SEGETHOVA, J. Numerical construction of the hill functions. SIAM J. Numer. Anal., 1972, 9, 199-204.

STEFFENSEN, J F. Interpolation. New York: Chelsea Publishing Company, 1927.

WICHMANN, B A. Estimating the execution speed of an Algol program. Report NAC 38. National Physical Laboratory, Teddington, Middlesex, 1973.

WILKINSON, J H. Error analysis of direct methods of matrix inversion. J. Assoc. Comput. Mach., 1961, 8, 281-330.

WILKINSON, J H. Rounding Errors in Algebraic Processes. Notes on Applied Science No 32. London: Her Majesty's Stationery Office, 1963.

WILKINSON, J H. The Algebraic Eigenvalue Problem. Oxford: Clarendon Press, 1965.

WILKINSON, J H. Private discussions. 1974.

WILKINSON, J H and REINSCH, C. (Eds). Handbook for Automatic Computation, Volume II, Linear Algebra. New York: Springer-Verlag, 1971.