Cranfield University
School of Engineering
Applied Mathematics & Computing Group

PhD Thesis
Academic Year 2003-2004

P. A. Sherar

Variational Based Analysis and Modelling using B-splines

Supervisor: Professor Chris Thompson

A thesis submitted in partial fulfillment of the
requirements for the degree of

Doctor of Philosophy

CRANFIELD UNIVERSITY
School of Engineering

P. A. Sherar

Variational Based Analysis and Modelling using B-splines

PhD Thesis

Abstract

The use of energy methods and variational principles is widespread in many fields of engineering of which structural mechanics and curve and surface design are two prominent examples. In principle many different types of function can be used as possible trial solutions to a given variational problem but where piecewise polynomial behaviour and user controlled cross segment continuity is either required or desirable, B-splines serve as a natural choice. Although there are many examples of the use of B-splines in such situations there is no common thread running through existing formulations that generalises from the one dimensional case through to two and three dimensions.

We develop a unified approach to the representation of the minimisation equations for B-spline based functionals in tensor product form and apply these results to solving specific problems in geometric smoothing and finite element analysis using the Rayleigh-Ritz method. We focus on the development of algorithms for the exact computation of the minimisation matrices generated by finding stationary values of functionals involving integrals of squares and products of derivatives, and then use these to seek new variational based solutions to problems in the above fields. By using tensor notation we are able to generalise the methods and the algorithms from curves through to surfaces and volumes.

The algorithms developed can be applied to other fields where a variational form of the problem exists and where such tensor product B-spline functions can be specified as potential solutions.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Preliminaries

The use of B-splines is very well established in the field of geometric modelling as testified by their mainstream integration in the majority of CAD/CAM packages. Since the early days of curve and surface design with the work of Bézier and de Casteljau, their flexibility and convenient mathematical and computational properties have been put to constant use in the design and development of products for which external shape is a critical factor. In addition to shape definition and spurred on by the ever increasing amount of computing power and resources available to the developer, there have been significant advances more recently in the use of B-splines for modelling and analysis of more complex phenomena such as human organs in medical imaging, [3], [71], [72], [74], image processing, [16], [45], and the physical behaviour of objects under external forces, [69], [81], [82]. Some of these developments have begun to employ B-spline volumes for their description.

An important tool in the repertoire of techniques for describing these systems is that of the principle of minimum total potential energy[1] and the associated use of variational methods to determine approximate solutions to the underlying displacement function. Here one seeks a stationary value of a functional with respect to a set of undetermined parameters representing a solution, where the functional may represent the total energy of the system or an integral representation of governing equations.

Variational principles are also used heavily in curve and surface design. Here the fundamental idea is the use of modelling tools which minimise a certain functional that can be interpreted in

[1] If a system is in equilibrium then its total potential energy is a minimum

terms of physics or geometry. For example, the process of removing unwanted imperfections, so-called *fairing* or *smoothing*, is an important aspect of free-form geometric modellers. Historically, thin elastic strips of wood were used for designing ship hulls and airplanes. Since elastic strips tend to minimise their bending energy and curves of least energy are considered fair, energy based functionals are used to improve shape quality. Such functionals can also be used as free-form design tools as demonstrated in, for example, [6], [14], [91] and [92].

In structural analysis the principle of minimum energy is used to obtain approximate solutions to mechanical models for deflections of objects under external loads. Using the equivalence between the differential equation and the variational form of the problem, a solution can be found by minimising a functional representing the total potential energy of the system subject to boundary conditions.

In this work we propose to develop a unified approach to the use of tensor product B-splines in curve, surface and volume form, in solving variational type problems based on the exact[2] computation of the minimisation matrices of functionals involving squares and products of derivatives, and to apply the results and associated algorithms to the specific fields of geometric smoothing and structural analysis.

1.2 Layout of the thesis

The outline of this document is as follows.

In Chapter 2 we begin by reviewing tensor notation and the associated contraction and product operators. The definition and principle algorithms for B-spline curves are covered and their tensor product extensions to surfaces and volumes. Whilst most of this material serves to introduce notation and basic properties of B-splines that are used in the rest of the document some new notation and algorithms are presented. In particular, the matrix system for expressing derivative control points in terms of the original and an explicit representation of a derivative of a B-spline curve as a B-spline curve. These results are generalised to the surface and volume forms.

Chapter 3 focuses on the derivation of the linear matrix systems for minimisation of B-spline functionals based on squares and products of derivatives. After reviewing the connection between the differential equation and variational formulations of a given continuum problem the matrix form of the derivative from Chapter 2 is used along with the tensor notation to produce minimisation matrices and formulae for the curve, surface and volume cases. Algorithms for the

[2]by 'exact' we mean within the limits of roundoff error, numerical approximations are not used

exact computation of these matrices are presented. The chapter proceeds with a review of the reduced transformation technique which is employed in this thesis for handling the boundary conditions in the variational formulation. The chapter concludes with algorithms for computing exactly the integral of a product B-spline for dealing with the source and boundary term integration.

The application of the functional minimisation formulae in Chapter 3 to B-spline curve, surface and volume smoothing is covered in Chapter 4. After reviewing the background to energy based minimisation methods for improving shape quality the principle techniques and contributions to the field are surveyed. There are two general types of algorithm used for smoothing, the first based on iterative local modification of control points and/or knots to reduce unwanted curvature variation and, secondly, those that use global methods based on minimising an energy based or similar functional. The contribution made here is to the latter group of techniques. We begin by presenting two curve algorithms, one which can be applied in conjunction with data fitting and the other for post construction smoothing, and give examples based on artificially perturbed curves. The technique is generalised to the surface case and examples presented which illustrate graphically and numerically the effectiveness of the algorithms. The chapter concludes with a section on volume methods. A survey of the use of volumetric models is presented and then, using the Kronecker product, the smoothing method for curves and surfaces is generalised to the volume case with functional and parametric examples provided.

Chapter 5 covers applications to the solution of certain types of finite element problem based on rectangular domains. We review briefly the use of B-splines in finite elements, convergence criteria and some relevant papers in the literature that are based on using the minimum energy principle. We then proceed to use this principle with the Rayleigh-Ritz method and the minimisation formulae from Chapter 3 to derive algorithms for constructing B-spline solutions to some problems in beam and plate bending and the deformation of isotropic solids. In each case the reduced transformation technique is used to deal with the geometric boundary conditions. The algorithms presented return exact polynomial solutions where they exist and approximations otherwise. A number of plate bending examples are tested and comparisons made with theoretical results based on trigonometric series solutions from Timoshenko, [85]. Within the limits of the comparison data, convergence trends for increasing segment number and order are shown. For the plate bending examples the developments here serve as a generalisation of the results presented in Antes, [2]. The chapter finishes with an application of the volume based algorithm to the solution of Laplace's equation for heat flow through a cube.

Finally, Chapter 6 draws some conclusions of the work and highlights some areas for further

development.

In Appendix A we present the full numerical results for the plate bending examples presented in Chapter 5. Some additional examples with varying boundary conditions are also given.

Appendix B provides definitions of the principle C++ classes used in the construction of the software that wraps the algorithms covered in the four chapters.

In Appendix C we present UML type diagrams illustrating the inheritance structure of the classes listed in Appendix B.

Appendix D presents implementations of the basic Kronecker product operations used in the surface and volume algorithms of Chapters 4 and 5.

In Appendix E we give details of the steps to derive the basic B-spline point and derivative evaluation formulae referred to in Chapter 2.

Finally, in Appendix F we provide the main formulae for the curvature properties of curves, surfaces and volumes used as smoothing criteria in Chapter 4.

Chapter 2

B-spline Curve, Surface and Volume Algorithms

2.1 Introduction

B-splines in curve and surface form have enjoyed great popularity in the broad field of geometric modelling over the past 30 years or so. This is principally due to their attractive mathematical properties, convenient notation, and the very well established and computationally efficient algorithms that have been developed for their manipulation. From the early work of de Casteljau and Bézier in the automobile industry on what today are known as Bézier curves and surfaces, through to the more recent development of dynamic models for free-form deformation and physics based manipulation of surfaces and solids, the B-spline form has remained a powerful standard for the representation of shape where complex data and physical properties have to be modelled.

In this chapter we focus on B-spline properties and associated algorithms. After reviewing tensor notation which we use to provide a common notational framework for curves, surface and volumes, we concentrate on the fundamental B-spline curve algorithms of evaluation, derivative, knot insertion/removal, product and integration and their tensor product generalisations which are used extensively in the minimisation algorithms to follow. In doing so we introduce some new notation and algorithms for explicit representation of the derivative of B-spline curves which we also generalise to surfaces and volumes.

2.2 Background on Tensors

A tensor of rank n is an array of 3^n quantities which obey certain rules of transformation when the coordinate axes are rotated. We are concerned here not with the details of these transformation laws but rather on the operations that can be defined on tensors. The main reason for using tensor notation in this and in subsequent chapters is that it enables us to unify the notation for the treatment of B-spline curve, surface and volume entities with respect to evaluation of derivatives and the formulation of the matrix equations resulting from the minimisation of certain functionals. We concentrate on tensors of rank 0 to 3, which correspond to scalars, vectors, matrices, and the three dimensional matrix analogue, as well as the basic allowable operations on these entities. In particular, we are concerned with how the familiar form of these operations for the rank 0 to 2 cases is extended using the tensor notation to the three dimensional case.

For the purposes of this thesis a tensor of rank 0 is a scalar quantity, a tensor of rank 1 is a vector and that of rank 2 a matrix. A tensor of rank 3 is a three dimensional matrix with rows, columns and what we shall call 'layers'. We refer to such an entity as a *matrix3D*. Since we are concerned primarily with the use of the notation we don't restrict ourselves to the operations being performed only on entities of size 3^n. We allow the vectors, matrices and matrix3D entities to be of any size subject to the compatibility restrictions on the numbers of rows, columns and layers that are necessary for the operations to be carried out. We also make use of the Einstein summation convention when appropriate which states that whenever two indices on a tensor are identified the implication is that the corresponding components of the tensor are summed over the range of the index in question (this is the contraction operation described later). Figure 2.1 illustrates the three basic entities we are dealing with. To demonstrate the allowable operations between these entities we will use the following notation for specific tensors of rank 1, 2 and 3 respectively:

$$\text{Rank 1:} \quad \mathbf{P} = \left(p_i\right)_{i=1}^{m}, \qquad \mathbf{Q} = \left(q_i\right)_{i=1}^{n}, \qquad \mathbf{R} = \left(r_i\right)_{i=1}^{p}$$

$$\text{Rank 2:} \quad \mathbf{A} = \left(a_{ij}\right)_{i,j=1}^{m,m}, \qquad \mathbf{B} = \left(b_{ij}\right)_{i,j=1}^{n,n}, \qquad \mathbf{C} = \left(c_{ij}\right)_{i,j=1}^{p,p},$$

$$\text{Rank 3:} \quad \mathbf{D} = \left(d_{ijk}\right)_{i,j,k=1}^{m,n,p}, \qquad \mathbf{E} = \left(e_{ijk}\right)_{i,j,k=1}^{m,n,p}.$$

We use $\mathbf{T}$ to denote the result of an allowable operation.

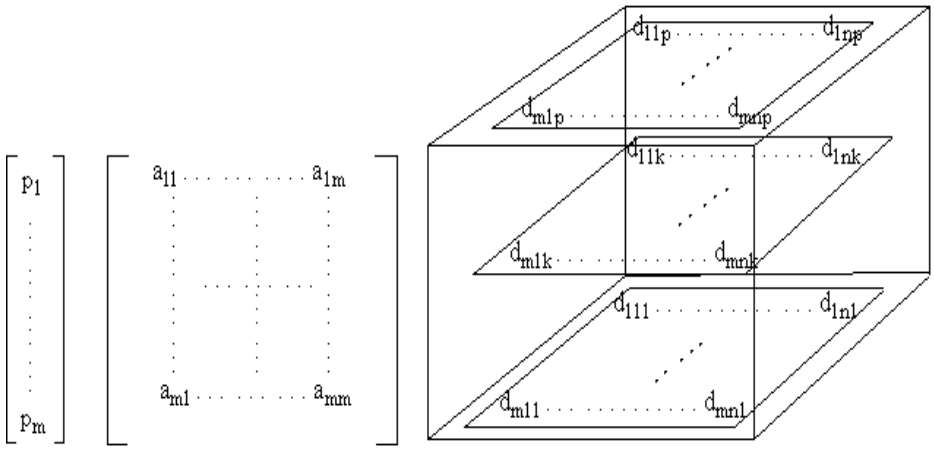


Figure 2.1: Tensors of rank 1, 2 and 3

2.2.1 Operations on tensors

Addition and Subtraction

Two tensors of the same rank can be added assuming that they have the same number of rows, rows and columns, or rows, columns and layers respectively. The sum is a tensor of the same order whose components are the sums of the corresponding components of the two tensors. This corresponds to familiar vector and matrix addition and its logical extension to the matrix3D case. We use the symbol $\oplus$ for this operation and $\ominus$ for subtraction.

Multiplication

We can multiply any two of the rank 1,2 or 3 tensors together using a product operation. The product is defined to be a tensor whose rank is the sum of the ranks of the two tensors and whose components are products of a component of one tensor with any component of the other tensor. This operation is denoted by the symbol $\otimes$. The cases of interest are listed below:

- **vector** $\otimes$ **vector**: For $\mathbf{P}$ and $\mathbf{Q}$ two tensors of rank 1 (vectors) their tensor product $\mathbf{P} \otimes \mathbf{Q}$ is the matrix of quantities obtained by multiplying each component of $\mathbf{P}$ with each

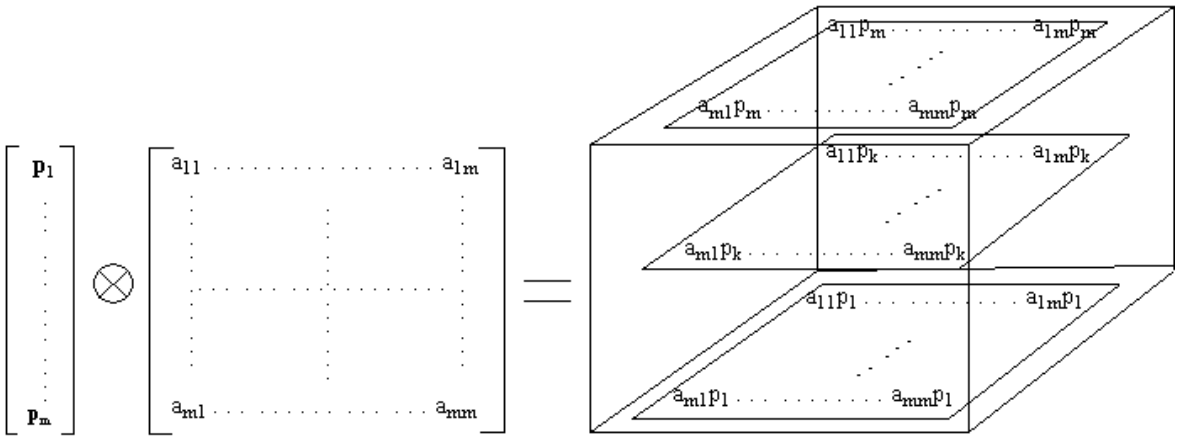


Figure 2.2: Tensor multiplication

component of **Q**:

$$\mathbf{T} = \mathbf{P} \otimes \mathbf{Q} = \begin{pmatrix} p_1 \\ \vdots \\ p_m \end{pmatrix} \otimes \begin{pmatrix} q_1 \\ \vdots \\ q_n \end{pmatrix} = \begin{pmatrix} p_1 q_1 & \cdots & p_1 q_n \\ \vdots & \ddots & \vdots \\ p_m q_1 & \cdots & p_m q_n \end{pmatrix}.$$

Using index notation we may also write this more simply as

$$\mathbf{T} = T_{ij} = p_i q_j.$$

- **vector** $\otimes$ **matrix**: For **P** a tensor of rank 1 and **A** a tensor of rank 2, their tensor product $\mathbf{T} = \mathbf{P} \otimes \mathbf{A}$ is the matrix3D of quantities obtained by multiplying each component of **P** with each component of **Q**, see figure 2.2. Using index notation we can write

$$\mathbf{T} = T_{ijk} = p_i a_{jk}.$$

- **matrix** $\otimes$ **vector**: For **A** a tensor of rank 2 and **P** a tensor of rank 1, their tensor product $\mathbf{T} = \mathbf{A} \otimes \mathbf{P}$ is the matrix3D of quantities obtained by multiplying each component of **A** with each component of **P**:

$$\mathbf{T} = T_{ijk} = a_{ij} p_k.$$

- **matrix $\otimes$ matrix**: **The Kronecker Product** For $\mathbf{A}$ and $\mathbf{B}$ both tensors of rank 2, their tensor product $\mathbf{T} = \mathbf{A} \otimes \mathbf{B}$ is the matrix of matrices (that is a *matrix4D*) of quantities obtained by multiplying each component of $\mathbf{A}$ with each component of $\mathbf{B}$:

$$\mathbf{T} = T_{ijkl} = a_{ij} b_{kl}. \tag{2.1}$$

Although this takes us outside the rank 1-3 tensors we are principally interested in, the product defined here has a particular significance when it comes to solving certain systems of linear equations that appear in later chapters and is called the *Kronecker product*. This product has many attractive algebraic properties and forms the basis of a wide variety of algorithms in scientific computing, see for example Graham 1981, [25], Henderson et al, [35].

It is always possible to represent a higher order tensor like 2.1 by a matrix in which each row and/or column is indexed by one of the set of of component indices and letting these sets be arranged in some systematic manner. For example, if

$$\mathbf{A} = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix}, \quad \mathbf{B} = \begin{pmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \\ b_{31} & b_{32} & b_{33} \end{pmatrix},$$

we can write

$$\mathbf{A} \otimes \mathbf{B} = \left(\begin{array}{ccc|ccc} a_{11}b_{11} & a_{11}b_{12} & a_{11}b_{13} & a_{12}b_{11} & a_{12}b_{12} & a_{12}b_{13} \\ a_{11}b_{21} & a_{11}b_{22} & a_{11}b_{23} & a_{12}b_{21} & a_{12}b_{22} & a_{12}b_{23} \\ a_{11}b_{31} & a_{11}b_{32} & a_{11}b_{33} & a_{12}b_{31} & a_{12}b_{32} & a_{12}b_{33} \\ \hline a_{21}b_{11} & a_{21}b_{12} & a_{21}b_{13} & a_{22}b_{11} & a_{22}b_{12} & a_{22}b_{13} \\ a_{21}b_{21} & a_{21}b_{22} & a_{21}b_{23} & a_{22}b_{21} & a_{22}b_{22} & a_{22}b_{23} \\ a_{21}b_{31} & a_{21}b_{32} & a_{21}b_{33} & a_{22}b_{31} & a_{22}b_{32} & a_{22}b_{33} \end{array}\right). \tag{2.2}$$

Here the rows and columns of the above matrix are indexed by the pair (i, j), with the pair (j, k) arranged in the normal manner. In general, if $\mathbf{A}$ and $\mathbf{B}$ are matrices of size $m_1 \times n_1$ and $m_2 \times n_2$ respectively, then we interpret $\mathbf{A} \otimes \mathbf{B}$ an m_1 by n_1 block matrix whose (i, j) block is the m_2 by n_2 matrix $b_{ij}\mathbf{C}$.

When working with Kronecker products, matrix and matrix3D entities are sometimes regarded as vectors and vectors are sometimes made into matrices and matrix3D entities. In particular, if $\mathbf{A}$ is an $m \times n$ matrix we can convert it into a vector by stacking the columns. We can perform a similar operation on matrix3D entities by treating the level matrices making up the 3D entity and stacking their columns layer by layer. We denote this generic operation of

converting higher order entities to vector form by the notation $vec(\mathbf{A})$ (for some background to this operator see Henderson & Searle [36]). The significance of this operation becomes apparent when considering the matrix equation, $\mathbf{CXB^T} = \mathbf{Y}$, where $\mathbf{C}, \mathbf{X}, \mathbf{B}$ are compatible matrices such that the product is defined. In this case we have the following equivalence

$$\mathbf{CXB^T} = \mathbf{Y} \equiv (\mathbf{B} \otimes \mathbf{C})vec(\mathbf{X}) = vec(\mathbf{Y}). \tag{2.3}$$

We use this result and the conversion operation in chapters 4 and 5. Appendix D gives code for the Kronecker product and the vector/matrix and vector/matrix3D conversion operations.

Contraction

A contraction of one tensor with another is the result of setting two component indices equal to each other and summing over the values of those indices. This operation can only be performed if the number of rows, columns and/or layers corresponding to the repeated indices are the same in the two tensors. A contraction operation can be applied more than one time. Each contraction produces a tensor whose rank is the sum of the ranks of the two tensors minus two. We use the symbol $\odot$ for this operation. To distinguish the various indices we can sum over we use an index on this symbol in the form $\odot_l$, where l can be any one of the r indices available for a tensor of rank r. For tensors of rank ≥ 2 the generic operation can be described as follows:

For $\mathbf{T} = \mathbf{T}_1 \odot_l \mathbf{T}_2$ *we fix an index different from* l *in* $\mathbf{T}_1$ *and then multiply what we can call sub-tensors (for a matrix this will be a vector and for a matrix3D a matrix) in that chosen direction by* $\mathbf{T}_2$ *putting the results into* $\mathbf{T}$ *again in the chosen direction.*

For completeness we list the cases of interest below:

- **vector** $\odot$ **vector**: Although there is only one index to sum over for a vector we will introduce two equivalent forms for the operation which will enable us to match with the rank 2 tensor contractions. Assuming that $m = n$ for the vectors $\mathbf{P}$ and $\mathbf{Q}$, the contraction $\mathbf{T} = \mathbf{P} \odot_i \mathbf{Q}$ is the following tensor of rank 0 (i.e. a scalar):

$$\mathbf{T} = \mathbf{P} \odot_i \mathbf{Q} = \begin{pmatrix} q_1 & \dots & q_m \end{pmatrix} \begin{pmatrix} p_1 \\ \vdots \\ p_m \end{pmatrix} = q_1 p_1 + \ldots + q_m p_m.$$

  In more familiar terms this is just the scalar product written as $\mathbf{Q}^T\mathbf{P}$. Using index notation we can write it as

$$\mathbf{T} = T = p_i q_i.$$

We also identify the same product with $\mathbf{T} = \mathbf{P} \odot_j \mathbf{Q}$:

$$\mathbf{T} = \mathbf{P} \odot_j \mathbf{Q} = \begin{pmatrix} p_1 & \dots & p_m \end{pmatrix} \begin{pmatrix} q_1 \\ \vdots \\ q_m \end{pmatrix} = p_1 q_1 + \ldots + p_m q_m,$$

which is just the scalar product written as $\mathbf{P}^T\mathbf{Q}$.

- **matrix $\odot$ vector**: There are two possible contractions of a matrix with a vector, both producing a vector and corresponding to familiar matrix-vector and vector-matrix multiplication:

$$\mathbf{T} = \mathbf{A} \odot_j \mathbf{P} = \begin{pmatrix} a_{11} & \dots & a_{1m} \\ \vdots & \ddots & \vdots \\ a_{m1} & \dots & a_{mm} \end{pmatrix} \begin{pmatrix} p_1 \\ \vdots \\ p_m \end{pmatrix} = \begin{pmatrix} \sum_i a_{1i} p_i \\ \vdots \\ \sum_i a_{mi} p_i \end{pmatrix},$$

$$\mathbf{T} = \mathbf{A} \odot_i \mathbf{P} = \begin{pmatrix} p_1 & \dots & p_m \end{pmatrix} \begin{pmatrix} a_{11} & \dots & a_{1m} \\ \vdots & \ddots & \vdots \\ a_{m1} & \dots & a_{mm} \end{pmatrix} = \begin{pmatrix} \sum_i a_{i1} p_i & \dots & \sum_i a_{im} p_i \end{pmatrix}.$$

In more familiar terms these are simply $\mathbf{AP}$ and $\mathbf{P}^T\mathbf{A}$. In index form they appear respectively as:

$$T_i = a_{ij} p_j \,, \qquad T_j = p_i a_{ij}.$$

- **matrix $\odot$ matrix**: There are two contractions of two tensors of rank 2 both producing a tensor of rank 2, and they correspond to familiar matrix-matrix multiplication schemes. Assuming that $m = n$, we have for $\mathbf{A} \odot_j \mathbf{B}$:

$$\mathbf{T} = \begin{pmatrix} a_{11} & \dots & a_{1m} \\ \vdots & \ddots & \vdots \\ a_{m1} & \dots & a_{mm} \end{pmatrix} \begin{pmatrix} b_{11} & \dots & b_{1n} \\ \vdots & \ddots & \vdots \\ b_{n1} & \dots & b_{nn} \end{pmatrix} = \begin{pmatrix} \sum_{i=1}^{m} a_{1i} b_{i1} & \dots & \sum_{i=1}^{m} a_{1i} b_{in} \\ \vdots & \ddots & \vdots \\ \sum_{i=1}^{m} a_{mi} b_{i1} & \dots & \sum_{i=1}^{m} a_{mi} b_{in} \end{pmatrix},$$

or, in index notation, $T_{ij} = a_{ik} b_{kj}$. This is equivalent to the normal matrix multiplication $\mathbf{AB}$. For $\mathbf{A} \odot_i \mathbf{B}$ the scheme is

$$\mathbf{T} = \begin{pmatrix} b_{11} & \dots & b_{n1} \\ \vdots & \ddots & \vdots \\ b_{1n} & \dots & b_{nn} \end{pmatrix} \begin{pmatrix} a_{11} & \dots & a_{1m} \\ \vdots & \ddots & \vdots \\ a_{m1} & \dots & a_{mm} \end{pmatrix} = \begin{pmatrix} \sum_{i=1}^{m} a_{i1}b_{i1} & \dots & \sum_{i=1}^{m} a_{im}b_{i1} \\ \vdots & \ddots & \vdots \\ \sum_{i=1}^{m} a_{i1}b_{in} & \dots & \sum_{i=1}^{m} a_{im}b_{in} \end{pmatrix},$$

or in index notation $T_{ij} = a_{ij}b_{ik}$. This corresponds to the multiplication scheme $\mathbf{B}^T\mathbf{A}$. For tensors of rank 2 we also have the relationship

$$\mathbf{A} \odot_i \mathbf{B} = \mathbf{B}^T \odot_j \mathbf{A}. \tag{2.4}$$

- **matrix3D $\odot$ vector**: The contraction of a tensor of rank 3 with a tensor of rank 1 is a tensor of rank 2. There are three possible indices we can sum over giving three possible contractions, which we denote by $\odot_l$ where l denotes an index, $l \in \{i, j, k\}$:

$$\mathbf{T} = \mathbf{D} \odot_i \mathbf{P} = T_{jk} = D_{ijk} \odot_i P_i = \Big[\sum_{i=1}^{m} d_{ijk}p_i\Big]_{jk},$$

$$\mathbf{T} = \mathbf{D} \odot_j \mathbf{Q} = T_{ik} = D_{ijk} \odot_j Q_j = \Big[\sum_{j=1}^{n} d_{ijk}q_j\Big]_{ik},$$

$$\mathbf{T} = \mathbf{D} \odot_k \mathbf{R} = T_{ij} = D_{ijk} \odot_k R_k = \Big[\sum_{k=1}^{p} d_{ijk}r_k\Big]_{ij}.$$

In effect to obtain the contraction of a rank 3 tensor by a vector with respect to an index l, we fix an index in $\{i, j, k\}$ different from l and multiply all level matrices in the matrix3D in that direction by the vector. This produces new vectors which are inserted in as the columns of the resultant matrix. Hence for each one of these three possible contractions there are two possible and equivalent multiplication methods. For the $\odot_i$ contraction they are:

1. Fix j, and using the notation $\mathbf{D}^j$ to indicate a level matrix in the j direction, we have

$$\mathbf{T} = \mathbf{D} \odot_i \mathbf{P} = \Big[\mathbf{D}^j\mathbf{P}\Big]_j = \Big[\Big[\sum_{i=1}^{m} d_{ijk}p_i\Big]_k\Big]_j = \Big[\sum_{i=1}^{m} d_{ijk}p_i\Big]_{jk},$$

2. Fix k, we have

$$\mathbf{T} = \mathbf{D} \odot_i \mathbf{P} = \Big[\mathbf{D}^k\mathbf{P}\Big]_k = \Big[\Big[\sum_{i=1}^{m} d_{ijk}p_i\Big]_j\Big]_k = \Big[\sum_{i=1}^{m} d_{ijk}p_i\Big]_{jk}.$$

```
Matrix M3DV1(Matrix3D D, Vector P)
{
  double sum=0.0;
  int m = D.GetNumRows();
  int n = D.GetNumCols();
  int p = D.GetNumLays();
  Matrix T(n,p);

  for (k=0; k<p; k++)
    for (j=0; j<n; j++) {
      for (i=0; i<m; i++)
        sum+=D[k][i][j]*P[i];
      T[j][k]=sum;
      sum=0.0
    }

  return T;
}
```

```
Matrix M3DV2(Matrix3D D, Vector P)
{
  double sum=0.0;
  int m = D.GetNumRows();
  int n = D.GetNumCols();
  int p = D.GetNumLays();
  Matrix T(n,p);

  for (j=0; j<n; j++)
    for (k=0; k<p; k++) {
      for (i=0; i<m; i++)
        sum+=D[k][i][j]*P[i];
      T[j][k]=sum;
      sum=0.0;
    }

  return T;
}
```

Figure 2.3: Two versions of $\mathbf{D} \odot_i \mathbf{P}$

The treatment of $\odot_j$ and $\odot_k$ follows similarly. Figure 2.3 presents a C++-style pseudocode function for the implementation of the two equivalent forms for $\odot_i$.

- **matrix3D $\odot$ matrix**: The contraction of a tensor of rank 3 with a tensor of rank 2 is another tensor of rank 3. Again there are three possible indices we can sum over giving three possible contractions:

$$\mathbf{T} = \mathbf{D} \odot_i \mathbf{A} = T_{ljk} = D_{ijk} \odot_i A_{il} = \Big[\sum_{i=1}^{m} d_{ijk} a_{il}\Big]_{ljk},$$

$$\mathbf{T} = \mathbf{D} \odot_j \mathbf{B} = T_{ilk} = D_{ijk} \odot_j B_{jl} = \Big[\sum_{j=1}^{n} d_{ijk} q_{jl}\Big]_{ilk},$$

$$\mathbf{T} = \mathbf{D} \odot_k \mathbf{C} = T_{ijl} = D_{ijk} \odot_k C_{kl} = \Big[\sum_{k=1}^{p} d_{ijk} r_{kl}\Big]_{ijl}.$$

To obtain the multiplication of a rank 3 tensor by a matrix with respect to an index l, we fix an index in $\{i, j, k\}$ different from l and multiply all level matrices in the matrix3D in that direction by the matrix in question. This produces new matrices which are inserted in as the level matrices in the chosen direction forming the resultant matrix3D. The two possible and equivalent multiplication methods for $\odot_i$ are:

```
Matrix3D M3DM1(Matrix3D D, Matrix A)
{
  double sum=0.0;
  int m = D.GetNumRows();
  int n = D.GetNumCols();
  int p = D.GetNumLays();
  int c = A.GetNumCols();
  Matrix3D T(c,n,p);

  for (k=0; k<p; k++)
    for (l=0; l<c; l++)
      for (j=0; j<n; j++) {
        for (i=0; i<m; i++)
          sum+=D[k][i][j]*A[i][l];
        T[k][l][j]=sum;
        sum=0.0;
      }

  return T;
}
```

```
Matrix3D M3DM2(Matrix3D D, Matrix A)
{
  double sum=0.0;
  int m = D.GetNumRows();
  int n = D.GetNumCols();
  int p = D.GetNumLays();
  int c = A.GetNumCols();
  Matrix3D T(c,n,p);

  for (j=0; j<n; j++)
    for (l=0; l<c; l++)
      for (k=0; k<p; k++) {
        for (i=0; i<m; i++)
          sum+=D[k][i][j]*A[i][l];
        T[k][l][j]=sum;
        sum=0.0;
      }

  return T;
}
```

Figure 2.4: Two versions of $\mathbf{D} \odot_i \mathbf{A}$

1. Fix j, and using the notation $\mathbf{D}^j$ to indicate a level matrix in the j direction, we have

$$\mathbf{T} = \mathbf{D} \odot_i \mathbf{A} = \left[\mathbf{D}^j\mathbf{A}\right]_j = \left[\left[\sum_{i=1}^{m} d_{ijk}a_{il}\right]_{kl}\right]_j = \left[\sum_{i=1}^{m} d_{ijk}a_{il}\right]_{ljk}.$$

2. Fix k, we have

$$\mathbf{T} = \mathbf{D} \odot_i \mathbf{A} = \left[\mathbf{D}^k\mathbf{A}\right]_k = \left[\left[\sum_{i=1}^{m} d_{ijk}a_{il}\right]_{jl}\right]_k = \left[\sum_{i=1}^{m} d_{ijk}a_{il}\right]_{ljk}.$$

Again the other two cases follow similarly. Figure 2.4 presents a C++-style pseudocode function for the implementation of the two equivalent forms for $\odot_i$ for the the **matrix3D** $\odot$ **matrix** cases. In summary we have the results:

- vector $\odot$ vector = scalar

$$\begin{cases} \text{vector} \odot_j \text{vector} & T = p_i q_i \\ \text{vector} \odot_i \text{vector} & T = q_i p_i \end{cases}$$

- matrix $\odot$ vector = vector,

$$\begin{cases} \text{matrix} \odot_j \text{vector} & T_i = a_{ij} p_j \\ \text{matrix} \odot_i \text{vector} & T_j = p_i a_{ij} \end{cases}$$

- matrix $\odot$ matrix = matrix,

$$\begin{cases} \text{matrix} \odot_j \text{matrix} & T_{ij} = a_{ik} b_{kj} \\ \text{matrix} \odot_i \text{matrix} & T_{kj} = a_{ij} b_{ik} \end{cases}$$

- matrix3D $\odot$ vector = matrix,

$$\begin{cases} \text{matrix3D} \odot_i \text{vector} & T_{jk} = d_{ijk} p_i \\ \text{matrix3D} \odot_j \text{vector} & T_{ik} = d_{ijk} q_j \\ \text{matrix3D} \odot_k \text{vector} & T_{ij} = d_{ijk} r_k \end{cases}$$

- matrix3D $\odot$ matrix = matrix3D,

$$\begin{cases} \text{matrix3D} \odot_i \text{matrix} & T_{ljk} = d_{ijk} a_{il} \\ \text{matrix3D} \odot_j \text{matrix} & T_{ilk} = d_{ijk} q_{jl} \\ \text{matrix3D} \odot_k \text{matrix} & T_{ijl} = d_{ijk} r_{kl} \end{cases}$$

Under the condition that the tensors are of compatible size for the multiplications and additions to occur, the contraction operator $\odot_l$ for rank 3 with rank 2 tensors satisfies the following properties, where $l, m \in \{i, j, k\}$ ($\mathbf{D}$ and $\mathbf{E}$ are matrix3Ds, $\mathbf{A}, \mathbf{B}$ are matrices and c is a constant).

$$\mathbf{D} \odot_l c\mathbf{A} = c\mathbf{D} \odot_l \mathbf{A} \tag{2.5}$$

$$\mathbf{D} \odot_l (\mathbf{A} \oplus \mathbf{B}) = (\mathbf{D} \odot_l \mathbf{A}) \oplus (\mathbf{D} \odot_l \mathbf{B}) \tag{2.6}$$

$$(\mathbf{D} \oplus \mathbf{E}) \odot_l \mathbf{A} = (\mathbf{D} \odot_l \mathbf{A}) \oplus (\mathbf{E} \odot_l \mathbf{A}) \tag{2.7}$$

$$(\mathbf{D} \odot_l \mathbf{A}) \odot_m \mathbf{B} = (\mathbf{D} \odot_m \mathbf{B}) \odot_l \mathbf{A} \tag{2.8}$$

For $\mathbf{P}$ and $\mathbf{Q}$ vectors of compatible size we have:

$$\mathbf{D} \odot_l c\mathbf{P} = c\mathbf{D} \odot_l \mathbf{P} \tag{2.9}$$

$$\mathbf{D} \odot_l (\mathbf{P} \oplus \mathbf{Q}) = (\mathbf{D} \odot_l \mathbf{P}) \oplus (\mathbf{D} \odot_l \mathbf{Q}) \tag{2.10}$$

$$(\mathbf{D} \oplus \mathbf{E}) \odot_l \mathbf{P} = (\mathbf{D} \odot_l \mathbf{P}) \oplus (\mathbf{E} \odot_l \mathbf{Q}) \tag{2.11}$$

$$(\mathbf{D} \odot_l \mathbf{P}) \odot_m \mathbf{Q} = (\mathbf{D} \odot_m \mathbf{P}) \odot_l \mathbf{Q} \tag{2.12}$$

Finally, for tensors $\mathbf{T}_1, \mathbf{T}_2, \mathbf{T}_3$ of rank 1,2 or 3 the following result holds:

$$\mathbf{T}_1 \odot_l \Big(\mathbf{T}_2 \otimes \mathbf{T}_3\Big) = \Big(\mathbf{T}_1 \odot_l \mathbf{T}_2\Big) \otimes \mathbf{T}_3. \tag{2.13}$$

2.3 B-spline Curves

A B-spline curve is a piecewise polynomial function expressed with respect to a set of B-spline basis functions which have local support and built-in cross-segment continuity defined by a knot set. The maximum allowable cross-segment continuity is equal to the order of the B-spline minus two (C^2 in the case of cubic B-splines) and the minimum is C^{-1}, which implies that the two adjacent segments are in fact disjoint. The basis functions are chosen so has to have the minimal possible support subject to their piecewise polynomial nature and the specified cross-segment continuity. The formula for a B-spline curve of order k is

$$f(t) = \sum_{i=1}^{n} d_i N_{i,k}(t), \quad \text{on the knot set} \quad (t_i)_{i=1}^{n+k}, \tag{2.14}$$

where the $N_{i,k}(t)$ are the B-spline basis functions defined over an associated knot set $(t_i)_{i=1}^{n+k}$. We can write this in tensor form as follows:

$$\mathbf{N} = \mathbf{N}^0(t) = \begin{pmatrix} N_{1,k}(t) & \dots & N_{n,k}(t) \end{pmatrix}^T, \quad \mathbf{d} = \begin{pmatrix} d_1 & \dots & d_n \end{pmatrix},$$

then

$$f(t) = \mathbf{d} \odot_i \mathbf{N}. \tag{2.15}$$

Each basis function $N_{i,k}(t)$ of order k is itself a functional B-spline curve and defined locally over $k+1$ knots, in the range $[t_i, t_{i+k}]$. A given knot multiplicity equal to r in one one of the knots implies a cross segment continuity of order C^{k-r+1} at that knot. The basis functions satisfy the following Cox de-Boor recursive formula, [10], (derivation also given in Appendix E):

$$\begin{aligned} N_{i,1}(t) &= \begin{cases} 1 & t_i \le t < t_{i+1} \\ 0 & \text{otherwise} \end{cases} \\ N_{i,k}(t) &= \frac{t - t_i}{t_{i+k-1} - t_i} N_{i,k-1}(t) + \frac{t_{i+k} - t}{t_{i+k} - t_{i+1}} N_{i+1,k-1}(t), \quad k \ge 2. \end{aligned} \tag{2.16}$$

The knot set itself can either be represented in vector form as one sequential list, or as two lists, the first giving the distinct knots (`dts`) and the second the multiplicities (`mult`). In the algorithms that follow we assume the coalesced form for the knot extension, that is if `nseg` represents the number of segments of the B-spline then

`mult[1]=mult[nseg-1]=k`.

In the following sections of this chapter we review briefly the important B-spline algorithms for curves, surfaces and volumes that are used extensively in the later chapters. Further details of B-spline theory are provided in, for example, [10], [11], [40]. We concentrate here on evaluation, derivative, integration, knot insertion/removal and product which have particular significance for this work. In doing so we introduce some new notation for expressing the derivative of a B-spline and derive an explicit method for expressing the control points of the derivative curve in terms of control points of the original.

Note on notation: The normal method of distinction between the functional and parametric forms is to use bold notation for the parametric case. In this thesis we use bold where the B-spline is explicitly taken to be parametric as in the geometric smoothing covered in Chapter 4, otherwise the control points are written without bold typeface.

2.3.1 Evaluation

The local non-zero nature of the B-spline basis functions means that to evaluate the B-spline function

$$f(t) = \sum_j d_j N_{j,k}(t)$$

at a point $t \in [t_i, t_{i+1})$, it is only necessary to calculate the k non-zero numbers

$$N_{j,k}(t)\,, \qquad j = i-k+1, \ldots, i,$$

then $f(t)$ is given by

$$f(t) = \sum_{j=i-k+1}^{i} d_j N_{j,k}(t).$$

The calculation of this reduced sum is carried out efficiently by using the recursive nature of the $N_{j,k}(t)$, 2.16, which allows us to write $f(t)$ in terms of B-spline basis functions of one order less:

$$f(t) = \sum_i d_i^1 N_{i,k-1}(t),$$

where

$$d_i^1(t) = \frac{t - t_i}{t_{i+k-1} - t_i} d_i + \frac{t_{i+k-1} - t}{t_{i+k-1} - t_i} d_{i-1}.$$

Applying this result recursively we obtain

$$f(t) = \sum_i d_i^j(t) N_{i,k-j}(t), \tag{2.17}$$

where

$$d_i^j(t) = \begin{cases} d_i\,, & j = 0 \\ \frac{t-t_i}{t_{i+k-j}-t_i} d_i^{j-1}(t) + \frac{t_{i+k-j}-t}{t_{i+k-j}-t_i} d_{i-1}^{j-1}(t)\,, & j > 0 \end{cases} \tag{2.18}$$

or equivalently

$$d_i^j = \lambda d_i^{j-1} + (1-\lambda) d_{i-1}^{j-1}, \tag{2.19}$$

where $\lambda = (t - t_i)/(t_{i+k-j} - t_i)$. Now $f(t)$ is written in terms of normalized B-splines of lower order. Since $N_{i,1}(t) = 1$ for $t_i \le t < t_{i+1}$ and zero otherwise, it follows that

$$f(t) = d_i^{k-1}(t)\,, \qquad t_i \le t < t_{i+1}.$$

Hence if $t \in [t_i, t_{i+1})$, $f(t)$ can be found from $d_{i-k+1}, \ldots, d_i$ by forming *convex* combinations according to 2.19. The convexity of the calculation assures the numerical stability of the process. Utilising the above formula 2.17 for $f(t)$ in terms of the d_i^j we need first to find i such that $t_i \le t < t_{i+1}$. Having found the correct i we generate all the entries in the following triangular table:

$d_{i-k+1}^0(t)$				
$d_{i-k+2}^0(t)$	$d_{i-k+2}^1(t)$			
$\vdots$	$\vdots$	$\ddots$		
$d_{i-1}^0(t)$	$d_{i-1}^1(t)$	$\ldots$	$d_{i-1}^{k-2}(t)$	
$d_i^0(t)$	$d_i^1(t)$	$\ldots$	$d_i^{k-2}(t)$	$d_i^{k-1}(t)$

Table 2.1: Triangular table for recursive evaluation of a B-spline function

The table is evaluated column by column with each entry a convex combination of the two adjacent elements in the preceding column. The right most entry in this table, $d_i^{k-1}(t)$, is the desired $f(t)$.

This algorithm can be used to compute the vector of coefficients expressing the value of a B-spline function $f(t)$ in terms of the original control points. We seek coefficients $(\alpha_i)_{i=1}^n$ such that

$$f(t^*) = \sum_{i=1}^{n} \alpha_i d_i.$$

The following algorithm computes the cofficients α_i:

Algorithm 2.1: To compute vector $(\alpha_i)_{i=1}^{n}$ ***such that*** $f(t^*) = \sum_{i=1}^{n} \alpha_i d_i$ *for* $t_k \le t^* \le t_{n+1}$

1. *create storage matrix* `mat(k,k)` *and vectors* `dp(k)`, `dm(k)`, *create vector* `res` *for storage of* α_i
2. *find index* `ind` *such that* $t_{ind} \le t^* < t_{ind+1}$
3. *initialise* `mat[0][0]=1.0`
4. *for* `j=0` *to* `k-2`
   - *4.1* `dp[j]=t[ind+j+1]-t*`
   - *4.2* `dm[j]=t*-t[ind-j]`
   - *4.3 for* `i=0` *to* `j`
     - *4.3.1* `temp=mat[i][j]/(dp[i]+dm[j-i])`
     - *4.3.2* `mat[i][j+1]+=dp[i]*temp`
     - *4.3.3* `mat[i+1][j+1]=dm[j-i]*temp`
5. *for* `i=0` *to* `k-1`
   - *5.1* `res[i+ind-k+1]=mat[i][k-1]`

Figure 2.5: Algorithm 2.1: Computation of B-spline evaluation coefficients

2.3.2 Derivatives

It can be shown that (for derivation see appendix E) that the following result holds for the derivative of a B-spline curve $f(t)$:

$$f'(t) = \sum_i d_i N'_{i,k}(t) = (k-1) \sum_i d_i^{(1)} N_{i,k-1}(t)\,, \tag{2.20}$$

where

$$d_i^{(1)} = (d_i - d_{i-1})/(t_{i+k-1} - t_i). \tag{2.21}$$

Generally, with $d_i^{(0)} = d_i$, and

$$d_i^{(j)} = (d_i^{(j-1)} - d_{i-1}^{(j-1)})/(t_{i+k-j} - t_i)\,, \qquad j > 0\,, \tag{2.22}$$

we have

$$f^{(j)}(t) = (k-1)\ldots(k-j) \sum_i d_i^{(j)} N_{i,k-j}(t). \tag{2.23}$$

It is possible to calculate $f^{(j)}(t)$ by noting that the $(k-j)$th column of the B-spline evaluation, table 2.1, contains the numbers needed for the derivative evaluation. The coefficients $d_i^{(j)}$ can likewise be arranged in a convenient triangular table as follows:

$d_{i-k+1}^{(0)}$				
$d_{i-k+2}^{(0)}$	$d_{i-k+2}^{(1)}$			
$\vdots$	$\vdots$	$\ddots$		
$d_{i-1}^{(0)}$	$d_{i-1}^{(1)}$	$\dots$	$d_{i-1}^{(k-2)}$	
$d_i^{(0)}$	$d_i^{(1)}$	$\dots$	$d_i^{(k-2)}$	$d_i^{(k-1)}$

Table 2.2: Triangular table for recursive evaluation of a B-spline derivative function

We use this result to develop an algorithm for expressing the rth derivative of a B-spline function as a B-spline function and for expressing the control points of the derivative B-spline as a linear combination of the control points of the undifferentiated curve. From 2.20 we can write the derivative of a B-spline basis function, $N_{i,k}(t)$, as a linear combination of B-spline basis functions of one order less:

$$f'(t) = \sum_{i=1}^{n-1} d_i^{(1)} N_{i,k-1}(t),$$

where (here $d_i^{(0)} = d_i$),

$$d_i^{(1)} = \sum_{j=1}^{n} \alpha_{ij}^{1\,0} d_j^{(0)}\,, \qquad i = 1, \dots, n-1,$$

and the coefficients α_{ij} are such that

$$\alpha_{ij} = \begin{cases} \frac{1}{t_{i+k-1} - t_i} & j = i \\ \frac{-1}{t_{i+k-1} - t_i} & j = i-1 \\ 0 & j \neq i, i-1. \end{cases}$$

By defining an $n-1$ by n matrix $\mathbf{D}_0^1$ as (logically $\mathbf{D}_0^0 = \mathbf{I}$, the identity matrix):

$$\mathbf{D}_0^1 = (\alpha_{ij}^{1\,0})_{i,j=1}^{n-1,n}, \tag{2.24}$$

we can then express the control points of the derivative curve in terms of the original:

$$\begin{pmatrix} d_1^{(1)} \\ \vdots \\ d_{n-1}^{(1)} \end{pmatrix} = \mathbf{D}_0^1 \begin{pmatrix} d_1 \\ \vdots \\ d_n \end{pmatrix}. \tag{2.25}$$

For the second derivative we have

$$f^{(2)}(t) = \sum_{i=1}^{n-2} d_i^{(2)} N_{i,k-2}(t), \qquad d_i^{(2)} = \sum_{j=1}^{n-1} \alpha_{ij}^{2\,1} d_j^{(1)}, \qquad i = 1, \ldots, n-2,$$

and we define the $n-2$ by $n-1$ matrix $\mathbf{D}_1^2$:

$$\mathbf{D}_1^2 = (\alpha_{ij}^{2\,1})_{i,j=1}^{n-2,n-1},$$

so that

$$\begin{pmatrix} d_1^{(2)} \\ \vdots \\ d_{n-2}^{(2)} \end{pmatrix} = \mathbf{D}_1^2 \begin{pmatrix} d_1^{(1)} \\ \vdots \\ d_{n-1}^{(1)} \end{pmatrix}.$$

More generally we have for $r \geq 0$

$$f^{(r)}(t) = \sum_{i=1}^{n-r} d_i^{(r)} N_{i,k-r}(t), \qquad d_i^{(r)} = \sum_{j=1}^{n-r} \alpha_{ij}^{r\,r-1} d_j^{(r-1)}, \qquad i = 1, \ldots, n-r,$$

and we define the matrix

$$\mathbf{D}_{r-1}^r = (\alpha_{ij}^{r\,r-1})_{i,j=1}^{n-r,n-r+1}, \tag{2.26}$$

so that

$$\begin{pmatrix} d_1^{(r)} \\ \vdots \\ d_{n-r}^{(r)} \end{pmatrix} = \mathbf{D}_{r-1}^r \begin{pmatrix} d_1^{(r-1)} \\ \vdots \\ d_{n-r+1}^{(r-1)} \end{pmatrix}.$$

If we define for $r > s \geq 0$

$$\mathbf{D}_s^r = (\alpha_{ij}^{r\,r-s})_{i,j=1}^{n-r,n-s},$$

then

$$f^{(r)}(t) = \sum_{i=1}^{n-r} d_i^{(r)} N_{i,k-r}(t), \qquad d_i^{(r)} = \sum_{j=1}^{n-s} \alpha_{ij}^{r\,r-s} d_j^{(r-s)},$$

and we have result

$$\mathbf{D}_s^r = \mathbf{D}_{r-1}^r \mathbf{D}_{r-2}^{r-1} \dots \mathbf{D}_s^{s+1} = \prod_{i=0}^{r-s-1} \mathbf{D}_{r-i-1}^{r-i},$$

from which

$$\mathbf{D}_0^r = \mathbf{D}_{r-1}^r \mathbf{D}_{r-2}^{r-1} \dots \mathbf{D}_0^1 = \prod_{i=0}^{r-1} \mathbf{D}_{r-i-1}^{r-i}. \tag{2.27}$$

Using the following notation

$$\mathbf{N} = \mathbf{N}^0(t) = \begin{pmatrix} N_{1,k}(t) & \dots & N_{n,k}(t) \end{pmatrix}^T, \tag{2.28}$$

$$\mathbf{N}^r = \mathbf{N}^r(t) = \begin{pmatrix} N_{1,k-r}(t) & \dots & N_{n-r,k-r}(t) \end{pmatrix}^T, \tag{2.29}$$

$$\mathbf{N}^{(r)} = \mathbf{N}^{(r)}(t) = \begin{pmatrix} N_{1,k}^{(r)}(t) & \dots & N_{n,k}^{(r)}(t) \end{pmatrix}^T, \tag{2.30}$$

we have

$$f^{(r)}(t) = \left(\mathbf{N}^{(r)}\right)^T \mathbf{d} = \left(\mathbf{N}^r\right)^T \mathbf{D}_0^r \, \mathbf{d},$$

and hence

$$\mathbf{N}^{(r)} = \left(\mathbf{D}_0^r\right)^T \mathbf{N}^r. \tag{2.31}$$

In tensor notation we can write

$$f(t) = \mathbf{d} \odot_i \mathbf{N}\,, \quad f^{(r)}(t) = \mathbf{d} \odot_i \mathbf{N}^{(r)} = \mathbf{d} \odot_i \left(\mathbf{D}_0^r\right)^T \mathbf{N}^r. \tag{2.32}$$

Algorithm for Derivative Evaluation

On the face of it the matrix $\mathbf{D}_0^1$ can be easily calculated using 2.24 and this will furnish the derivative control points. However, in order to generate the derivative curve we also need to consider the knot set. The derivative knot set will be of one order less and contain the same knots as the original curve whilst maintaining their multiplicity. For example, a cubic B-spline on the knot set $(0,0,0,0,1,2,2,2,3,3,3,3)$ will have a derivative set equal to $(0,0,0,1,2,2,2,3,3,3)$. Here the original knot set implies basis function continuity equal to C^2 at 1 and C^2 at 2 whilst the derivative knot set, being one degree less, is such that the continuity is C^1 at 1 and C^{-1} at 2. Differentiate again and we hit a problem since it is impossible for a degree 1 curve to have a triple knot (we can't have a continuity level less than C^{-1}) and indeed the recursive algorithm 2.16 upon which B-splines are based will fail. In practice a suitable knot set in this case is $(0,0,1,2,2,3,3)$. Hence in order to cope with the most general case we need to revert to differentiating the curve segment by segment and then eliminate any redundancy in

the resulting derivative expression. To illustrate this we look at expressing the second derivative of the following B-spline function as a B-spline

$$f(t) = \sum_{i=1}^{12} d_i N_{i,4}(t), \quad \text{on the knot set } \ (t_i)_{i=1}^{16} = (0,0,0,0,1,1,2,2,2,3,4,4,5,5,5,5).$$

The second derivative of this function is given by an expression of the form

$$f^{(2)}(t) = \sum_i d_i^{(2)} N_{i,2}(t).$$

To determine a suitable knot set and the associated control points we proceed as follows:

Step 1: Compute the control points, $d_i^{(2)}$, corresponding to the second derivative for each segment of the B-spline using table 2.2. In this example we have five segments, and the five triangular arrays (corresponding to the indices 4, 6, 9, 10, 12) gives us the following control points

$$(d_3^{(2)}, d_4^{(2)}, d_5^{(2)}, d_6^{(2)}, d_8^{(2)}, d_9^{(2)}, d_9^{(2)}, d_{10}^{(2)}, d_{11}^{(2)}, d_{12}^{(2)}).$$

We need to delete any repetitions in the control point list. This can be conveniently done by forming what we call an overlap list, specifying the overlap of control points from one segment to the next. For the original function this list is $(2,1,3,2)$ and for the rth derivative function it will be $(2-r, 1-r, 3-r, 2-r)$. For $r = 2$ this gives (0,-1,1,0) which indicates an overlap of one control point from the 3rd to the 4th segment (i.e $d_9^{(2)}$), and hence this one can be left out on computing the control points for the 4th segment (numbers less than equal to 0 indicate no overlap meaning that the two consecutive segments have become disjoint in the derivative curve).

Step 2: From the list of control points in *Step 1* with indices (3,4,5,6,8,9,10,11,12), we form the knot set by selecting the knots with these indices from the original set. In addition we take the r knots following the last one selected. This gives us the list

$$(t_3, t_4, t_5, t_6, t_8, t_9, t_{10}, t_{11}, t_{12}, t_{13}, t_{14}) = (0,0,1,1,2,2,3,4,4,5,5).$$

We now have the control points $(d_i^{(2)})_{i=1}^{9}$ and knot set $(t_i^2)_{i=1}^{11}$ for the B-spline representation for $f^{(2)}(t)$:

$$f^{(2)}(t) = \sum_{i=1}^{9} d_i^{(2)} N_{i,2}(t).$$

We summarise this section on derivatives with a set of five algorithms leading to the computation of the derivative of a B-spline curve $f(t)$ as a B-spline curve:

1. Computation of the rth derivative knot set
2. Computation of $\mathbf{D}_0^1$
3. Computation of $\mathbf{D}_0^r$, using 2.27
4. For a given point t^*, expressing the value of $f^{(r)}(t^*)$ in terms of original control points using $\mathbf{D}_0^r$ and algorithm 2.1
5. Computation of $f^{(r)}(t)$ as a B-spline

Algorithm 2.2: To compute rth derivative knot set
Original knot set has distinct knot vector `dts` *and multiplicities vector* `mult`
Derivative knot set has distinct knot vector `rdts` *and multiplicities vector* `rmult`

1. initialise an overlap vector, `overlap`, *size* `nseg` *with* `overlap[i]=k-mult[i]-r`
2. set the end knot multiplicities to `k-r`
3. for each distinct internal knot in `dts`
  3.1 if `overlap[i]<=0`
    3.1.1 `rmult[i]=k-r`
  3.2 else `rmult[i]=k-r-overlap[i]`
  3.3 `rdts[i]=dts[i]`

Figure 2.6: Algorithm 2.2: Computation of the rth derivative knot set

Algorithm 2.3: To compute matrix $\mathbf{D}_0^1$

1. *initialise an overlap vector,* `overlap` *, size* `nseg` *with* `overlap[i]=k-mult[i]-1`
2. *create the first derivative knot set* `kset`
3. *create a matrix* `mat` *size* `(n-1)*n` *to store the result*
4. *initialise an index counter for the rows of* `mat, row_ind=1`
5. *for each distinct knot* `dts[i]`
   - *5.1 find index* `ind` *such that* $\texttt{t}_{\texttt{ind}}$ `<=dts[i]<` $\texttt{t}_{\texttt{ind+1}}$
   - *5.2 determine offset for row of* `mat` *from overlap vector*
     - *if* `(overlap[i] > 0) row_ind -=overlap[i]`
   - *5.3 compute coefficients for row* `row_ind` *of* `mat` *according to equation 2.23*
6. *multiply by the degree:* `mat=(k-1)*mat`

Figure 2.7: Algorithm 2.3: Computation of first derivative matrix $\mathbf{D}_0^1$

Algorithm 2.4: To compute matrix $\mathbf{D}_0^r$

1. *compute matrix for first derivative* $\mathbf{D}_0^1$
2. *create knot set,* `kset,` *initialised with current knot set*
3. *for* `i = 1` *to* `r`
   - *3.1 update* `kset` *with the first derivative knot set*
   - *3.2 create* $\mathbf{D}_0^1$ *from* `kset`
   - *3.3 update matrix from 1 by multiplying on left by matrix from 3.2*

Figure 2.8: Algorithm 2.4: Computation of the rth derivative matrix $\mathbf{D}_0^r$

Algorithm 2.5: To compute vector $(\beta_i)_{i=1}^{n}$ ***such that***
$f^{(r)}(t^*) = \sum_{i=1}^{n} \beta_i d_i \quad$ *for* $\quad t_k \le t^* \le t_{n+1}$

1. *compute matrix for rth derivative* $\mathbf{D}_0^r$
2. *create rth derivative knot set,* `kset`
3. *from* `kset` *create vector* $\alpha_{\mathtt{i}}$ *using algorithm 2.1*
4. *then* $(\beta_{\mathtt{i}})_{\mathtt{i=1}}^{\mathtt{n}} = (\alpha_{\mathtt{i}})_{\mathtt{i=1}}^{\mathtt{n-r}} * \mathtt{D}_{\mathtt{0}}^{\mathtt{r}}$

Figure 2.9: Algorithm 2.5: Computation of the rth derivative in terms of the d_i

Algorithm 2.6: To compute $f^{(r)}(t)$ ***as a B-spline***

1. *compute matrix for rth derivative,* $\mathbf{D}_0^r$
2. *create rth derivative knot set from the original,* `rkset`
3. *create new control points as* $\mathbf{D}_0^r\mathbf{d}$
4. *build new B-spline or order* $\mathtt{k-r}$ *from 2 and 3*

Figure 2.10: Algorithm 2.6: Computation of the rth derivative as a B-spline curve

2.3.3 Integration

The indefinite integral of a B-spline function $f(t)$

$$f(t) = \sum_{i=1}^{n} d_i N_{i,k}(t)$$

on the knot set $(t_i)_{i=1}^{n+k}$, is given by the B-spline function $g(t)$ where (see de Boor, [10])

$$g(t) = \int_{t_1}^{t} \sum_{i=1}^{n} d_i N_{i,k}(u)du = \sum_{i=1}^{n} \Big(\frac{t_{i+k+1} - t_i}{k} \sum_{j=1}^{i} d_j \Big) N_{i,k+1}(t)\,, \quad t_k \le t \le t_{n+1}. \tag{2.33}$$

Hence the integral of a B-spline is represented as:

$$g(t) = \int_{t_1}^{t} f(u)du = \sum_{i=1}^{n+1} e_i N_{i,k+1}(t),$$

where

$$e_1 = 0, \;\; e_{i+1} = \Big(\frac{t_{i+k+1} - t_i}{k}\Big) \sum_{j=1}^{i} d_j, \quad 1 \le i \le n. \tag{2.34}$$

The knot set for $g(t)$ matches that of the original curve except for one extra knot at either end due to the increased degree. For a definite integral of a B-spline we have:

$$\int_{x_1}^{x_2} f(t)dt = \int_{t_1}^{x_2} f(t)dt - \int_{t_1}^{x_1} f(t)dt = g(x_2) - g(x_1)\,, \quad t_k \le x_1, x_2 \le t_{n+1}. \tag{2.35}$$

2.3.4 Knot insertion and removal

The knot insertion process is well documented, see for example, [10], [11], [40]. It permits the addition of knots to an existing B-spline curve in such a way that the resulting function is geometrically identical to the original. There are numerous applications of this process in the area of Computer Aided Design, ranging from evaluation and refinement to subdivision and intersection testing. For a B-spline curve $f(t)$ on the knot set $(t_i)_{i=1}^{n+k}$, the basic procedure can be described as follows.

After adding a knot $\hat{t}$ say, coinciding with the knot t_{p+1} which already has multiplicity s say (if the knot does not already occur in the sequence then we take $s = 0$) we obtain a new knot set which we denote by (t_i^1), such that

$$t_i^1 = \begin{cases} t_i & i \le p \\ \hat{t} = t_{p+1} & i = p+1 \\ t_{i-1} & i \ge p+2 \end{cases}.$$

If we denote the set of basis functions on this new knot set as $(N_{i,k}^1(t))$ we can express $f(t)$ in the form

$$f(t) = \sum_i d_i^1 N_{i,k}^1(t),$$

for some coefficients d_i^1. These coefficients can be found by using the following expression for $N_{i,k}(t)$ written as a linear combination of $N_{i,k}^1(t)$ and $N_{i+1,k}^1(t)$ (for derivation see Appendix E):

$$N_{i,k}(t) = \begin{cases} N_{i,k}^1(t) & i \leq p-k+s \\ \frac{\hat{t}-t_i^1}{t_{i+k}^1-t_i^1} N_{i,k}^1(t) + \frac{t_{i+k+1}^1-\hat{t}}{t_{i+k+1}^1-t_{i+1}^1} N_{i+1,k}^1(t), & p-k+s+1 \leq i \leq p. \\ N_{i+1,k}^1(t) & i \geq p+1 \end{cases} \tag{2.36}$$

Summing $d_i N_{i,k}(t)$ over the range $p-k+s+1 \leq i \leq p$ using the above expression, and changing the variable of summation ($i \to i-1$) for the terms involving $N_{i+1,k}^1(t)$, we obtain (noting that $\hat{t} - t_{p+1}^1 = 0$),

$$\sum_{i=p-k+s+1}^{p} d_i N_{i,k}(t) = \sum_{i=p-k+s+1}^{p+1} \left(\frac{\hat{t}-t_i^1}{t_{i+k}^1-t_i^1} d_i + \frac{t_{i+k}^1-\hat{t}}{t_{i+k}^1-t_i^1} d_{i-1} \right) N_{i,k}^1(t). \tag{2.37}$$

Hence, by the uniqueness of the representation of $f(t)$ with respect to a given B-spline basis we have, by comparing 2.36 with 2.37, the result

$$d_i^1 = \alpha_i d_i + (1-\alpha_i) d_{i-1}, \tag{2.38}$$

where

$$\alpha_i = \begin{cases} 1 & i \leq p-k+s+1 \\ (\hat{t}-t_i^1)/(t_{i+k}^1-t_i^1) = (\hat{t}-t_i)/(t_{i+k-1}-t_i) & p-k+s+2 \leq i \leq p \\ 0 & i \geq p+1. \end{cases}$$

Equation 2.38 has the simple interpretation of d_i^1 dividing the line joining d_{i-1} and d_i in the ratio $\alpha_i : 1-\alpha_i$, a convex combination of control points. Multiple knot insertions are achieved by using the above process iteratively.

Knot Removal

Knot removal is essentially the reverse process of knot insertion. However, whereas knot insertion is theoretically exact in that it produces a geometrically identical curve, knot removal will only reproduce the original when the knot being removed does not change the continuity level of the curve at that join. This is the case, for example, when converting a piecewise composite Bézier curve of order k with internal knots of multiplicity equal to $k-1$, to its minimal B-spline representation. This occurs in one version of the B-spline product algorithm described below and indeed the main interest here in this algorithm relates to this situation. In general, if there

is a continuity change then knot removal will produce a curve that approximates the original. To check whether a knot is removable or not it is necessary to check to see whether two computed control points are within a given tolerance of each other. This tolerance can be used to control the maximum deviation of the knot reduced curve from the original. More details are provided in Tiller, [84].

2.3.5 Product

The product of two B-splines can be expressed as a B-spline by using parameter normalisation, knot insertion and a specific multiplication method. Given two B-spline curves of possibly differing orders

$$f_1(t) = \sum_{i=1}^{n_1} d_i^1 N_{i,k_1}(t)\,, \quad f_2(t) = \sum_{i=1}^{n_2} d_i^2 N_{i,k_2}(t)$$

and on differing knot sets $(t_i^1)_{i=1}^{n_1+k_1}$, $(t_i^2)_{i=1}^{n_2+k_2}$, we seek a representation of the product function $f(t)$ as a B-spline:

$$f(t) = f_1(t)f_2(t) = \Big(\sum_{i=1}^{n_1} d_i^1 N_{i,k_1}(t)\Big)\Big(\sum_{i=1}^{n_2} d_i^2 N_{i,k_2}(t)\Big) = \sum_{i=1}^{n} d_i N_{i,k}(t),$$

on the knot set $(t_i)_{i=1}^{n+k}$ where $k = k_1 + k_2 - 1$. There are two well known algorithms in the literature for finding the control points $(d_i)_{i=1}^{n}$ and the corresponding knot set $(t_i)_{i=1}^{n+k}$. Morken, [60], derives an involved but explicit formula for the new control points after forming the product knot set $(t_i)_{i=1}^{n+k}$. The distinct knots in the product are formed from the union of the two knot sets (t_1) and (t_2) after deleting repetitions. Each knot t_i in the product knot set has a multiplicity m, where

$$m = \begin{cases} max(k_1 - 1 + m_2, k_2 - 1 + m_1) & m_1 > 0, m_2 > 0 \\ k_1 - 1 + m_2 & m_1 = 0, m_2 > 0 \\ k_2 - 1 + m_1 & m_1 > 0, m_2 = 0 \\ 0 & m_1 = 0, m_2 = 0, \end{cases}$$

Here m_1 and m_2 are the multiplicities of the knots in $(t_1)_{i=1}^{n_1+k_1}$ and $(t_2)_{i=1}^{n_2+k_2}$ respectively.

Piegl and Tiller, [66], take another approach. They normalise the knot sets $(t_1), (t_2)$ through knot insertion so that finally both curves have the same breakpoints and number of segments. The resulting curves are then converted to piecewise Bézier format again using knot insertion. Multiplication of the functions can then be carried out segment by segment using the following

Algorithm 2.7: To compute the product B-spline $f(t) = f_1(t)f_2(t)$

1. *test to see whether knot sets for* $\mathtt{f_1}, \mathtt{f_2}$ *are identical or not*
2. *if not identical, make the two knot sets break compatible using process of knot normalisation and knot insertion*
3. *convert both curves to composite Bezier format*
4. *multiply them segment by segment using Bezier product formula 2.39*
5. *create the minimal product knot set using Morken's algorithm*
6. *form a difference knot set between the product set and the composite Bezier form*
7. *use knot removal to remove knots given by difference knot set to convert to the final B-spline form*

Figure 2.11: Algorithm 2.7: Computation of the product B-spline

result for the multiplication of two Bézier curves. For $f(t), g(t)$ two Bézier curves:

$$f(t) = \sum_{i=0}^{p} b_i^1 B_i^p(t), \quad g(t) = \sum_{i=0}^{q} b_i^2 B_i^q(t), \quad B_i^r(t) = \binom{r}{i} t^i (1-t)^{r-i}, \quad \binom{r}{i} = \frac{r!}{i!(r-i)!}$$

then

$$h(t) = f(t)g(t) = \sum_{i=0}^{p+q} b_i B_i^{p+q}(t)\,, \quad b_i = \sum_{l=\max(0,i-q)}^{\min(p,i)} \frac{\binom{p}{l}\binom{q}{i-l}}{\binom{p+q}{i}} b_l^1 b_{i-l}^2 \quad i = 0, \ldots, p+q. \tag{2.39}$$

After segment by segment multiplication a B-spline is recovered by removing the knots (using the knot removal algorithm) formed by the difference of the piecewise Bézier knot set and the product knot set as given in the Morken algorithm. Due to its superior computational performance we use Piegl and Tiller's version of the product algorithm in this thesis. The steps involved are given in algorithm 2.7 (figure 2.11).

2.4 B-spline Surfaces

A B-spline surface is formed from the tensor product of the curve representation. In particular, if we take two B-spline curve schemes

$$f(u) = \sum_{i=1}^{p} d_i^1 N_{i,k}(u), \text{ knot set } (u_i)_{i=1}^{m+k}, \; g(v) = \sum_{j=1}^{q} d_j^2 N_{j,l}(v), \text{ knot set } (v_j)_{j=1}^{q+l}$$

and write them in the following tensor form

$$f(u) = \mathbf{d_1} \odot_i \mathbf{N}_u, \qquad g(v) = \mathbf{d_2} \odot_j \mathbf{N}_v,$$

their tensor product is given by

$$f \otimes g = \left(\mathbf{d_1} \odot_i \mathbf{N}_u\right) \otimes \left(\mathbf{d_2} \odot_j \mathbf{N}_v\right) = \mathbf{N}_u \odot_i (\mathbf{d}_1 \otimes \mathbf{d}_2) \odot_j \mathbf{N}_v = \mathbf{d}_1 \odot_i \left(\mathbf{N}_u \otimes \mathbf{N}_v^T\right) \odot_j \mathbf{d}_2$$

$$= \sum_{i=1}^{m} \sum_{j=1}^{n} d_i^1 d_j^2 N_{i,k}(u) N_{j,l}(v), \quad \text{knot set } (u_i)_{i=1}^{p+k} \times (v_j)_{j=1}^{q+l}.$$

This is a B-spline surface whose basis functions are products of the curve basis functions. By allowing the control points to take arbitrary values we obtain the general formula for a B-spline surface:

$$x(u,v) = \sum_{i=1}^{p} \sum_{j=1}^{q} d_{ij} N_{i,k}(u) N_{j,l}(v) = \mathbf{N}_u^T \, \mathbf{d} \, \mathbf{N}_v = \mathbf{d} \odot_i \mathbf{N}_u \odot_j \mathbf{N}_v, \tag{2.40}$$

where

$$\mathbf{N}_u = \begin{pmatrix} N_{1,k}(u) & \dots & N_{p,k}(u) \end{pmatrix}^T, \quad \mathbf{N}_v = \begin{pmatrix} N_{1,l}(v) & \dots & N_{q,l}(v) \end{pmatrix}^T,$$

and $\mathbf{d}$ is a matrix of control points

$$\mathbf{d} = \begin{pmatrix} d_{11} & \dots & d_{1q} \\ \vdots & \ddots & \vdots \\ d_{p1} & \dots & d_{pq} \end{pmatrix}.$$

The B-spline basis functions, $N_{i,k}(u)N_{j,l}(v)$, are themselves functional B-spline surfaces defined locally over the kl knots with domain $[u_i, u_{i+k}] \times [v_j, v_{j+l}]$.

Since a B-spline surface is formed from a tensor product of the curve scheme it is not surprising that the isoparametric curves making up a B-spline surface have a B-spline representation. For example, letting $u = u_0$ say, we obtain the v curve at u_0 :

$$\sum_{i=1}^{p} \sum_{j=1}^{q} d_{ij} N_{i,k}(u_0) N_{j,l}(v) = \sum_{j=1}^{q} \left(\sum_{i=1}^{p} d_{ij} N_{i,k}(u_0) \right) N_{j,l}(v) = \sum_{j=1}^{q} w_j N_{j,l}(v),$$

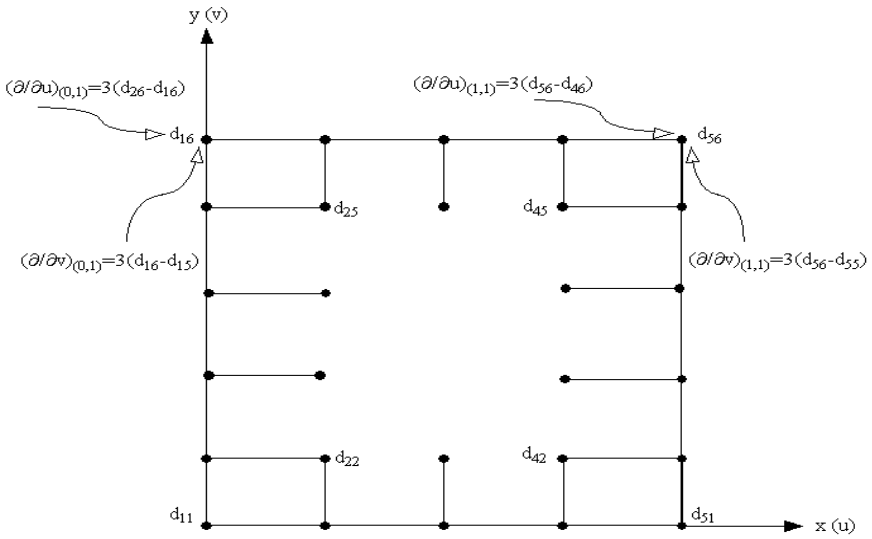


Figure 2.12: B-spline surface boundary derivatives for bi-cubic example over $[0,1] \times [0,1]$

defined by the control polygon $(w_j)_{j=1}^{q}$, where

$$\mathbf{w}_j = \sum_{i=1}^{p} d_{ij} N_{i,k}(u_0).$$

An important consequence of this is that the boundary control points actually define the boundary curves for the surface. In addition the derivatives of a B-spline surface along the boundaries can be conveniently computed using the first two rows or columns of control points making up the control net. We illustrate this important fact for a bi-cubic example in the schematic figure 2.12.

A significant property of higher dimensional entities derived from the tensor product of lower dimensional ones is that the evaluation, manipulation and creation methods (such as interpolation, approximation etc) generalise very conveniently. In each case the higher dimensional method involves repetition of the same 1-dimensional version in each coordinate direction. As well as having useful theoretical consequences this also leads to important benefits computationally. In the following sections, we review briefly the extensions of the curve algorithms considered in section 2.3 to the surface case and derive tensor forms for the surface derivatives.

Note: In all the following tensor product extensions of the curve algorithms one must decide on a direction to process first. Whereas there can be some small computational advantage of one

selection over another depending on the dimensions of the B-spline in the different directions, the end result is the same.

2.4.1 Evaluation

The evaluation of a point on a B-spline surface $x(u, v)$ at a parameter pair (u_0, v_0) is achieved via the following two step process:

1. Apply the curve evaluation algorithm to the each of the columns of control points $(d_{ij})_{i=1}^{p}, j = 1, \ldots, q$ using the u knot set $(u_i)_{i=1}^{p+k}$ and the u evaluation point u_0. This results in q points, one for each column, corresponding to the apex of the appropriate triangular array table 2.1 generated by the curve algorithm.

2. Apply the curve algorithm once more to the q points generated from step 1, using the v knot set $(v_j)_{j=1}^{q+l}$ and the v evaluation point v_0. The resulting point is the value of $x(u, v)$ at (u_0, v_0).

2.4.2 Derivatives

Explicit derivative evaluation follows the two step point evaluation process above using the curve derivative evaluation in both steps. In terms of tensor notation formulae for the derivatives we have the following:

First Derivative

$$\frac{\partial x}{\partial u} = \sum_{i=1}^{p-1} \sum_{j=1}^{q} d_{ij}^{10} N_{i,k-1}(u) N_{j,l}(v),$$

where

$$(d_{ij}^{1\,0})_{i,j=1}^{p-1,q} = \mathbf{D}_u^1 \mathbf{d},$$

and where $\mathbf{D}_u^1$ is the first derivative matrix with respect to the u knot set (more correctly using 2.24 we should write $\mathbf{D}_{0u}^1$ but for clarity we omit the 0). Then we can write

$$\frac{\partial x}{\partial u} = (\mathbf{N}_u^1)^T \, \mathbf{D}_u^1 \, \mathbf{d} \, \mathbf{N}_v,$$

where

$$\mathbf{N}_u^1 = \begin{pmatrix} N_{1,k-1}(u) & \ldots & N_{p-1,k-1}(u) \end{pmatrix}^T.$$

In tensor form this becomes

$$\mathbf{d} \odot_i \left(\left(\mathbf{D}_u^1 \right)^T \mathbf{N}_u^1 \right) \odot_j \mathbf{N}_v.$$

Similarly for the v partial derivative we have

$$\frac{\partial x}{\partial v} = \mathbf{N}_u^T \, \mathbf{d} \, (\mathbf{D}_v^1)^T \, \mathbf{N}_v^1 = \mathbf{d} \odot_i \mathbf{N}_u \odot_j \left(\left(\mathbf{D}_v^1 \right)^T \mathbf{N}_v^1 \right),$$

where

$$\mathbf{N}_v^1 = \begin{pmatrix} N_{1,l-1}(v) & \dots & N_{q-1,l-1}(v) \end{pmatrix}^T.$$

Higher Derivatives

Using tensor notation we have, for example:

$$\frac{\partial^2 x}{\partial u \partial v} = (\mathbf{N}_u^1)^T \, \mathbf{D}_u^1 \, \mathbf{d} \, (\mathbf{D}_v^1)^T \, \mathbf{N}_v^1 = \mathbf{d} \odot_i \left(\left(\mathbf{D}_u^1 \right)^T \mathbf{N}_u^1 \right) \odot_j \left(\left(\mathbf{D}_v^1 \right)^T \mathbf{N}_v^1 \right),$$

$$\frac{\partial^2 x}{\partial u^2} = (\mathbf{N}_u^2)^T \, \mathbf{D}_u^2 \, \mathbf{d} \, \mathbf{N}_v = \mathbf{d} \odot_i \left(\left(\mathbf{D}_u^2 \right)^T \mathbf{N}_u^2 \right) \odot_j \mathbf{N}_v^T,$$

$$\frac{\partial^2 x}{\partial v^2} = (\mathbf{N}_u)^T \, \mathbf{d} \, (\mathbf{D}_v^2)^T \, \mathbf{N}_v^2 = \mathbf{d} \odot_i \mathbf{N}_u \odot_j \left(\left(\mathbf{D}_v^2 \right)^T \mathbf{N}_v^2 \right).$$

For the general case with $a = a_1 + a_2$ we have

$$\frac{\partial^a x}{\partial u^{a_1} \partial v^{a_2}} = \left(\mathbf{N}_u^{a_1} \right)^T \, \mathbf{D}_u^{a_1} \, \mathbf{d} \, \left(\mathbf{D}_v^{a_2} \right)^T \mathbf{N}_v^{a_2} = \mathbf{d} \odot_i \left(\left(\mathbf{D}_u^{a_1} \right)^T \mathbf{N}_u^{a_1} \right) \odot_j \left(\left(\mathbf{D}_v^{a_2} \right)^T \mathbf{N}_v^{a_2} \right). \tag{2.41}$$

Finally, the extension of algorithm 2.6 to computing the (a_1, a_2)th derivative surface as a B-spline surface is achieved via the tensor product generalisation of the curve algorithm. After computing the (a_1, a_2)th derivative knot sets using algorithm 2.2 we process the rows of control points as B-spline curves defined over the v knot set, computing the a_2th derivative for each row. We then process the control points from these derivative curves columnwise as B-spline curves defined over the u knot set computing the a_1th derivative for each column. The resulting control points together with the new knot sets form the derivative surface.

2.4.3 Integration

The integration formula extends directly to surfaces through the tensor product construction. For

$$x(u, v) = \sum_{i=1}^{p} \sum_{j=1}^{q} d_{ij} N_{i,k}(u) N_{j,l}(v),$$

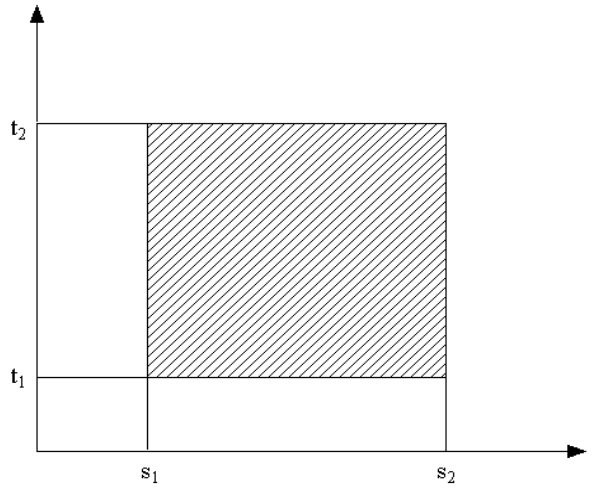


Figure 2.13: B-spline definite integral summation

on the knot set $(u_i)_{i=1}^{p+k} \times (v_j)_{j=1}^{q+l}$, we can compute the integral B-spline surface in u or v by integrating over the columns or rows of control points respectively. The indefinite integral of the surface $x(u, v)$,

$$y(u, v) = \int_{u_1}^{u} \int_{v_1}^{v} x(s, t) ds\, dt,$$

is another B-spline surface (up to an additive constant), formed by composing the integration in u with that in v in the tensor product fashion. The new knot sets in u and v are derived from the original ones by augmenting one knot at either end to tie in with the increased order. For the definite integral we have, by adding and subtracting appropriate areas (see figure 2.13), the result

$$\int_{s_1}^{s_2} \int_{t_1}^{t_2} x(u, v) du\, dv = y(s_2, t_2) - y(s_2, t_1) - y(s_1, t_2) + y(s_1, t_1). \tag{2.42}$$

2.4.4 Knot insertion and removal

Knot insertion for B-spline surfaces is the tensor product generalizations of the curve algorithm. By writing the B-spline surface $x(u, v)$ as

$$x(u, v) = \sum_{j=1}^{q} \Big(\sum_{i=1}^{p} d_{ij} N_{i,k}(u) \Big) N_{j,l}(v),$$

we see that a knot $\hat{u}$ say, can be inserted into the u knotline by treating each column of control points $(d_{ij})_{i=1}^{p}, j = 1, \ldots, q$ as defining a B-spline curve in u and applying the curve knot insertion algorithm q times, using the u knot set. This results in one extra control point for each column, and new surface

$$x(u,v) = \sum_{i=1}^{p+1} \sum_{j=1}^{q} d_{ij}^{1} N_{i,k}(u) N_{j,l}(v)$$

say, where the u knot set $(u_i)_{i=1}^{p+k+1}$ now includes $\hat{u}$. In an analogous fashion and by writing the surface as

$$x(u,v) = \sum_{i=1}^{p} \Big(\sum_{j=1}^{q} d_{ij} N_{j,l}(v) \Big) N_{i,k}(u),$$

we can insert a knot into the v knotline by processing the rows of control points $(d_{ij})_{j=1}^{q}, i = 1, \ldots p$, using the v knot set. Multiple knot insertions are achieved by applying the above process iteratively. The surface knot removal algorithm follows an analogous approach.

2.4.5 Product

The product algorithm extends to B-spline surfaces. Both versions described in section 2.3.5 have natural tensor product generalisations. After normalisation of the u and v knot sets of the two surfaces, the Bézier based algorithm relies on the corresponding formula for the product of two Bézier surfaces. For f and g two Bézier surfaces of orders (p_1, q_1) and (p_2, q_2):

$$f(s,t) = \sum_{i=0}^{p_1} \sum_{j=0}^{q_1} b_{ij}^{1} B_i^{p_1}(s) B_j^{q_1}(t)\,, \quad g(s,t) = \sum_{i=0}^{p_2} \sum_{j=0}^{q_2} b_{ij}^{2} B_i^{p_2}(s) B_j^{q_2}(t),$$

the product h is a Bézier surface of order $(p_1 + p_2, q_1 + q_2)$ such that

$$h = fg = \sum_{i=0}^{p_1+p_2} \sum_{j=0}^{q_1+q_2} b_{ij} B_i^{p_1+p_2}(s) B_j^{q_1+q_2}(t)\,,$$

where

$$b_{ij} = \sum_{k=\max(0,i-p_2)}^{\min(p_1,i)} \sum_{l=\max(0,j-q_2)}^{\min(q_1,j)} \frac{\binom{p_1}{k}\binom{p_2}{i-k}\binom{q_1}{l}\binom{q_2}{j-l}}{\binom{p_1+p_2}{i}\binom{q_1+q_2}{j}} b_{kl}^{1} b_{i-k\,j-l}^{2}. \tag{2.43}$$

Using the product surface knot set from Morken's algorithm, [60], knot removal in u and v is applied to recreate the minimal B-spline representation for the product surface from the piecewise Bézier representation.

2.5 B-spline Volumes

A B-spline volume (or solid) of order (l, m, n) is a tensor product generalisation of three curve schemes of orders l, m, n. The formula is

$$x(u, v, w) = \sum_{i=1}^{p} \sum_{j=1}^{q} \sum_{k=1}^{r} d_{ijk} N_{i,l}(u) N_{j,m}(v) N_{k,n}(w), \tag{2.44}$$

with associated knot vectors in u, v, w:

$$(u_i)_{i=1}^{p+l}, \quad (v_j)_{j=1}^{q+m}, \quad (w_k)_{k=1}^{r+n}.$$

Here $(d_{ijk})_{i,j,k=1}^{p,q,r}$ is a Matrix3D control points. The basis functions, $N_{i,l}(u)N_{j,m}(v)N_{k,n}(w)$, are themselves functional B-spline volumes defined locally over the domain $[u_i, u_{i+l}] \times [v_j, v_{j+m}] \times [w_k, w_{k+n}]$.

We can write formula 2.44 in tensor form as follows:

$$x(u, v, w) = \mathbf{d} \odot_i \mathbf{N}_u \odot_j \mathbf{N}_v \odot_k \mathbf{N}_w, \quad \mathbf{d} = (d_{ijk})_{i,j,k=1}^{p,q,r}.$$

This is effectively a triple contraction of the third rank tensor $\mathbf{d}$ with the basis function vectors $\mathbf{N}_u, \mathbf{N}_v, \mathbf{N}_w$. The first tensor contraction produces a matrix, the second a vector and the third a scalar equal to the triple sum 2.44. In an analogous fashion to the lower dimensional B-spline entities, the isoparametric surfaces making up a B-spline volume have a B-spline surface representation and the boundary control points of the volume define the boundary surfaces. In addition the boundary derivatives of a B-spline volume can be conveniently calculated using the boundary surface control points and the next layer of control points into the volume in the three directions. Figure 2.14 gives a schematic representation of the six isoparametric faces making up the volume and the boundary derivative property for the bottom face $w = 0$.

The principal algorithms for B-spline volumes derive their form from the tensor product generalisation of the corresponding curve and surface algorithms. We review these in the following sections and extend the tensor form of the derivative expressions to the volume case.

2.5.1 Evaluation

The evaluation of a point on a B-spline volume $x(u, v, w)$ at a parameter triple (u_0, v_0, w_0) can be achieved via the following two step process:

1. apply the surface evaluation algorithm to the each of the layers of control points $(d_{ijk})_{i,j=1}^{p,q}, k = 1, \ldots, r$ using the uv knot sets $(u_i)_{i=1}^{p+l}, (v_j)_{j=1}^{q+m}$ and the uv parameter pair (u_0, v_0). This results in r points, one for each layer.

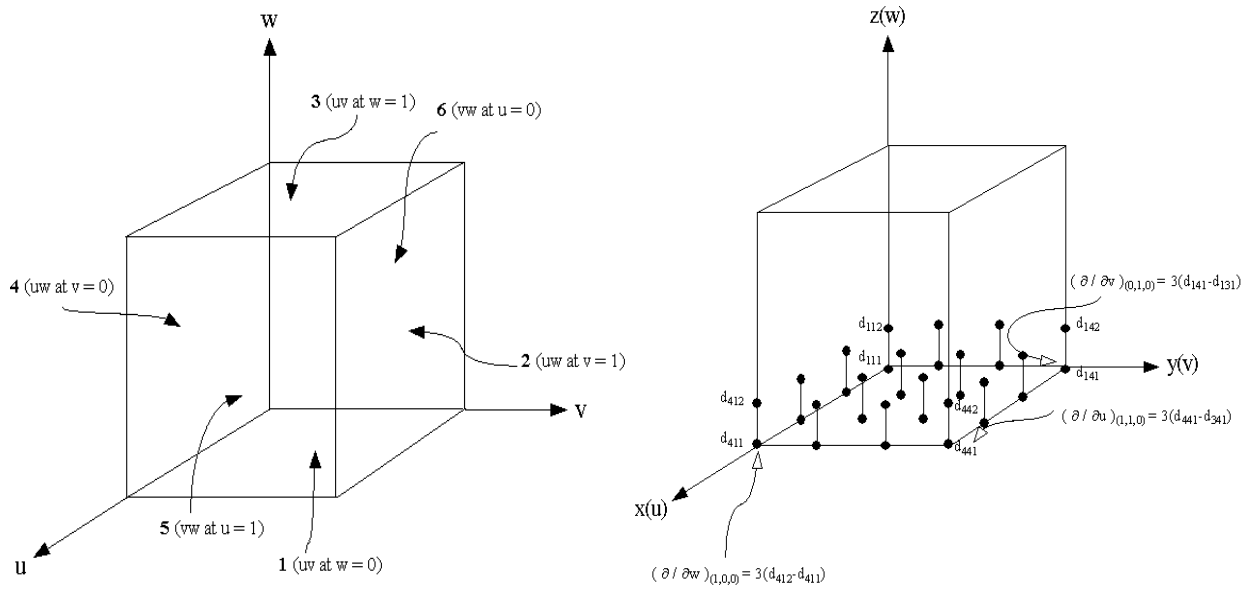


Figure 2.14: B-spline volume isoparametric faces and boundary derivative property

2. apply the curve algorithm to the r points generated from step 1, using the w knot set $(w_k)_{k=1}^{r+n}$ and the w evaluation point w_0. The resulting point is the value of x at (u_0, v_0, w_0).

2.5.2 Derivatives

Explicit derivative evaluation is achieved by using the above method with the curve and surface derivative algorithms in place of point evaluation. For the tensor form of the derivative B-spline volume of order (l, m, n) we can express the first derivatives in u, v, w as follows:

$$\frac{\partial x}{\partial u} = \sum_{i=1}^{p-1} \sum_{j=1}^{q} \sum_{k=1}^{r} d_{ijk}^{100} N_{i,l-1}(u) N_{j,m}(v) N_{k,n}(w),$$

where $\left(d_{ijk}^{100}\right)_{i,j,k=1}^{p-1,q,r} = \mathbf{d} \odot_i \mathbf{D}_u^1$. In tensor notation this becomes

$$\mathbf{d} \odot_i \left(\left(\mathbf{D}_u^1 \right)^T \mathbf{N}_u^1 \right) \odot_j \mathbf{N}_v \odot_k \mathbf{N}_w.$$

Similarly

$$\frac{\partial x}{\partial v} = \sum_{i=1}^{p} \sum_{j=1}^{q-1} \sum_{k=1}^{r} d_{ijk}^{010} N_{i,l}(u) N_{j,m-1}(v) N_{k,n}(w),$$

where $\left(d_{ijk}^{010}\right)_{i,j,k=1}^{p,q-1,r} = \mathbf{d} \odot_j \mathbf{D}_v^1$, in tensor form this becomes

$$\mathbf{d} \odot_i \mathbf{N}_u \odot_j \left(\left(\mathbf{D}_v^1 \right)^T \mathbf{N}_v^1 \right) \odot_k \mathbf{N}_w.$$

Finally

$$\frac{\partial x}{\partial w} = \sum_{i=1}^{p} \sum_{j=1}^{q} \sum_{k=1}^{r-1} d_{ijk}^{001} N_{i,l}(u) N_{j,m}(v) N_{k,n-1}(w),$$

where $\left(d_{ijk}^{001}\right)_{i,j,k=1}^{p,q,r-1} = \mathbf{d} \odot_k \mathbf{D}_w^1$. In tensor form this appears as

$$\frac{\partial x}{\partial w} = \mathbf{d} \odot_i \mathbf{N}_u \odot_j \mathbf{N}_v \odot_k \left(\left(\mathbf{D}_w^1 \right)^T \mathbf{N}_w^1 \right).$$

The mixed partial in u, v, w is:

$$\frac{\partial^3 x}{\partial u \partial v \partial w} = \mathbf{d} \odot_i \left(\left(\mathbf{D}_u^1 \right)^T \mathbf{N}_u^1 \right) \odot_j \left(\left(\mathbf{D}_v^1 \right)^T \mathbf{N}_v^1 \right) \odot_k \left(\left(\mathbf{D}_w^1 \right)^T \mathbf{N}_w^1 \right),$$

where

$$\left(d_{ijk}^{111}\right)_{i,j,k=1}^{p-1,q-1,r-1} = \mathbf{d} \odot_i \mathbf{D}_u^1 \odot_j \mathbf{D}_v^1 \odot_k \mathbf{D}_w^1.$$

More generally for $a_1 + a_2 + a_3 = a$ we have

$$\frac{\partial^a x}{\partial u^{a_1} \partial v^{a_2} \partial w^{a_3}} = \sum_{i=1}^{p-a_1} \sum_{j=1}^{q-a_2} \sum_{k=1}^{r-a_3} d_{ijk}^{a_1 a_2 a_3} N_{i,l-a_1}(u) N_{j,m-a_2}(v) N_{k,n-a_3}(w),$$

where

$$(d_{ijk}^{a_1 a_2 a_3})_{i,j,k=1}^{p-a_1,q-a_2,r-a_3} = \mathbf{d} \odot_i \mathbf{D}_u^{a_1} \odot_j \mathbf{D}_v^{a_2} \odot_k \mathbf{D}_w^{a_3},$$

and in tensor form this becomes

$$\mathbf{d} \odot_i \left(\left(\mathbf{D}_u^{a_1} \right)^T \mathbf{N}_u^{a_1} \right) \odot_j \left(\left(\mathbf{D}_v^{a_2} \right)^T \mathbf{N}_v^{a_2} \right) \odot_k \left(\left(\mathbf{D}_w^{a_3} \right)^T \mathbf{N}_w^{a_3} \right). \tag{2.45}$$

The extension of algorithm 2.6 to computing the (a_1, a_2, a_3)th derivative as a B-spline volume is analogous to the surface case. After computing the (a_1, a_2, a_3)th derivative knot sets we process the w direction layers of control points as B-spline surfaces defined over the u, v knot sets, computing the (a_1, a_2)th derivative for each layer. We then process the control points from these derivative surfaces columnwise as B-spline curves defined over the w knot set computing the a_3th derivative for each column. The resulting control points together with the new knot sets form the derivative B-spline volume.

2.5.3 Integral

The integral of a tensor product B-spline volume of order (l, m, n)

$$x(u, v, w) = \sum_{i=1}^{p} \sum_{j=1}^{q} \sum_{k=1}^{r} d_{ijk} N_{i,l}(u) N_{j,m}(v) N_{k,n}(w)$$

on the knot set $(u_i)_{i=1}^{p+l} \times (v_j)_{j=1}^{q+m} \times (w_k)_{k=1}^{r+n}$, can be formed by composing three separate integrations in the u, v and w directions. The indefinite integral of the volume $x(u, v, w)$ given by

$$y(u, v, w) = \int_{u_1}^{u} \int_{v_1}^{v} \int_{w_1}^{w} x(s, t, r) ds\, dt\, dr,$$

and is a B-spline volume of one higher order in u, v and w. The definite integral is obtained by adding and subtracting appropriate volumes (see figure 2.15) with the following result:

$$\int_{s_1}^{s_2} \int_{t_1}^{t_2} \int_{r_1}^{r_2} x(u, v, w) du\, dv\, dw = y(s_2, t_2, r_2) - y(s_2, t_2, r_1) - y(s_2, t_1, r_2) + y(s_2, t_1, r_1)$$

$$+ y(s_1, t_2, r_1) + y(s_1, t_1, r_2) - y(s_1, t_1, r_1). \tag{2.46}$$

2.5.4 Knot insertion and removal

Knot insertion for B-spline volumes follows the tensor product generalization of the curve algorithm. By writing the B-spline volume $x(u, v, w)$ as

$$x(u, v, w) = \sum_{i=1}^{p} \sum_{j=1}^{q} \Big(\sum_{k=1}^{r} d_{ijk} N_{k,n}(w) \Big) N_{i,l}(u) N_{j,m}(v),$$

we see that a knot $\hat{w}$ say, can be inserted into the w knotline by treating each line of control points in the w direction $(d_{ijk})_{i,j=1}^{p,q}, k = 1, \ldots, r$ as defining a B-spline curve in w and applying the curve knot insertion algorithm $p * q$ times, using the w knot set. This results in one extra control point for each line, and new volume

$$x(u, v, w) = \sum_{i=1}^{p} \sum_{j=1}^{q} \sum_{k=1}^{r+1} d^1_{ijk} N_{i,l}(u) N_{j,m}(v) N_{k,n}(w)$$

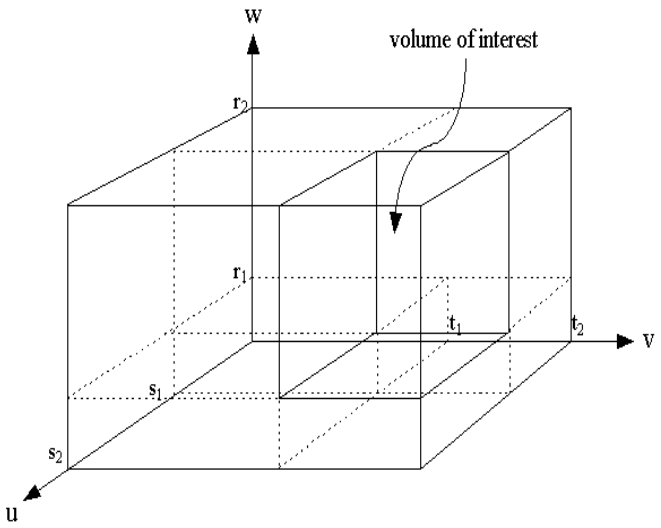


Figure 2.15: B-spline volume definite integral summation

say, where the w knot set $(w_k)_{k=1}^{n+r+1}$ now includes $\hat{w}$. In an analogous fashion and by writing the volume as

$$x(u,v,w) = \sum_{j=1}^{l}\sum_{k=1}^{r}\Big(\sum_{i=1}^{p} d_{ijk}N_{i,l}(u)\Big)N_{j,m}(v)N_{k,n}(w)$$

and

$$x(u,v,w) = \sum_{i=1}^{p}\sum_{k=1}^{r}\Big(\sum_{j=1}^{q} d_{ijk}N_{j,m}(v)\Big)N_{i,l}(u)N_{k,n}(w)$$

we can insert knots into the u and v knotlines respectively. Multiple knot insertions are achieved by applying the above process iteratively. The knot removal algorithm follows this procedure in an analogous fashion.

2.5.5 Product

The product algorithm extends in a natural way to B-spline volumes. After normalisation of the u, v and w knot sets of the two entities, the Bézier based algorithm uses the corresponding formula for the product of two Bézier volumes. For f and g two Bézier volumes of orders

(p_1, q_1, r_1) and (p_2, q_2, r_2):

$$f(u,v,w) = \sum_{i=0}^{p_1}\sum_{j=0}^{q_1}\sum_{k=0}^{r_1} b^1_{ijk} B_i^{p_1}(u) B_j^{q_1}(v) B_k^{r_1}(w),$$

$$g(u,v,w) = \sum_{i=0}^{p_2}\sum_{j=0}^{q_2}\sum_{k=0}^{r_2} b^2_{ijk} B_i^{p_2}(u) B_j^{q_2}(v) B_k^{r_2}(w),$$

the product h of order $(p_1+p_2, q_1+q_2, r_1+r_2)$ is such that

$$h = fg = \sum_{i=0}^{p_1+p_2}\sum_{j=0}^{q_1+q_2}\sum_{k=0}^{r_1+r_2} b_{ijk} B_i^{p_1+p_2}(u) B_j^{q_1+q_2}(v) B_k^{r_1+r_2}(w),$$

where

$$b_{ijk} = \sum_{l=\max(0,i-p_2)}^{\min(p_1,i)} \sum_{m=\max(0,j-q_2)}^{\min(q_1,j)} \sum_{n=\max(0,k-r_2)}^{\min(r_1,k)} \frac{\binom{p_1}{l}\binom{p_2}{i-l}\binom{q_1}{m}\binom{q_2}{j-m}\binom{r_1}{n}\binom{r_2}{k-n}}{\binom{p_1+p_2}{i}\binom{q_1+q_2}{j}\binom{r_1+r_2}{k}} b^1_{lmn} b^2_{i-l,j-m,k-n}. \quad (2.47)$$

Finally, using the product knot sets in u, v, w from Morken's algorithm, knot removal in u, v, w is used to recreate the minimal B-spline representation for the product volume from the piecewise Bézier representation.

2.6 Summary

In this chapter we have introduced and reviewed tensor notation which allows us to deal with the basic operations of product and contraction on vector, matrix and matrix3D entities. Starting from the basic B-spline curve point and derivative evaluation formulae we have developed a matrix form and a notation for representing the derivative B-spline which allows us to express the control points of the derivative curve in terms of the original set. As a by product we have presented an algorithm to compute the derivative of a general B-spline curve as a new B-spline curve. We have taken these results together with some additional important algorithms for curve entities and, using tensor notation, reviewed their extensions and generalisations to the surface and volume forms. In the next chapter we use the notation and results presented here to derive explicit formulae and algorithms for computing the minimisation of certain types of B-spline functionals which will form the basis for the applications presented in Chapters 4 and 5.

Chapter 3

Functional Minimisation Formulae & Algorithms

3.1 Introduction and Background

There is a rich branch of mathematics that concerns itself with problems in which it is necessary to determine the maximum and minimum values of functions. A related subject to this and one which is often encountered in problems that have a physical basis is that where one seeks to find the maximum and minimum values of so called *functionals*.

Functionals are variable quantities whose values are determined by the choice of one or several functions. For example the arc length l of a plane or space curve connecting two points a and b is a functional as is the area of a surface since they are both determined by a choice of a curve or surface. Moments of inertia, static moments, the coordinates of the center of gravity of a curve or surface are also functionals in the same way. In all these examples we have a relationship that is characteristic of functionals: *to a function there corresponds a real value*. The *order* of the functional is the order of the highest partial derivative that enters the functional and a functional is said to be linear, quadratic, cubic etc if the highest powers of the derivatives occurring in the integral are those of order one, two, three etc.

Numerous laws of mechanics and physics reduce to the statement that a certain functional in a given process has to reach a minimum or maximum. Such principles are called variational principles of mechanics or physics and the subject of the *calculus of variations* investigates methods that permit finding the maximum or minimum values of functionals.

Minimising a functional is equivalent to requiring that the first variation of that functional is zero which gives rise to the *Euler* differential equation. This is Euler's theorem of variational

calculus: a solution that satisfies the essential boundary conditions and renders the functional stationary also satisfies the Euler differential equation.

The variational formulation of a continuum problem has the following advantages over the differential equation formulation:

- the functional I usually possess a clear physical meaning,
- the functional I contains lower order derivatives of the field variable compared to the governing differential equation making it easier to find approximate solutions,
- the variational formulation enables us to treat complicated boundary conditions as natural boundary conditions which are implicitly imposed by the variational statement of the problem and hence only geometric boundary conditions need be imposed.

In this chapter we review the functional/differential equation connection in more detail from both directions and then using the variational form and the results from Chapter 2 derive formulae and corresponding algorithms for the minimisation of various B-spline curve, surface and volume functionals, involving integral expressions of squares and products of derivatives. These developments form the basis for specific algorithms for solving B-spline based variational problems in geometric smoothing and finite elements presented in chapters 4 and 5. We also look at a method called 'the reduced transformation technique' that is employed for dealing with geometric constraints. Finally, we present algorithms for computing the exact product of two B-spline functions in curve surface and volume form enabling us to treat the source terms appearing in the variational form of a given problem.

3.2 Functionals and Differential Equations

3.2.1 Functionals dependent on functions of one variable

The basic relationship between differential equations and functionals dependent on functions of one variable can be seen by considering the differential equation in linear differential operator form. For such an operator L and associated linear equation $L\phi = f$, where $\phi = \phi(t)$ and there are given boundary conditions at a and b, we consider the following functional

$$I(\phi) = \int_a^b \big(L\phi(t)\big)\phi(t)dt - 2\int_a^b f\phi(t)dt.$$

F is minimised at a particular function $\phi = \psi$ only if its derivative vanishes there and the condition for this can be shown to be the equation $L\phi = f$, called, in this respect, the Euler equation of the functional. An example of this is the following differential equation

$$\frac{d^2\phi}{dt^2} + \phi + t = 0\,, \quad 0 \leq t \leq 1,$$

with boundary conditions $\phi(0) = \phi(1) = 0$. In this case $L = \left(d^2/dt^2 + 1\right)$ and $f = -t$. Using integration by parts and the given boundary conditions we have

$$\begin{aligned} I(\phi) &= \int_0^1 \Big((\phi'(t))' + \phi(t)\Big)\phi(t)dt + 2\int_0^1 t\phi(t)dt \\ &= \int_0^1 \Big(-(\phi'(t))^2 + \phi^2(t)\Big)dt + \phi'(t)\phi(t)\Big]_0^1 + 2\int_0^1 t\phi(t)dt \\ &= \int_0^1 \Big(-(\phi'(t))^2 + \phi^2(t) + 2t\phi(t)\Big)dt. \end{aligned} \tag{3.1}$$

By minimising the first order, quadratic functional 3.1 subject to the boundary conditions we effectively solve the original differential equation.

Looking at the problem from the other way round, that is starting from the definition of a functional, the task is to find a function $x(t)$ that makes the following integral $I(x)$ take a minimum value

$$I(x) = \int_a^b F(t, x, x')dt,$$

where boundary conditions on the unknown function are prescribed

$$x(a) = x_0, \quad x(b) = x_1.$$

We find a solution $x(t)$ by constructing comparison or trial solutions

$$x = x(t) + \alpha\eta(t).$$

such that when $\alpha = 0$, x becomes the solution, and we require that $\eta(a) = \eta(b) = 0$ so that x still satisfies all the boundary conditions. Considering the minimisation of I as a function of α, an alternative way of saying that at a minimum of a function, $f(t)$, the derivative vanishes

is to say that the variation $\delta f = f_t \delta t$ vanishes. If we consider an increment of the introduced parameter α, then we have

$$\delta I[x] = I[x + \delta x] - I[x] = \int_a^b F(t, x + \delta x, x' + \delta x')dt - \int_a^b F(t, x, x')dt$$

$$= \int_a^b (F_x \delta x + F_{x'} \delta x')dt.$$

Hence we have an expression for the variation δI with x. Using integration by parts as necessary all derivatives of y can be eliminated from under the integral sign, giving

$$\delta I[x] = \int_a^b [F_x - (F_{x'})']\delta x dt.$$

For I to be a minimum this variation must be zero for any δx. This can only be guaranteed if the expression in the brackets is zero. Hence the differential equation for the solution $x(t)$ is the (second order) Euler equation:

$$F_x - (F_{x'})' = 0.$$

The integral curves of Euler's equation, $x = x(t, C_1, C_2)$, are called extremals. It is only on the extremals that the functional

$$I[x(t)] = \int_{t_0}^{t_1} F(t, x(t), x'(t))dt$$

can be extremised.

More generally it can be shown that (see for example Esgolts, [18]) the function $x(t)$ that extremises the value of the functional

$$I[x(t)] = \int_{t_0}^{t_1} F(t, x(t), x'(t), \ldots, x^{(n)}(t))dt,$$

where the boundary conditions are of the form

$$x(t_0) = x_0,\ x'(t_0) = x_0^{'},\ \ldots\ , x^{(n-1)}(t_0) = x_0^{(n-1)},$$

$$x(t_1) = x_1,\ x'(t_1) = x_1^{'},\ \ldots,\ x^{(n-1)}(t_1) = x_1^{(n-1)},$$

is a solution to the equation

$$F_x - \frac{d}{dt}F_{x'} + \frac{d^2}{dt^2}F_{x''} + \ldots + (-1)^n \frac{d^n}{dt^n}F_{x^{(n)}} = 0.$$

Applying this to the example above we see that minimisation of the functional

$$\int_0^1 \Big(-(\phi'(t))^2 + \phi^2(t) + 2t\phi(t) \Big) dt,$$

with its boundary conditions gives rise to the original differential equation

$$\frac{d^2\phi}{dt^2} + \phi + t = 0\,, \quad 0 \le t \le 1.$$

3.2.2 Functionals dependent on functions of several variables

Starting from a functional dependent on functions of several independent variables a similar differential equation can be found. For example, the following functional

$$I[x(u,v)] = \int_D F(u,v,x,\frac{\partial x}{\partial u},\frac{\partial x}{\partial v})du\,dv,$$

where the values of $x(u,v)$ are given on the boundary C of the domain D, is such that $x(u,v)$ is a solution of the equation

$$F_x - \frac{\partial}{\partial u}F_p - \frac{\partial}{\partial v}F_q = 0,$$

where

$$p = \frac{\partial x}{\partial u}, \quad q = \frac{\partial x}{\partial v}.$$

This second order pde must be satisfied by the extremising function $x(u,v)$. Some specific instances of this are:

1. $$I[x(u,v)] = \int_D \Big[\Big(\frac{\partial x}{\partial u}\Big)^2 + \Big(\frac{\partial x}{\partial v}\Big)^2 \Big] du\,dv,$$

   where the corresponding pde is of the form

   $$\nabla^2 x = \frac{\partial^2 x}{\partial u^2} + \frac{\partial^2 x}{\partial v^2} = 0.$$

   This is the familiar Laplace equation.

2.
$$I[x(u,v)] = \int_D \Big[\Big(\frac{\partial x}{\partial u}\Big)^2 + \Big(\frac{\partial x}{\partial v}\Big)^2 - 2xf(u,v) \Big] du\, dv,$$
where the function x is given on the boundary of D. Here the Euler pde equation is of the form
$$\nabla^2 x = \frac{\partial^2 x}{\partial u^2} + \frac{\partial^2 x}{\partial v^2} = f(u,v),$$
which is Poisson's equation.

3.
$$I[x(u,v)] = \int_D \Big[\Big(\frac{\partial x}{\partial u}\Big)^2 + \Big(\frac{\partial x}{\partial v}\Big)^2 + 2\Big(\frac{\partial^2 x}{\partial u \partial v}\Big)^2 \Big] du\, dv,$$
which leads to the so called biharmonic equation
$$0 = \triangle^2 x = \Big(\frac{\partial^2}{\partial u^2} + \frac{\partial^2}{\partial v^2}\Big)^2 x = \frac{\partial^4 x}{\partial u^4} + 2\frac{\partial^4 x}{\partial u^2 \partial v^2} + \frac{\partial^4 x}{\partial v^4}.$$
Minimising I subject to interpolatory constraints gives the so called 'thin plate splines', the 2D analog of the the 1D cubic spline.

4.
$$I = \int_D \left[\Big(\frac{\partial^2 x}{\partial u^2}\Big)^2 + \Big(\frac{\partial^2 x}{\partial v^2}\Big)^2 + 2\Big(\frac{\partial^2 x}{\partial u \partial v}\Big)^2 - 2xf(u,v) \right] du\, dv,$$
where the extremising function $x = f(u,v)$ must satisfy the equation
$$\triangle^2 x = f(u,v).$$

Starting from the differential equation side, a general linear second order elliptic pde in two independent variables is given by
$$\alpha(u,v)\frac{\partial^2 x}{\partial u^2} + \beta(u,v)\frac{\partial^2 x}{\partial u \partial v} + \gamma(u,v)\frac{\partial^2 x}{\partial v^2} + \delta(u,v)x = f(u,v),$$
where $\alpha, \beta, \gamma, \delta$ may be functions of u and v, and $\beta^2 - 4\alpha\gamma$ is satisfied throughout the domain D of solution. The corresponding first order, quadratic functional is
$$I(x(u,v)) = \frac{1}{2}\int_D \Big[\alpha\Big(\frac{\partial x}{\partial u}\Big)^2 + \beta\Big(\frac{\partial x}{\partial u}\Big)\Big(\frac{\partial x}{\partial v}\Big) + \gamma\Big(\frac{\partial x}{\partial v}\Big)^2 - f(u,v)x(u,v) \Big] du\, dv.$$

A more general linear second order pde with respect to three independent variables (u, v, w) is given by

$$a_{11}\frac{\partial^2 x}{\partial u^2} + a_{22}\frac{\partial^2 x}{\partial v^2} + a_{33}\frac{\partial^2 x}{\partial w^2} + a_{12}\frac{\partial^2 x}{\partial u \partial v} + a_{13}\frac{\partial^2 x}{\partial u \partial w} + a_{23}\frac{\partial^2 x}{\partial v \partial w} + bx = f(u, v, w),$$

with corresponding variational functional

$$I(x) = \frac{1}{2}\int_V \Big[a_{11}\Big(\frac{\partial x}{\partial u}\Big)^2 + a_{22}\Big(\frac{\partial x}{\partial v}\Big)^2 + a_{33}\Big(\frac{\partial x}{\partial w}\Big)^2 + a_{12}\Big(\frac{\partial x}{\partial u}\Big)\Big(\frac{\partial x}{\partial v}\Big) + a_{13}\Big(\frac{\partial x}{\partial u}\Big)\Big(\frac{\partial x}{\partial w}\Big) +$$
$$a_{23}\Big(\frac{\partial x}{\partial v}\Big)\Big(\frac{\partial x}{\partial w}\Big) - f(u, v, w)x(u, v, w)\Big]\, du\, dv\, dw. \tag{3.2}$$

A special case of this is Laplace's equation in 3D

$$\frac{\partial^2 x}{\partial u^2} + \frac{\partial^2 x}{\partial v^2} + \frac{\partial^2 x}{\partial w^2} = 0,$$

which has corresponding variational functional

$$I(x(u, v, w)) = \frac{1}{2}\int_V \Big[\Big(\frac{\partial x}{\partial u}\Big)^2 + \Big(\frac{\partial x}{\partial v}\Big)^2 + \Big(\frac{\partial x}{\partial w}\Big)^2\Big]\, du\, dv\, dw. \tag{3.3}$$

In each of these cases we have to find a solution, continuous in D or V, of the equation that takes on specified values on the boundary of D. This is called the *Dirichlet problem* and we look in more detail at specific B-spline solutions to this problem using the variational form in chapter 5.

In order to use the above variational forms derived either explicitly from a differential equation or from more general considerations, to solve constrained problems in geometric modelling and finite element analysis using B-splines, we need to look in more detail at the explicit minimisation of B-spline functionals involving squares and products of derivatives. In the following sections we present this analysis looking first at the single variable curve case. We use tensor notation and the results given by 2.5, 2.8, 2.9, 2.12 and 2.13 to find expressions for the minimisation of the square of the various derivatives. We also use the notation $\partial/\partial\mathbf{d}$ to indicate the collective differentiation of all the d_i, that is

$$\frac{\partial}{\partial \mathbf{d}} = \begin{pmatrix} \frac{\partial}{\partial d_1} & \cdots & \frac{\partial}{\partial d_n} \end{pmatrix}^T.$$

Finally, for ease of notation we omit the limits on the integrals.

3.3 Curve Functional Minimisation

3.3.1 Minimisation of the zeroth derivative

For $f(t)$ a B-spline curve of order k on the knot set $(t_i)_{i=1}^{n+k}$ we wish to compute the minimisation of B-spline functionals involving squares of derivative terms. To begin with we look at minimisation of the 0th derivative i.e. the integral of the function squared itself (in finite element terminology this corresponds to the mass matrix term):

Note: The linear systems for the control points that we derive in the following sections have as a solution the trivial one $\mathbf{d} = \mathbf{0}$. In practice we are interested in the matrices that appear in these systems knowing that in applications to functional minimisation they are always combined with suitable constraints to ensure a non-zero solution.

$$\frac{\partial}{\partial d_i}\int \big(f(t)\big)^2 dt = \frac{\partial}{\partial d_i}\int \Big(\sum_j d_j N_{j,k}(t)\Big)^2 dt.$$

We have

$$\frac{\partial}{\partial d_i}\int \big(f(t)\big)^2 dt = \frac{\partial}{\partial \mathbf{d}}\int \big(\mathbf{d}\odot_i \mathbf{N}\big)^2 dt = \int \frac{\partial}{\partial \mathbf{d}}\big(\mathbf{d}\odot_i \mathbf{N}\big)^2 dt$$

$$= 2\int \big(\mathbf{d}\odot_i \mathbf{N}\big)\otimes \mathbf{N}\, dt = 2\int \mathbf{N}\otimes\big(\mathbf{d}\odot_i \mathbf{N}\big)\, dt = 2\int \mathbf{N}\otimes\big(\mathbf{N}^T\odot_j \mathbf{d}\big)\, dt$$

$$= 2\int \big(\mathbf{N}\otimes\mathbf{N}^T\big)\odot_j \mathbf{d}\, dt = 2\Big(\int \mathbf{N}\otimes\mathbf{N}^T dt\Big)\odot_j \mathbf{d}.$$

Writing

$$\mathbf{A}_0 = \int \big(\mathbf{N}\otimes\mathbf{N}^T\big)\, dt = \begin{pmatrix} \int N_1(t)N_1(t)dt & \dots & \dots & \int N_1(t)N_n(t)dt \\ \vdots & \ddots & & \vdots \\ \vdots & & \ddots & \vdots \\ \int N_1(t)N_n(t)dt & \dots & \dots & \int N_n(t)N_n(t)dt \end{pmatrix},$$

and equating the result to zero we have the following n by n system of equations for the control points $\mathbf{d}$:

$$\mathbf{A}_0\mathbf{d} = \mathbf{0}. \tag{3.4}$$

3.3.2 Minimisation of higher derivatives

For the higher derivative case we seek the system of equations generated by the minimisation of the integral of the rth derivative squared. In finite element terminology for the cases $r = 1$ and $r = 2$ we are dealing with the stiffness and bending matrix components of the problem respectively. Using 2.32 we have

$$\frac{\partial}{\partial \mathbf{d}} \int \left(f^{(r)}(t)\right)^2 dt = \frac{\partial}{\partial \mathbf{d}} \int \left(\mathbf{d} \odot_i \left(\mathbf{D}_0^r\right)^T \mathbf{N}^r\right)^2 dt = \int \frac{\partial}{\partial \mathbf{d}} \left(\mathbf{d} \odot_i \left(\mathbf{D}_0^r\right)^T \mathbf{N}^r\right)^2 dt.$$

The integrand is equal to

$$\left(\mathbf{d} \odot_i \left(\mathbf{D}_0^r\right)^T \mathbf{N}^r\right) \otimes \left(\left(\mathbf{D}_0^r\right)^T \mathbf{N}^r\right) \tag{3.5}$$

$$= \left(\left(\mathbf{D}_0^r\right)^T \mathbf{N}^r\right) \otimes \left(\mathbf{d} \odot_i \left(\mathbf{D}_0^r\right)^T \mathbf{N}^r\right)$$

$$= \left(\left(\mathbf{D}_0^r\right)^T \mathbf{N}^r\right) \otimes \left(\left(\mathbf{N}^r\right)^T \mathbf{D}_0^r \odot_j \mathbf{d}\right)$$

$$= \left(\mathbf{D}_0^r\right)^T \left(\mathbf{N}^r \otimes \left(\mathbf{N}^r\right)^T\right) \mathbf{D}_0^r \odot_j \mathbf{d}.$$

Hence we have the equation

$$\left(\mathbf{D}_0^r\right)^T \left(\int \left(\mathbf{N}^r \otimes \left(\mathbf{N}^r\right)^T\right) dt\right) \mathbf{D}_0^r \odot_j \mathbf{d} = \mathbf{0}.$$

Writing

$$\mathbf{A}_r = \int \left(\mathbf{N}^r \otimes \left(\mathbf{N}^r\right)^T\right) dt$$

$$= \begin{pmatrix} \int N_{1,k-r}(t) N_{1,k-r}(t) dt & \dots & \dots & \int N_{1,k-r}(t) N_{n-r,k-r}(t) dt \\ \vdots & \ddots & & \vdots \\ \vdots & & \ddots & \vdots \\ \int N_{n-r,k-r}(t) N_{1,k-r}(t) dt & \dots & \dots & \int N_{n-r,k-r}(t) N_{n-r,k-r}(t) dt \end{pmatrix},$$

we obtain the following matrix equation for the vector of control points $\mathbf{d} = (d_i)_i^n$:

$$\left(\mathbf{D}_0^r\right)^T \mathbf{A}_r \, \mathbf{D}_0^r \, \mathbf{d} = \mathbf{0}. \tag{3.6}$$

Writing $\mathbf{D}_0^r = \left(\alpha_{ij}^r\right)_{i,j=1}^{n-r,n}$, the product $\left(\mathbf{D}_0^r\right)^T \mathbf{A}_r \, \mathbf{D}_0^r$, which we call the minimisation matrix and denote $\mathbf{M}_r$, multiplies out to give us the following matrix:

$$\begin{pmatrix} \sum_{i=1}^{n-r}\sum_{j=1}^{n-r} \alpha_{i1}^r \alpha_{j1}^r \int N_{i,k-r}(t) N_{j,k-r}(t) dt & \dots & \dots & \sum_{i=1}^{n-r}\sum_{j=1}^{n-r} \alpha_{i1}^r \alpha_{jn}^r \int N_{i,k-r}(t) N_{j,k-r}(t) dt \\ \vdots & \ddots & & \vdots \\ \vdots & & \ddots & \vdots \\ \sum_{i=1}^{n-r}\sum_{j=1}^{n-r} \alpha_{in}^r \alpha_{j1}^r \int N_{i,k-r}(t) N_{j,k-r}(t) dt & \dots & \dots & \sum_{i=1}^{n-r}\sum_{j=1}^{n-r} \alpha_{in}^r \alpha_{jn}^r \int N_{i,k-r}(t) N_{j,k-r}(t) dt \end{pmatrix}.$$

We present an algorithm to compute exactly the symmetric banded matrix $\mathbf{A}_r$ for $0 \leq r \leq k-1$, where the integration extends over the limits $[t_1, t_2]$. After finding the rth derivative knot set from algorithm 2.2, we find the first and last basis functions that will contribute within the limits for the integration. Consecutive basis functions are then tested for possible intersection with each other and the segment overlap determined. B-spline representations for the intersecting parts of the basis functions are determined and they are then multiplied together using algorithm 2.7. Finally the product B-spline function is integrated over the limits using 2.35. Since the matrix $\mathbf{A}_r$ is symmetric we only need test approximately half the basis functions. To aid in the intersection testing we use a mathematical set representation for the knot set making up a particular basis function. Algorithm 3.1 performs this calculation and algorithm 3.2 computes the minimisation matrix $\mathbf{M}_r$.

With a few modifications to the above algorithm we can also treat the non-symmetrical case which is required for the surface product derivative minimisation. We define

$$\mathbf{A}_r^s = \int \left(\mathbf{N}^r \otimes \left(\mathbf{N}^s\right)^T \right) dt$$

$$= \begin{pmatrix} \int N_{1,k-r}(t) N_{1,k-s}(t) dt & \dots & \dots & \int N_{1,k-r}(t) N_{n-s,k-s}(t) dt \\ \vdots & \ddots & & \vdots \\ \vdots & & \ddots & \vdots \\ \int N_{n-r,k-r}(t) N_{1,k-s}(t) dt & \dots & \dots & \int N_{n-r,k-r}(t) N_{n-s,k-s}(t) dt \end{pmatrix},$$

and

$$\mathbf{M}_r^s = (\mathbf{D}_0^r)^T \mathbf{A}_r^s \mathbf{D}_0^s.$$

Algorithms 3.3 and 3.4 perform the exact computation of $\mathbf{A}_r^s$ and $\mathbf{M}_r^s$ respectively:

Algorithm 3.1: Computation of $\mathbf{A}_r$

1. *find the* `r`*th derivative knot set with* `nr` *elements,* $(\mathtt{t}_\mathtt{i})_{\mathtt{i}=1}^{\mathtt{nr}}$
2. *create a square matrix,* `mat,` *of size* `(nr-k-r, nr-k-r)`
3. *create a basis function set* $(\mathtt{N}_{\mathtt{i},\mathtt{k}-\mathtt{r}}(\mathtt{t}))_{\mathtt{i}=1}^{\mathtt{nr}-\mathtt{k}-\mathtt{r}}$, *from the knot set in 1*
4. *find index* `ind1` *of first basis function for which the last distinct knot is greater than* `t1`
5. *find index* `ind2` *of first basis function for which the last distinct knot is less than* `t2`
6. *for* `(i=ind1; i<=ind2-1; i++)`
   - *6.1 for* `(j=ind1; j<=i; j++)`
     - *6.1.1 create two sets* `s1, s2` *representing basis function knots from 3,*
     - *6.1.2 test for a possible intersection between* `s1` *and* `s2`
     - *6.1.3 if there is an intersection*
       - *6.1.3.1 form the intersection as a set,* `s3`
       - *6.1.3.2 if the size of* `s3 > 1`
         - *6.1.3.2.1 create the* `i` *and* `j` *basis functions as B-spline curves*
         - *6.1.3.2.2 form two composite Bezier curves from the overlapping segments*
         - *6.1.3.2.3 multiply the composite Bezier curves together*
         - *6.1.3.2.4 convert the result to a B-spline curve and integrate over* `(t1,t2)`
         - *6.1.3.2.5 store result in matrix mat from 2*
7. *fill in other half of matrix from symmetry*

Figure 3.1: Algorithm 3.1: Computation of the the matrix $\mathbf{A}_r$

Algorithm 3.2: Computation of $\mathbf{M}_r$

1. *create the matrix* $\mathtt{A}_\mathtt{r}$
2. *create the* `r`*th derivative matrix,* $\mathtt{D}_0^\mathtt{r}$, *alg 2.4*
3. *form* $\mathtt{M}_\mathtt{r} = \left(\mathtt{D}_0^\mathtt{r}\right)^\mathtt{T} \mathtt{A}_\mathtt{r}\, \mathtt{D}_0^\mathtt{r}$

Figure 3.2: Algorithm 3.2: Computation of the minimisation matrix $\mathbf{M}_r$

Algorithm 3.3: Computation of $\mathbf{A}_r^s$

1. *create the* `r`*th derivative knot set, size* `nr`, $(\mathtt{t}_\mathtt{i}^1)_{\mathtt{i}=1}^{\mathtt{nr}}$
2. *create the* `s`*th derivative knot set, size* `ns`, $(\mathtt{t}_\mathtt{i}^2)_{\mathtt{i}=1}^{\mathtt{ns}}$
3. *form a matrix,* `mat`, *of size* `(nr-k-r, ns-k-s)`
4. *create a basis function set* $(\mathtt{N}_{\mathtt{i},\mathtt{k}-\mathtt{r}}(\mathtt{t}))_{\mathtt{i}=1}^{\mathtt{nr}-\mathtt{k}-\mathtt{r}}$, *from knot set in 1*
5. *create a basis function set* $(\mathtt{N}_{\mathtt{i},\mathtt{k}-\mathtt{s}}(\mathtt{t}))_{\mathtt{i}=1}^{\mathtt{nr}-\mathtt{k}-\mathtt{s}}$, *from knot set in 2*
6. *find index* `ind1` *of first basis function for which the last distinct knot in 4 is greater than* `t1`
7. *find index* `ind2` *of first basis function for which the last distinct knot in 4 is less than* `t2`
8. *find index* `ind3` *of first basis function for which the last distinct knot in 5 is greater than* `t1`
9. *find index* `ind4` *of first basis function for which the last distinct knot in 5 is less than* `t2`
10. *sort the indices*
    if `(ind1 < ind3) i1 = ind1,` *else* `i1 = ind3`
    if `(ind2 > ind4) i2 = ind2,` *else* `i2 = ind4`
11. *for* `(i=i1; i<=i2-1; i++)`
    11.1 for `(j=i1; j<=i2-1; j++)`
    11.1.1 create two sets `s1, s2` *representing basis function knots from 4,5*
    11.1.2 test for a possible intersection between `s1` *and* `s2`
    11.1.3 if there is an intersection
    11.1.3.1 form the intersection set, `s3`
    11.1.3.2 if size of `s3>1`
    11.1.3.2.1 create the `i` *and* `j` *basis functions as B-spline curves*
    11.1.3.2.2 form two composite Bezier curves from the overlapping segments
    11.1.3.2.3 multiply the composite Bezier curves together
    11.1.3.2.4 convert the result to a B-spline curve and integrate over `(t1, t2)`
    11.1.3.2.5 store result in matrix `mat` *from 3*

Figure 3.3: Algorithm 3.3: Computation of the matrix $\mathbf{A}_r^s$

Algorithm 3.4: Computation of $\mathbf{M}_r^s$

1. *create the matrix* $\mathbf{A}_r^s$ *from algorithm 3.3*
2. *create the* `r`*th derivative matrix* $\mathbf{D}_0^r$ *from the original knot set, alg 2.4*
3. *create the* `s`*th derivative matrix* $\mathbf{D}_0^s$ *from the original knot set, alg 2.4*
4. *form the minimisation matrix* $\mathbf{M}_r^s = \left(\mathbf{D}_0^r\right)^T \mathbf{A}_r^s \, \mathbf{D}_0^s$

Figure 3.4: Algorithm 3.4: Computation of the non-symmetrical minimisation matrix $\mathbf{M}_r^s$

3.4 Surface Functional Minimisation

3.4.1 Minimisation of the zeroth derivative

For $x(u,v)$ a B-spline surface of order k by l on the knot set $(u_i)_{i=1}^{p+k} \times (v_j)_{j=1}^{q+l}$ and using $\partial/\partial\mathbf{d}$ to represent the collective differentiation of all the d_{ij}:

$$\frac{\partial}{\partial \mathbf{d}} = \begin{pmatrix} \frac{\partial}{\partial d_{11}} & \cdots & \frac{\partial}{\partial d_{1q}} \\ \vdots & \ddots & \vdots \\ \frac{\partial}{\partial d_{p1}} & \cdots & \frac{\partial}{\partial d_{pq}} \end{pmatrix},$$

we have

$$\frac{\partial}{\partial \mathbf{d}} \int\int \big(x(u,v)\big)^2 du\, dv = \frac{\partial}{\partial \mathbf{d}} \int\int \big(\mathbf{N}_u^T \, \mathbf{d} \, \mathbf{N}_v\big)^2 du\, dv = \int\int \frac{\partial}{\partial \mathbf{d}} \big(\mathbf{d} \odot_i \mathbf{N}_u \odot_j \mathbf{N}_v\big)^2.$$

The integrand is equal to (ignoring the factor 2)

$$\big(\mathbf{d} \odot_i \mathbf{N}_u \odot_j \mathbf{N}_v\big) \otimes \big(\mathbf{N}_u \otimes \mathbf{N}_v^T\big)$$

$$= \mathbf{N}_u \otimes \big(\mathbf{d} \odot_i \mathbf{N}_u \odot_j \mathbf{N}_v\big) \otimes \mathbf{N}_v^T$$

$$= \mathbf{N}_u \otimes \big(\mathbf{N}_u^T \odot_j \mathbf{d} \odot_j \mathbf{N}_v\big) \otimes \mathbf{N}_v^T$$

$$= \big(\mathbf{N}_u \otimes \mathbf{N}_u^T\big) \odot_j \mathbf{d} \odot_j \big(\mathbf{N}_v \otimes \mathbf{N}_v^T\big).$$

Hence we have the equation

$$\int \left(\mathbf{N}_u \otimes \mathbf{N}_u^T\right) du \odot_j \mathbf{d} \odot_j \int \left(\mathbf{N}_v \otimes \mathbf{N}_v^T\right) dv = \mathbf{0}.$$

Writing

$$\mathbf{A}_0^u = \int \left(\mathbf{N}_u \otimes \mathbf{N}_u^T\right) du = \begin{pmatrix} \int N_1(u)N_1(u)du & \dots & \dots & \int N_1(u)N_p(u)du \\ \vdots & \ddots & & \vdots \\ \vdots & & \ddots & \vdots \\ \int N_1(u)N_p(u)du & \dots & \dots & \int N_p(u)N_p(u)du \end{pmatrix},$$

$$\mathbf{A}_0^v = \int \left(\mathbf{N}_v \otimes \mathbf{N}_v^T\right) dv = \begin{pmatrix} \int N_1(v)N_1(v)dv & \dots & \dots & \int N_1(v)N_q(v)dv \\ \vdots & \ddots & & \vdots \\ \vdots & & \ddots & \vdots \\ \int N_1(v)N_q(v)dv & \dots & \dots & \int N_q(v)N_q(v)dv \end{pmatrix},$$

we have the following n by n system of equations for matrix of control points $\mathbf{d} = (d_{ij})_{i,j=1}^{p,q}$:

$$\mathbf{A}_0^u \, \mathbf{d} \mathbf{A}_0^v = \mathbf{0}.$$

3.4.2 Minimisation of higher derivatives

For the general derivative case with $a = a_1 + a_2$ we have:

$$\frac{\partial}{\partial \mathbf{d}} \int \left(\frac{\partial^a x}{\partial u^{a_1} \partial v^{a_2}}\right)^2 du\, dv = \frac{\partial}{\partial \mathbf{d}} \int \int \left(\left(\mathbf{N}_u^{a_1}\right)^T \mathbf{D}_u^{a_1} \, \mathbf{d} \left(\mathbf{D}_v^{a_2}\right)^T \mathbf{N}_v^{a_2}\right)^2 du\, dv$$

$$= \int \int \frac{\partial}{\partial \mathbf{d}} \left(\mathbf{d} \odot_i \left(\mathbf{D}_u^{a_1}\right)^T \mathbf{N}_u^{a_1} \odot_j \left(\mathbf{D}_v^{a_2}\right)^T \mathbf{N}_v^{a_2}\right)^2 du\, dv.$$

The integrand is equal to

$$\Big(\mathbf{d}\odot_i\big(\mathbf{D}_u^{a_1}\big)^T\mathbf{N}_u^{a_1}\odot_j\big(\mathbf{D}_v^{a_2}\big)^T\mathbf{N}_v^{a_2}\Big)\otimes\Big(\big(\mathbf{D}_u^{a_1}\big)^T\mathbf{N}_u^{a_1}\Big)\otimes\Big(\big(\mathbf{N}_v^{a_2}\big)^T\mathbf{D}_v^{a_2}\Big)$$

$$=\Big(\big(\mathbf{D}_u^{a_1}\big)^T\mathbf{N}_u^{a_1}\Big)\otimes\Big(\mathbf{d}\odot_i\big(\mathbf{D}_u^{a_1}\big)^T\mathbf{N}_u^{a_1}\odot_j\big(\mathbf{D}_v^{a_2}\big)^T\mathbf{N}_v^{a_2}\Big)\otimes\Big(\big(\mathbf{N}_v^{a_2}\big)^T\mathbf{D}_v^{a_2}\Big)$$

$$=\Big(\big(\mathbf{D}_u^{a_1}\big)^T\mathbf{N}_u^{a_1}\Big)\otimes\Big(\big(\mathbf{N}_u^{a_1}\big)^T\mathbf{D}_u^{a_1}\Big)\odot_j\mathbf{d}\odot_j\Big(\big(\mathbf{D}_v^{a_2}\big)^T\mathbf{N}_v^{a_2}\Big)\otimes\Big(\big(\mathbf{N}_v^{a_2}\big)^T\mathbf{D}_v^{a_2}\Big)$$

$$=\big(\mathbf{D}_u^{a_1}\big)^T\Big(\mathbf{N}_u^{a_1}\otimes\big(\mathbf{N}_u^{a_1}\big)^T\Big)\mathbf{D}_u^{a_1}\odot_j\mathbf{d}\odot_j\big(\mathbf{D}_v^{a_2}\big)^T\Big(\mathbf{N}_v^{a_2}\otimes\big(\mathbf{N}_v^{a_2}\big)^T\Big)\mathbf{D}_v^{a_2}.$$

Hence we have the equation

$$\big(\mathbf{D}_u^{a_1}\big)^T\Big(\int\mathbf{N}_u^{a_1}\otimes\big(\mathbf{N}_u^{a_1}\big)^T du\Big)\,\mathbf{D}_u^{a_1}\odot_j\mathbf{d}\odot_j\big(\mathbf{D}_v^{a_2}\big)^T\Big(\int\mathbf{N}_v^{a_2}\otimes\big(\mathbf{N}_v^{a_2}\big)^T dv\Big)\,\mathbf{D}_v^{a_2}=\mathbf{0}.$$

Writing

$$\mathbf{A}_r^u=\int\Big(\mathbf{N}_u^r\otimes\big(\mathbf{N}_u^r\big)^T\Big)\,du\,,\quad\mathbf{A}_s^v=\int\Big(\mathbf{N}_v^s\otimes\big(\mathbf{N}_v^s\big)^T\Big)dv,\tag{3.7}$$

we have the following matrix form for the control points $\mathbf{d}=(d_{ij})_{i,j=1}^{p,q}$:

$$\big((\mathbf{D}_u^{a_1})^T\,\mathbf{A}_{a_1}^u\,\mathbf{D}_u^{a_1}\big)\,\mathbf{d}\,\big((\mathbf{D}_v^{a_2})^T\mathbf{A}_{a_2}^v\mathbf{D}_v^{a_2}\big)=\mathbf{0}.\tag{3.8}$$

Finally, if we write

$$\mathbf{M}_r^u=(\mathbf{D}_u^r)^T\,\mathbf{A}_r^u\,\mathbf{D}_u^r,\quad\mathbf{M}_s^v=(\mathbf{D}_v^s)^T\mathbf{A}_s^v\mathbf{D}_v^s,$$

the matrix equation becomes

$$\mathbf{M}_{a_1}^u\,\mathbf{d}\,\mathbf{M}_{a_2}^v=\mathbf{0}.\tag{3.9}$$

3.4.3 Products of derivatives

For the minimisation of an expression involving the product of a u with a v derivative we have, for $a_1,a_2>0$:

$$\frac{\partial}{\partial\mathbf{d}}\int\int\frac{\partial^{a_1}x}{\partial u^{a_1}}\frac{\partial^{a_2}x}{\partial v^{a_2}}\,du\,dv=\int\int\frac{\partial}{\partial\mathbf{d}}\Big(\big(\mathbf{N}_u^{a_1}\big)^T\mathbf{D}_u^{a_1}\,\mathbf{d}\,\mathbf{N}_v\Big)\Big(\big(\mathbf{N}_u\big)^T\,\mathbf{d}\,\big(\mathbf{D}_v^{a_2}\big)^T\mathbf{N}_v^{a_2}\Big)\,du\,dv$$

$$=\int\int\frac{\partial}{\partial\mathbf{d}}\Big(\mathbf{d}\odot_i\big(\mathbf{D}_u^{a_1}\big)^T\mathbf{N}_u^{a_1}\odot_j\mathbf{N}_v\Big)\Big(\mathbf{d}\odot_i\mathbf{N}_u\odot_j\big(\mathbf{D}_v^{a_2}\big)^T\mathbf{N}_v^{a_2}\Big)\,du\,dv.$$

Using the product rule for integration the integrand is equal to

$$\begin{aligned}
&\Big(\mathbf{d}\odot_i \mathbf{N}_u \odot_j \left(\mathbf{D}_v^{a_2}\right)^T\mathbf{N}_v^{a_2}\Big)\otimes\Big(\left(\mathbf{D}_u^{a_1}\right)^T\mathbf{N}_u^{a_1}\otimes\mathbf{N}_v^T\Big)\\
&+\Big(\mathbf{d}\odot_i \left(\mathbf{D}_u^{a_1}\right)^T\mathbf{N}_u^{a_1}\odot_j \mathbf{N}_v\Big)\otimes\Big(\mathbf{N}_u\otimes\left(\mathbf{N}_v^{a_2}\right)^T\mathbf{D}_v^{a_2}\Big)\\
&\\
&=\left(\mathbf{D}_u^{a_1}\right)^T\mathbf{N}_u^{a_1}\otimes\Big(\mathbf{N}_u^T\odot_j\mathbf{d}\odot_j\left(\mathbf{D}_v^{a_2}\right)^T\mathbf{N}_v^{a_2}\Big)\otimes\mathbf{N}_v^T\\
&+\mathbf{N}_u\otimes\Big(\left(\mathbf{N}_u^{a_1}\right)^T\mathbf{D}_u^{a_1}\odot_j\mathbf{d}\odot_j\mathbf{N}_v\Big)\otimes\left(\mathbf{N}_v^{a_2}\right)^T\mathbf{D}_v^{a_2}\\
&\\
&=\left(\mathbf{D}_u^{a_1}\right)^T\left(\mathbf{N}_u^{a_1}\otimes\mathbf{N}_u^T\right)\odot_j\mathbf{d}\odot_j\left(\mathbf{D}_v^{a_2}\right)^T\left(\mathbf{N}_v^{a_2}\otimes\mathbf{N}_v^T\right)\\
&+\Big(\mathbf{N}_u\otimes\left(\mathbf{N}_u^{a_1}\right)^T\Big)\mathbf{D}_u^{a_1}\odot_j\mathbf{d}\odot_j\Big(\mathbf{N}_v\otimes\left(\mathbf{N}_v^{a_2}\right)^T\Big)\mathbf{D}_v^{a_2}.
\end{aligned}$$

Writing

$$\mathbf{A}_r^{0u}=\int\Big(\mathbf{N}_u^r\otimes\left(\mathbf{N}_u\right)^T\Big)\,du\,,\quad \mathbf{A}_s^{0v}=\int\Big(\mathbf{N}_v^s\otimes\left(\mathbf{N}_v\right)^T\Big)dv,$$

and using 3.7 we obtain the following matrix form for the control points $\mathbf{d}=(d_{ij})_{i,j=1}^{p,q}$:

$$(\mathbf{D}_u^{a_1})^T\,\mathbf{A}_{a_1}^{0u}\,\mathbf{d}\,(\mathbf{D}_v^{a_2})^T\mathbf{A}_{a_2}^{0v}+(\mathbf{A}_{a_1}^{0u})^T\,\mathbf{D}_u^{a_1}\,\mathbf{d}\,(\mathbf{A}_{a_2}^{0v})^T\,\mathbf{D}_v^{a_2}=\mathbf{0}. \tag{3.10}$$

Writing further

$$\mathbf{M}_r^{0u}=(\mathbf{D}_u^r)^T\mathbf{A}_r^{0u},\qquad \mathbf{M}_s^{0v}=(\mathbf{D}_v^s)^T\mathbf{A}_s^{0v},$$

we have the matrix equation

$$\mathbf{M}_{a_1}^{0u}\,\mathbf{d}\,\mathbf{M}_{a_2}^{0v}+\left(\mathbf{M}_{a_1}^{0u}\right)^T\mathbf{d}\left(\mathbf{M}_{a_2}^{0v}\right)^T=\mathbf{0}. \tag{3.11}$$

3.5 Volume Functional Minimisation

3.5.1 Minimisation of the zeroth derivative

For $x(u,v,w)$ a B-spline volume of order l,m,n on the knot set $(u_i)_{i=1}^{p+l}\times(v_j)_{j=1}^{q+m}\times(w_k)_{k=1}^{r+n}$ we have (using $\partial/\partial\mathbf{d}$ for the collective differentiation of all the d_{ijk}),

$$\frac{\partial}{\partial\mathbf{d}}\int\int\int x(u,v,w)^2du\,dv\,dw=\frac{\partial}{\partial\mathbf{d}}\int\int\int\Big(\mathbf{N}_u^T\,\mathbf{d}\,\mathbf{N}_v\,\mathbf{N}_w\Big)^2du\,dv\,dw=\mathbf{0}.$$

Using tensor notation this is equal to

$$\int\int\int \frac{\partial}{\partial \mathbf{d}}\Big(\mathbf{d} \odot_i \mathbf{N}_u \odot_j \mathbf{N}_v \odot_k \mathbf{N}_w\Big)^2 du\, dv\, dw$$

$$= 2\int\int\int \Big(\mathbf{d} \odot_i \mathbf{N}_u \ \odot_j \ \mathbf{N}_v \odot_k \mathbf{N}_w\Big) \otimes \Big(\mathbf{N}_u \otimes \mathbf{N}_v^T \otimes \mathbf{N}_w^T\Big) du\, dv\, dw.$$

The integrand is equal to

$$\mathbf{N}_u \otimes \Big(\mathbf{d} \odot_i \mathbf{N}_u \ \odot_j \ \mathbf{N}_v \odot_k \mathbf{N}_w\Big) \otimes \Big(\mathbf{N}_v^T \otimes \mathbf{N}_w^T\Big)$$

$$= \mathbf{N}_u \otimes \Big(\mathbf{N}_u^T \odot_i \mathbf{d} \odot_j \ \mathbf{N}_v \odot_k \mathbf{N}_w\Big) \otimes \Big(\mathbf{N}_v^T \otimes \mathbf{N}_w^T\Big)$$

$$= \Big(\mathbf{N}_u \otimes \mathbf{N}_u^T\Big) \odot_i \mathbf{d} \odot_j \ \mathbf{N}_v \odot_k \mathbf{N}_w \otimes \Big(\mathbf{N}_v^T \otimes \mathbf{N}_w^T\Big)$$

$$= \Big(\mathbf{N}_u \otimes \mathbf{N}_u^T\Big) \odot_i \mathbf{d} \odot_k \ \mathbf{N}_w \odot_j \mathbf{N}_v \otimes \Big(\mathbf{N}_v^T \otimes \mathbf{N}_w^T\Big)$$

$$= \Big(\mathbf{N}_u \otimes \mathbf{N}_u^T\Big) \odot_i \mathbf{d} \odot_k \ \mathbf{N}_w \odot_j \Big(\mathbf{N}_v \otimes \mathbf{N}_v^T\Big) \otimes \mathbf{N}_w^T$$

$$= \Big(\mathbf{N}_u \otimes \mathbf{N}_u^T\Big) \odot_i \mathbf{d} \odot_j \ \Big(\mathbf{N}_v \otimes \mathbf{N}_v^T\Big) \odot_k \Big(\mathbf{N}_w \otimes \mathbf{N}_w^T\Big).$$

Writing

$$\mathbf{A}_0^w = \int \Big(\mathbf{N}_w \otimes \mathbf{N}_w^T\Big)\, dw,$$

we have the equation

$$\mathbf{A}_0^u \ \odot_i \ \mathbf{d} \ \odot_j \ \mathbf{A}_0^v \ \odot_k \ \mathbf{A}_0^w = \mathbf{0}.$$

3.5.2 Minimisation of higher derivatives

For the derivative case, letting $a = a_1 + a_2 + a_3$, we look at the system generated by the following equation:

$$\frac{\partial}{\partial \mathbf{d}} \int\int\int \Big(\frac{\partial^a x}{\partial u^{a_1} \partial v^{a_2} \partial w^{a_3}}\Big)^2 du\, dv\, dw = \mathbf{0}.$$

Using tensor notation the left hand side is equal to

$$\int\int\int \frac{\partial}{\partial \mathbf{d}}\Big(\mathbf{d} \odot_i \left(\mathbf{D}_u^{a_1}\right)^T \mathbf{N}_u^{a_1} \odot_j \left(\mathbf{D}_v^{a_2}\right)^T \mathbf{N}_v^{a_2} \odot_k \left(\mathbf{D}_w^{a_3}\right)^T \mathbf{N}_w^{a_3}\Big)^2 du\, dv\, dw.$$

The integrand is equal to

$$\Big(\mathbf{d} \odot_i \left(\mathbf{D}_u^{a_1}\right)^T \mathbf{N}_u^{a_1} \odot_j \left(\mathbf{D}_v^{a_2}\right)^T \mathbf{N}_v^{a_2} \odot_k \left(\mathbf{D}_w^{a_3}\right)^T \mathbf{N}_w^{a_3}\Big) \otimes \Big(\left(\mathbf{D}_u^{a_1}\right)^T \mathbf{N}_u^{a_1}\Big) \otimes \Big((\mathbf{N}_v^{a_2})^T \mathbf{D}_v^{a_2}\Big) \otimes \Big((\mathbf{N}_w^{a_3})^T \mathbf{D}_w^{a_3}\Big)$$

$$= \Big(\left(\mathbf{D}_u^{a_1}\right)^T \mathbf{N}_u^{a_1}\Big) \otimes \Big(\mathbf{d} \odot_i \left(\mathbf{D}_u^{a_1}\right)^T \mathbf{N}_u^{a_1} \odot_j \left(\mathbf{D}_v^{a_2}\right)^T \mathbf{N}_v^{a_2} \odot_k \left(\mathbf{D}_w^{a_3}\right)^T \mathbf{N}_w^{a_3}\Big) \otimes \Big((\mathbf{N}_v^{a_2})^T \mathbf{D}_v^{a_2}\Big) \otimes \Big((\mathbf{N}_w^{a_3})^T \mathbf{D}_w^{a_3}\Big)$$

$$= \left(\mathbf{D}_u^{a_1}\right)^T \mathbf{N}_u^{a_1} \otimes \Big((\mathbf{N}_u^{a_1})^T \mathbf{D}_u^{a_1} \odot_i \mathbf{d} \odot_k \left(\mathbf{D}_w^{a_3}\right)^T \mathbf{N}_w^{a_3} \odot_j \left(\mathbf{D}_v^{a_2}\right)^T \mathbf{N}_v^{a_2}\Big) \otimes \Big((\mathbf{N}_v^{a_2})^T \mathbf{D}_v^{a_2}\Big) \otimes \Big((\mathbf{N}_w^{a_3})^T \mathbf{D}_w^{a_3}\Big)$$

$$= \Big(\left(\mathbf{D}_u^{a_1}\right)^T \mathbf{N}_u^{a_1} \otimes \left((\mathbf{N}_u^{a_1})^T \mathbf{D}_u^{a_1}\right)\Big) \odot_i \mathbf{d} \odot_k \Big(\left(\mathbf{D}_w^{a_3}\right)^T \mathbf{N}_w^{a_3}\Big) \odot_j \Big(\left(\mathbf{D}_v^{a_2}\right)^T \mathbf{N}_v^{a_2}\Big) \otimes \Big((\mathbf{N}_v^{a_2})^T \mathbf{D}_v^{a_2}\Big) \otimes \Big((\mathbf{N}_w^{a_3})^T \mathbf{D}_w^{a_3}\Big)$$

$$= \left(\mathbf{D}_u^{a_1}\right)^T \left(\mathbf{N}_u^{a_1} \otimes (\mathbf{N}_u^{a_1})^T\right) \mathbf{D}_u^{a_1} \odot_i \mathbf{d} \odot_j \left(\mathbf{D}_v^{a_2}\right)^T \left(\mathbf{N}_v^{a_2} \otimes (\mathbf{N}_v^{a_2})^T\right) \mathbf{D}_v^{a_2} \odot_k \left(\mathbf{D}_w^{a_3}\right)^T \left(\mathbf{N}_w^{a_3} \otimes (\mathbf{N}_w^{a_3})^T\right) \mathbf{D}_w^{a_3}. \tag{3.12}$$

Hence we have the following equation for the control points:

$$\left(\mathbf{D}_u^{a_1}\right)^T \int \left(\mathbf{N}_u^{a_1} \otimes (\mathbf{N}_u^{a_1})^T\right) du\ \mathbf{D}_u^{a_1} \odot_i \mathbf{d} \odot_j \left(\mathbf{D}_v^{a_2}\right)^T \int \left(\mathbf{N}_v^{a_2} \otimes (\mathbf{N}_v^{a_2})^T\right) dv\ \mathbf{D}_v^{a_2}$$

$$\odot_k \left(\mathbf{D}_w^{a_3}\right)^T \int \left(\mathbf{N}_w^{a_3} \otimes (\mathbf{N}_w^{a_3})^T\right) dw\ \mathbf{D}_w^{a_3} = \mathbf{0}.$$

or

$$\mathbf{M}_{a_1}^u \odot_i \mathbf{d} \odot_j \mathbf{M}_{a_2}^v \odot_k \mathbf{M}_{a_3}^w = \mathbf{0}. \tag{3.13}$$

3.5.3 Products of derivatives

For $a_1, a_2, a_3 > 0$, and by analogy with the surface case, we look to minimise expressions involving products of derivatives in pairs. For the uv product:

$$\frac{\partial}{\partial \mathbf{d}} \int\int\int \frac{\partial^{a_1} x}{\partial u^{a_1}} \frac{\partial^{a_2} x}{\partial v^{a_2}}\, du\, dv\, dw$$

$$= \int\int\int \frac{\partial}{\partial \mathbf{d}} \Big(\left(\mathbf{N}_u^{a_1}\right)^T \mathbf{D}_u^{a_1}\, \mathbf{d}\, \mathbf{N}_v \mathbf{N}_w \Big) \Big(\left(\mathbf{N}_u\right)^T \mathbf{d}\, \left(\mathbf{D}_v^{a_2}\right)^T \mathbf{N}_v^{a_2} \mathbf{N}_w \Big)\, du\, dv\, dw$$

$$= \int\int\int \frac{\partial}{\partial \mathbf{d}} \Big(\mathbf{d} \odot_i \left(\mathbf{D}_u^{a_1}\right)^T \mathbf{N}_u^{a_1} \odot_j \mathbf{N}_v \odot_k \mathbf{N}_w \Big) \Big(\mathbf{d} \odot_i \mathbf{N}_u \odot_j \left(\mathbf{D}_v^{a_2}\right)^T \mathbf{N}_v^{a_2} \odot_k \mathbf{N}_w \Big)\, du\, dv\, dw.$$

Using the product rule the integrand is equal to

$$\begin{aligned}
&\Big(\mathbf{d} \odot_i \mathbf{N}_u \odot_j \left(\mathbf{D}_v^{a_2}\right)^T \mathbf{N}_v^{a_2} \odot_k \mathbf{N}_w \Big) \otimes \Big(\left(\mathbf{D}_u^{a_1}\right)^T \mathbf{N}_u^{a_1} \otimes \mathbf{N}_v^T \otimes \mathbf{N}_w^T \Big) \\
&+ \Big(\mathbf{d} \odot_i \left(\mathbf{D}_u^{a_1}\right)^T \mathbf{N}_u^{a_1} \odot_j \mathbf{N}_v \odot_k \mathbf{N}_w \Big) \otimes \Big(\mathbf{N}_u \otimes \left(\mathbf{N}_v^{a_2}\right)^T \mathbf{D}_v^{a_2} \otimes \mathbf{N}_w \Big)
\end{aligned}$$

$$\begin{aligned}
&= \left(\mathbf{D}_u^{a_1}\right)^T \mathbf{N}_u^{a_1} \otimes \Big(\mathbf{N}_u^T \odot_i \mathbf{d} \odot_j \left(\mathbf{D}_v^{a_2}\right)^T \mathbf{N}_v^{a_2} \odot_k \mathbf{N}_w \Big) \otimes \mathbf{N}_v^T \otimes \mathbf{N}_w^T \\
&+ \mathbf{N}_u \otimes \Big(\left(\mathbf{N}_u^{a_1}\right)^T \mathbf{D}_u^{a_1} \odot_i \mathbf{d} \odot_k \mathbf{N}_w \odot_j \mathbf{N}_v \Big) \otimes \left(\mathbf{N}_v^{a_2}\right)^T \mathbf{D}_v^{a_2} \otimes \mathbf{N}_w
\end{aligned}$$

$$\begin{aligned}
&= \left(\mathbf{D}_u^{a_1}\right)^T \left(\mathbf{N}_u^{a_1} \otimes \mathbf{N}_u^T\right) \odot_i \mathbf{d} \odot_j \left(\mathbf{D}_v^{a_2}\right)^T \left(\mathbf{N}_v^{a_2} \otimes \mathbf{N}_v^T\right) \odot_k \left(\mathbf{N}_w \otimes \mathbf{N}_w^T\right) \\
&+ \Big(\mathbf{N}_u \otimes \left(\mathbf{N}_u^{a_1}\right)^T \Big) \mathbf{D}_u^{a_1} \odot_i \mathbf{d} \odot_j \Big(\mathbf{N}_v \otimes \left(\mathbf{N}_v^{a_2}\right)^T \Big) \mathbf{D}_v^{a_2} \odot_k \Big(\mathbf{N}_w \otimes \mathbf{N}_w^T \Big).
\end{aligned}$$

This gives the following matrix3D equation for the control points $\mathbf{d} = (d_{ijk})_{i,j,k=1}^{p,q,r}$:

$$\mathbf{M}_{a_1}^{0u} \odot_i \mathbf{d} \odot_j \mathbf{M}_{a_2}^{0v} \odot_k \mathbf{M}_0^w + (\mathbf{M}_{a_1}^{0u})^T \odot_i \mathbf{d} \odot_j (\mathbf{M}_{a_2}^{0v})^T \odot_k \mathbf{M}_0^w = \mathbf{0}. \tag{3.14}$$

For the vw derivative product we proceed similarly by expanding

$$\frac{\partial}{\partial \mathbf{d}} \int\int\int \frac{\partial^{a_2} x}{\partial v^{a_2}} \frac{\partial^{a_3} x}{\partial w^{a_3}}\, du\, dv\, dw.$$

This is equal to

$$\int\int\int \frac{\partial}{\partial \mathbf{d}} \Big(\mathbf{d} \odot_i \mathbf{N}_u \odot_j \left(\mathbf{D}_v^{a_2}\right)^T \mathbf{N}_v^{a_2} \odot_k \mathbf{N}_w \Big) \Big(\mathbf{d} \odot_i \mathbf{N}_u \odot_j \mathbf{N}_v \odot_k \left(\mathbf{D}_w^{a_3}\right)^T \mathbf{N}_w^{a_3} \Big)\, du\, dv\, dw,$$

which, by defining,

$$\mathbf{A}_t^{0w} = \int \Big(\mathbf{N}_w^t \otimes \big(\mathbf{N}_w\big)^T \Big) dw, \qquad \mathbf{M}_t^{0w} = \mathbf{D}_w^t \mathbf{A}_t^{0w},$$

gives us the following matrix3D equation

$$\mathbf{M}_0^u \odot_i \mathbf{d} \odot_j \mathbf{M}_{a_2}^{0v} \odot_k \mathbf{M}_{a_3}^{0w} + \mathbf{M}_0^u \odot_i \mathbf{d} \odot_j (\mathbf{M}_{a_2}^{0v})^T \odot_k (\mathbf{M}_{a_3}^{0w})^T = \mathbf{0}. \tag{3.15}$$

Finally, for uw derivative product we have

$$\frac{\partial}{\partial \mathbf{d}} \int\int\int \frac{\partial^{a_1} x}{\partial u^{a_1}} \frac{\partial^{a_3} x}{\partial w^{a_3}} \, du \, dv \, dw$$

$$= \int\int\int \frac{\partial}{\partial \mathbf{d}} \Big(\mathbf{d} \odot_i \big(\mathbf{D}_u^{a_1}\big)^T \mathbf{N}_u^{a_1} \odot_j \mathbf{N}_v \odot_k \mathbf{N}_w \Big) \Big(\mathbf{d} \odot_i \mathbf{N}_u \odot_j \mathbf{N}_v \odot_k \big(\mathbf{D}_w^{a_3}\big)^T \mathbf{N}_w^{a_3} \Big) \, du \, dv \, dw,$$

which, in an analogous fashion, produces the equation

$$\mathbf{M}_{a_1}^{0u} \odot_i \mathbf{d} \odot_j \mathbf{M}_0^v \odot_k \mathbf{M}_{a_3}^{0w} + (\mathbf{M}_{a_1}^{0u})^T \odot_i \mathbf{d} \odot_j \mathbf{M}_0^v \odot_k (\mathbf{M}_{a_3}^{0w})^T = \mathbf{0}. \tag{3.16}$$

Having obtained explicit expressions for the minimisation of these quadratic B-spline functionals there remains two other important aspects of a given variational problem that need to dealt with. The first concerns the geometric boundary conditions which must be translated into constraint equations involving the control points. These equations can then be used to produce a minimal set of independent variables from the linear system generated by the minimisation of the functional in question prior to solving. The second concerns the computation of the source terms which involve the integral of a product function.

3.6 Boundary Conditions and Constraint Handling

When solving either the differential equation or minimising the corresponding functional, boundary conditions have to be taken into account. There are three basic types that are commonly dealt with

- geometric boundary conditions (also called essential or forced)
- free or natural boundary conditions

- mixed geometric and free conditions

The first two are called Dirichlet and Neumann conditions respectively. The Dirichlet conditions require that the solution match a given function on part or all of the boundary of the domain under consideration. In the variational form of the problem these conditions are constraints that have to be satisfied when constructing a solution. The Neumann conditions require that the directional derivative along the outward normal to the boundary match a given function on all or part of the boundary. For the variational form of the problem the Neumann conditions are satisfied automatically. The mixed case specifies that some linear combination of the solution and its normal derivative match a given function on all or part of the boundary. In general only one of the three types of condition will be specified on a given portion of the boundary.

There are a number of techniques that can be used to handle the geometric boundary conditions on a given variational problem. General geometric constraints can be imposed by augmenting the original functional with Lagrange multipliers. This method changes a constrained minimisation problem into an unconstrained one such that the solution to the new problem is also a solution to the original one. For each constraint a number, called a Lagrange multiplier, is associated with the constraint. The mth Lagrange multiplier in a system is the correction needed to satisfy the mth constraint. To solve for the multipliers, the original equation is transformed into a new form called the Lagrangian. Minimising this new function gives the multipliers as a set of linear equations. These are substituted back into the Lagrangian to obtain the solution vector to the original problem. Although this general technique is often employed, its drawbacks are that the resulting equations often lose the linearity and complicate the problem.

The Penalty method also eliminates most or all of the original constraints. Here a quadratic energy term is added to the original equation and acts as the penalty. When a particular constraint is violated, the method penalises the violation by adding energy into the system and guiding it back to a valid state. The severity of the penalty is controlled by a parameter whose magnitude determines how much of this energy is added by the penalty term. As the severity of the penalty increases the accuracy of the method improves. However, although this is a simple and generally fast approach to dealing with the constraints, such methods are not as accurate as other techniques for solving constrained minimisation problems.

If we restrict the class of constraints by considering only those composed of linear combinations of the degrees of freedom, it is possible to find a solution by using the so-called reduced transformation technique. This is a technique used to enforce linear equality constraints when solving general quadratic minimisation problems and in view of the convenient property of B-spline boundary point and derivative interpolation (see figures 2.12 and 2.14) this is the solution

adopted here and in the following chapters. The details are summarised in the following section.

3.6.1 The reduced transformation technique

We assume that we have a linear system resulting from the functional minimisation and consisting of a vector $\mathbf{d}$ containing n variables. We assume also that the geometric equality constraints are expressed in the form

$$h_k(\mathbf{d}) = 0.0, \quad k = 1, \ldots, m,$$

where the h_k consist of linear combinations of the variables. These can be written in matrix form as $\mathbf{Ad} = \mathbf{b}$ for some matrix $m \times n$ matrix $\mathbf{A}$ and vector $\mathbf{b}$ of size m. Each linear constraint reduces the problem dimension by one. We select the independent variables and represent the dependent variables by the selected independent variables. By premultiplying a permutation matrix we rearrange $\mathbf{d}$ so that the dependent and independent variables are separate. To illustrate this we take the example of a system consisting of five degree of freedom $(x_1 \ldots x_5)$ subject to the following two constraints:

$$\begin{aligned} x_1 + 2x_2 - x_4 &= 3 \\ x_3 + 2x_4 &= 5, \end{aligned}$$

which can be written as $\mathbf{Ad} = \mathbf{B}$:

$$\mathbf{A} = \begin{pmatrix} 1 & 2 & 0 & -1 & 0 \\ 0 & 0 & 1 & 2 & 0 \end{pmatrix}, \qquad \mathbf{b} = \begin{pmatrix} 3 \\ 5 \end{pmatrix}.$$

Taking the dependent variables as x_1, x_3 we construct a permutation matrix $\mathbf{P}$ such that $\mathbf{Pd} = \left[\mathbf{d}_{\text{indep}} \mid \mathbf{d}_{\text{dep}}\right]^T$:

$$\begin{pmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{pmatrix} = \begin{pmatrix} x_2 \\ x_4 \\ x_5 \\ x_1 \\ x_3 \end{pmatrix},$$

and

$$\mathbf{A}_1 \mathbf{d}_{\text{indep}} + \mathbf{A}_2 \mathbf{d}_{\text{dep}} = \mathbf{b},$$

where $\mathbf{A}_1$ is a matrix of size m by $n - m$ and matrix $\mathbf{A}_2$ m by m:

$$\begin{pmatrix} 2 & -1 & 0 \\ 0 & 2 & 0 \end{pmatrix} \begin{pmatrix} x_2 \\ x_4 \\ x_5 \end{pmatrix} + \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_3 \end{pmatrix} = \begin{pmatrix} 3 \\ 5 \end{pmatrix}.$$

Here $\mathbf{d}_{\text{indep}}$ is the reduced column vector containing the $n-m$ independent variables and $\mathbf{d}_{\text{dep}}$ is a column vector of dimension m containing the dependent variables. We can express the dependent variables by the independent variables as:

$$\mathbf{d}_{\text{dep}} = \mathbf{A}_2^{-1}\mathbf{b} - \mathbf{A}_2^{-1}\mathbf{A}_1\mathbf{d}_{\text{indep}} = \mathbf{D}_0 + \mathbf{D}_1\mathbf{d}_{\text{indep}},$$

where

$$\mathbf{D}_0 = \begin{pmatrix} 3 \\ 5 \end{pmatrix}, \quad \mathbf{D}_1 = \begin{pmatrix} -2 & 1 & 0 \\ 0 & -2 & 0 \end{pmatrix},$$

and

$$\begin{pmatrix} x_1 \\ x_3 \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}\begin{pmatrix} 3 \\ 5 \end{pmatrix} + \begin{pmatrix} -2 & 1 & 0 \\ 0 & -2 & 0 \end{pmatrix}\begin{pmatrix} x_2 \\ x_4 \\ x_5 \end{pmatrix}.$$

We also have

$$\mathbf{d} = \mathbf{P}^{-1}\big[\mathbf{d}_{\text{indep}} \mid \mathbf{d}_{\text{dep}}\big]^T = \mathbf{P}^{-1}\Big(\big[\mathbf{I} \mid \mathbf{D}_1\big]\mathbf{d}_{\text{indep}} + \big[\mathbf{Z} \mid \mathbf{D}_0\big]^T\Big) = \mathbf{D}_3\mathbf{d}_{\text{indep}} + \mathbf{D}_2, \tag{3.17}$$

where $\mathbf{I}$ is the identity matrix of size $n-m$ and $\mathbf{Z}$ is the null vector of size $n-m$. Then

$$\begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{pmatrix} = \begin{pmatrix} 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{pmatrix}\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ -2 & 1 & 0 \\ 0 & -2 & 0 \end{pmatrix}\begin{pmatrix} x_2 \\ x_4 \\ x_5 \end{pmatrix} + \begin{pmatrix} 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{pmatrix}\begin{pmatrix} 0 \\ 0 \\ 0 \\ 3 \\ 5 \end{pmatrix},$$

and hence

$$\mathbf{D}_3 = \begin{pmatrix} -2 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & -2 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \quad \mathbf{D}_2 = \begin{pmatrix} 3 \\ 0 \\ 5 \\ 0 \\ 0 \end{pmatrix}.$$

In general by substituting equation 3.17 into a standard quadratic minimisation problem of the form $I = \frac{1}{2}\mathbf{d}^T\mathbf{K}\mathbf{d} - \mathbf{F}^T\mathbf{d}$ we obtain

$$\begin{aligned} &\frac{1}{2}\big(\mathbf{D}_3\mathbf{d}_{\text{indep}} + \mathbf{D}_2\big)^T\mathbf{K}\big(\mathbf{D}_3\mathbf{d}_{\text{indep}} + \mathbf{D}_2\big) - \mathbf{F}^T\big(\mathbf{D}_3\mathbf{d}_{\text{indep}} + \mathbf{D}_2\big) \\ &= \frac{1}{2}\Big(\big(\mathbf{D}_3\mathbf{d}_{\text{indep}}\big)^T\mathbf{K}\mathbf{D}_3\mathbf{d}_{\text{indep}} + \big(\mathbf{D}_3\mathbf{d}_{\text{indep}}\big)^T\mathbf{K}\mathbf{D}_2 + \mathbf{D}_2^T\mathbf{K}\big(\mathbf{D}_3\mathbf{d}_{\text{indep}}\big) + \mathbf{D}_2^T\mathbf{K}\mathbf{D}_2\Big) \\ &- \mathbf{F}^T\mathbf{D}_3\mathbf{d}_{\text{indep}} - \mathbf{F}^T\mathbf{D}_2. \end{aligned} \tag{3.18}$$

Using the fact that $(\partial/\partial\mathbf{d})(\mathbf{d}^T\mathbf{K}\mathbf{d}) = 2\mathbf{K}\mathbf{d}$ we have

$$\frac{\partial}{\partial \mathbf{d}_{\text{indep}}} I = \mathbf{D}_3^T\mathbf{K}\mathbf{D}_3\mathbf{d}_{\text{indep}} + \frac{1}{2}\mathbf{D}_3^T\mathbf{K}\mathbf{D}_2 + \frac{1}{2}\mathbf{D}_2^T\mathbf{K}\mathbf{D}_3 - \mathbf{F}^T\mathbf{D}_3 = \big(\mathbf{D}_3\mathbf{K}\mathbf{D}_3^T\big)\mathbf{d}_{\text{indep}} + \mathbf{D}_3^T\mathbf{K}\mathbf{D}_2 - \mathbf{D}_3^T\mathbf{F}.$$

Hence, equating to zero we get

$$\big(\mathbf{D}_3\mathbf{K}\mathbf{D}_3^T\big)\mathbf{d}_{\text{indep}} = \mathbf{D}_3^T\mathbf{K}\mathbf{D}_2 - \mathbf{D}_3^T\mathbf{F}. \tag{3.19}$$

This automatically satisfies the linear constraints while the solution is found that minimises the quadratic function. Having found the vector $\mathbf{d}_{\text{indep}}$, we use the elimination information to find the vector $\mathbf{d}_{\text{dep}}$ and hence build the complete solution.

3.7 Source and Boundary Term Integration

To cope with the source and boundary integration terms we make two simplifying assumptions about the geometric shape of the domain and the form of the source term. Firstly we restrict ourselves to dealing with rectangular domains and secondly we assume that the source term is in B-spline form or can be converted to such a form either exactly (i.e. if it polynomial in nature) or approximately (if it is some other non-polynomial function). Under these assumptions we can compute exact results using the product rule for integration in its various forms for one, two and three dimensional domains. A comparison of techniques for performing this calculation numerically is given in [86]. Without loss of generality in the following algorithms we restrict ourselves to dealing with the basis function terms for the B-splines appearing in the product formulae.

3.7.1 Curve product

Using the explicit formula for the integral of a B-spline in equation 2.34, the one dimensional integration by parts formula for a product of a B-spline function of order k representing our solution

$$f = \sum_{i=1}^{n} d_i N_{i,k}(t), \quad \text{knot set} \quad (t_i)_{i=1}^{n+k}$$

and a multiplying B-spline function g of order k_1 say, becomes, (focusing in on a single basis function of f),

$$\int_{t1}^{t2} N_{i,k}(t)g(t)dt = \Big[g(t)\int N_{i,k}(t)\Big]_{t1}^{t2} - \int_{t1}^{t2}\Big(g'(t)\int N_{i,k}(t)\Big)dt.$$

By applying this result iteratively $k_1 - 1$ times so reducing the degree of g down to 1 we obtain the following algorithm for the exact integral of the product.

Algorithm 3.5: To integrate a curve product B-spline:
$\int_{t1}^{t2} N_{i,k}(t)g(t)dt$

1. *Create a vector* `vec` *of size* `n` *to store the result*
2. *for* `(i=0; i<n; i++)`
   - *2.1 for* `(j=1; j<=k1; j++)`
     - *2.1.1 create B-spline representation* `c1` *of* `i`*th basis function of* `f`
     - *2.1.2 integrate* `c1` *to the level* `j` *as a B-spline curve* `c2` *(2.34)*
     - *2.1.3 determine limits* `(x1,x2)` *of* `c2`
     - *2.1.4 form the* `(j-1)`*th derivative* `c3` *of the curve* `g` *(alg 2.6)*
     - *2.1.5 compute* `vec[i] += (-1)^(j-1) * c2(x2) * c3(x2) - c2(x1) * c3(x1)`

Figure 3.5: Algorithm 3.5: Integral of a product B-spline

3.7.2 Surface product

For the surface case the corresponding formula for integrating a product is Green's formula:

$$\int\int_D \psi \frac{\partial \phi}{\partial x} dx\, dy = -\int\int_D \frac{\partial \psi}{\partial x}\, \phi dx\, dy + \int_{y_1}^{y_2} \psi\phi \Big]_{x_l}^{x_r} dy \tag{3.20}$$

By using this formula and the explicit formula for the integral of a B-spline, 2.42, the two dimensional integration by parts algorithm for a product of a function representing our solution

$$f = \sum_{i=1}^{p}\sum_{j=1}^{q} d_{ij} N_{i,k}(u) N_{j,l}(v), \quad (u_i)_{i=1}^{p+k} \times (v_j)_{j=1}^{q+l},$$

and a multiplying B-spline function g of order (k_1, l_1) becomes:

Algorithm 3.6: To integrate a surface product B-spline:
$\int_{u1}^{u2}\int_{v1}^{v2} N_{i,k}(u)N_{j,l}(v)g(u,v)du\,dv$

1. *Create matrix* `mat` *of size* `p,q` *to store the result*
2. *for* `(i=0; i<p; i++)`
   - *2.1 for* `(j=0; j<q; j++)`
     - *2.1.1 for* `(r=1; r<=k1; r++)`
       - *2.1.1.1 create B-spline surface representation* `s1` *of* `(i,j)`*th basis function*
       - *2.1.1.2 integrate* `s1` *in* `u` *to the level* `l` *as a B-spline surface,* `s2`
       - *2.1.1.3 determine limits* `(u1,u2,v1,v2)` *of* `s2`
       - *2.1.1.4 form the* `(r-1)`*th derivative of* `g, s3`
       - *2.1.1.5 subdivide* `s3` *over the limits* `(u1,u2,v1,v2), s4`
       - *2.1.1.6 create the* `u` *isoparametric curve* `c1` *from* `s2` *at* `u1`
       - *2.1.1.7 create the* `u` *isoparametric curve* `c2` *from* `s4` *at* `u1`
       - *2.1.1.8 create the* `u` *isoparametric curve* `c3` *from* `s2` *at* `u2`
       - *2.1.1.9 create the* `u` *isoparametric curve* `c4` *from* `s4` *at* `u2`
       - *2.1.1.10 form the product curves,* `p1=prod(c1,c2), p2=prod(c3,c4), p3 = p2 - p1`
       - *2.1.1.11 compute* `mat[i][j] +=` $(-1)^{r-1}*$ *integral of* `p3` *over* `(v1,v2)`

Figure 3.6: Algorithm 3.6: Integral of a surface B-spline product

3.7.3 Volume product

Finally for the volume case the corresponding product formula is

$$\int\int\int_V \psi\frac{\partial\phi}{\partial x}dx\,dy\,dz = -\int\int\int_V \frac{\partial\psi}{\partial x}\phi\,dx\,dy\,dz + \int\int \psi\phi\Big]_{x_l}^{x_r} dy\,dz.$$

Using the explicit formula for the integral of a B-spline volume, equation 2.46, the three dimensional integration by parts algorithm for a product of a B-spline function of order (l,m,n) representing our solution

$$f = \sum_{i=1}^{p}\sum_{j=1}^{q}\sum_{k=1}^{r} d_{ijk}N_{i,l}(u)N_{j,m}(v)N_{k,n}(w), \quad (u_i)_{i=1}^{p+l} \times (v_j)_{j=1}^{q+m} \times (w_k)_{k=1}^{r+n}$$

and a multiplying B-spline function g of order (l_1,m_1,n_1) becomes

Algorithm 3.7: To integrate a volume product B-spline:
$\int_{u1}^{u2}\int_{v1}^{v2}\int_{w1}^{w2} N_{i,l}(u)N_{j,m}(v)N_{k,n}(w)g(u,v,w)du\,dv\,dw$

1. Create matrix3D `mat` *of size* $\mathtt{p},\mathtt{q},\mathtt{r}$ *to store the result*
2. for `(k=0; k<r; k++)`
  2.1 for `(i=0; i<p; i++)`
    2.1.1 for `(j=0; j<q; j++)`
      2.1.1.1 for $(\mathtt{s}=1;\ \mathtt{s}<\ \mathtt{l}_1;\ \mathtt{l}{+}{+})$
      2.1.1.1.1 create B-spline volume representation `V1` *of* $(\mathtt{i},\mathtt{j},\mathtt{k})$ *basis function*
      2.1.1.1.2 integrate `V1` *in* `u` *to the level* `k` *as a B-spline volume,* `V2`
      2.1.1.1.3 determine limits $(\mathtt{u1},\mathtt{u2},\mathtt{v1},\mathtt{v2},\mathtt{w1},\mathtt{w2})$ *of* `v2`
      2.1.1.1.4 form the $(\mathtt{s}-1)$*th derivative of* `g,` *volume* `V3`
      2.1.1.1.5 subdivide `v3` *over the limits* $(\mathtt{u1},\mathtt{u2},\mathtt{v1},\mathtt{v2},\mathtt{w1},\mathtt{w2})$ *volume* `V4`
      2.1.1.1.6 create the `u` *isoparametric surface* `s1` *from* `V2` *at* `u1`
      2.1.1.1.7 create the `u` *isoparametric surface* `s2` *from* `V4` *at* `u1`
      2.1.1.1.8 create the `u` *isoparametric surface* `s3` *from* `V2` *at* `u2`
      2.1.1.1.9 create the `u` *isoparametric surface* `s4` *from* `V4` *at* `u2`
      2.1.1.1.10 form the product surfaces, `p1=prod(s1,s2), p2=prod(s3,s4), p3=p2-p1`
      2.1.1.1.12 compute `mat[k][i][j] +=` $(-1)^{\mathtt{s}-1}$**integral of* `p3` *over* $(\mathtt{v1},\mathtt{v2},\mathtt{w1},\mathtt{w2})$

Figure 3.7: Algorithm 3.7: Integration of a B-spline product volume

3.8 Summary

The equivalence between the statement of a continuum problem in terms of a differential equation and its corresponding variational form allows us to concentrate on solving a given problem by minimising a certain functional subject to specified geometric and/or natural boundary conditions. By seeking a solution in B-spline form and using tensor notation we have developed explicit formulae and corresponding algorithms for the exact minimisation of general quadratic functionals in curve, surface and volume form. The convenient point and derivative boundary properties of B-spline entities allows us to treat the restricted class of geometric boundary conditions involving linear point and derivative expressions in a relatively simple manner. By using the reduced transformation technique we are able to take the constraint equations involving the

control points and generate a minimal linearly independent set of free variables prior to solving. These are then used to build the solution. By assuming the source terms are in B-spline form and the domain is rectangular we have also presented algorithms for computing exactly the product integral terms appearing in the variational form. In the following chapter we put these elements to use in producing B-spline solutions to some problems in the field of geometric smoothing.

Chapter 4

Applications to Geometric Smoothing

4.1 Introduction and Background

One of the principal motivations for studying smoothness and looking for algorithms that can compute fair[1] curves and surfaces originates from the fact that even when very efficient schemes based on splines are used for construction the resulting curve or surface is often not fair enough and has extraneous bumps and wiggles. In this respect energy-based minimisation algorithms have been used by researchers to incrementally improve the fairness of a shape by tuning parameters of the model. The motivation for this approach to solving the problem comes from the following.

In the design of free-form plane curves, the quantity

$$E = \int \kappa^2 ds \tag{4.1}$$

where κ is the curvature[2] as a function of arc length s, is often invoked as a measure of the fairness of loci that satisfy given constraints. The theory of linear splines deals with loci interpolating data points with a given order of continuity and minimising an approximation to E. Under appropriate conditions, E is proportional to the work done in bending a thin elastic beam so as to assume the shape characterised by the curvature profile $\kappa(s)$. This integral represents the bending energy of such a beam under the following conditions:

- the material is homogeneous and obey's Hooke's law, i.e a linear relation between stress and strain at each point,

[1] We use the terms smooth/fair and their derivatives synonymously in this chapter

[2] We use curvature terms in this chapter without giving formal definitions. Appendix F summarises the main formulae for the curve, surface and volume cases

- the beam has a constant cross-sectional shape, with a symmetry axis that lies in the plane of bending,
- transverse plane sections of the beam remain plane upon bending.

Under these conditions the bending moment due to internal stresses across each section is proportional to κ and E measures the strain energy stored in the bent beam. Determining the shape of a bent beam subject to given constraints can thus be formulated as a variational problem in terms of minimising a functional. Curves that realise global minima of E possibly under various constraints, points/tangents, arc length are sought. In most fairing algorithms the integral in 4.1, which represent a geometric measure, is linearised by the assumption that the actual parameter t of the spline curve $\mathbf{f}$ nearly represents the arc length, meaning that $\|\mathbf{x}'(t)\|$ is nearly constant. This gives the simpler but parameter dependent integral measure

$$\int_a^b \left\|\mathbf{f}^{(2)}(t)\right\|^2 dt.$$

Fairness then is a quantifiable property of the geometric quality in a curve or surface. It can be described by a numerical fairness measure and the strain energy is an example of such a criterion. To test the efficacy of a given smoothing algorithm one normally formulates a measure of the 'goodness' at each curve/surface point and then integrates this measure over all or part of the entity in question to get a single number which characterises the desirability of the surface shape under that metric. Using a variational based energy minimisation algorithm or similar we seek shapes that optimise this quantity. If geometric constraints such as interpolation need to be satisfied then we have a case of constrained optimisation. A suitable smoothing algorithm will minimise an energy functional subject to the constraints. Without constraints the curve or surface will simply be smoothed.

In this chapter we provide a review of the main contributions to the field of curve and surface smoothing and the principles upon which they are based and present a number of new fairing algorithms both integrated with and separate from data approximation using least squares. In both cases, we employ the exact B-spline functional minimisation procedures from Chapter 3. We use a number of integral measures to demonstrate the utility of these algorithms in removing unwanted irregularities as well as curvature/environment maps for a visual representation of the quality of the resulting curves and surfaces. We then look at the volume case and after presenting a brief review of the use of volumetric models derive a generalisation of the curve/surface algorithms and the associated integral measures to the smoothing of a B-spline volume entity in

both functional and parametric form. In each of these cases we concentrate on the smoothing algorithm itself and the quality of the results it generates rather than the imposition of constraints. However, in principle, and without significant complication, point/tangent and other linear interpolatory constraints can be integrated with the algorithms developed here using the reduced transformation technique of Chapter 3.

4.2 Curve Smoothing

In general to avoid unfair effects, which can originate from, for example, digitising errors, two broad approaches are used in general. The first one consists of incorporating the fairness criterion already into the interpolation or approximation process through the minimisation of energy based functionals. The second consists of fitting first to a given precision using an interpolation/approximation technique and then smoothing the resulting curve by some iterative fairing process (which may again use functional minimisation).

Figure 4.1 illustrates the elements for the first technique. A given construction process, for example least squares, is combined with a smoothing functional that represents an energy terms in analogy with material mechanics. By minimising the combined expression we, in principle, obtain curves that are both accurate in terms of closeness to the original data and smooth in terms of curvature variation. In practice there is a fundamental contradiction between the distance least squares error term and the smoothing functional. If the curve is too smooth it will not meet the tight tolerances that will be required in areas where the data imply high curvature. On the other hand if the fit is too accurate then the curve will follow unwanted undulations and irregularities that may be inherent in the data. However, by carefully balancing these two terms with the help of a smoothing factor that controls the relative importance of the two goals and possibly by varying the terms in the functional, smooth and accurate curves can be obtained.

The integrated smoothness term is normally taken to be some linear combination of the derivatives squared of the unknown curve:

$$\int_a^b \Big(\sum_{j=1}^{m} \alpha_j \left\| \mathbf{f}^{(j)}(t) \right\|^2 \Big) dt.$$

As stated in the introduction the motivation for this comes from considering strain energy in material mechanics. In particular the first two derivative terms in the smoothness functional have the following physical interpretation:

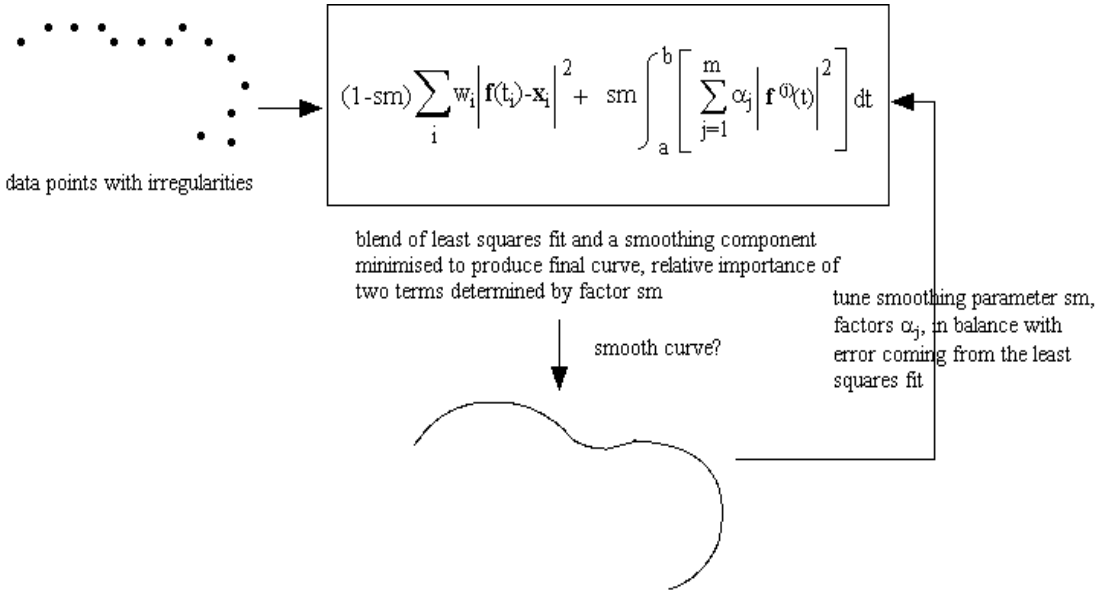


Figure 4.1: Incorporating smoothing process with least squares construction

- first derivative: The integrand refers to the square of the norm of $\mathbf{f}^{(1)}(t)$ which is related to the length of the curve. The squared first derivative term stands for the strain energy due to stretching of a beam.

- second derivative: The integrand refers to the square of the norm of $\mathbf{f}^{(2)}(t)$, which is proportional to the curvature. The integrand corresponds to the work done by the bending moment, and the integral criterion measures the total elastic energy accumulated in the beam or curve segment.

A curve that minimises some combination of these two terms will naturally resist stretching and bending. Some attention has also been focused on the use of higher order measures, $m > 2$. The third derivative term (also called 'jerk' with reference to its interpretation as a rate of change of acceleration) is a rough estimate of the rate of change of curvature, $\frac{d\kappa}{ds}$ and hence by minimising this integral term, gradual changes in curvature are obtained. The interpretation here is geometric rather than physical. Since the integral of the squared magnitude of the derivative of curvature evaluates to zero for circular arcs and straight lines, while minimising the integral of the squared magnitude of the first and second derivatives makes the curve stretch and bend

as little as possible, introducing the third derivative term enables the curve to form a circular arc approximately assuming that constraints allow it.

Various authors have investigated energy based techniques for curve smoothing. In Nowacki and Lu, [61], the curve is required to minimise a fairness functional which is based on a linear combination of the second and the third derivative squared integrals of the curve

$$J = \beta \int_a^b \left\| \frac{d^2\mathbf{f}(t)}{dt^2} \right\|^2 dt + \gamma \int_a^b \left\| \frac{d^3\mathbf{f}(t)}{dt^3} \right\|^2 dt,$$

where β and γ are constants and satisfy $0 \leq \beta, \gamma \leq 1, \beta + \gamma = 1$. They combine this with a least squares criterion D and look for curves that minimise the functional $J + D$ subject to an area constraint which leads to a non linear system of equations. Meier and Nowacki, [57], consider norms of the form

$$J_m = \int_0^1 \left\| \mathbf{f}^{(m)}(t) \right\|^2 dt$$

and provide some motivation for considering the cases $m = 2, 3$ and 4. The authors suggest the use of a combination of the three fairness functionals J_2, J_3, J_4 as a best approach in many cases. In Pottmann, [68], the object is to pass a smooth and visually pleasing curve with tensions parameters through a given finite set of data points. A smooth curve will not possess large variations in derivative vectors and the author chooses minimisation of a functional based on the first, second and third derivative terms. Fang and Gossard, [19], also use a functional based on first, second and third derivatives.

In practice, in addition to the smoothing functional, parameterisation effects are important, particularly where it is necessary to meet tight tolerances. A standard technique for improving the parameterisation of a least squares type data fit is given in Hoschek, [38]. Wang et al, [86], consider the relative effects of the parameterisation and various energy based functionals on the smoothed curves and surfaces. Weiss et al, [90], consider carefully the interaction between the smoothing factor and the tolerance and investigate procedures for automatic smoothing based on an iterative process of reparameterisation and knot modification.

The second way to obtain smooth spline curves is to separate the construction and fairing process, see figure 4.2. Here algorithms either work on the principles of iterative local control point/knot vector modification to smooth out higher derivative discontinuities, or are based on the application of global energy based functionals. Notable contributions to the former technique are found in Farin et al, [20], Eck and Hadenfeld, [17], Sapadis and Farin, [76]. They use an automated knot removal and reinsertion process process by determining where a knot should be

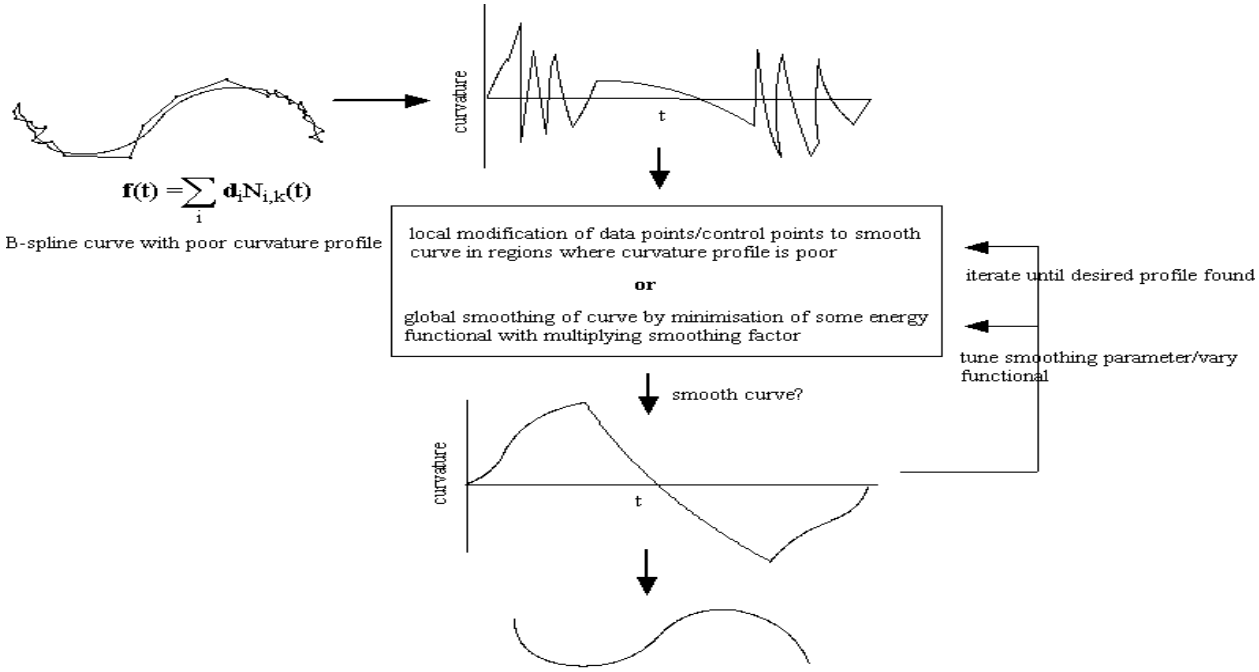


Figure 4.2: Smoothing existing curves

removed and a new one inserted using the curvature plot as the fairness criteria. This provides an approximation to the original curve. Other techniques dealing with this approach are found in Kjellander, [47] and Poliakoff, [67].

Based on the developments in Chapter 2 and 3 we present two algorithms for providing solutions to the two cases depicted in figures 4.1, 4.2. In the first case we combine the least squares error minimisation with a linear system derived from the exact minimisation of a given smoothness functional based on an integral of the squares of some linear combination of the derivatives of the unknown curve. In the second we take an existing B-spline curve and solve a system obtained by combining the smoothing functional term with evaluation of the curve at a set of predefined points. Our focus here and in the rest of the chapter is on the effects of the algorithms themselves rather than the process of integrating them into a larger iterative and automated scheme.

4.2.1 Smoothing combined with least squares data fitting

Starting with a given set of data points $\left(\mathbf{x}_i\right)_{i=1}^{M}$, optional weights $(w_i)_{i=1}^{M}$, a smoothing factor sm and a parameterisation, $(\tau_i)_{i=1}^{M}$, for the data points, we seek a solution to the combined least squares/smoothing functional[3]:

$$\mathrm{I}(\mathbf{f}) = (1-sm)\sum_{i=1}^{M} w_i \left\|\mathbf{f}(\tau_i) - \mathbf{x}_i\right\|^2 + sm \int_a^b \left(\sum_{j=1}^{m} \alpha_j \left\|\mathbf{f}^{(j)}(t)\right\|^2\right) dt = (1-sm)\mathrm{I}_{lsq}(\mathbf{f}) + sm\mathrm{I}_{smooth}(\mathbf{f}), \tag{4.2}$$

where $\mathbf{f}(t)$ is a parametric B-spline with a specified order k and dimension n:

$$\mathbf{f}(t) = \sum_{i=1}^{n} \mathbf{d}_i N_{i,k}(t), \quad \text{knot set } (t_i)_{i=1}^{n+k}.$$

The minimisation condition of the least squares error term

$$\frac{\partial \mathrm{I}_{lsq}}{\partial \mathbf{d}} = \mathbf{0},$$

gives the following n by n system of equations, $\mathbf{A}\mathbf{d} = \mathbf{y}$, where

$$\mathbf{A} = \left(\sum_{i=1}^{M} w_i N_{j,k}(\tau_i) N_{l,k}(\tau_i)\right)_{j,l=1}^{n}, \quad \mathbf{d} = \begin{pmatrix} \mathbf{d}_1 \\ \mathbf{d}_2 \\ \vdots \\ \mathbf{d}_n \end{pmatrix}, \quad \mathbf{y} = \begin{pmatrix} \sum_i w_i N_{1,k}(\tau_i)\mathbf{x}_i \\ \sum_i w_i N_{2,k}(\tau_i)\mathbf{x}_i \\ \vdots \\ \sum_i w_i N_{n,k}(\tau_i)\mathbf{x}_i \end{pmatrix},$$

($\mathbf{A}$ is of full rank if the Schoenberg-Whitney conditions are satisfied, see appendix E). We minimise the smoothing part I_{smooth} in 4.2 using algorithm 3.2, providing minimisation matrices $\mathbf{M}_1, \mathbf{M}_2, \ldots, \mathbf{M}_m$ for the derivative terms included in the functional. Combining these with the error term we obtain the following system of equations for the control points $\mathbf{d}$:

$$\left((1-sm)\mathbf{A} + sm\sum_{i=1}^{m} \alpha_i \mathbf{M}_i\right)\mathbf{d} = \mathbf{y}. \tag{4.3}$$

This can be solved efficiently either by straightforward LU factorisation or, taking into account the banded and symmetric positive definite nature of the matrices $\mathbf{A}$ and $\mathbf{M}_i$, by a banded Choleski factorisation. Although the least squares part of 4.3 is well conditioned, at least for

[3]In the following sections we use the symbols I and J to represent functionals

moderate degrees, [10], it can become ill-conditioned in the case of uneven segment distribution determined by the knot spacing relative to the parameterisation. However, in this respect, the matrix representing the smoothing part in 4.3 actually has a stabilising influence on the system as as a whole, something which is demonstrated in regularization theory, see Tikhonov & Aresnin, [83].

Based on 4.3, algorithm 4.1 presents the steps required to compute the B-spline solution for this case.

Algorithm 4.1: B-spline curve least squares fitting combined with smoothing

data set $\left(\mathbf{x}_i\right)_{i=1}^{M}$, *optional weights* $(w_i)_{i=1}^{M}$

smoothing functional, $\int_a^b \Big(\sum_{j=1}^{m} \alpha_j \left\| \mathbf{f}^{(j)}(t) \right\|^2 \Big) dt$

Solution: $\mathbf{f}(t) = \sum_{i=1}^{n} \mathbf{d}_i N_{i,k}(t)$ *on knot set* $(\mathtt{t_i})_{\mathtt{i=1}}^{\mathtt{n+k}}$

1. *create a parameterisation* $(\tau_\mathtt{i})_{\mathtt{i=1}}^{\mathtt{M}}$ *from the data points*
2. *given* `n` *create a suitable knot set* $(\mathtt{t_i})_{\mathtt{i=1}}^{\mathtt{n+k}}$
3. *create the basis function set* $\left(\mathtt{N_{i,k}(t)}\right)_{\mathtt{i=1}}^{\mathtt{n}}$
4. *create the B-spline least squares matrix,* $\mathtt{A} = (\mathtt{a_{ij}})_{\mathtt{i,j=1}}^{\mathtt{n}}, \mathtt{a_{ij}} = \sum_{\mathtt{l=1}}^{\mathtt{M}} \mathtt{w_l N_{i,k}}(\tau_\mathtt{l})\mathtt{N_{j,k}}(\tau_\mathtt{l})$
5. *create the right hand side vector,* $\mathtt{y} = (\mathtt{y_i})_{\mathtt{i=1}}^{\mathtt{n}}$, $\mathtt{y_i} = \sum_{\mathtt{j}} \mathtt{w_j N_{i,k}}(\tau_\mathtt{j})\mathtt{x_j}$
6. *create the minimisation matrices* $\mathtt{M_1}, \ldots, \mathtt{M_m}$ *from the basis function set*
7. *for a given smoothing factor* `sm`, *and coefficients* (α_j), *form the matrix*
   $\mathtt{B} = (1 - \mathtt{sm})\mathtt{A} + \mathtt{sm} \sum_{\mathtt{j}} \alpha_\mathtt{j} \mathtt{M_j}$
8. *solve the system* `Bd=y`
9. *construct the B-spline from the control points and knot set*

Figure 4.3: Algorithm 4.1: Least squares fitting combined with smoothing

As illustrated in figure 4.1, the process to achieve a desired solution is normally iterative. Whereas k is normally fixed, for a given n and I_{smooth}, the smoothing factor sm needs to be chosen carefully (see for example Weiss et al, [87]) to obtain the desired balance between accuracy as measured by the least squares error and curvature properties of the resulting curve as measured

by numerical values of I_{smooth}. However, it is worth noting that for a given order k and knot set, which is typically independent from the parameterisation, the smoothing matrices making up I_{smooth} are fixed and hence need be computed only once in an iterative scheme that seeks the 'best' solution.

4.2.2 Smoothing an existing B-spline curve

To smooth an existing B-spline curve we combine the matrices derived from the minimisation of the smoothing functional with matrices derived from evaluating the basis functions and B-spline curve, $\mathbf{f}(t)$, at the knot averages, $(\eta_i)_{i=1}^n$, given by

$$\eta_i = \frac{1}{k-1}(t_{i+1} + \ldots + t_{i+k-1}).$$

We compute the matrix of B-spline basis functions and the vector of curve evaluations at these points,

$$\mathbf{Q} = \big(N_{i,k}(\eta_j)\big)_{i,j=1}^n, \quad \mathbf{v} = \big(\mathbf{f}(\eta_i)\big)_{i=1}^n,$$

and then solve the following system for the new set of control points:

$$\Big(sm \sum_j \alpha_j \mathbf{M}_j + \mathbf{Q}\Big)\mathbf{d} = \mathbf{v}.$$

Algorithm 4.2 presents the steps.

4.3 Examples

To examine the effectiveness of algorithm 4.2 (and indirectly 4.1) in smoothing B-spline curves we test it on some examples and compare the results based on the following five functionals:

$$J_1 = \int_a^b \left\| \frac{d^2\mathbf{f}}{dt^2} \right\| dt, \quad J_2 = \int_a^b \left\| \frac{d^3\mathbf{f}}{dt^3} \right\| dt, \quad J_3 = \int_a^b \Big(0.25 \left\| \frac{d^2\mathbf{f}}{dt^2} \right\| + 0.75 \left\| \frac{d^3\mathbf{f}}{dt^3} \right\|\Big) dt,$$

$$J_4 = \int_a^b \Big(0.5 \left\| \frac{d^2\mathbf{f}}{dt^2} \right\| + 0.5 \left\| \frac{d^3\mathbf{f}}{dt^3} \right\|\Big) dt, \quad J_5 = \int_a^b \Big(0.75 \left\| \frac{d^2\mathbf{f}}{dt^2} \right\| + 0.25 \left\| \frac{d^3\mathbf{f}}{dt^3} \right\|\Big) dt.$$

To quantify the smoothing produced we calculate the following four global smoothing measures as an indication of the effectiveness of the functional:

$$\int_a^b \kappa^2 dt, \quad \int_a^b \left\| \frac{d^2\mathbf{f}}{dt^2} \right\| dt, \quad \int_a^b \left\| \frac{d^3\mathbf{f}}{dt^3} \right\| dt, \quad \int_a^b \left\| \frac{d\mathbf{f}}{dt} \right\| dt.$$

Algorithm 4.2: Smoothing an existing B-spline curve

$\mathbf{f}(t) = \sum_{i=1}^{n} \mathbf{d}_i N_{i,k}(t)$ *on the knot set* $(t_i)_{i=1}^{n+k}$
smoothing functional, $\int_a^b \Big(\sum_{j=1}^{m} \alpha_j \left\| \mathbf{f}^{(j)}(t) \right\|^2 \Big) dt$

1. *create the basis function set* $\left(\mathtt{N_{i,k}(t)}\right)_{\mathtt{i=1}}^{\mathtt{n}}$
2. *using algorithm 3.2 create the minimisation matrices* $\mathtt{M_1}, \ldots, \mathtt{M_m}$
3. *create the matrix of the basis functions evaluated at the knot averages,*
   $\mathtt{Q}=\left(\mathtt{N_{i,k}}(\eta_\mathtt{j})\right)_{\mathtt{i,j=1}}^{\mathtt{n}}$
4. *create the vector of curve evaluations at knot averages,* $\mathtt{v} = \left(\mathtt{f}(\eta_i)\right)_{i=1}^{n}$
5. *for a given smoothing factor* `sm`, *and coefficients* (α_j), *form the matrix*
   $\mathtt{B} = \mathtt{Q} + \mathtt{sm} \sum_{\mathtt{j}} \alpha_\mathtt{j} \mathtt{M_j}$
6. *solve the system* `Bd=v`
7. *build final smoothed B-spline curve from control points* `d` *from 6 and original knot set*

Figure 4.4: Algorithm 4.2: Smoothing an existing B-spline curve

The fourth measure represents the length of the curve. In addition we compute an estimate of the error between the original and the smoothed curve obtained by sampling.

4.3.1 Example 1

For the curve itself we construct an example with unwanted irregularities along the lines of Eck and Hadenfeld, [17]. We begin with the a B-spline curve containing 61 control points given by

$$\mathbf{d}_i = (x_i, sin\, x_i), \quad x_i = -3 + \frac{i}{10}, \quad 0 \leq i \leq 60,$$

and defined over the knot vector $(0, 0, 0, 0, 1, 2, \ldots, 57, 58, 58, 58, 58)$. The internal control points are then perturbed (keeping the end points fixed) using a random number generator in such a way the maximal allowed disturbance of the control points is 0.02 in absolute value. The curvature plot 4.6 shows the large number of unwanted inflection points introduced into the curve. The modified curve has significantly worse values for the smoothing measures compared

to the original. Table 4.1 displays the results of the test using a smoothing factor of 0.1. The results show good agreement with the original, with the blended second (energy) and third derivative (jerk) based functionals producing slightly better results than the unblended ones. Figure 4.5 displays the original (top), perturbed (bottom) and the five smoothed curves obtained by minimising $J_1, \ldots, J_5$ (top to bottom). The curvature plots of the curves are shown in figures 4.7 and 4.8.

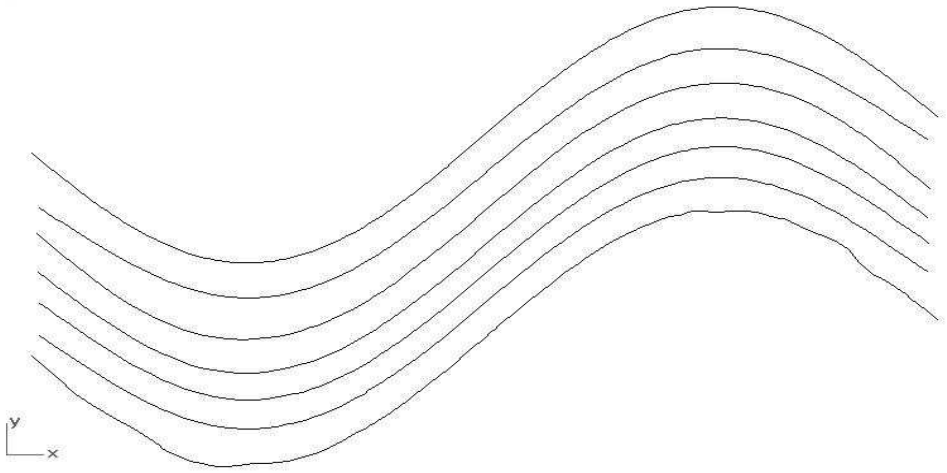


Figure 4.5: original (top), perturbed (bottom) and smoothed B-spline curves using functionals $J_1, \ldots, J_5$ top to bottom, smoothing factor=0.1

Example 1: Smoothing functionals applied to perturbed B-spline curve					
	$\int_a^b \kappa^2\, dt$	$\int_a^b \left\| \frac{d^2\mathbf{f}}{dt^2} \right\| dt$	$\int_a^b \left\| \frac{d^3\mathbf{f}}{dt^3} \right\| dt$	$\int_a^b \left\| \frac{d\mathbf{f}}{dt} \right\| dt$	*error*
original	28.2504	0.003095	2.70562e-05	6.92835	0.0
perturbed	47.671	0.0310638	0.0542622	6.53564	0.03258
method 1, J_1	25.3826	0.002737	2.28735e-05	6.88206	0.14277
method 2, J_2	24.8185	0.00418198	1.97002e-05	7.12918	0.0567637
method 3, J_3	27.0858	0.00310781	2.32442e-05	7.000426	0.0952497
method 4, J_4	26.8976	0.00310638	2.63855e-05	6.9527	0.114375
method 5, J_5	26.2211	0.00284096	2.056421e-05	6.91463	0.12951

Table 4.1: Integral measures before and after smoothing a B-spline curve using five different functionals

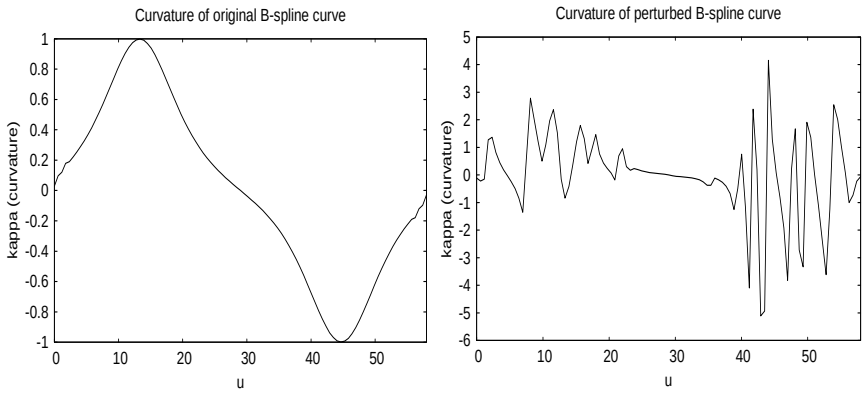


Figure 4.6: Curvature plots of original and perturbed curves

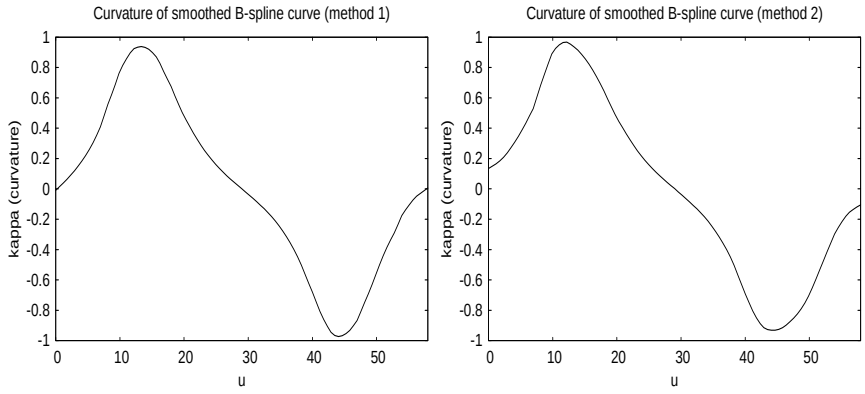


Figure 4.7: Curvature plots for methods 1 and 2, (J_1, J_2)

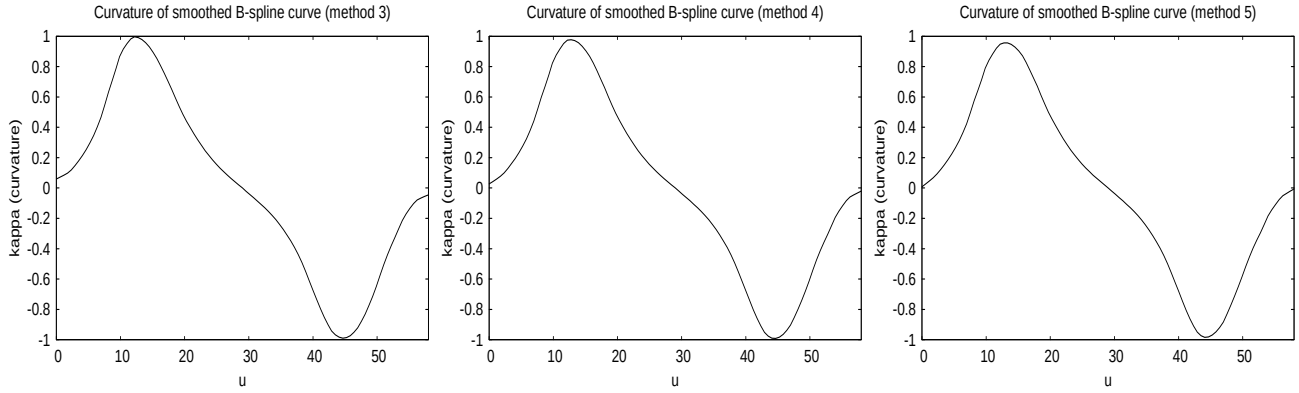


Figure 4.8: Curvature plots for methods 3, 4 and 5: (J_3, J_4, J_5)

4.3.2 Example 2

In the second example we test the algorithm on a more extreme example of unwanted behavior, again along the lines of Eck and Hadenfeld. We begin with the perturbed curve used in example 1 and then remove the following 26 internal knots:

$$(1, 3, 4, 5, 6, 9, 11, 14, 15, 19, 20, 25, 26, 27, 28, 31, 33, 34, 37, 38, 45, 46, 51, 52, 55, 57).$$

This produces a curve with additional irregularities due to the uneven knot distribution. The (bad) curvature plot of the curve is shown in figure 4.9 (this time using porcupines).

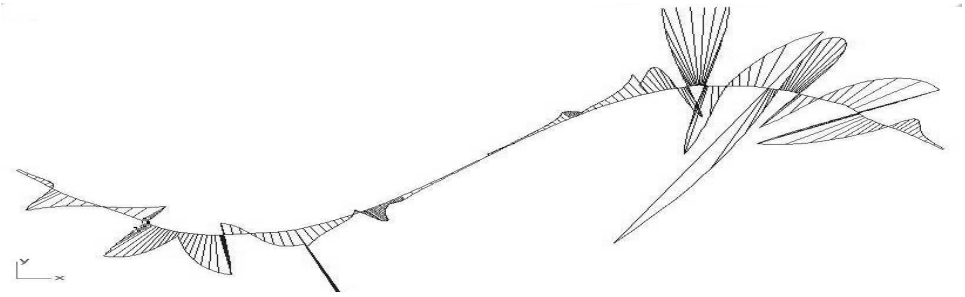


Figure 4.9: Perturbed curve with curvature map, smoothing example 2

The curve is smoothed using the functionals from example 1. The presence of the third derivative component in $J_2, \ldots, J_5$ actually produces significantly worse results for the resulting curve in comparison to the second derivative functional J_1. This is in line with the results obtained by Eck and Hadenfeld and supports their conclusion that in such cases of poor knot distribution

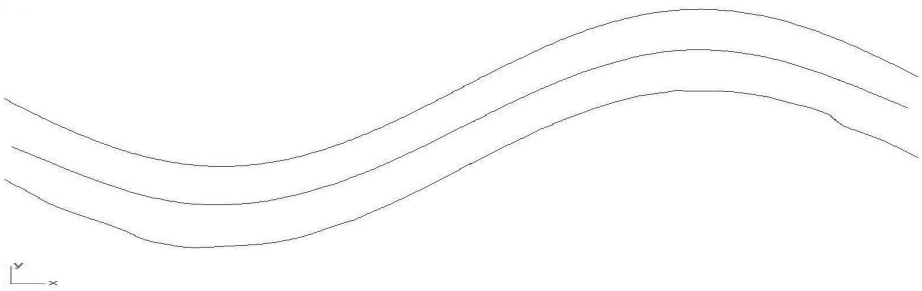


Figure 4.10: original (top), perturbed (bottom) and smoothed B-spline curve (middle) using functional J_1, smoothing factor=0.1

smoothing functionals involving the second derivative perform best. Figure 4.10 shows the original and perturbed curves with the result obtained by using functional J_1. Table 4.2 gives the numerical results.

Example 2: Smoothing functionals applied to perturbed B-spline curve					
	$\int \kappa^2\, dt$	$\int \left\| \frac{d^2\mathbf{f}}{dt^2} \right\| dt$	$\int \left\| \frac{d^3\mathbf{f}}{dt^3} \right\| dt$	$\int \left\| \frac{d\mathbf{f}}{dt} \right\| dt$	*error*
original	28.2504	0.003095	2.70562e-05	6.92835	0.0
perturbed	54.9652	0.02129970	0.0255867	6.38524	0.105748
method 1, J_1	29.939	0.00288389	3.0269e-05	6.85471	0.131429

Table 4.2: Integral measures before and after smoothing a B-spline curve using J_1

4.3.3 Example 3

In the final example we use the original curve from example 1 but this time perturb the control points to a greater extent: the maximum absolute control point deviation is increased by a factor 5 to 0.1. The smoothing factor is increased to 10. Figure 4.11 displays the original and perturbed curve with curvature values displayed and figure 4.12 shows the curvature map of the perturbed curve. Figure 4.13 shows the original and the five curves resulting from the smoothing with their curvature profiles. Table 4.3 gives the numerical results and figure 4.14 shows the curvature plots of some of the smoothed curves. The results demonstrate visually and numerically a good reconstruction with significant reductions in the smoothing measures compared to the perturbed curve whilst maintaining reasonable agreement with the original length.

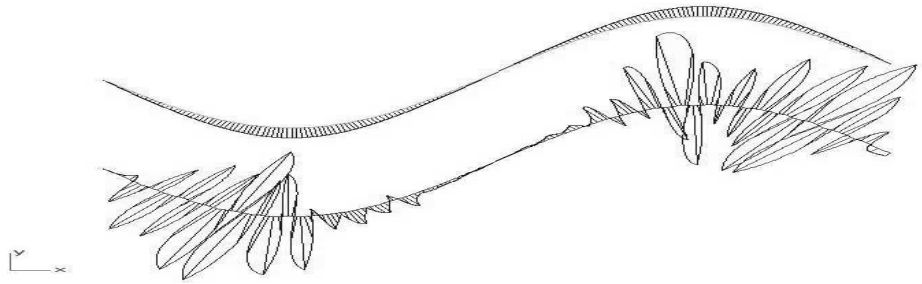


Figure 4.11: Original and perturbed curve with curvature displayed

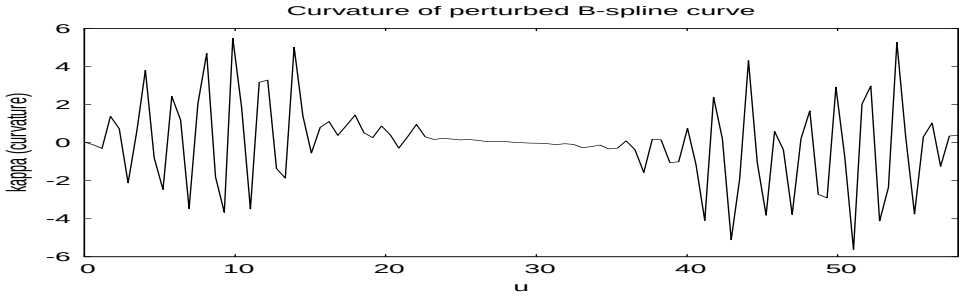


Figure 4.12: Perturbed curve curvature graph (example 3)

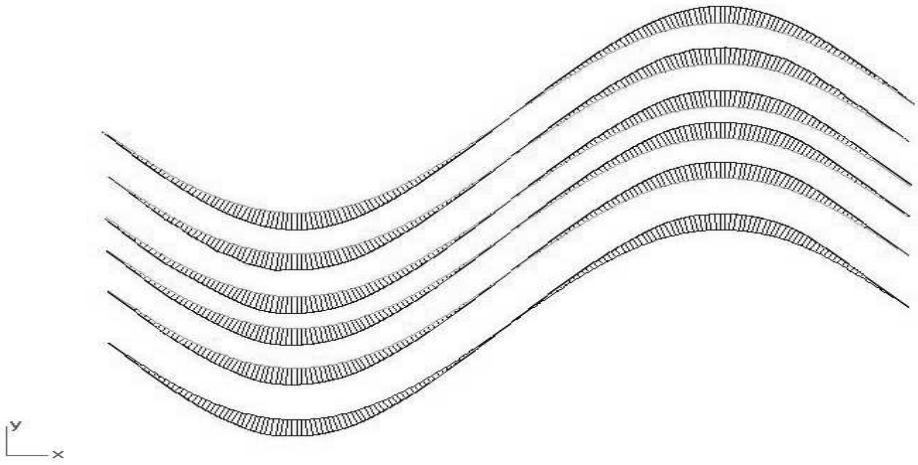


Figure 4.13: Original curve (bottom) and the five smoothed curves using functionals $J_1, \ldots, J_5$, top to bottom, with curvature porcupines, smoothing factor=10

Example 3: Smoothing functionals applied to perturbed B-spline curve					
	$\int \kappa^2\, dt$	$\int \left\| \frac{d^2\mathbf{f}}{dt^2} \right\| dt$	$\int \left\| \frac{d^3\mathbf{f}}{dt^3} \right\| dt$	$\int \left\| \frac{d\mathbf{f}}{dt} \right\| dt$	*error*
original	28.2504	0.003095	2.70562e-05	6.92835	0.0
perturbed	85.6386	0.0989454	0.996293	7.02165	0.0157374
method 1	25.4188	0.00404659	5.63528e-05	7.12099	0.0653389
method 2	26.8356	0.00707109	0.000432305	7.11573	0.0378333
method 3	26.0398	0.00535781	7.01335e-05	7.13534	0.0503984
method 4	25.7832	0.00989454	2.56868e-05	7.13612	0.0573036
method 5	25.6389	0.00431652	2.45208e-05	7.13026	0.0619578

Table 4.3: Integral measures before and after smoothing a B-spline curve using five different functionals

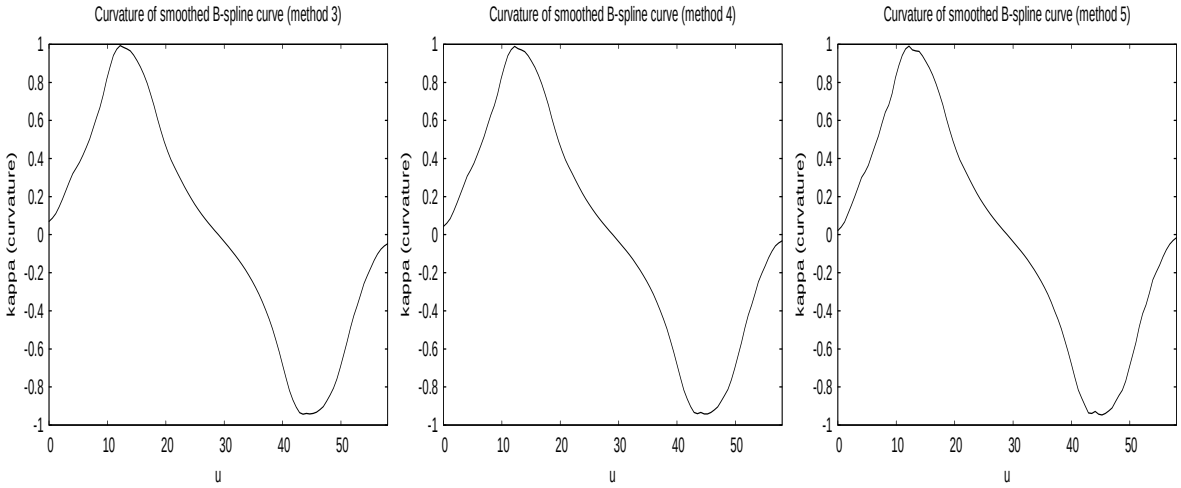


Figure 4.14: Smoothed curve curvature plots, methods 3, 4 and 5, example 3

4.4 Surface Smoothing

Surface smoothing is based upon similar principles to curve smoothing. The basic idea is to take a functional which is typically comprised of a part representing the integral of the square of the surface's curvature or its higher derivatives, and a part which represents how well the surface approximates a specified data set. We then seek a surface that minimises this combined measure. Often it can be likened to seeking the surface with the lowest elastic energy which satisfies certain interpolation/approximation constraints.

Figure 4.15 illustrates the basic process where least squares approximation is combined with a smoothing functional that represents, again in analogy with material mechanics, an energy term. By minmising the combined expression we can obtain obtain surfaces that are both accurate in terms of closeness to the original data and smooth in terms of curvature distribution. In an analogous fashion to the curve case, the combined least squares and smoothing functional

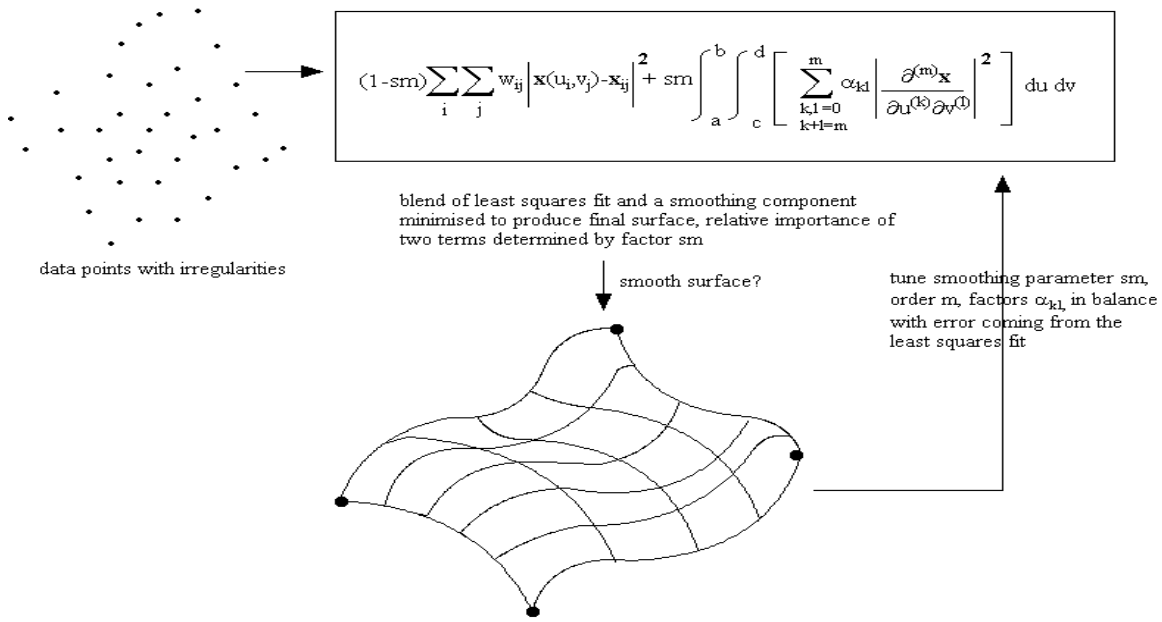


Figure 4.15: Incorporating smoothing process with least squares construction

represents for the surface generation process a balance between between two competing influences: a tendency of the surface to pass close to the data points $\mathbf{x}_{ij}$ and a tendency for curvature variations in the surface to be reduced.

The functional to be minimised in 4.15 is given by

$$\mathrm{I}(\mathbf{x}) = (1-sm)\mathrm{I}_{lsq}(\mathbf{x}) + sm\mathrm{I}_{smooth}(\mathbf{x})$$

$$= (1-sm)\sum_i\sum_j \|\mathbf{x}(u_i,v_j) - \mathbf{x}_{ij}\|^2 + sm\int_a^b\int_c^d \left(\sum_{i,j=0,i+j=m}^{m} \alpha_{ij}\left\|\frac{\partial^{(m)}\mathbf{x}}{\partial u^{(i)}\partial v^{(j)}}\right\|^2\right) du\,dv,$$

where the limits $[a,b]\times[c,d]$ lie within the parametric range of the surface. If the ratio $\frac{sm}{1-sm} \leq 1$, then the surface closely approximates the data points at the expense perhaps of surface fairness. If $\frac{sm}{1-sm} \geq 1$ then the dominant influence in determining the shape of the surface will be the requirement to minimise the integral of some energy term involving the integral of squared derivatives. The first order derivative terms in I_{smooth} are called membrane terms, for they make the surface resistant to stretching, much like an elastic string in one dimension, or a membrane in two dimensions. The second order derivative terms make the model resistant to bending forces, and hence can be interpreted as a rod in one dimension or a thin plate model in two dimensions. The weighting terms α_{ij} are used to control the elastic properties of the model. Increasing the value of the first order derivative weighting terms make the model more resistant to stretching, and likewise, increasing the second order terms make the model more resistant to bending.

Much of the published literature on energy minimisation for surfaces concentrates on functional based upon the sum of squares of the principal curvatures (the generalisation of 4.1 to surfaces) or suitable approximations thereof. Although this measure is insensitive to signed curvature variations it is sensitive to the relative magnitudes of curvature and has a convenient physical analogy. The strain energy of a thin rectangular elastic plate is given by

$$I = c\int_R \left[\left(\frac{\partial^2 x}{\partial u^2} + \frac{\partial^2 x}{\partial v^2}\right)^2 - 2(1-\mu)\left(\frac{\partial^2 x}{\partial u^2}\frac{\partial^2 x}{\partial v^2} - \frac{\partial^2 x}{\partial u \partial v}\right)^2\right] du\,dv, \tag{4.4}$$

where μ is Poisson's ratio, c is a constant and R represents the domain of the plate. This expression is minimised with respect to the free variables in the surface equation and in view of any given constraints, which leads to a variational formulation of the surface fairing problem. For practical purposes the expression is often replaced by simplified approximations. Neglecting the Poisson ratio for example leads to:

$$I = c\int_R \left[\left(\frac{\partial^2 x}{\partial u^2}\right)^2 + \left(\frac{\partial^2 x}{\partial v^2}\right)^2 + 2\left(\frac{\partial^2 x}{\partial u \partial v}\right)^2\right] du\,dv. \tag{4.5}$$

This is the small deflection equivalent (measuring the strain energy of flexure and torsion in a thin rectangular elastic plate with small deflection, (discussed in more detail in chapter 5) of the

functional

$$I = \int_R (k_1^2 + k_2^2) d\omega, \tag{4.6}$$

i.e. the integrated sum of the squares of the principal curvatures k_1, k_2.

Normally a unique measure of some geometric property should be independent of the parameterisation of the surface. Examples of such measures are given by 4.6 and the Gaussian or mean curvature and their generalisations to higher order derivatives. Hence some authors, notably Lott & Pullin, [53], Moretin & Sequin, [59]), have investigated using functionals based directly on these geometric measures rather than their parametrisation dependent cousins as in 4.4, 4.5. The drawback however is that whilst these methods produce very good results, evaluation of parameter invariant measures is costly and time-consuming because of the non linear relationship between defining coefficients (i.e control points) and surface derivatives. Hence they are unattractive as fairness optimisation criteria. As a compromise between the two, Greiner in [27], [28], [29], derives so-called data dependent functionals based on evaluating derivatives from a reference surface. The process is iterative with the reference surface updated with each iteration and is in fact independent of the surface parameterisation. Since it remains linear it shares some of the advantageous computational properties of the parameter dependent cases.

In analogy with the curve case, variations in curvature can be considered too as in Hagen, Santarelli, [32], and Hagen and Bonneau, [31]. Here the construction algorithm combines a least squares approximation with automatic surface smoothing. The smoothing criterion is the approximate minimisation of curvature variation based on the integral of the sum of the third order partial derivatives squared. More generally, an mth order functional can be created by using the so called Frobenius norm of the mth derivative:

$$\sum_{i=0}^{m} \binom{m}{i} \left\| \frac{\partial^{(m)} \mathbf{x}}{\partial u^{(i)} \partial v^{(m-i)}} \right\|^2 .$$

Figure 4.16 illustrates the case of smoothing an existing surface. Here the algorithms can broadly be thought of as dividing into two cases,

1. locally based modifications where small step changes to groups of control points/data points derived from derivative and curvature considerations are applied iteratively to obtain a desired solution, as in for example, Kjellander [46], Poliakoff, [67], and,

2. global techniques where typically an energy based functional is applied to part of or the whole surface, see for example, Nowacki et al, [62], Westgaard & Nowacki, [93].

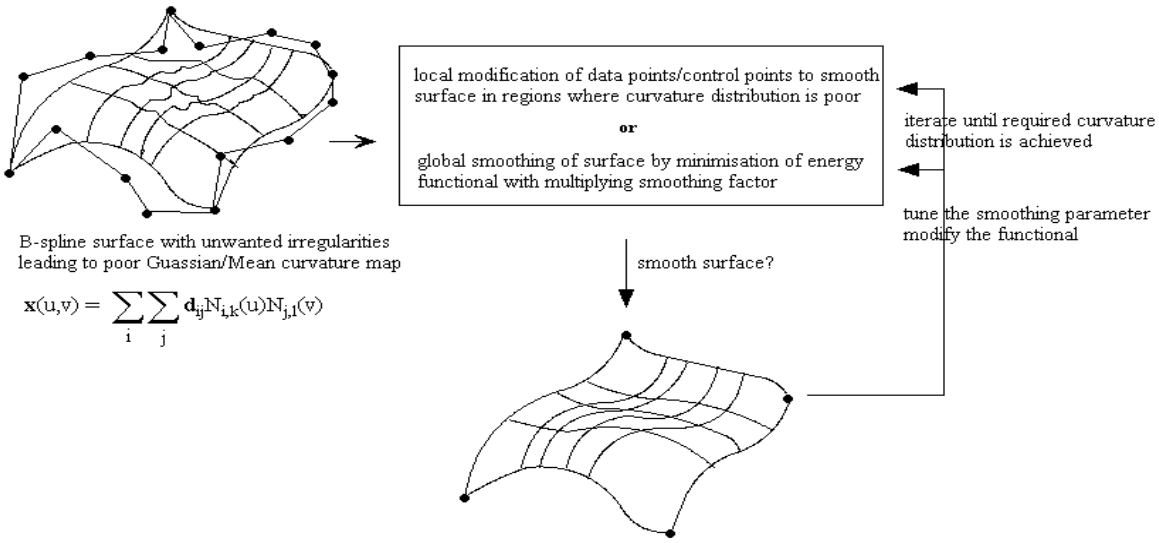


Figure 4.16: Smoothing an existing surface

We regard the parameterisation dependent measures as reasonable approximations to the invariant ones for fairing purposes and in the following sections present algorithms based on the second case for smoothing surfaces with or without data fitting. Examples are provided.

4.4.1 Smoothing combined with least squares data fitting

Starting with a given set of data points $\left(\mathbf{x}_{ij}\right)_{i,j=1}^{M,N}$, and u, v parameterisations, $(\tau_i)_{i=1}^{M}, (\mu_j)_{j=1}^{N}$ respectively, we seek a B-spline solution

$$\mathbf{x}(u,v) = \sum_{i=1}^{p}\sum_{j=1}^{q} \mathbf{d}_{ij} N_{i,k}(u) N_{j,l}(v), \quad (u_i)_{i=1}^{p+k} \times (v_j)_{j=1}^{q+l},$$

to the combined least squares/smoothing functional:

$$\mathrm{I}(\mathbf{x}) = (1-sm)\mathrm{I}_{lsq}(\mathbf{x}) + sm\mathrm{I}_{smooth}(\mathbf{x})$$

$$= (1-sm)\sum_{i=1}^{M}\sum_{j=1}^{N} \left\|\mathbf{x}(\tau_i,\mu_j) - \mathbf{x}_{ij}\right\|^2 + sm \int_a^b \int_c^d \left(\sum_{i,j=0, i+j=m}^{m} \alpha_{ij} \left\| \frac{\partial^{(m)}\mathbf{x}}{\partial u^{(i)} \partial v^{(j)}} \right\|^2 \right) du\, dv.$$

The minimisation of the least squares error term, $\partial \mathrm{I}_{lsq}/\partial \mathbf{d} = 0$, gives the following matrix equation

$$\mathbf{AdB} = \mathbf{C},$$

where

$$\mathbf{A} = \Big(\sum_{i=1}^{M} N_{r,k}(\tau_i) N_{s,k}(\tau_i)\Big)_{r,s=1}^{p}, \quad \mathbf{B} = \Big(\sum_{j=1}^{N} N_{r,l}(\mu_j) N_{s,l}(\mu_j)\Big)_{r,s=1}^{q},$$

$$\mathbf{C} = \Big(\sum_{i=1}^{M}\sum_{j=1}^{N} N_{r,k}(\tau_i) N_{s,l}(\mu_j)\mathbf{x}_{ij}\Big)_{r,s=1}^{p,q}.$$

From 3.9 the minimisation of the smoothness term gives

$$\mathbf{0} = \frac{\partial \mathrm{I}_{smooth}}{\partial \mathbf{d}} = \sum_{i,j} \alpha_{ij} \mathbf{M}_i^u \mathbf{d} \mathbf{M}_j^v.$$

By combining with the error term we obtain the following system of equations for the control point matrix $\mathbf{d}$:

$$(1 - sm)\mathbf{AdB} + sm \sum_{i,j} \alpha_{ij} \mathbf{M}_i^u \mathbf{d} \mathbf{M}_j^v = \mathbf{C}. \tag{4.7}$$

We solve this system using the generalisation of the Kronecker product equivalence given by equation 2.3:

$$\mathbf{A}_1\mathbf{X}\mathbf{B}_1 + \ldots + \mathbf{A}_n\mathbf{X}\mathbf{B}_n = \mathbf{C} \equiv \Big((\mathbf{B}_1)^T \otimes \mathbf{A}_1 + \ldots + (\mathbf{B}_n)^T \otimes \mathbf{A}_n\Big) vec(\mathbf{X}) = vec(\mathbf{C}). \tag{4.8}$$

Using this generalisation combined with a conventional LU factorisation, algorithm 4.3 presents the steps required to compute the smoothed B-spline approximating surface.

4.4.2 Smoothing an existing B-spline surface

For smoothing an existing B-spline surface we combine the matrices derived from the minimisation of the smoothing functional with matrices derived from evaluating the basis functions and B-spline surface at the knot averages, $(\eta_i)_{i=1}^{p}, (\psi_j)_{j=1}^{q}$, where

$$\eta_i = \frac{1}{k-1}(u_{i+1} + \ldots + u_{i+k-1}), \quad \psi_j = \frac{1}{l-1}(v_{j+1} + \ldots + v_{j+l-1}).$$

The matrices in question

$$\mathbf{C} = \big(N_{i,k}(\eta_j)\big)_{i,j=1}^{p}, \quad \mathbf{D} = \big(N_{i,l}(\psi_j)\big)_{i,j=1}^{q}, \quad \mathbf{E} = \big(\mathbf{x}(\eta_i, \psi_j)\big)_{i,j=1}^{p,q},$$

Algorithm 4.3: B-spline least squares surface fitting with smoothing

data set $\left(\mathbf{x}_{ij}\right)_{i,j=1}^{M,N}$,

smoothing functional, $\int_a^b \int_c^d \sum_{i,j=0,i+j=m}^{m} \alpha_{ij} \left\| \frac{\partial^{(m)}\mathbf{x}}{\partial u^{(i)} \partial v^{(j)}} \right\|^2 du\, dv$

solution formed: $\mathbf{x}(u,v) = \sum_{i=1}^{p} \sum_{j=1}^{q} \mathbf{d}_{ij} N_{i,k}(u) N_{j,l}(v)$, *knot set* $(u_i)_{i=1}^{p+k} \times (v_j)_{j=1}^{q+l}$

1. *create* `u,v` *parameterisations* $(\tau_{\mathtt{i}})_{\mathtt{i=1}}^{\mathtt{M}}, (\mu_{\mathtt{i}})_{\mathtt{i=1}}^{\mathtt{N}}$, *from the data points*
2. *create suitable knot sets* $(\mathtt{u_i})_{\mathtt{i=1}}^{\mathtt{p+k}}$, $(\mathtt{v_j})_{\mathtt{j=1}}^{\mathtt{q+1}}$
3. *create the basis function set* $\left(\mathtt{N_{i,k}(u)N_{j,l}(v)}\right)_{\mathtt{i,j=1}}^{\mathtt{p,q}}$
4. *create the minimisation matrices* $\mathtt{M_1^u}, \ldots, \mathtt{M_m^u}, \mathtt{M_1^v}, \ldots, \mathtt{M_m^v}$ *from the basis function set*
5. *create the least squares matrices,*
$$\mathtt{A} = (\sum_{\mathtt{i=1}}^{\mathtt{M}} \mathtt{N_{r,k}}(\tau_{\mathtt{i}})\mathtt{N_{s,k}}(\tau_{\mathtt{i}}))_{\mathtt{r,s=1}}^{\mathtt{p,p}}, \quad \mathtt{B} = (\sum_{\mathtt{j=1}}^{\mathtt{N}} \mathtt{N_{r,l}}(\mu_{\mathtt{j}})\mathtt{N_{s,l}}(\mu_{\mathtt{j}}))_{\mathtt{r,s=1}}^{\mathtt{q,q}}$$
6. *create the right hand side matrix,* $\mathtt{C} = (\sum_{\mathtt{i=1}}^{\mathtt{M}} \sum_{\mathtt{j=1}}^{\mathtt{N}} \mathtt{N_{r,k}}(\tau_{\mathtt{i}})\mathtt{N_{s,l}}(\mu_{\mathtt{j}})\mathtt{x_{ij}})_{\mathtt{r,s=1}}^{\mathtt{p,q}}$
7. *for a given factor* `sm` *form the Kronecker product matrix* $\mathtt{D} = (\mathtt{1} - \mathtt{sm})(\mathtt{B}^{\mathtt{T}} \otimes \mathtt{A})$
8. *for a given factor* `sm` *form the Kronecker product matrix* $\mathtt{E} = \mathtt{sm} \sum_{\mathtt{ij}} \alpha_{\mathtt{ij}} ((\mathtt{M_j^v})^{\mathtt{T}} \otimes \mathtt{M_i^u})$
9. *solve the system* `(D+E)vec(d)=vec(C)`
10. *reconstruct the B-spline control point matrix from the vector* `vec(d)` *and the B-spline surface*

Figure 4.17: Algorithm 4.3: Least squares fitting combined with smoothing

are combined with the smoothing functional equation derived from minimising I_{smooth}:

$$\frac{\partial \mathrm{I}_{smooth}}{\partial \mathbf{d}} = \sum_{i,j} \alpha_{ij} \mathbf{M}_i^u \mathbf{d} \mathbf{M}_j^v = \mathbf{0},$$

to get the matrix equation

$$sm \sum_{i,j} \alpha_{ij} \mathbf{M}_i^u \mathbf{d} \mathbf{M}_j^v + \mathbf{C}\mathbf{d}\mathbf{D} = \mathbf{E}. \tag{4.9}$$

Again we solve this system by using the Kronecker product method and LU factorisation. Equa-

tion 4.9 becomes

$$\Big(sm \sum_{ij} \alpha_{ij}((\mathbf{M}_j^v)^T \otimes \mathbf{M}_i^u) + (\mathbf{D}^T \otimes \mathbf{C})\Big) vec(\mathbf{d}) = vec(\mathbf{E}). \tag{4.10}$$

Algorithm 4.4 list the steps required to compute the smooth surface based on 4.9:

Algorithm 4.4: Smoothing an existing B-spline surface

$\mathbf{x}(u,v) = \sum_{i=1}^{p}\sum_{j=1}^{q} \mathbf{d}_{ij} N_{i,k}(u) N_{j,l}(v)$, *knot set* $(u_i)_{i=1}^{p+k} \times (v_j)_{j=1}^{q+l}$

smoothing functional, $\int_a^b \int_c^d \sum_{i,j=0, i+j=m}^{m} \alpha_{kl} \left\| \frac{\partial^{(m)}\mathbf{x}}{\partial u^{(i)} \partial v^{(j)}} \right\|^2 du\, dv$

1. *create the basis function set* $\big(\mathtt{N_{i,k}(u)N_{j,l}(v)}\big)_{\mathtt{i,j=1}}^{\mathtt{p,q}}$
2. *create the minimisation matrices* $\mathtt{M_1^u}, \ldots, \mathtt{M_m^u}$ $\mathtt{M_1^v}, \ldots, \mathtt{M_m^v}$ *from the basis function set*
3. *create matrices of* `u,v` *basis functions of* `x` *evaluated at knot averages*
   $\mathtt{C} = \big(\mathtt{N_{i,k}}(\eta_\mathtt{j})\big)_{\mathtt{i,j=1}}^{\mathtt{p}}$, $\mathtt{D} = \big(\mathtt{N_{i,l}}(\psi_\mathtt{j})\big)_{\mathtt{i,j=1}}^{\mathtt{q}}$
4. *create the matrix of surface evaluations at knot averages,* $\mathtt{E} = \big(\mathbf{x}(\eta_i, \psi_j)\big)_{i,j=1}^{p,q}$
5. *form the Kronecker product matrix* $\mathtt{A} = \mathtt{D^T} \otimes \mathtt{C}$
6. *for a given smoothing factor* `sm` *form the Kronecker product matrix* $\mathtt{B} = \mathtt{sm} \sum_{\mathtt{ij}} \alpha_{\mathtt{ij}} ((\mathtt{M_j^v})^\mathtt{T} \otimes \mathtt{M_i^u})$
7. *solve the system* `(A+B)vec(d)=vec(E)`
8. *reconstruct the B-spline control point matrix from the vector* `vec(d),` *hence the surface*

Figure 4.18: Algorithm 4.4: Smoothing an existing B-spline surface

4.4.3 Examples

To examine the effectiveness of algorithm 4.3/4.4 in smoothing existing surfaces we start with a B-spline surface that is an approximation to a torus section. The surface is of order 5 by 5 with 5 segments in u and v and represents an approximation to a quarter torus with major radius 10 and minor radius 4 defined over the u, v limits $[0, \pi] \times [0, \pi]$:

$$\mathbf{x}(u,v) = \sum_{i=1}^{14}\sum_{j=1}^{14} \mathbf{d}_{ij} N_{i,5}(u) N_{j,5}(v).$$

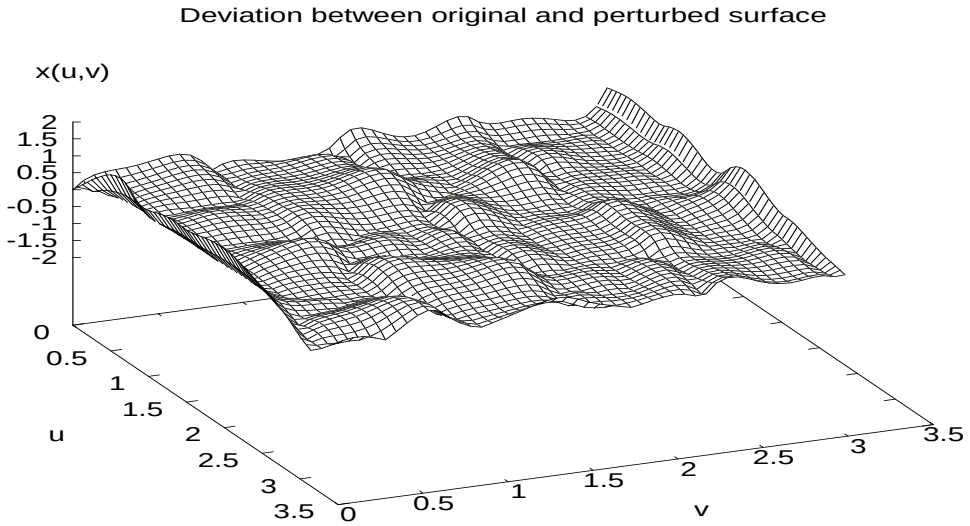


Figure 4.19: Deviation of perturbed B-spline surface from original

As in the curve smoothing examples, to produce a surface with unwanted undulations we perturb the torus using a random number generator producing random numbers in the range 0 to 1. These 'errors' are successively added to and subtracted from the original control points to produce the perturbed surface. Figure 4.19 shows the resulting deviation surface as a function of u and v illustrating the irregularities introduced throughout the domain of the surface. Figures 4.20 and 4.21 illustrate the torus section before and after perturbing the control points in both wireframe and environment mapped forms and 4.22 displays the Gaussian curvature surfaces for the two entities.

We use the following four smoothing functionals to reconstruct the surface:

$$J^2_{uv} = \int_R \left(\mathbf{x}^2_{uu} + \mathbf{x}^2_{vv}\right) du\, dv, \qquad J^2_{pas} = \int_R \left(\mathbf{x}^2_{uu} + 2\mathbf{x}^2_{uv} + \mathbf{x}^2_{vv}\right) du\, dv,$$
$$J^3_{uv} = \int_R \left(\mathbf{x}^2_{uuu} + \mathbf{x}^2_{vvv}\right) du\, dv, \qquad J^3_{pas} = \int_R \left(\mathbf{x}^2_{uuu} + 3\mathbf{x}^2_{uuv} + 3\mathbf{x}^2_{uvv} + \mathbf{x}^2_{vvv}\right) du\, dv, \tag{4.11}$$

where R is the relevant smoothing region of the u, v plane (in this example the whole domain of the surface). To examine the effectiveness of the algorithm and the functionals we evaluate the

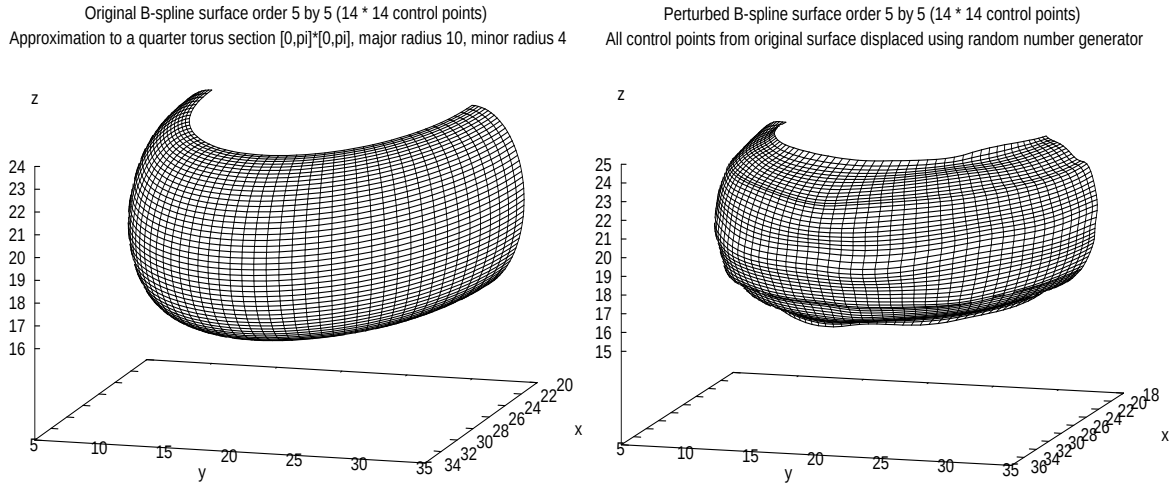


Figure 4.20: Original torus section and surface after perturbing the control points

Figure 4.21: Environments maps of the original and perturbed surfaces

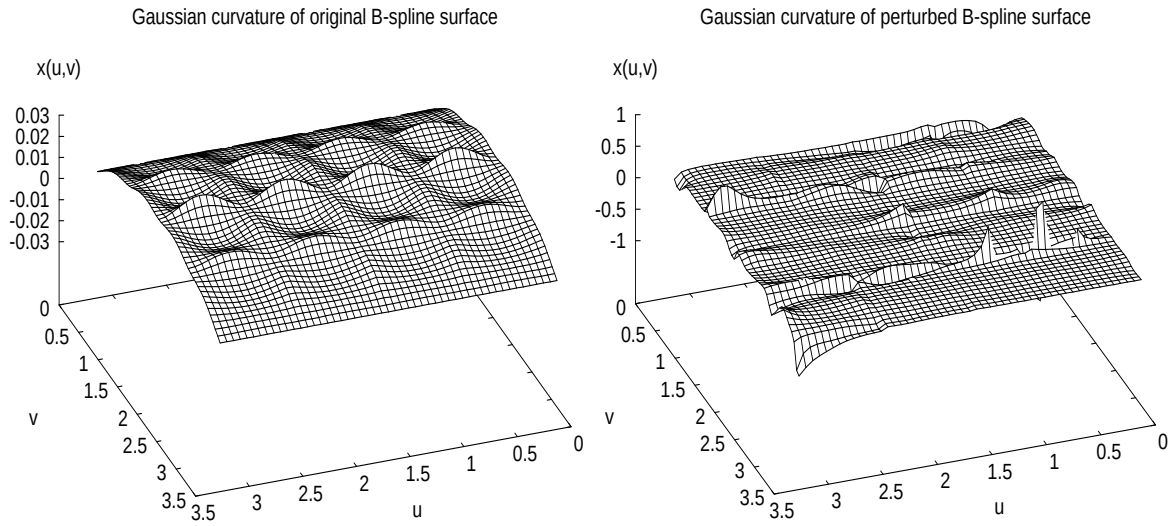


Figure 4.22: Gaussian curvature of original and perturbed surfaces

following selection of global fairness measures for the resulting surface:

$$
\begin{aligned}
J^2_u(\mathbf{x}) &= \int_R \mathbf{x}^2_{uu} du\, dv, & J^2_v(\mathbf{x}) &= \int_R \mathbf{x}^2_{vv} du\, dv,\\
J^2_{uv}(\mathbf{x}) &= \int_R \big(\mathbf{x}^2_{uu} + \mathbf{x}^2_{vv}\big) du\, dv, & J^2_{pas}(\mathbf{x}) &= \int_R \big(\mathbf{x}^2_{uu} + 2\mathbf{x}^2_{uv} + \mathbf{x}^2_{vv}\big) du\, dv,\\
J^3_u(\mathbf{x}) &= \int_R \mathbf{x}^2_{uuu} du\, dv, & J^3_v(\mathbf{x}) &= \int_R \mathbf{x}^2_{vvv} du\, dv,\\
J^3_{uv}(\mathbf{x}) &= \int_R \big(\mathbf{x}^2_{uuu} + \mathbf{x}^2_{vvv}\big) du\, dv, & J^3_{pas}(\mathbf{x}) &= \int_R \big(\mathbf{x}^2_{uuu} + 3\mathbf{x}^2_{uuv} + 3\mathbf{x}^2_{uvv} + \mathbf{x}^2_{vvv}\big) du\, dv.
\end{aligned} \tag{4.12}
$$

For the purpose of the figures to follow we label these functional measures as $J_1, \ldots, J_8$ reading left to right and top to bottom. As with the curve case, the smaller the value obtained for the global smoothness measure the better the fairness of the surface. In addition, four parameter independent measures based on the principal curvatures k_1, k_2 are evaluated, Gaussian (K), Mean (H), Absolute (A) and Total (T):

$$J_K = \int_R (k_1 k_2)^2 du\, dv, \qquad J_H = \int_R \big(\frac{k_1 + k_2}{2}\big)^2 du\, dv,$$
$$J_A = \int_R \big(\|k_1\| + \|k_2\|\big)^2 du\, dv, \qquad J_T = \int_R \big(k_1^2 + k_2^2\big)^2 du\, dv.$$

Finally, we also compute the pointwise error between the original and the smoothed surfaces obtained by sampling and the area,

$$Area = \int_R \|\mathbf{x}_u \times \mathbf{x}_v\|\, du\, dv,$$

as measures of the accuracy of the reconstruction.

The four smoothing functionals in 4.11 are applied to the perturbed surface using a notional smoothing factor, sm, of 0.1. The measures $J_1, \ldots, J_8$ are then evaluated to help determine the quality of the reconstruction. Table 4.4 gives the results of the tests for the smoothing of the perturbed torus. Figures 4.24 and 4.26 show the resulting surfaces in wireframe form and figures 4.25 and 4.27 the corresponding environment mapped images. From the figures and tabular results the functionals J_3 and J_4 appear to do better at reproducing the curvature properties of the original surface for this example. This is backed by closer examination of the Gaussian curvature surfaces displayed in figures 4.22, 4.28 and 4.29 and the Guassian and mean curvature maps for the original and smoothed surfaces presented in figures 4.23 to 4.33.

Evaluation Measure	*Optimisation Criteria*				*Perturbed Surface*	*Original Surface*
	$min \longrightarrow J^2_{uv}$	$min \longrightarrow J^2_{pas}$	$min \longrightarrow J^3_{uv}$	$min \longrightarrow J^3_{pas}$		
J^2_u	1205.43	1226.63	5382.38	4684.04	4764.59	1590.56
J^2_v	110.137	114.515	663.29	473.321	14679.5	166.585
J^2_{uv}	1802.39	1837.01	7712.14	6061.77	17505.6	2452.2
J^2_{pas}	2113.36	2132.25	8140.91	6370.33	18553	2769.18
J^3_u	7848.46	7258.76	148658	114399	73210.1	1639.92
J^3_v	831.981	811.836	25770.3	11758.9	747252	212.437
J^3_{uv}	8713.6	8092.25	201548	126797	758863	1851.88
J^3_{pas}	11030.9	10197	229550	124280	1.3747e+06	7741.65
J_K	3.03332e-21	3.18595e-21	2.83922e-18	4.28346e-19	7.88762e-07	7.29018e-21
J_H	0.00301197	0.0031941	0.0844391	0.04071	74334.7	0.00562191
J_A	0.0481914	0.0511055	1.35103	0.65136	1.18935e+06	0.0899506
J_T	0.000301968	0.000334955	1.65967	0.141795	2.50147e+12	0.00091451
Area	492.077	490.118	501.858	495.366	599.906	495.295
Max dev perturbed	1.2695	1.26419	1.3438	1.33566	0.0	1.28526
Max dev Original	0.1764	0.158105	0.613125	0.46787	1.28526	0.0

Table 4.4: Smoothing measures and values for totally perturbed torus section, $sm = 0.1$

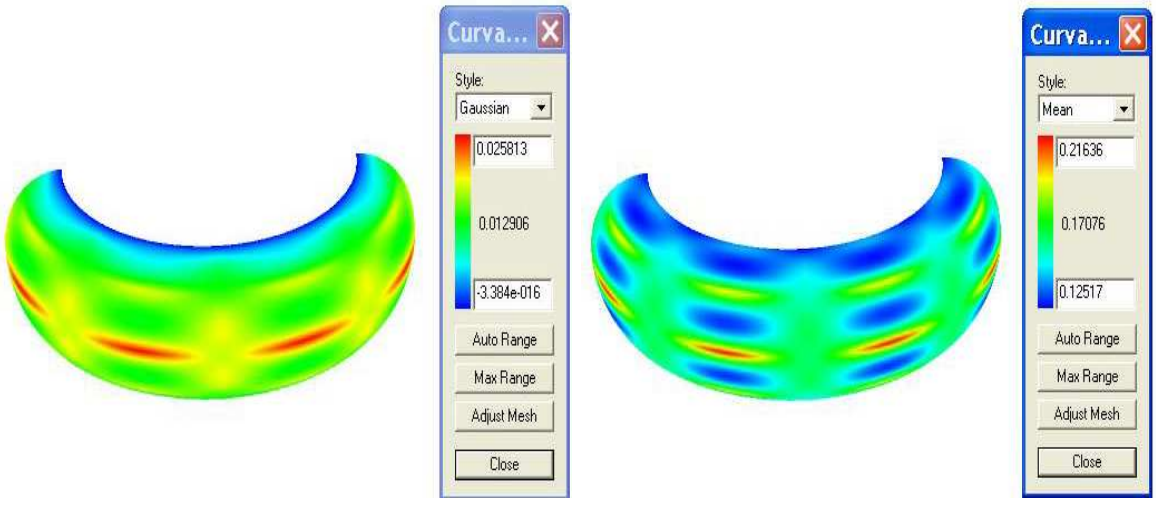


Figure 4.23: Gaussian and mean curvature map of original surface

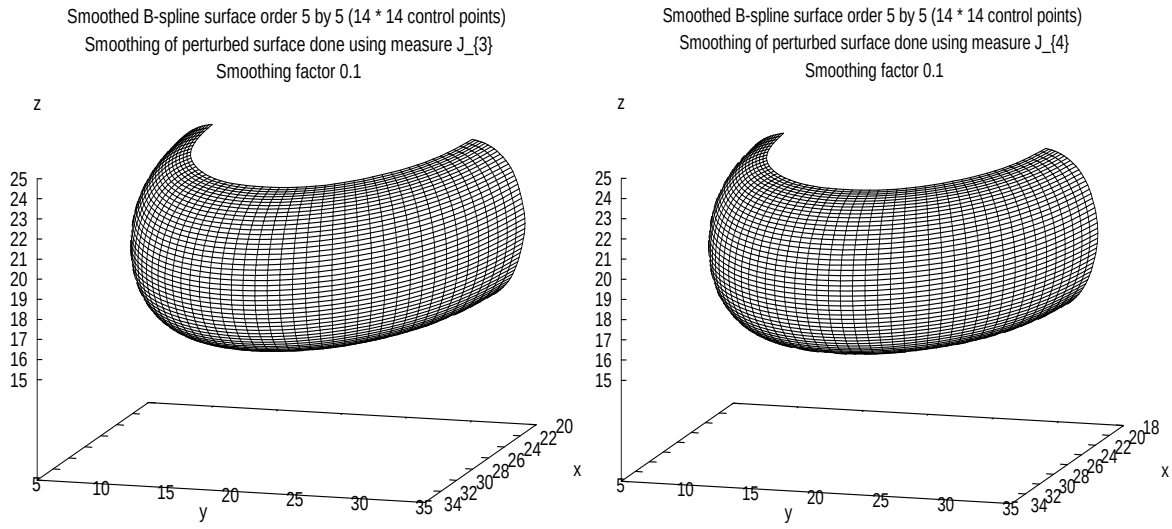


Figure 4.24: Smoothed surfaces using functionals J_3 and J_4

Figure 4.25: Environment maps for J_3 and J_4 smoothed surfaces

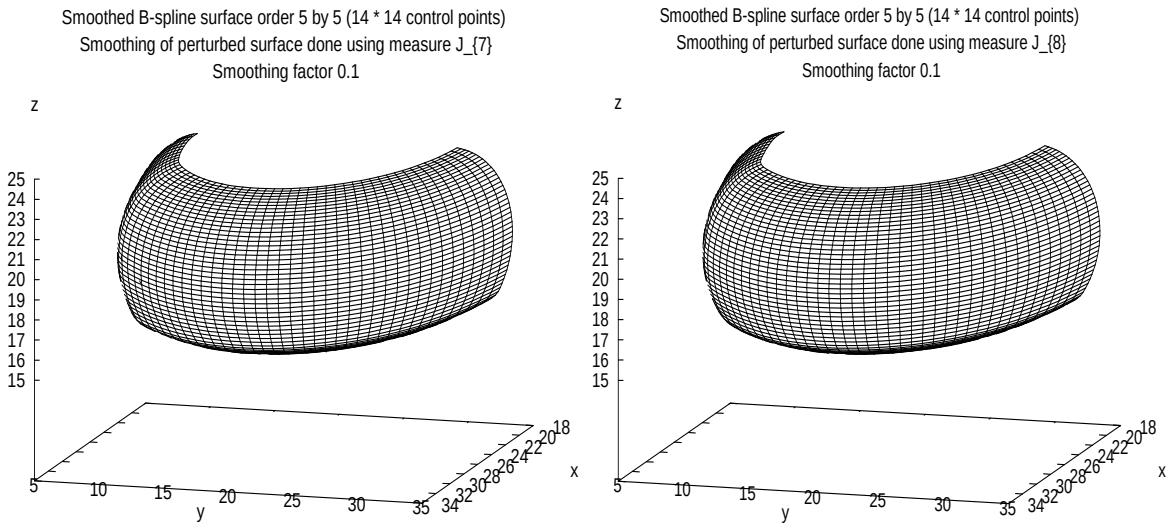


Figure 4.26: Smoothed surfaces using functionals J_7 and J_8

Figure 4.27: Environment maps for J_7 and J_8 smoothed surfaces

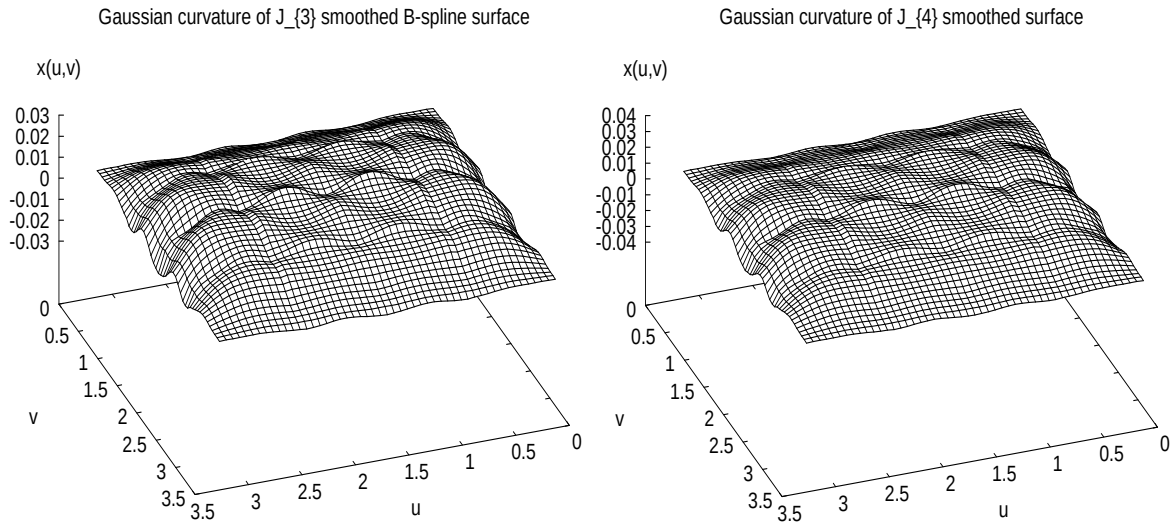


Figure 4.28: Gaussian curvature of J_3 and J_4 smoothed surfaces

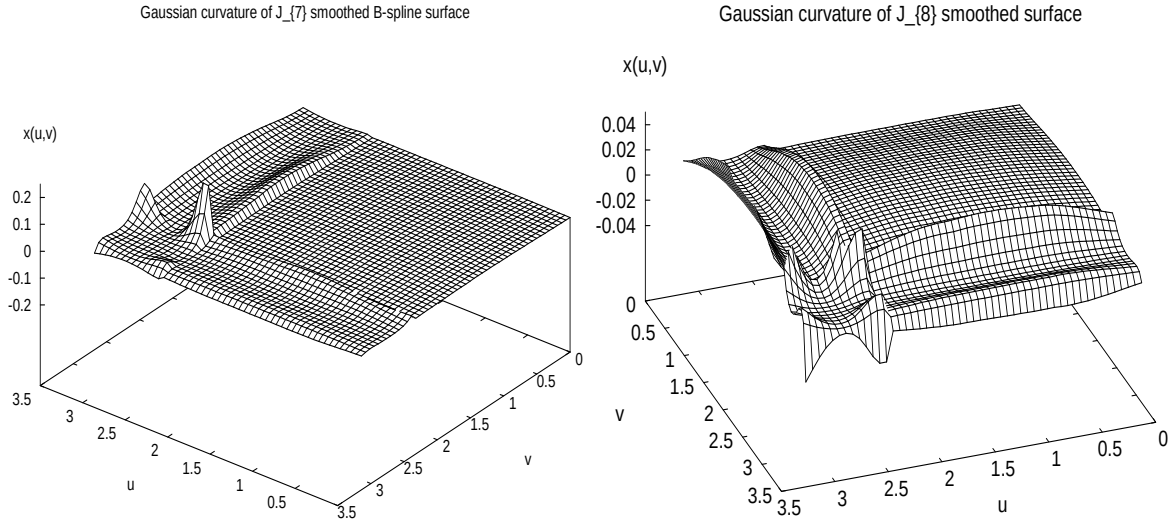


Figure 4.29: Gaussian curvature of J_7 and J_8 smoothed surfaces

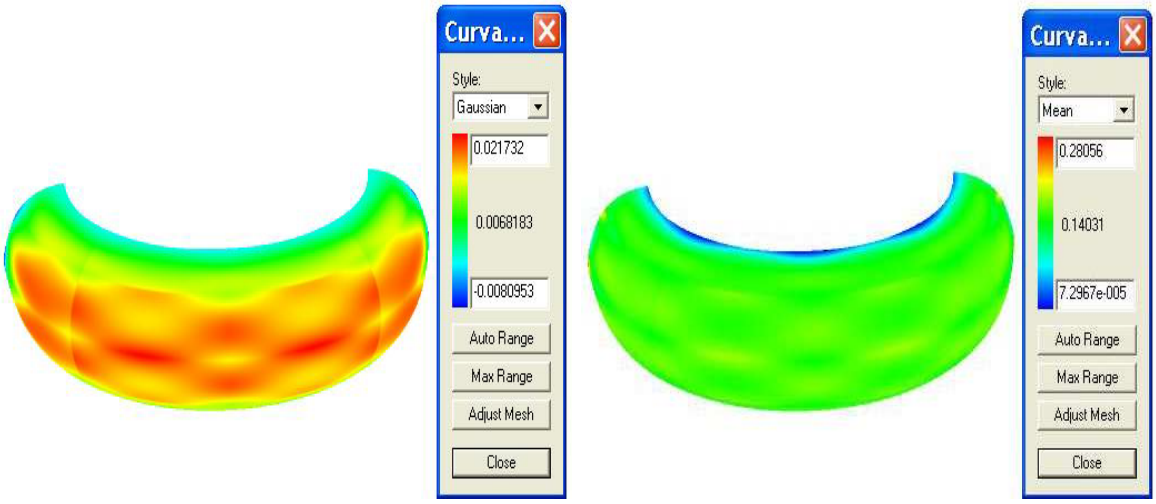


Figure 4.30: Gaussian and mean curvature map of J_3 smoothed surface

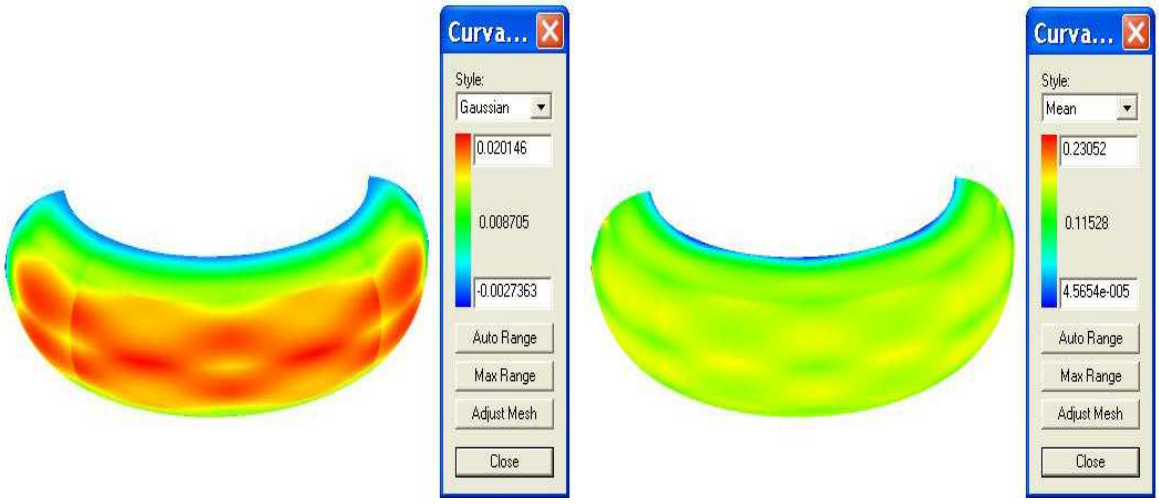


Figure 4.31: Gaussian and mean curvature map of J_4 smoothed surface

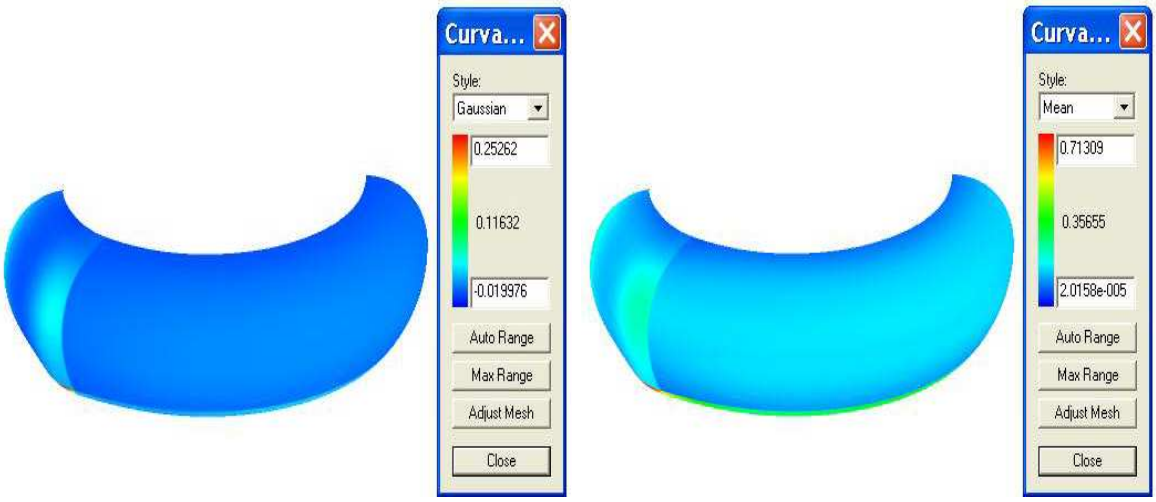


Figure 4.32: Gaussian and mean curvature map of J_7 smoothed surface

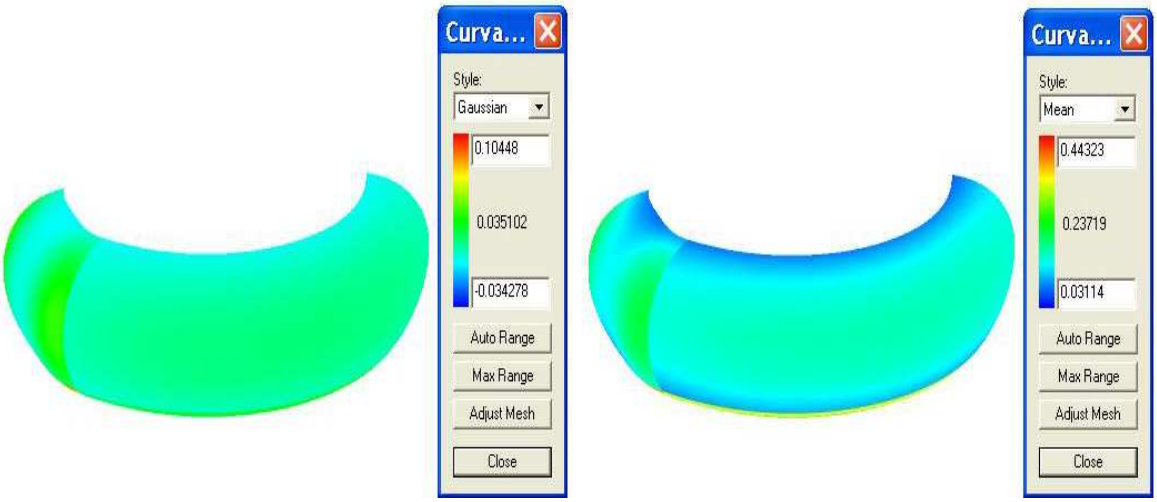


Figure 4.33: Gaussian and mean curvature map of J_8 smoothed surface

4.4.4 Local smoothing

It is of course not always necessary to smooth the whole surface. Irregularities may only be present over a particular section of the domain and a surface that needs to be smoothed in only a small region doesn't require a change in shape everywhere. Hence it is important that a smoothing algorithm deal with locality. To illustrate the effectiveness of the algorithm in fairing over a specified region of the surface we take a second example with the same original surface but with control points randomly perturbed in the range 5 to 8 in u and v. The partially perturbed torus section is then smoothed over the u, v section $[a, b] \times [c, d] = [0.589, 1.963] \times [0.589, 1.963]$, these values corresponding to the parametric region of the surface that was modified. Figure 4.34 displays the deviation map and the perturbed surface in environment mapped form. Table 4.5 gives the evaluation measures and figures 4.35 to 4.38 the results in wireframe and environment mapped forms. Figures 4.39 to 4.41 provide displays of the Gaussian and mean curvature maps. Again functionals J_3 and J_4 appear to do slightly better in producing a surface that comes 'close' to matching the original.

Evaluation	*Optimisation Criteria*				*Perturbed*	*Original*
Measure	$min \longrightarrow J^2_{uv}$	$min \longrightarrow J^2_{pas}$	$min \longrightarrow J^3_{uv}$	$min \longrightarrow J^3_{pas}$	*Surface*	*Surface*
$J^2_u\,(J_1)$	1352.63	1379.37	1725.06	1730.19	2101.09	1590.56
$J^2_v\,(J_2)$	141.014	144.57	181.643	184.154	746.313	166.585
$J^2_{uv}\,(J_3)$	2028.89	2084.11	2612.67	2656.61	2191.91	2452.2
$J^2_{pas}\,(J_4)$	2372.77	2425.77	2930.53	3114.06	3085.19	2769.18
$J^3_u\,(J_5)$	5318.45	4783.49	14373.8	12697.7	5753.77	1639.92
$J^3_v\,(J_6)$	1033.56	865.488	1641.41	1343.44	11681.6	212.437
$J^3_{uv}\,(J_7)$	6231.48	5599.14	15305.7	13616.7	11767.9	1851.88
$J^3_{pas}\,(J_8)$	23024.1	20369	27006.7	28295.7	117943	7741.65
J_K	4.17905e-21	4.30838e-21	7.82576e-21	8.21328e-21	5.82889e-21	7.29018e-21
J_H	0.00439791	0.00447648	0.00631404	0.00616064	0.00424065	0.00562191
J_A	0.0703665	0.0716236	0.101025	0.0985702	0.0678515	0.0899506
J_T	0.000903488	0.000862518	0.00176585	0.00127207	0.000674266	0.00091451
Area	495.396	495.279	502.57	504.153	529.898	495.295
Max dev perturbed	0.825255	0.820711	0.817691	0.792637	0.0	0.777434
Max dev Original	0.126234	0.122933	0.287779	0.302422	0.777434	0.0

Table 4.5: Smoothing measures and values for partially perturbed torus section smoothed with factor 0.1 over the range $[0.589, 1.963]$ in u and v

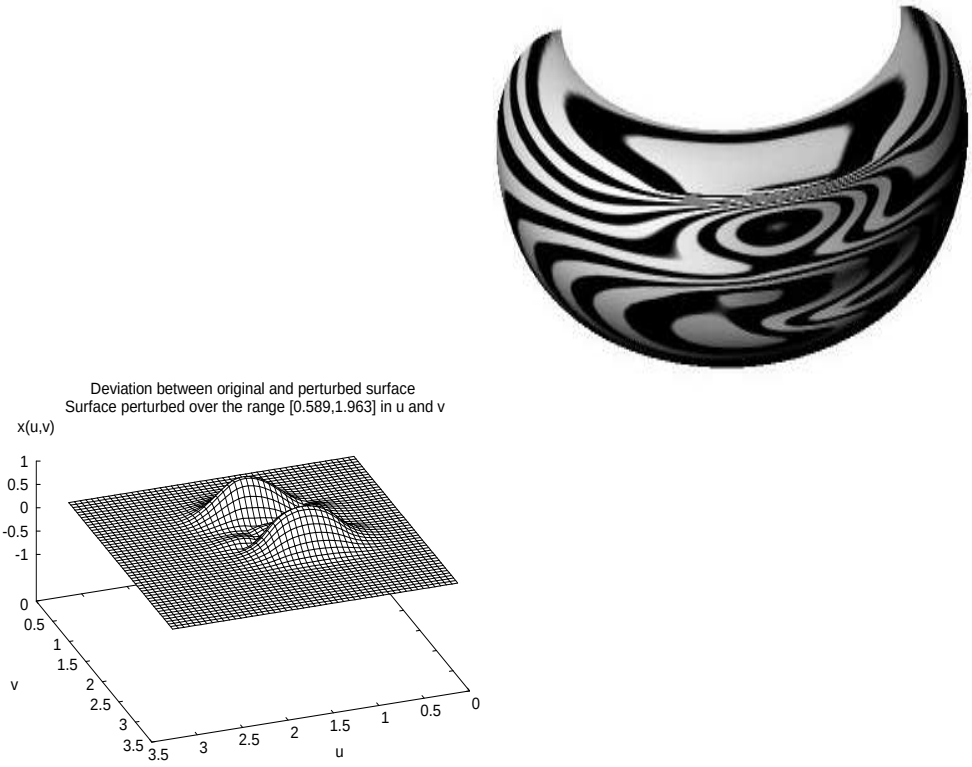


Figure 4.34: Partially perturbed surface, deviation from original and environment mapped views

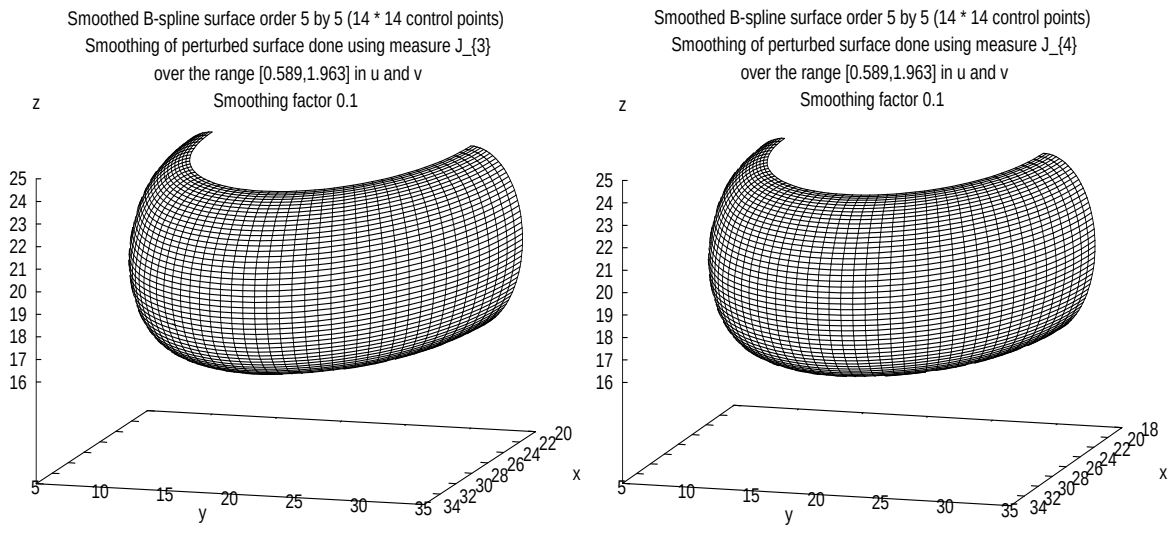


Figure 4.35: Smoothed surfaces using J_3 and J_4 functionals to partially perturbed torus section

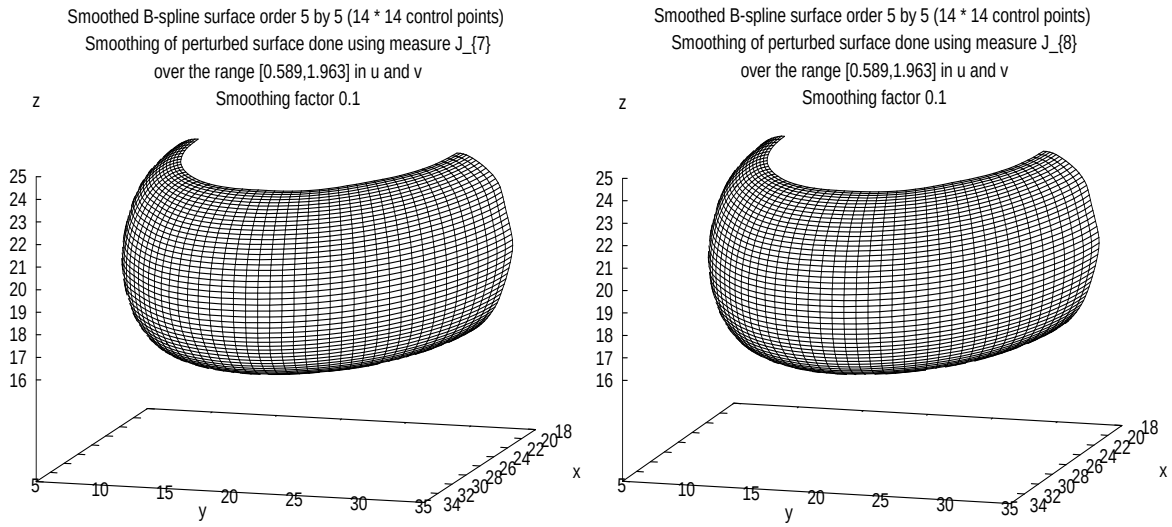


Figure 4.36: Smoothed surfaces using J_7 and J_8 functionals to partially perturbed torus section

Figure 4.37: Environment maps for J_3 and J_4 smoothed surfaces

Figure 4.38: Environment maps for J_7 and J_8 smoothed surfaces

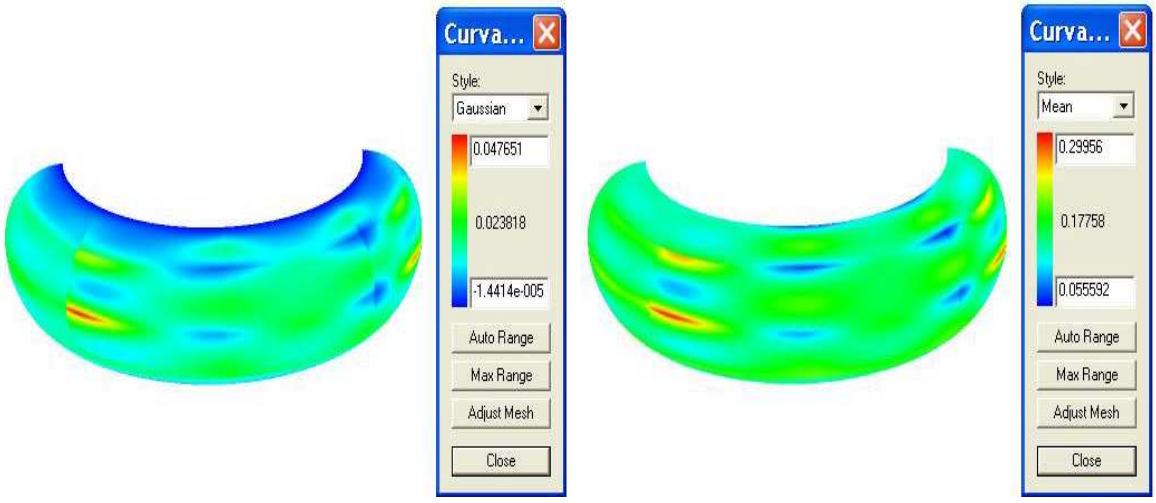


Figure 4.39: Gaussian and mean curvature map of J_3 smoothed surface

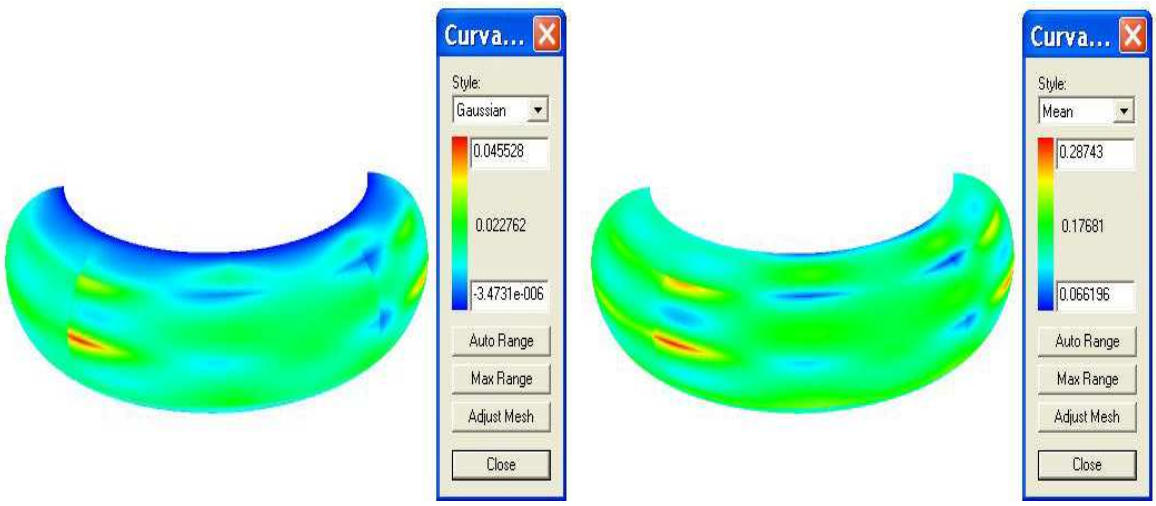


Figure 4.40: Gaussian and mean curvature map of J_4 smoothed surface

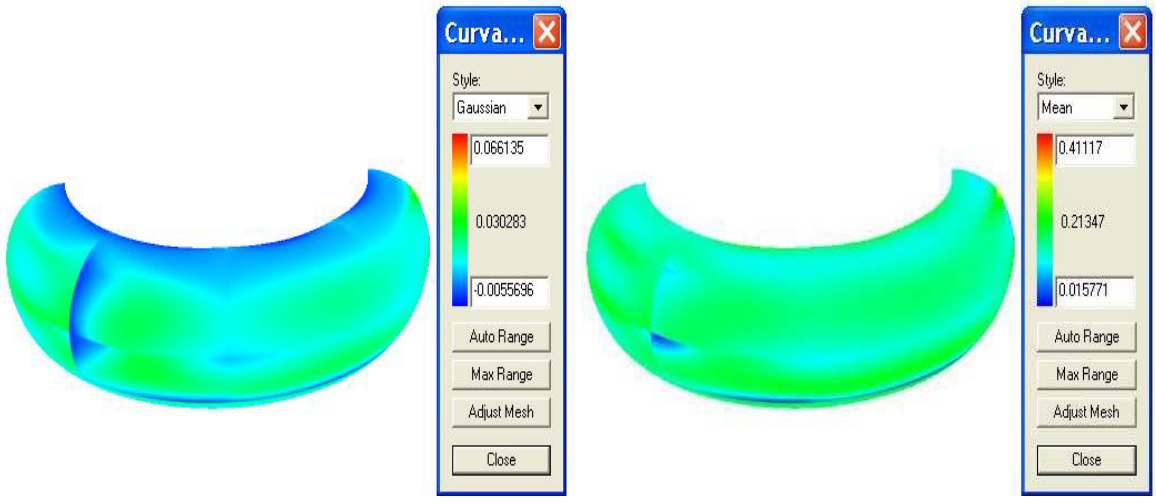


Figure 4.41: Gaussian and mean curvature map of J_7 smoothed surface

4.4.5 Alternative computational method

An alternative computational method for smoothing an existing surface based on equation 4.7 is to take individual terms, $\alpha_{ij}\mathbf{M}_i^u\mathbf{dM}_j^v$, resulting from minimisation of the smoothing functional and to repeatedly solve as necessary the following two term Kronecker equation:

$$sm * \alpha_{ij}\mathbf{M}_i^u\mathbf{dM}_j^v + \mathbf{CdD} = \mathbf{E}. \tag{4.13}$$

The motivation for this approach come from the paper of Gardiner et al, [23], who document an algorithm for solving this type of matrix equation (called the Sylvester equation) based on a Hessenberg-Schur method. Their implementation has a computational cost $O(n^3)$, where $n = max(p, q)$, whereas the conventional LU factorisation for the Kronecker product method is $O((pq)^3)$ ([24]).

We solve 4.13 for $\mathbf{d}$ and then take this new surface as input to the equation for applying the next term in the smoothing functional. Algorithm 4.5 presents the steps involved.

We test this alternative technique for smoothing as in examples 1 and 2. Table 4.6 presents the evaluation measures for the perturbed torus section smoothed over its whole domain in u and v. Figures 4.43 and 4.44 show the surfaces in wireframe form and figures 4.45 and 4.46 illustrate the corresponding Guassian curvature maps.

Finally, we apply algorithm 4.5 to the case of the partially perturbed torus section. Table 4.7 gives the numerical results, figures 4.47 and 4.48 display the resulting surfaces and figure 4.49 two of the corresponding Gaussian surfaces. The results as shown for these two tests are comparable in terms of quality to the Kronecker product method.

Algorithm 4.5: Smoothing an existing B-spline surface, alternative method

$\mathbf{x}(u,v) = \sum_{i=1}^{p}\sum_{j=1}^{q} \mathbf{d}_{ij} N_{i,k}(u) N_{j,l}(v), \quad$ *knot set* $\quad (u_i)_{i=1}^{p+k} \times (v_j)_{j=1}^{q+l}$

smoothing functional, $\int_a^b \int_c^d \sum_{i,j=0, i+j=m}^{m} \alpha_{ij} \left\| \frac{\partial^{(m)}\mathbf{x}}{\partial u^{(i)} \partial v^{(j)}} \right\|^2 du\, dv$

method uses Gardiner et al algorithm for solving system of the form $\mathbf{AXB^T + CXD^T = E}$

1. *create the basis function set* $\left(\mathtt{N_{i,k}(u)N_{j,l}(v)}\right)_{\mathtt{i,j=1}}^{\mathtt{p,q}}$
2. *create the minimisation matrices* $\mathtt{M_1^u}, \ldots, \mathtt{M_m^u}, \mathtt{M_1^v}, \ldots, \mathtt{M_m^v}$ *from the basis function set*
3. *create matrices of* $\mathtt{u, v}$ *basis functions of* $\mathtt{x}$ *evaluated at knot averages*
   $\mathtt{C} = \left(\mathtt{N_{i,k}}(\eta_\mathtt{j})\right)_{\mathtt{i,j=1}}^{\mathtt{p}}, \mathtt{D} = \left(\mathtt{N_{i,l}}(\psi_\mathtt{j})\right)_{\mathtt{i,j=1}}^{\mathtt{q}}$
4. *for each* $\alpha_{\mathtt{ij}}$
   4.1 solve the system $\mathtt{sm} * \alpha_{\mathtt{ij}} \mathbf{M_i^u d M_j^v} + \mathbf{CdD} = \mathbf{E}$
   4.2 build B-spline surface from $\mathbf{d}$ *and knot sets* $(u_i)_{i=1}^{p+k} \times (v_j)_{j=1}^{q+l}$
   4.3 update surface for next iteration

Figure 4.42: Algorithm 4.5: Smoothing an existing surface, alternative method

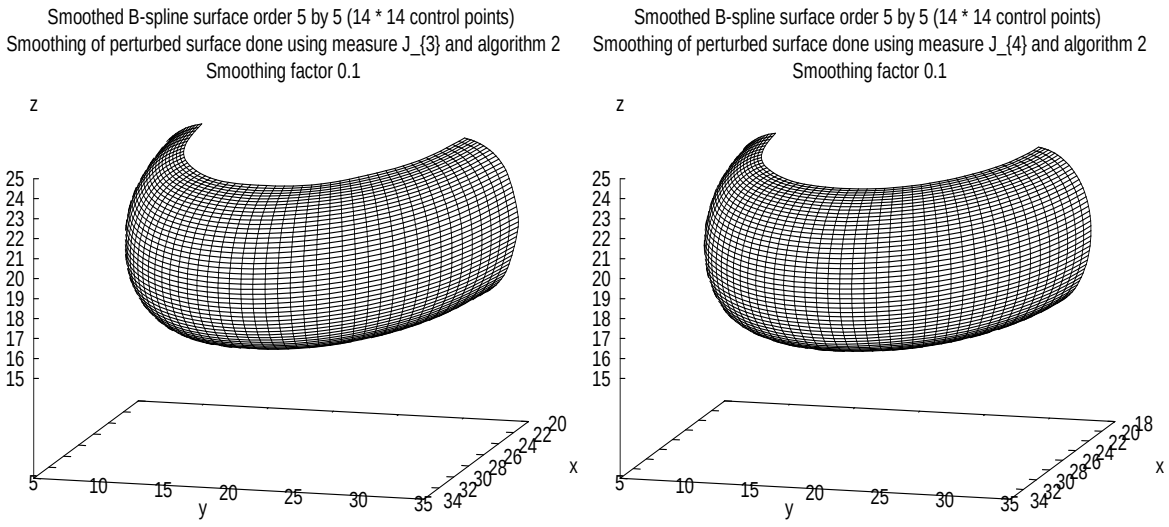


Figure 4.43: Smoothed surfaces using functionals J_3 and J_4

Evaluation Measure	*Optimisation Criteria*				*Perturbed Surface*	*Original Surface*
	$min \longrightarrow J^2_{uv}$	$min \longrightarrow J^2_{pas}$	$min \longrightarrow J^3_{uv}$	$min \longrightarrow J^3_{pas}$		
J^2_u	1246.65	1670.78	2681.17	2756.55	4764.59	1590.56
J^2_v	119.899	201.454	339.418	313.967	14679.5	166.585
J^2_{uv}	1848.42	2431.74	3871.36	3745.13	17505.6	2452.2
J^2_{pas}	2157.61	2752.92	4252.16	4026.15	18553	2769.18
J^3_u	5889.75	4356.95	30075.8	32151.3	73210.1	1639.92
J^3_v	786.363	1567.93	7359.36	4405.28	747252	212.437
J^3_{uv}	6699.78	6910.8	44721.6	37220.3	758863	1851.88
J^3_{pas}	9338.57	10440.6	56922.6	36770.4	1.3747e+06	7741.65
J_K	3.99326e-21	1.22622e-20	2.0972e-19	7.92166e-20	7.88762e-07	7.29018e-21
J_H	0.00370058	0.00670396	0.0212138	0.0173101	74334.7	0.00562191
J_A	0.0592092	0.107263	0.339421	0.276962	1.18935e+06	0.0899506
J_T	0.000490429	0.00228712	0.0799141	0.0228024	2.50147e+12	0.00091451
Area	489.079	498.559	498.694	493.25	599.906	495.295
Max dev perturbed	1.2607	1.28866	1.30139	1.27686	0.0	1.28526
Max dev Original	0.284613	0.206033	0.33812	0.328612	1.28526	0.0

Table 4.6: Smoothing measures and values for totally perturbed torus section smoothed with factor 0.1 over complete surface, algorithm 4.5

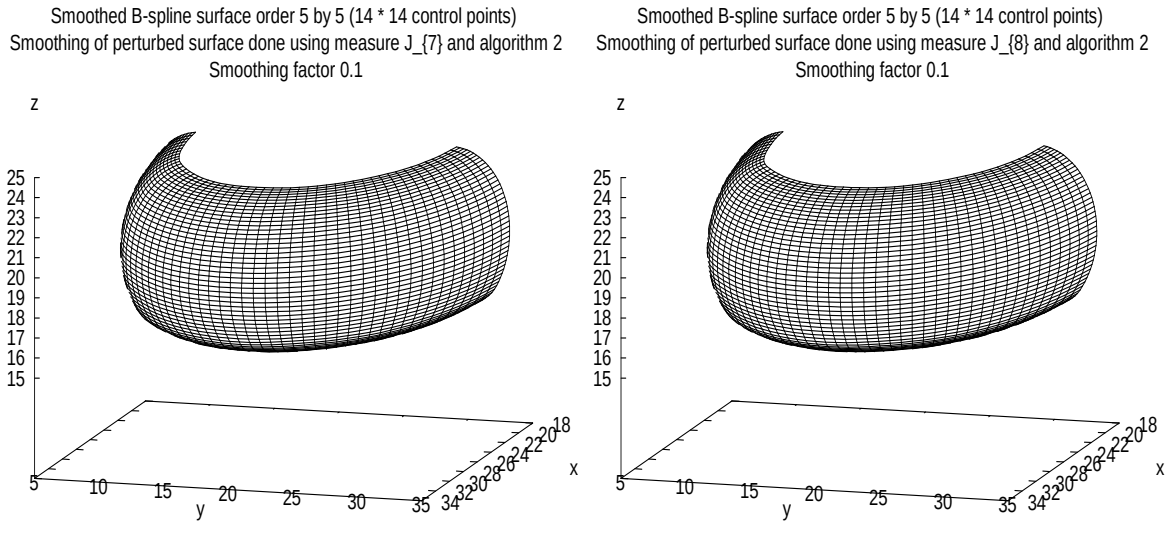


Figure 4.44: Smoothed surfaces using functionals J_7 and J_8

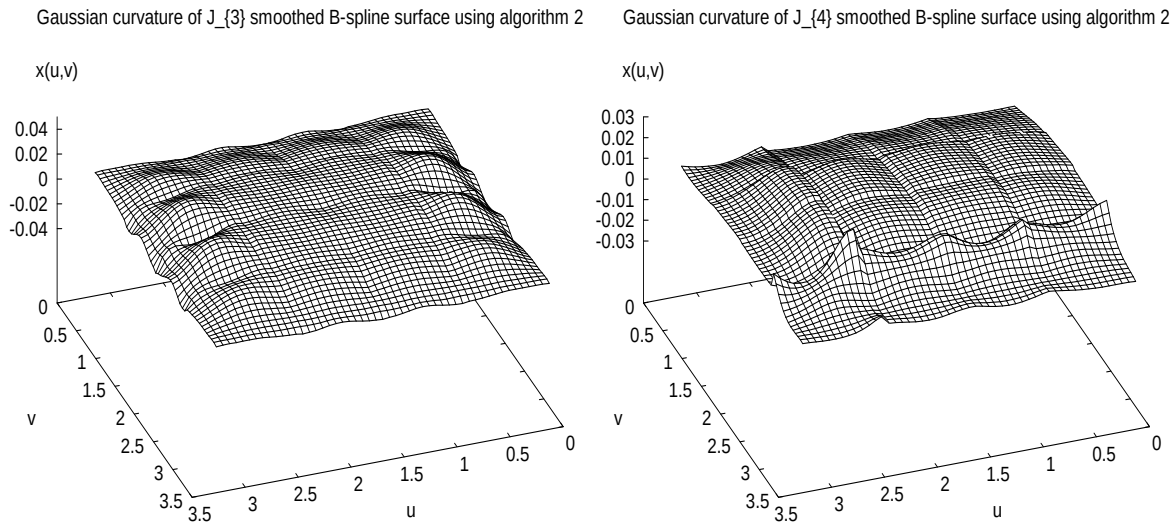


Figure 4.45: Gaussian curvature of J_3 and J_4 surfaces

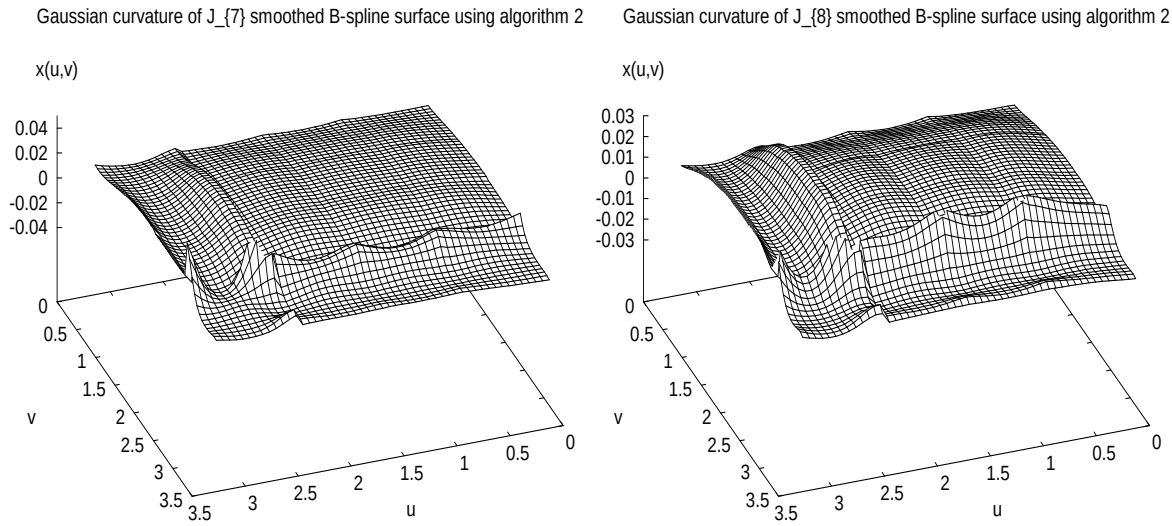


Figure 4.46: Gaussian curvature of J_7 and J_8 surfaces

Evaluation	*Optimisation Criteria*				*Perturbed*	*Original*
Measure	$min \longrightarrow J^2_{uv}$	$min \longrightarrow J^2_{pas}$	$min \longrightarrow J^3_{uv}$	$min \longrightarrow J^3_{pas}$	*Surface*	*Surface*
$J^2_u\ (J_1)$	1289.15	1684.93	1750.29	1696.88	2101.09	1590.56
$J^2_v\ (J_2)$	146.804	206.298	213.909	189.854	746.313	166.585
$J^2_{uv}\ (J_3)$	1931.78	2655.05	2758.68	2626.73	2191.91	2452.2
$J^2_{pas}\ (J_4)$	2300.35	3077.17	3199.48	3049.73	3085.19	2769.18
$J^3_u\ (J_5)$	9917.74	5024.12	9132.25	7567.91	5753.77	1639.92
$J^3_v\ (J_6)$	1961.82	1361.52	1723	1197.81	11681.6	212.437
$J^3_{uv}\ (J_7)$	12414.6	6874.94	10856.1	8177.97	11767.9	1851.88
$J^3_{pas}\ (J_8)$	42270	24963.7	33691.1	21074.4	117943	7741.65
J_K	3.64812e-21	2.48069e-20	2.71768e-20	9.45523e-21	5.82889e-21	7.29018e-21
J_H	0.00495916	0.00918412	0.00976201	0.00667339	0.00424065	0.00562191
J_A	0.0793465	0.146946	0.156192	0.106774	0.0678515	0.0899506
J_T	0.00175634	0.00758169	0.00921676	0.00188962	0.000674266	0.00091451
Area	497.469	500.694	501.84	501.51	529.898	495.295
Max dev perturbed	0.858635	0.809541	0.81418	0.816421	0.0	0.777434
Max dev Original	0.220561	0.210453	0.291759	0.292765	0.777434	0.0

Table 4.7: Smoothing/Fairness measures and values for partially perturbed torus section smoothed with factor 0.1 over a range $[0.589, 1.963]$, algorithm 4.5 used)

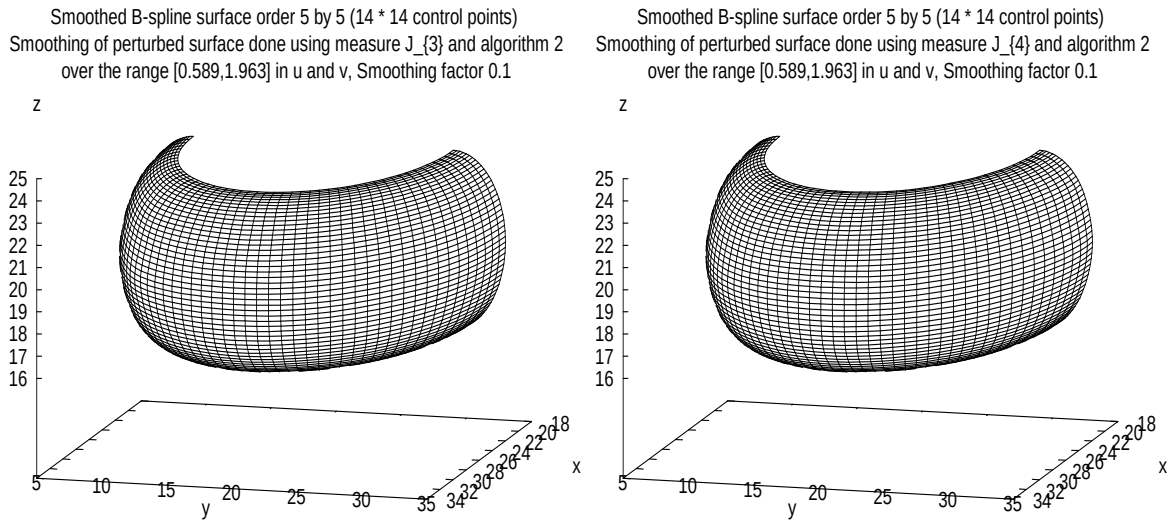


Figure 4.47: Smoothed surface using J_3, J_4 functionals (alg. 4.5)

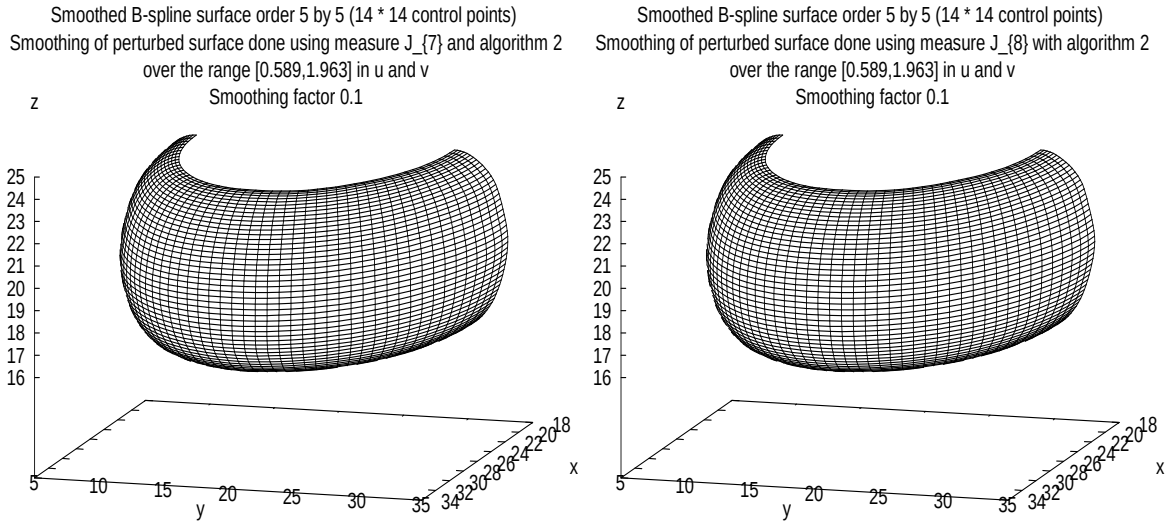


Figure 4.48: Smoothed surfaces using J_7 and J_8 functionals (alg. 4.5)

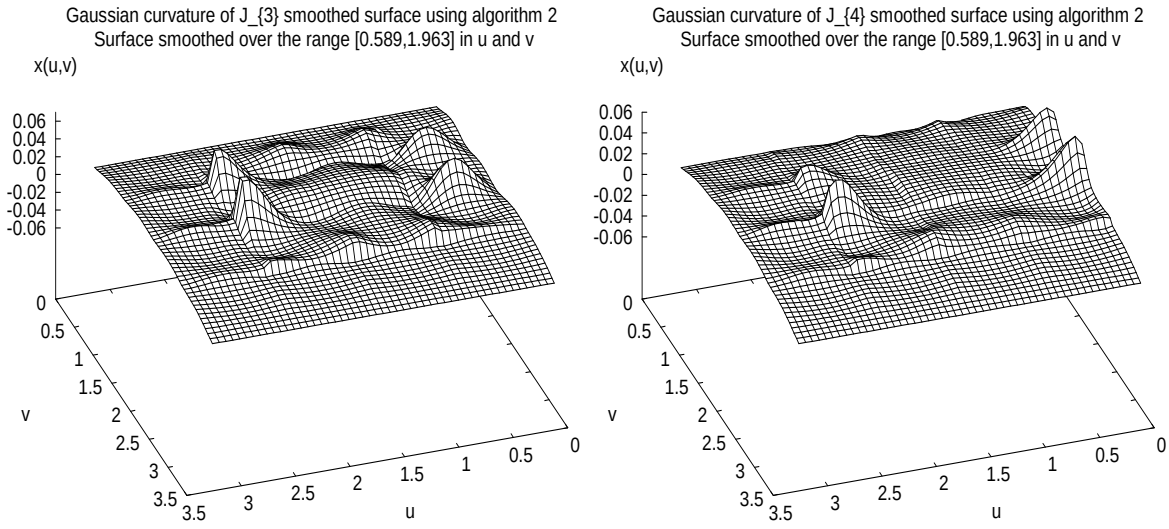


Figure 4.49: Gaussian curvature of J_3 and J_4 smoothed surfaces (alg. 4.5)

4.5 Volume Modelling

The development of algorithms surrounding the modelling and visualisation of three (and higher) dimensional data is a relatively new endeavor held in check in the past due to the large amount of computing power and memory required to process such models. Over the past decade or so there has been much interest and research into the use of 'solid' models and the potential they have to model processes in a more accurate and realistic manner. The fields of geometric modelling, image processing, data fitting and visualisation, medical imaging and others have been active in developing tools and techniques to simulate structure and behaviour of volumetric objects.

In geometric modelling tensor product solids or volumes are a natural extension of tensor product surfaces to a third dimension. As shown in section 2.5, algorithms and properties of B-splines extend naturally from the curve/surface case to volumes. B-spline volumes have local control for manipulation, satisfy the convex hull property and can be efficiently evaluated and subdivided. Derivatives and integrals can be found analytically and represented as entities of the same type. B-spline volumes have been used by the solid modelling community to model freeform primitives, Joy, [41], [42], Lasser, [50], [51], to effect freeform deformations of space curves and surfaces, Sedeberg & Parry [78], and to model both geometry and material properties of composites, Stanton & Crain [79], Marston & Dutta, [55].

In traditional solid modelling the emphasis is on describing the boundaries of an object where it is assumed that the interior of the object is homogeneous. The potential to analyse and build objects which are composed of multiple materials and/or have varying material properties throughout (such as density) has brought the need for an enhanced solid model which is capable of containing this type of information. Such a model is called a heterogeneous solid model. Tensor product functions can be used for describing such properties which vary through space in addition to geometry.

The medical research community has also been active in the use of free-form solids. 3D images from magnetic resonance and CAT scanned data have been used to construct free-form volumes for the elastic modelling of soft tissue, Bro-Neilson [3], Roth et al [74], and the tracking of deformations of human organs over time, Radeva et al [71], [72]. Assuming a regular grid of voxels, the conversion to a tensor product solid is carried out here by assigning a value which corresponds to some property of the object being modelled, such as density. By using an interpolatory or approximative (given the typical size of 3D images) data fitting technique such as least squares, the control points of a functional B-spline volume can be found which interpolate or approximate the voxel value.

A tensor product solid can be described in either parametric or functional form. The para-

metric function is written as

$$\mathbf{x}(u,v,w) = \begin{pmatrix} x(u,v,w) & y(u,v,w) & z(u,v,w) \end{pmatrix}$$

describing some freeform homogeneous solid, or as,

$$\mathbf{x}(u,v,w) = \begin{pmatrix} x(u,v,w) & y(u,v,w) & z(u,v,w) & p(u,v,w) \end{pmatrix}$$

where p is some property at the point (u,v,w). This representation can be used to describe a free-form volume which has heterogeneous properties throughout its interior and on its surfaces. The geometry and the material properties of the object are explicitly represented and the surfaces of the object are the isoparametric surfaces at the limiting parameter values. However, there are two drawbacks with the parametric form: first, we are limited to modelling objects that are six sided (or tubular if periodic in one or two directions) and second, that there is no direct way to get material properties at a particular point in space. The user must specify a parameter triple instead. As in the curve/surface case a point/parameter inversion algorithm such as a generalisation of Hoschek's, [38], can be used at a computational cost to overcome this problem.

The other form of the tensor product solid is the nonparametric (or functional) representation given by $p = f(u,v,w)$, where p describes a material property, such as temperature or density, throughout an object as a function of u,v,w. It can also be written parametrically as

$$\mathbf{x}(u,v,w) = \begin{pmatrix} u & v & w & f(u,v,w) \end{pmatrix}.$$

Here the parameter values correspond directly to a point in space $u = x, v = y, w = z$ and are bounded by the limits of the bounding box around the object ($u_{min} \leq u \leq u_{max}, v_{min} \leq v \leq v_{max}, w_{min} \leq w \leq w_{max}$). The geometry of the boundaries of the object are stored explicitly as separate curve and surface functions, or they are determined by setting p equal to a constant. A wide range of applications produce this type of data consisting of three independent and one scalar dependent variable, (x_i, y_i, z_i, F_i). For example, MRI and CAT scans as mentioned above, measurements of mineral concentrations from core samples, 3D CFD simulations and pressure or temperature readings. Although not dealt with explicitly in this report it is also possible to have several scalar dependent variables such as used in certain CFD simulations where the velocity and the pressure of the flow are measured at a point.

Although there are papers reviewing extensions of tensor product data fitting algorithms to the volume case, for example Nielson, [63], little has been written on generalisations of smoothing algorithms developed for the curve and surface case. We concentrate here on the use of volume B-splines to produce 'smooth' approximations to regular (that is based on a rectangular topology)

data sets by extending and generalising the smoothing algorithms developed in sections 4.2.1 and 4.4.1 for curves and surfaces. As before we take the least squares algorithm and combine it with a smoothing measure based on a suitable generalisation of a curve/surface based energy term. The combined functional is then minimised to produce a volume that both approximates and is 'smooth' in a number of ways in which we can quantify.

4.5.1 Volume smoothing combined with least squares data fitting

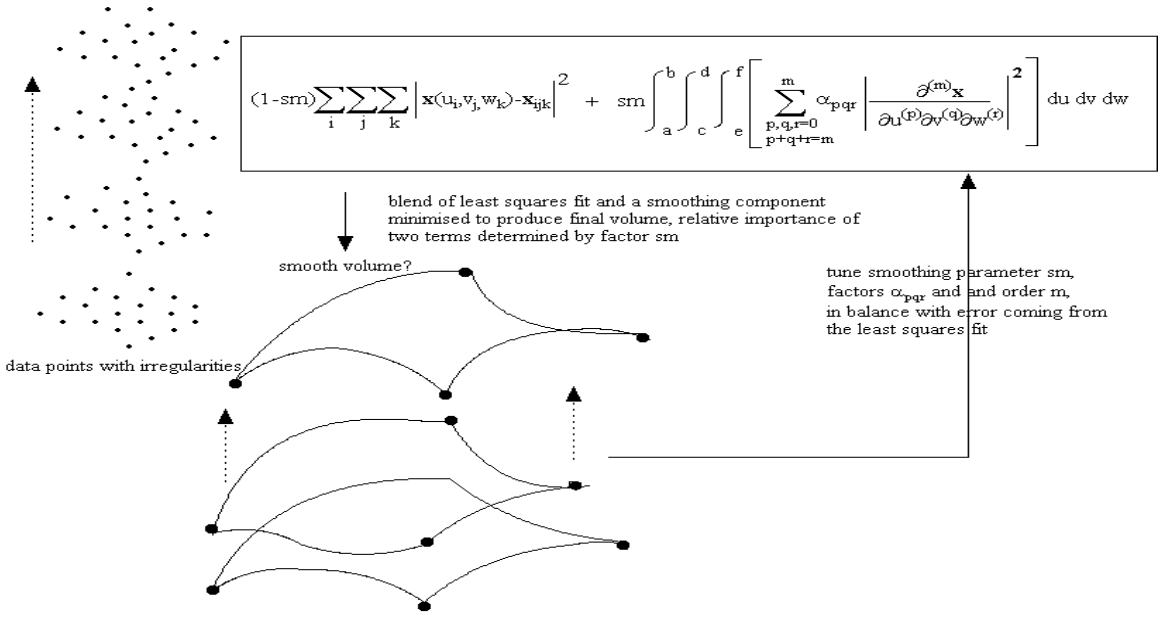


Figure 4.50: Incorporating volume smoothing process with least squares construction

Figure 4.50 illustrates the setup for combining volume data fitting with smoothing. Starting with a given set of data points $\left(\mathbf{x}_{ijk}\right)_{i,j,k=1}^{M,N,P}$, with u, v, w parameterisations , $(\tau_i)_{i=1}^{M}, (\mu_j)_{j=1}^{N}, (\nu_k)_{k=1}^{P}$, respectively, we seek a B-spline solution

$$\mathbf{x}(u,v,w) = \sum_{i=1}^{p}\sum_{j=1}^{q}\sum_{k=1}^{r}\mathbf{d}_{ijk}N_{i,l}(u)N_{j,m}(v)N_{k,n}(w), \quad (u_i)_{i=1}^{p+l} \times (v_j)_{j=1}^{q+m} \times (w_k)_{k=1}^{n}$$

to the combined least squares/smoothing functional:

$$\mathrm{I}(\mathbf{x}) = (1-sm)\mathrm{I}_{lsq}(\mathbf{x}) + sm\mathrm{I}_{smooth}(\mathbf{x})$$

$$= (1-sm)\sum_{i=1}^{M}\sum_{j=1}^{N}\sum_{k=1}^{P}\|\mathbf{x}(\tau_i,\mu_j,\nu_k)-\mathbf{x}_{ijk}\|^2+sm\int_V\left(\sum_{i,j,k=0,i+j+k=s}^{s}\alpha_{ijk}\left\|\frac{\partial^{(s)}\mathbf{x}}{\partial u^{(i)}\partial v^{(j)}\partial w^{(k)}}\right\|^2\right)du\,dv\,dw.$$

The minimisation of the least squares error term, $\partial \mathrm{I}_{lsq}/\partial \mathbf{d} = 0$, gives the following matrix equation

$$\mathbf{A}\odot_i\mathbf{d}\odot_j\mathbf{B}\odot_k\mathbf{C}=\mathbf{D},$$

where

$$\mathbf{d}=\Big(\mathbf{d}_{ijk}\Big)_{i,j,k=1}^{p,q,r},\quad \mathbf{A}=\Big(\sum_{i=1}^{M}N_{s,l}(\tau_i)N_{t,l}(\tau_i)\Big)_{s,t=1}^{p},\quad \mathbf{B}=\Big(\sum_{j=1}^{N}N_{s,m}(\mu_j)N_{t,m}(\mu_j)\Big)_{s,t=1}^{q},$$

$$\mathbf{C}=\Big(\sum_{k=1}^{P}N_{s,n}(\nu_k)N_{t,n}(\nu_k)\Big)_{s,t=1}^{r},\quad \mathbf{D}=\Big(\sum_{i=1}^{M}\sum_{j=1}^{N}\sum_{k=1}^{P}N_{r_1,l}(\tau_i)N_{s_1,m}(\mu_j)N_{t_1,n}(\nu_k)\mathbf{x}_{ijk}\Big)_{r_1,s_1,t_1=1}^{p,q,r}.$$

From 3.13 the minimisation of the smoothness term gives

$$\mathbf{0}=\frac{\partial \mathrm{I}_{smooth}}{\partial \mathbf{d}}=\sum_{i_1,j_1,k_1}\alpha_{i_1j_1k_1}\mathbf{M}_{i_1}^{u}\odot_i\mathbf{d}\odot_j\mathbf{M}_{j_1}^{v}\odot_k\mathbf{M}_{k_1}^{w}.$$

By combining this with the least squares error term we obtain the following system of equations for the control point matrix3D $\mathbf{d}$:

$$(1-sm)\mathbf{A}\odot_i\mathbf{d}\odot_j\mathbf{B}\odot_k\mathbf{C}+sm\sum_{i_1,j_1,k_1}\alpha_{i_1j_1k_1}\mathbf{M}_{i_1}^{u}\odot_i\mathbf{d}\odot_j\mathbf{M}_{j_1}^{v}\odot_k\mathbf{M}_{k_1}^{w}=\mathbf{D}. \tag{4.14}$$

We solve this system by using the Kronecker product generalisation of 2.3 to matrix3D entities, reducing it to the following conventional matrix equation:

$$\Big((1-sm)\mathbf{C}\otimes\mathbf{B}\otimes\mathbf{A}+sm\sum_{i_1,j_1,k_1}\alpha_{i_1j_1k_1}\mathbf{M}_{k_1}^{w}\otimes\mathbf{M}_{j_1}^{v}\otimes\mathbf{M}_{i_1}^{u}\Big)vec(\mathbf{d})=vec(\mathbf{D}). \tag{4.15}$$

Algorithm 4.6 presents the steps required to compute the B-spline solution for this case.

4.5.2 Smoothing an existing B-spline volume

Figure 4.52 illustrates the local/global approach to smoothing an existing volume, highlighting the functional/parametric distinction. For smoothing an existing B-spline volume we combine the matrices derived from the minimisation of the smoothing functional with matrices derived from evaluating the basis functions and B-spline volume at the knot averages, $(\eta_i)_{i=1}^{p},(\psi_j)_{j=1}^{q},(\zeta_k)_{k=1}^{r}$, where

$$\eta_i=\frac{1}{l-1}(u_{i+1}+\ldots+u_{i+l-1}),\quad \psi_j=\frac{1}{m-1}(v_{j+1}+\ldots+v_{j+m-1}),\quad \zeta_k=\frac{1}{n-1}(w_{k+1}+\ldots+w_{k+n-1}).$$

The matrices in question

$$\mathbf{A} = \big(N_{i,l}(\eta_j)\big)_{i,j=1}^{p}, \quad \mathbf{B} = \big(N_{i,m}(\psi_j)\big)_{i,j=1}^{q},, \quad \mathbf{C} = \big(N_{i,n}(\zeta_j)\big)_{i,j=1}^{r}, \quad \mathbf{D} = \big(\mathbf{x}(\eta_i,\psi_j,\zeta_k)\big)_{i,j,k=1}^{p,q,r},$$

are combined with the smoothing functional equation derived from minimising I_{smooth}:

$$\mathbf{0} = \frac{\partial \mathrm{I}_{smooth}}{\partial \mathbf{d}} = \sum_{i_1,j_1,k_1} \alpha_{i_1 j_1 k_1} \mathbf{M}_{i_1}^{u} \odot_i \mathbf{d} \odot_j \mathbf{M}_{j_1}^{v} \odot_k \mathbf{M}_{k_1}^{w},$$

to get the matrix equation

$$\Big(\mathbf{C}\otimes\mathbf{B}\otimes\mathbf{A} + sm \sum_{i_1,j_1,k_1} \alpha_{i_1 j_1 k_1} \mathbf{M}_{k_1}^{w} \otimes \mathbf{M}_{j_1}^{v} \otimes \mathbf{M}_{i_1}^{u}\Big) vec(\mathbf{d}) = vec(\mathbf{D}). \tag{4.16}$$

Algorithm 4.6: B-spline volume least squares fitting with smoothing

data set $\big(\mathtt{x_{ijk}}\big)_{\mathtt{i,j,k=1}}^{\mathtt{M,N,P}}$

smoothing functional, $\int_{\mathtt{V}} \Big(\sum_{\mathtt{i,j,k=0,i+j+k=s}}^{\mathtt{s}} \alpha_{\mathtt{ijk}} \left\| \frac{\partial^{(\mathtt{s})}\mathtt{x}}{\partial \mathtt{u}^{(\mathtt{i})} \partial \mathtt{v}^{(\mathtt{j})} \partial \mathtt{w}^{(\mathtt{k})}} \right\|^2 \Big) \mathtt{du\,dv\,dw}$

1. *create* $\mathtt{u,v,w}$ *parameterisations* $(\tau_\mathtt{i})_{\mathtt{i=1}}^{\mathtt{M}}, (\mu_\mathtt{i})_{\mathtt{i=1}}^{\mathtt{N}}, (\nu_\mathtt{k})_{\mathtt{k=1}}^{\mathtt{P}}$ *from the data points*
2. *create suitable knot sets* $(\mathtt{u_i})_{\mathtt{i=1}}^{\mathtt{p+l}}, (\mathtt{v_j})_{\mathtt{j=1}}^{\mathtt{q+m}}, (\mathtt{w_k})_{\mathtt{k=1}}^{\mathtt{r+n}}$
3. *create the basis function set* $\big(\mathtt{N_{i,l}(u)N_{j,m}(v)N_{k,n}(w)}\big)_{\mathtt{i,j,k=1}}^{\mathtt{p,q,r}}$
4. *create minimisation matrices* $\mathtt{M_1^u},\ldots,\mathtt{M_s^u},\ \mathtt{M_1^v},\ldots,\mathtt{M_s^v}\ \ \mathtt{M_1^w},\ldots,\mathtt{M_s^w}$**,** *from the basis function set*
5. *create the least squares matrices,*
   $\mathtt{A} = (\sum_{\mathtt{i=1}}^{\mathtt{M}} \mathtt{N_{s,l}}(\tau_\mathtt{i})\mathtt{N_{t,l}}(\tau_\mathtt{i}))_{\mathtt{s,t=1}}^{\mathtt{p}},\ \mathtt{B} = (\sum_{\mathtt{j=1}}^{\mathtt{N}} \mathtt{N_{s,m}}(\mu_\mathtt{j})\mathtt{N_{t,m}}(\mu_\mathtt{j}))_{\mathtt{s,t=1}}^{\mathtt{q}},\ \mathtt{C} = (\sum_{\mathtt{k=1}}^{\mathtt{P}} \mathtt{N_{s,n}}(\nu_\mathtt{k})\mathtt{N_{t,n}}(\nu_\mathtt{k}))_{\mathtt{s,t=1}}^{\mathtt{r}}$
6. *create the right hand side matrix,* $\mathtt{D} = \Big(\sum_{\mathtt{i=1}}^{\mathtt{M}} \sum_{\mathtt{j=1}}^{\mathtt{N}} \sum_{\mathtt{k=1}}^{\mathtt{P}} \mathtt{N_{r_1,l}}(\tau_\mathtt{i})\mathtt{N_{s_1,m}}(\mu_\mathtt{j})\mathtt{N_{t_1,n}}(\nu_\mathtt{k})\mathbf{x}_{\mathtt{ijk}}\Big)_{\mathtt{r_1,s_1,t_1=1}}^{\mathtt{p,q,r}}$
7. *for a given smoothing factor* $\mathtt{sm}$ *form the Kronecker product matrix*
   $\mathtt{E} = (\mathtt{1}-\mathtt{sm})\mathbf{C}\otimes\mathtt{B}\otimes\mathtt{A}$
8. *for a given smoothing factor* $\mathtt{sm}$ *form the Kronecker product matrix*
   $\mathtt{F} = \mathtt{sm} \sum_{\mathtt{i_1,j_1,k_1}} \alpha_{\mathtt{i_1 j_1 k_1}} \mathtt{M_{k_1}^w} \otimes \mathtt{M_{j_1}^v} \otimes \mathtt{M_{i_1}^u}$
9. *solve the system* $\mathtt{(E+F)vec(d)=vec(D)}$
10. *reconstruct the B-spline control point matrix3D from the vector* $\mathtt{vec(d)}$

Figure 4.51: Algorithm 4.6: Volume least squares fitting combined with smoothing

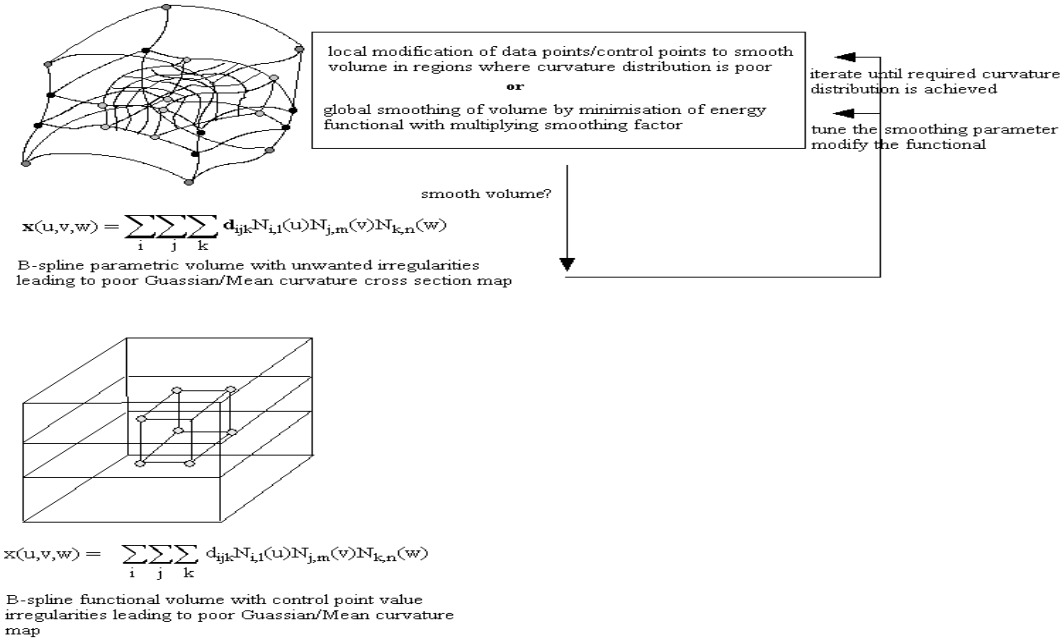


Figure 4.52: Smoothing an existing volume

Using 4.16, algorithm 4.7 extends algorithm 4.4 to the volume case.

4.6 Volume Smoothing Examples: functional case

4.6.1 Example 1

To test the effectiveness of algorithm 4.7 in smoothing existing volumes we test it first on an example functional B-spline volume of order 4 in u, v and w with 10 control points in each direction:

$$x(u,v,w) = \sum_{i=1}^{10} \sum_{j=1}^{10} \sum_{k=1}^{10} d_{ijk} N_{i,4}(u) N_{j,4}(v) N_{k,4}(w).$$

The control points d_{ijk} are set to be constant in a given k level, varying linearly in a range from -5 at $k = 1$ to 5 at $k = 10$. As before, to simulate a perturbed volume we disturb the control points using a random number generator such that the maximum absolute deviation of a given control point is 0.1. The perturbed volume is smoothed using algorithm 4.6 and employing the

Algorithm 4.7: Smoothing an existing B-spline volume

$\mathtt{x}(\mathtt{u},\mathtt{v},\mathtt{w}) = \sum_{\mathtt{i}=1}^{\mathtt{p}} \sum_{\mathtt{j}=1}^{\mathtt{q}} \sum_{\mathtt{k}=1}^{\mathtt{r}} \mathtt{d}_{\mathtt{ijk}} \mathtt{N}_{\mathtt{i},\mathtt{l}}(\mathtt{u}) \mathtt{N}_{\mathtt{j},\mathtt{m}}(\mathtt{v}) \mathtt{N}_{\mathtt{k},\mathtt{n}}(\mathtt{w}), \quad (\mathtt{u}_\mathtt{i})_{\mathtt{i}=1}^{\mathtt{p}+\mathtt{l}} \times (\mathtt{v}_\mathtt{j})_{\mathtt{j}=1}^{\mathtt{q}+\mathtt{m}} \times (\mathtt{w}_\mathtt{k})_{\mathtt{k}=1}^{\mathtt{r}+\mathtt{n}}$

smoothing functional, $\int_{\mathtt{V}} \Big(\sum_{\mathtt{i},\mathtt{j},\mathtt{k}=0,\mathtt{i}+\mathtt{j}+\mathtt{k}=\mathtt{s}}^{\mathtt{s}} \alpha_{\mathtt{ijk}} \left\| \frac{\partial^{(\mathtt{s})} \mathtt{x}}{\partial \mathtt{u}^{(\mathtt{i})} \partial \mathtt{v}^{(\mathtt{j})} \partial \mathtt{w}^{(\mathtt{k})}} \right\|^2 \Big) \mathtt{du}\,\mathtt{dv}\,\mathtt{dw}$

1. *create the basis function set* $\big(\mathtt{N}_{\mathtt{i},\mathtt{l}}(\mathtt{u}) \mathtt{N}_{\mathtt{j},\mathtt{m}}(\mathtt{v}) \mathtt{N}_{\mathtt{k},\mathtt{n}}(\mathtt{w})\big)_{\mathtt{i},\mathtt{j},\mathtt{k}=1}^{\mathtt{p},\mathtt{q},\mathtt{r}}$
2. *create minimisation matrices* $\mathtt{M}_1^{\mathtt{u}}, \ldots, \mathtt{M}_{\mathtt{s}}^{\mathtt{u}}$, $\mathtt{M}_1^{\mathtt{v}}, \ldots, \mathtt{M}_{\mathtt{s}}^{\mathtt{v}}$ $\mathtt{M}_1^{\mathtt{w}}, \ldots, \mathtt{M}_{\mathtt{s}}^{\mathtt{w}}$,
   from the basis function set
3. *create matrices of* $\mathtt{u},\mathtt{v},\mathtt{w}$ *basis functions evaluated at corresponding knot averages*
   $\mathtt{N}_1 = \big(\mathtt{N}_{\mathtt{i},\mathtt{l}}(\eta_\mathtt{j})\big)_{\mathtt{i},\mathtt{j}=1}^{\mathtt{p}}$, $\mathtt{N}_2 = \big(\mathtt{N}_{\mathtt{i},\mathtt{m}}(\psi_\mathtt{j})\big)_{\mathtt{i},\mathtt{j}=1}^{\mathtt{q}}$, $\mathtt{N}_3 = \big(\mathtt{N}_{\mathtt{i},\mathtt{n}}(\zeta_\mathtt{j})\big)_{\mathtt{i},\mathtt{j}=1}^{\mathtt{r}}$
3. *create a matrix3D of volume evaluations at knot averages* $\mathtt{G} = \big(\mathtt{x}(\eta_\mathtt{i}, \psi_\mathtt{j}, \zeta_\mathtt{k})\big)_{\mathtt{i},\mathtt{j},\mathtt{k}=1}^{\mathtt{p},\mathtt{q},\mathtt{r}}$
4. *for a given smoothing factor* $\mathtt{sm}$ *form the Kronecker product matrix*
   $\mathtt{F} = \mathtt{sm} \sum_{\mathtt{i}_1,\mathtt{j}_1,\mathtt{k}_1} \alpha_{\mathtt{i}_1\mathtt{j}_1\mathtt{k}_1} \mathtt{M}_{\mathtt{k}_1}^{\mathtt{w}} \otimes \mathtt{M}_{\mathtt{j}_1}^{\mathtt{v}} \otimes \mathtt{M}_{\mathtt{i}_1}^{\mathtt{u}}$
5. *form the kronecker product* $\mathtt{H} = \mathtt{N}_3 \otimes (\mathtt{N}_2 \otimes \mathtt{N}_1)$
6. *solve the system* `(F+H)vec(d)=vec(G)`
7. *reconstruct the B-spline control point matrix from the vector* `vec(d)`

Figure 4.53: Algorithm 4.7: Smoothing an existing B-spline volume

following four functionals:

$$
\begin{aligned}
J^2_{uvw} &= \int_V \Big(\mathbf{x}^2_{uu} + \mathbf{x}^2_{vv} + \mathbf{x}^2_{ww} \Big) du\, dv\, dw, \\
J^2_{pas} &= \int_V \Big(\mathbf{x}^2_{uu} + 2\mathbf{x}^2_{uv} + 2\mathbf{x}^2_{uw} + \mathbf{x}^2_{vv} + 2\mathbf{x}^2_{vw} + \mathbf{x}^2_{ww} \Big) du\, dv\, dw, \\
J^3_{uvw} &= \int_V \Big(\mathbf{x}^2_{uuu} + \mathbf{x}^2_{vvv} + \mathbf{x}^2_{www} \Big) du\, dv\, dw, \\
J^3_{pas} &= \int_V \Big(\mathbf{x}^2_{uuu} + 3\mathbf{x}^2_{uuv} + 3\mathbf{x}^2_{uuw} + 3\mathbf{x}^2_{uvv} + 6\mathbf{x}_{uvw} + 3\mathbf{x}^2_{uww} + \mathbf{x}^2_{vvv} + 3\mathbf{x}^2_{vvw} + 3\mathbf{x}^2_{vww} + \mathbf{x}^2_{www} \Big) du\, dv\, dw.
\end{aligned}
\tag{4.17}
$$

The terms and coefficients appearing here originate from the extension of Pascal's triangle coefficients in two dimensions (used in the surface smoothing) to three dimensions forming the

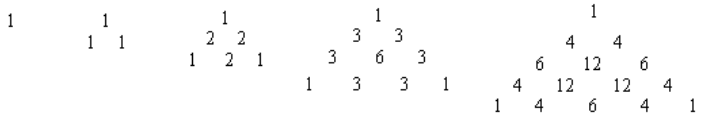


Figure 4.54: Pascal tetrahedron coefficients for orders 0 through to 4

so called Pascal pyramid or tetrahedron, see figure 4.54. These coefficients can be generated by expanding the trinomial $(x+y+z)^n$ for $n = 0, \ldots, 4$. The same four functionals are used as evaluation measures to examine the effectiveness of the smoothing operation in each case. In addition we use two 3D curvature measures, Gaussian (K), and Mean (H), based on evaluating the principal curvatures k_1, k_2 of the w-level surfaces of the functional volume (appendix F gives more details of these measures).

$$J_K = \int_V (k_1 k_2)^2 du\, dv\, dw, \quad J_H = \int_V \big(\frac{k_1+k_2}{2}\big)^2 du\, dv\, dw.$$

In terms of the partial derivatives of $x(u,v,w)$, the Gaussian and Mean curvatures of these isosurfaces of the functional volume are given by

$$K = \Big(x_w^2(x_{uu}x_{vv} - x_{uv}^2) + x_v^2(x_{uu}x_{ww} - x_{uw}^2) + x_u^2(x_{vv}x_{ww} - x_{vw}^2) + 2\big[x_u x_v(x_{uw}x_{vw} - x_{uv}x_{ww}) + x_u x_w(x_{uv}x_{vw} - x_{uw}x_{vv}) + x_v x_w(x_{uv}x_{uw} - x_{vw}x_{uu})\big] \Big) / (x_u^2 + x_v^2 + x_w^2)^2 \tag{4.18}$$

$$H = \frac{\Big(x_u^2(x_{vv} + x_{ww}) + x_v^2(x_{uu} + x_{ww}) + x_w^2(x_{uu} + x_{vv}) - 2x_u x_v x_{uv} - 2x_u x_w x_{uw} - 2x_v x_w x_{vw} \Big)}{2(x_u^2 + x_v^2 + x_w^2)^{\frac{3}{2}}}. \tag{4.19}$$

In addition to the numerical values obtained from the smoothed volume, we also illustrate graphically, in terms of cross sectional curvatures and environment mapped images, the effectiveness of the algorithm and the functional. To do this we take three section planes in the w direction. Figures 4.55, 4.56 and 4.57 present these views for the original and perturbed volume. The original volume has zero curvature measures and the modifications to the control points introduce irregularities into the object as observed in the curvature and environment mapped images.

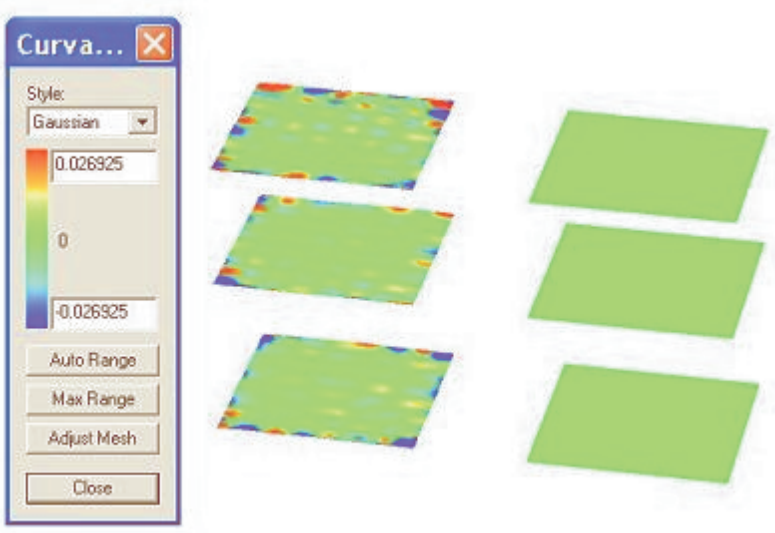


Figure 4.55: Perturbed and original volume using sectional view with Gaussian curvature mapped image

For the four functionals tested, five smoothing factors, $(0.1, 0.2, 0.5, 1, 10)$ are applied and the measures $J^2_{uvw}, J^2_{pas}, J^3_{uvw}, J^3_{pas}$ compared as well as the resulting deviation from the original (obtained by sampling the volume over a grid of u, v, w values). The numerical comparisons are given in tables 4.8 to 4.12. As can be seen from the tables the smoothing algorithm produces new volumes with significantly reduced mean and Gaussian curvature variations, predictably getting smaller as the smoothing factor increases. The functionals containing the full complement of Pascal's coefficients, J^2_{pas}, J^3_{pas}, appear to produce somewhat better results in this respect. Increasing the smoothing factor, while leading to more desirable curvature distributions, has the expected side effect of increasing the point deviation from the original. Figures 4.59 through to 4.65 illustrate graphically the sectional mean and Gaussian curvatures and the environment mapped images of the smoothed volumes for the smoothing factors $(0.1, 0.2, 0.5, 1)$ and figures 4.66, 4.67 for $sm = 10$.

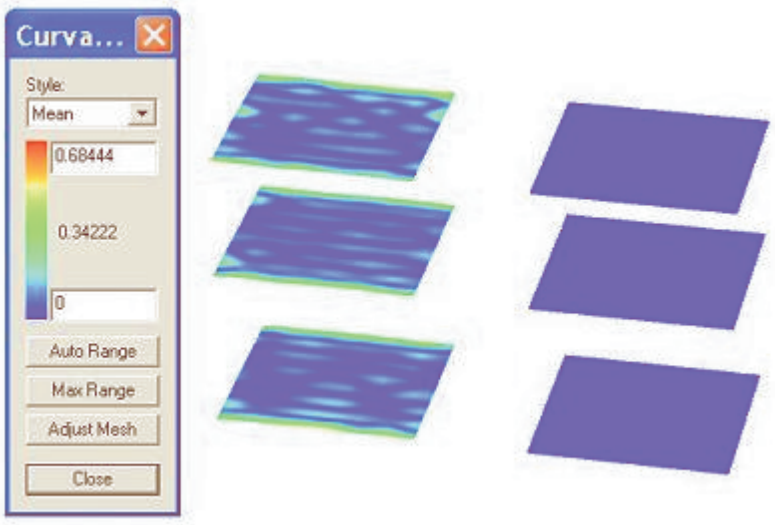


Figure 4.56: Perturbed and original volume using sectional view with mean curvature mapped image

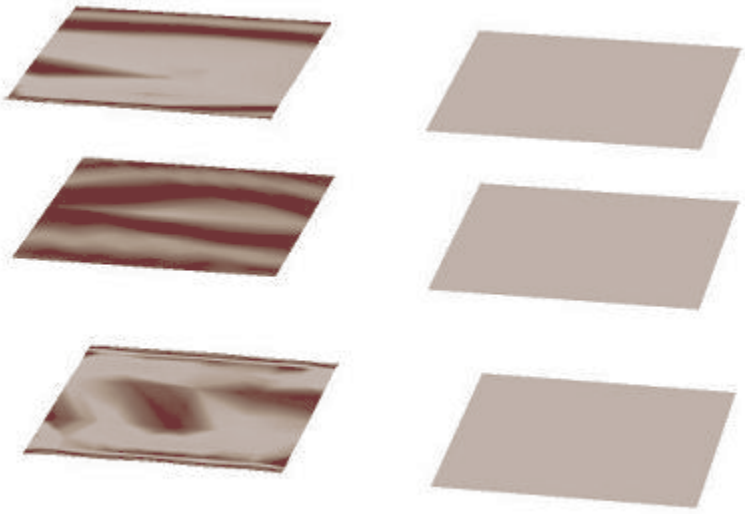


Figure 4.57: Perturbed and original volume using sectional view with environment mapped image

Evaluation	*Optimisation Criteria, Smoothing factor 0.1*				*Perturbed*	*Original*
Measure	$min \rightarrow J^2_{uvw}$	$min \rightarrow J^2_{pas}$	$min \rightarrow J^3_{uvw}$	$min \rightarrow J^3_{pas}$	*Volume*	*Volume*
J^2_{uvw}	68.2443	69.4734	106.175	107.56	816.365	776.688
J^2_{pas}	70.0268	70.7226	111.13	112.005	819.453	776.688
J^3_{uvw}	211.022	212.154	82.0886	82.7701	1967.83	1640.5
J^3_{pas}	500.224	443.177	171.915	137.711	2159.21	1640.5
J_K	0.000730434	0.000607229	0.000114043	8.11102e-05	0.0260893	0.0
J_H	0.315361	0.307859	0.12353	0.117451	5.77858	0.0
Max dev perturbed	0.164342	0.16419	0.153374	0.153837	0.0	0.0901385
Max dev Original	0.182314	0.181014	0.145298	0.145322	0.0901385	0.0

Table 4.8: Smoothing/Fairness measures and values for totally perturbed functional volume section smoothed with factor 0.1 over complete volume

Evaluation	*Optimisation Criteria, Smoothing factor 0.2*				*Perturbed*	*Original*
Measure	$min \rightarrow J^2_{uvw}$	$min \rightarrow J^2_{pas}$	$min \rightarrow J^3_{uvw}$	$min \rightarrow J^3_{pas}$	*Volume*	*Volume*
J^2_{uvw}	42.2833	43.5482	79.5072	81.2599	816.365	776.688
J^2_{pas}	49.3039	48.2824	86.7169	86.5393	819.453	776.688
J^3_{uvw}	133.957	136.3	45.3546	45.9521	1967.83	1640.5
J^3_{pas}	279.344	210.101	112.723	73.3937	2159.21	1640.5
J_K	0.000177267	0.000113541	4.32194e-05	2.5255e-05	0.0260893	0.0
J_H	0.148497	0.140146	0.0828473	0.0761597	5.77858	0.0
Max dev perturbed	0.214919	0.214124	0.188367	0.189002	0.0	0.0901385
Max dev Original	0.22673	0.222737	0.177893	0.177771	0.0901385	0.0

Table 4.9: Smoothing/Fairness measures and values for totally perturbed functional volume section smoothed with factor 0.2 over complete volume

Evaluation	*Optimisation Criteria, Smoothing factor 0.5*				*Perturbed*	*Original*
Measure	$min \rightarrow J^2_{uvw}$	$min \rightarrow J^2_{pas}$	$min \rightarrow J^3_{uvw}$	$min \rightarrow J^3_{pas}$	*Volume*	*Volume*
J^2_{uvw}	22.9293	24.4178	52.9643	55.1923	816.365	776.688
J^2_{pas}	32.9718	28.5016	59.9636	59.0252	819.453	776.688
J^3_{uvw}	59.5442	61.3634	18.8717	19.0245	1967.83	1640.5
J^3_{pas}	100.667	59.7379	68.8404	31.5993	2159.21	1640.5
J_K	2.44132e-05	6.81157e-06	1.67577e-05	5.35277e-06	0.0260893	0.0
J_H	0.0666896	0.0523931	0.0650042	0.0475262	5.77858	0.0
Max dev perturbed	0.293676	0.287434	0.231838	0.232212	0.0	0.0901385
Max dev Original	0.294669	0.279079	0.218715	0.218832	0.0901385	0.0

Table 4.10: Smoothing/Fairness measures and values for totally perturbed functional volume section smoothed with factor 0.5 over complete volume

Evaluation	*Optimisation Criteria, Smoothing factor 1.0*				*Perturbed*	*Original*
Measure	$min \rightarrow J^2_{uvw}$	$min \rightarrow J^2_{pas}$	$min \rightarrow J^3_{uvw}$	$min \rightarrow J^3_{pas}$	*Volume*	*Volume*
J^2_{uvw}	13.9075	15.658	38.417	40.8543	816.365	776.688
J^2_{pas}	22.6828	17.5689	44.0678	43.2773	819.453	776.688
J^3_{uvw}	27.8302	27.867	9.60166	9.5517	1967.83	1640.5
J^3_{pas}	45.787	25.4135	48.6028	16.8132	2159.21	1640.5
J_K	9.23314e-06	9.50741e-07	1.05381e-05	1.79373e-06	0.0260893	0.0
J_H	0.0465829	0.024013	0.0589051	0.0330301	5.77858	0.0
Max dev perturbed	0.358805	0.338399	0.260815	0.25819	0.0	0.0901385
Max dev Original	0.350154	0.317285	0.245931	0.243653	0.0901385	0.0

Table 4.11: Smoothing/Fairness measures and values for totally perturbed functional volume section smoothed with factor 1 over complete volume

Evaluation Measure	*Optimisation Criteria, Smoothing factor 10*				*Perturbed Volume*	*Original Volume*
	$min \rightarrow J^2_{uvw}$	$min \rightarrow J^2_{pas}$	$min \rightarrow J^3_{uvw}$	$min \rightarrow J^3_{pas}$		
J^2_{uvw}	1.95863	2.68722	11.3683	13.0721	816.365	776.688
J^2_{pas}	4.44804	2.70585	14.8125	13.4469	819.453	776.688
J^3_{uvw}	1.42586	1.20513	1.07077	1.10913	1629.74	1640.5
J^3_{pas}	5.90919	1.18066	15.3256	1.21023	1967.83	1640.5
J_K	9.7446e-07	8.13845e-10	3.64999e-06	2.14684e-08	0.0260893	0.0
J_H	0.0157271	0.000782721	0.0353609	0.00529835	5.77858	0.0
Max dev perturbed	0.588441	0.519463	0.373793	0.345677	0.0	0.0901385
Max dev Original	0.55856	0.49737	0.341467	0.323212	0.0901385	0.0

Table 4.12: Smoothing/Fairness measures and values for totally perturbed functional volume section smoothed with factor 10.0 over complete volume

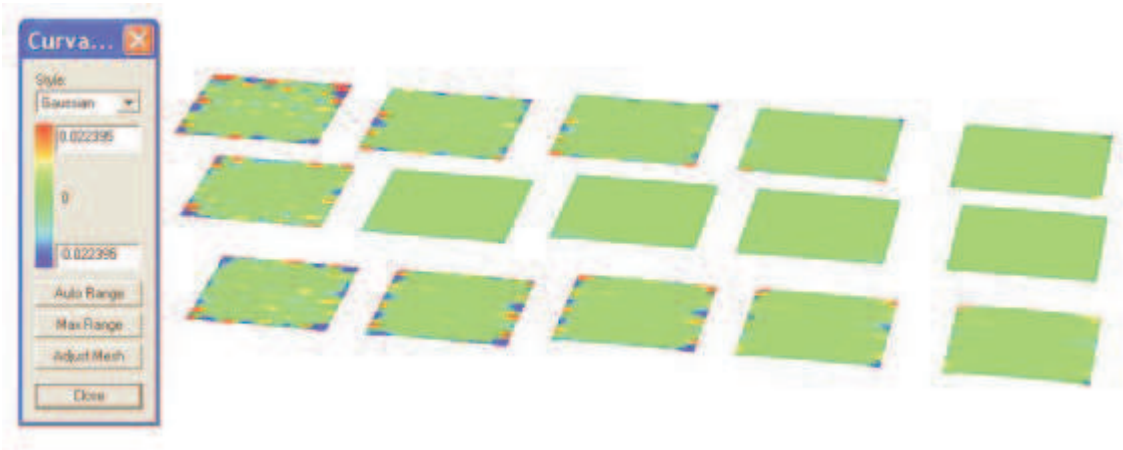


Figure 4.58: Perturbed and smoothed volume using sectional view with Gaussian curvature mapped image, smoothing factors 0.1,0.2,0.5,1.0,functional J^2_{uvw}

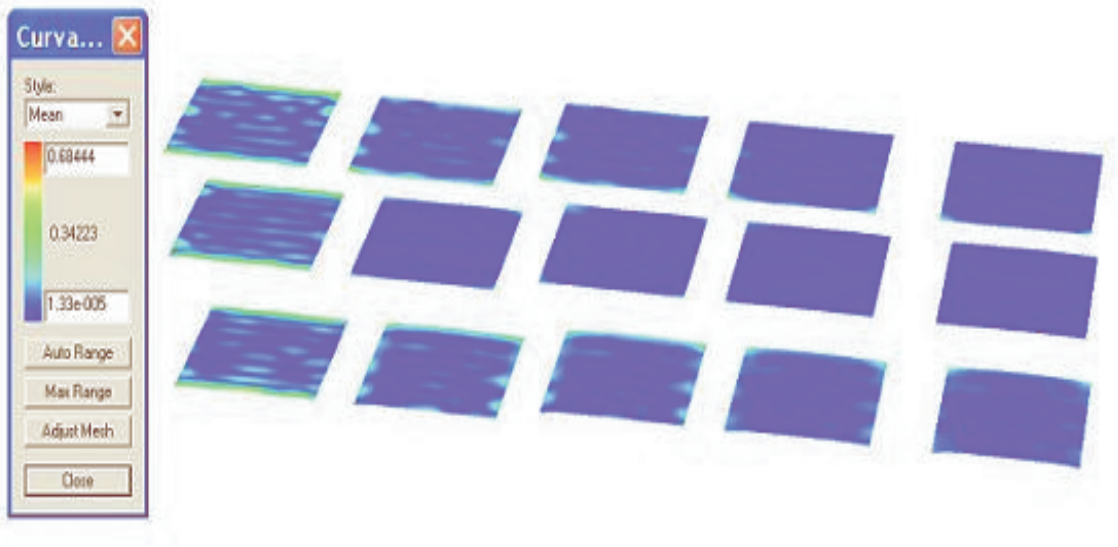


Figure 4.59: Perturbed and smoothed volume using sectional view with mean curvature mapped image, smoothing factors 0.1,0.2,0.5,1.0, functional J^2_{uvw}

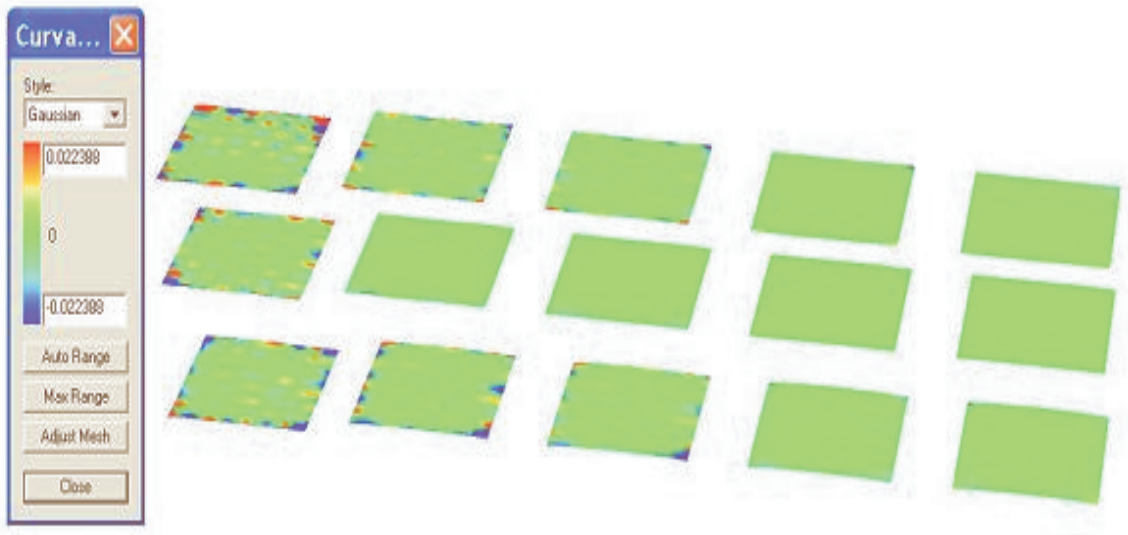


Figure 4.60: Perturbed and smoothed volume using sectional view with Gaussian curvature mapped image, smoothing factors 0.1,0.2,0.5,1.0, functional J^2_{pas}

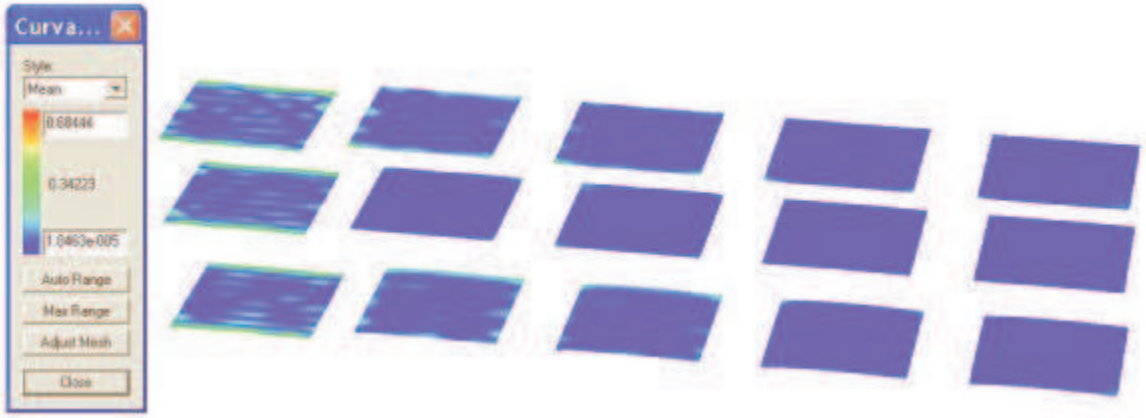


Figure 4.61: Perturbed and smoothed volume using sectional view with mean curvature mapped image, smoothing factors 0.1,0.2,0.5,1.0,functional J^2_{pas}

Figure 4.62: Perturbed and smoothed volume using sectional view with Gaussian curvature mapped image, smoothing factors 0.1,0.2,0.5,1.0,functional J^3_{uvw}

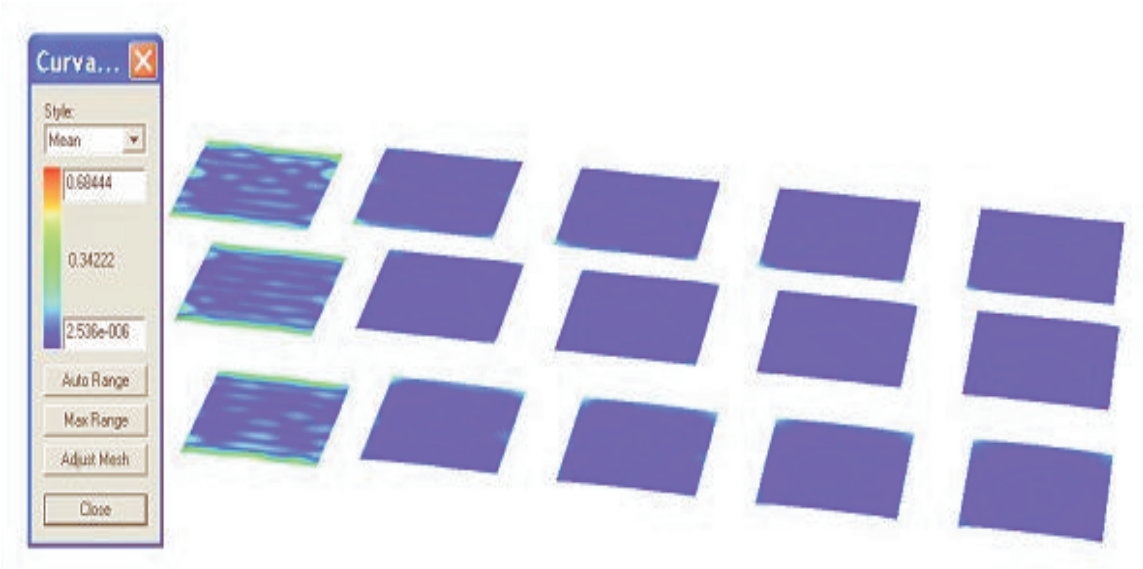


Figure 4.63: Perturbed and smoothed volume using sectional view with mean curvature mapped image, smoothing factors 0.1,0.2,0.5,1.0, functional J^3_{uvw}

Figure 4.64: Perturbed and smoothed volume using sectional view with Gaussian curvature mapped image, smoothing factors 0.1,0.2,0.5,1.0, functional J^3_{pas}

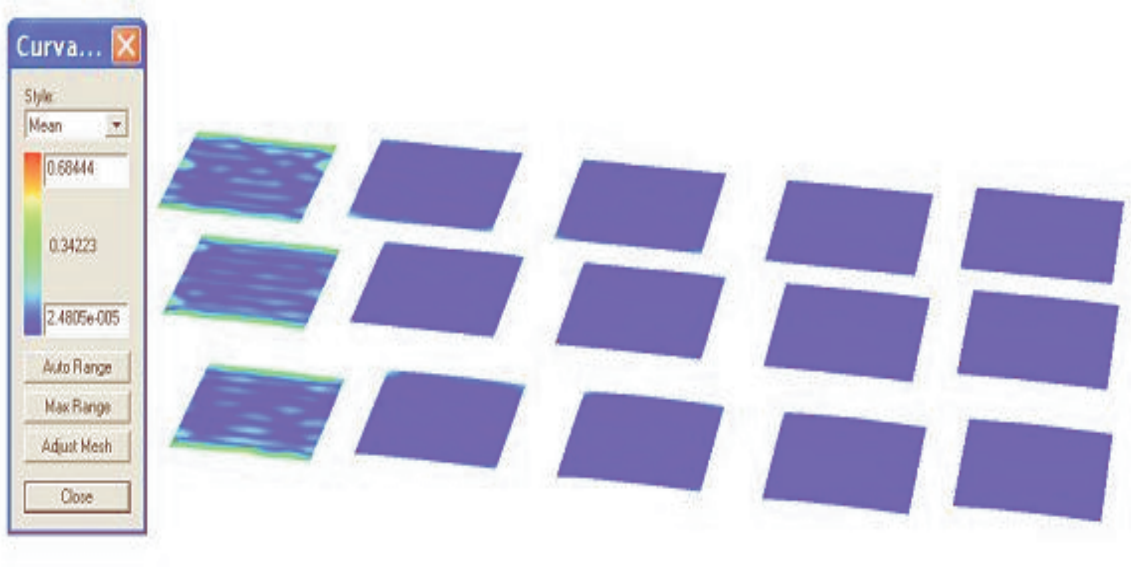


Figure 4.65: Perturbed and smoothed volume using sectional view with mean curvature mapped image, smoothing factors 0.1,0.2,0.5,1.0,functional J^3_{pas}

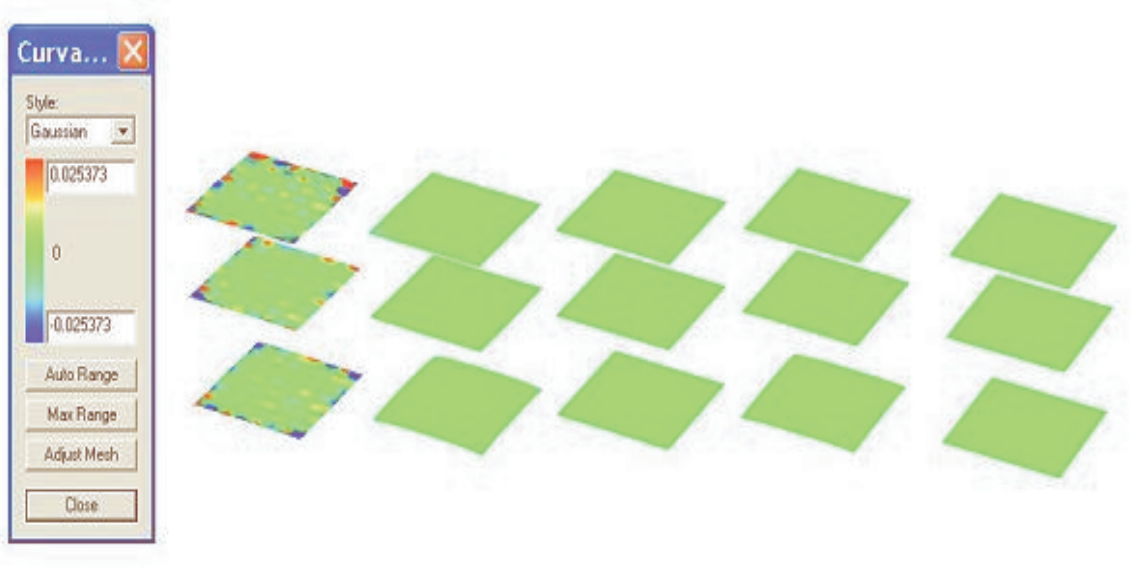


Figure 4.66: Perturbed and smoothed volume using sectional view with Gaussian curvature mapped image, smoothing factor 10.0, functionals J^2_{uvw}, J^2_{pas}, J^3_{uvw}, J^3_{pas}

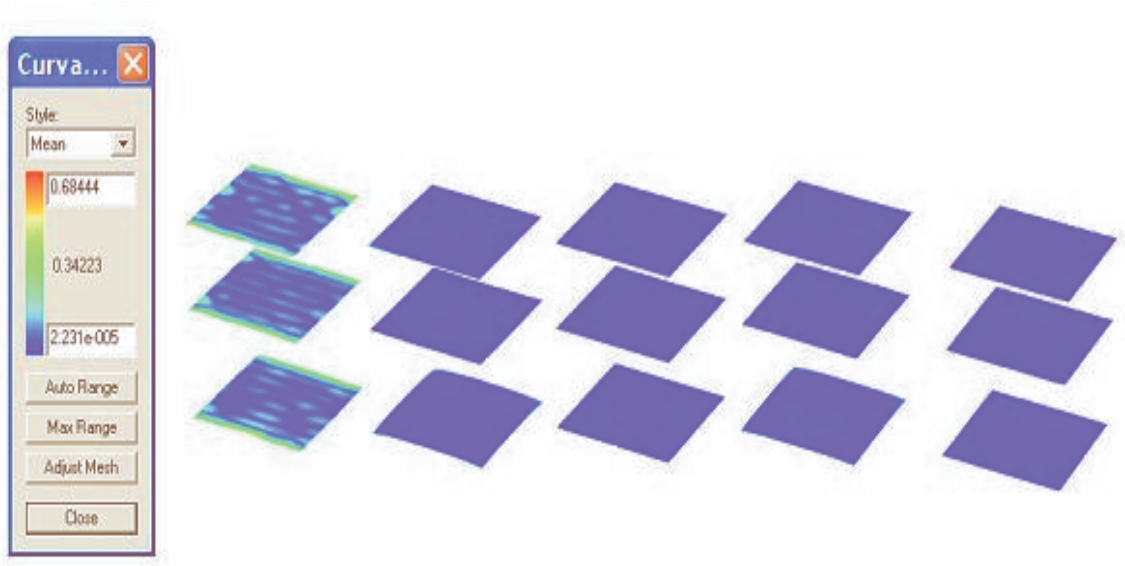


Figure 4.67: Perturbed and smoothed volume using sectional view with mean curvature mapped image, smoothing factor 10.0, functionals $J^2_{uvw}, J^2_{pas}, J^3_{uvw}, J^3_{pas}$

4.7 Volume Smoothing Examples: parametric case

To test the effectiveness of algorithm 4.7 to the smoothing of parametrically defined volumes we again use an example B-spline representing a cube:

$$\mathbf{x}(u,v,w) = \sum_{i=1}^{10}\sum_{j=1}^{10}\sum_{k=1}^{10} \mathbf{d}_{ijk} N_{i,4}(u) N_{j,4}(v) N_{k,4}(w), \quad (u_i)_{i=1}^{14} \times (v_j)_{j=1}^{14} \times (w_k)_{k=1}^{14}.$$

The control points $\mathbf{d}_{ijk}$ are set to vary from $(0,0,0)$ to $(9,9,9)$ and the uniformly spaced knot set is taken to be $(0,0,0,0,1/7,2/7,3/7,4/7,5/7,6/7,1,1,1,1)$ in u, v and w. As in the functional case, to simulate irregularities we perturb the volume by disturbing the control points using a random number generator such that the maximum absolute displacement of any given control point is 0.1 in each of its x, y, z components. The perturbed volume is smoothed using algorithm 4.7, employing the same four functionals, $J^2_{uvw}, J^2_{pas}, J^3_{uvw}, J^3_{pas}$. We test the algorithm using five smoothing factors, $(0.1, 0.2, 0.5, 1.0, 10)$.

The original and perturbed volumes are represented graphically using two views. In the first we take the three isoparametric surfaces at $u = 0.5, v = 0.5, w = 0.5$, and in the second we use the bounding isoparametric surfaces of the volume, that is the six surfaces at $u = 0, 1, v = 0, 1, w = 0, 1$. In each case we display Gaussian and mean curvature maps and environment

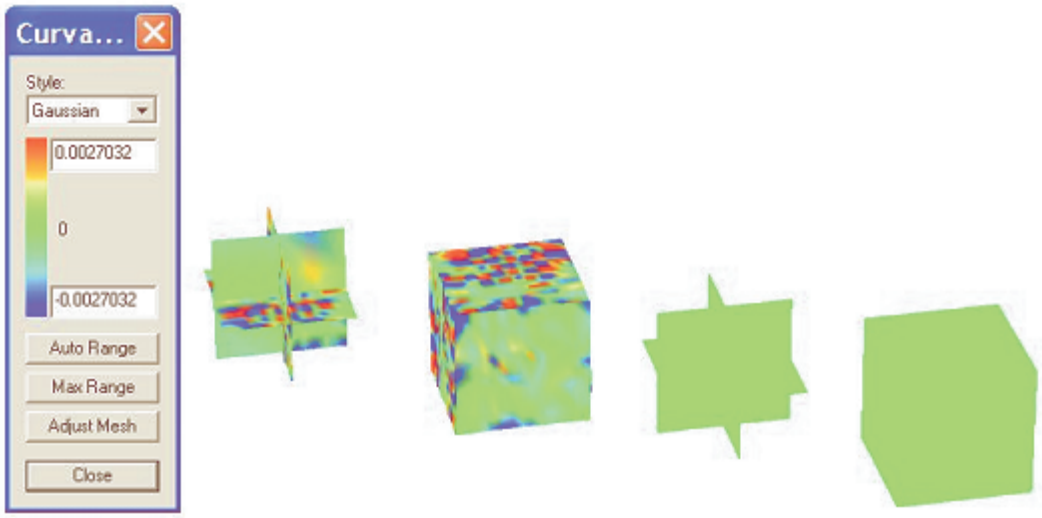


Figure 4.68: Original and perturbed cube, sectional view with Gaussian curvature map

mapped images. Figures 4.68, 4.69 and 4.70 present these views. The original volume has zero curvature measures and the modifications to the control points introduce irregularities into the perturbed volume which can be clearly seen in the curvature and environment mapped images.

The numerical comparisons are given in tables 4.13 for a smoothing factor of 0.1 to table 4.17 for a smoothing factor of 10. In addition to the four functional measures and the point deviation (obtained by sampling), for the parametric case we also compare the resulting volumes. As can be seen the perturbations to the control points result in an increased volume with the smoothing algorithm reducing this in each case (more significantly with the second derivative functionals than the third derivative ones). Increasing the smoothing factor results in successively smaller volumes but with also smaller curvature variations illustrating the tradeoff between smoothness and accuracy.

Figures 4.71 through to 4.81 illustrate graphically the mean and Gaussian curvatures of the two isoparametric views and the environment mapped images of the resulting volumes for the smoothing factors $(0.1, 0.2, 0.5, 1, 10)$. In each case the image at the top left is the original perturbed volume followed by the smoothed volumes for $sm = 0.1$ to 10 moving to the right and down (so that the image at the bottom right is for $sm = 10$).

Evaluation Measure	*Optimisation Criteria, Smoothing factor 0.1*				*Perturbed Volume*	*Original Volume*
	$min \longrightarrow J^2_{uvw}$	$min \longrightarrow J^2_{pas}$	$min \longrightarrow J^3_{uvw}$	$min \longrightarrow J^3_{pas}$		
J^2_{uvw}	119.683	121.201	215.083	218.643	1175.86	1059.88
J^2_{pas}	126.235	126.702	224.28	225.458	1175.86	1066.08
J^3_{uvw}	452.232	454.101	158.846	161.049	1646.86	1813.17
J^3_{pas}	802.161	725.908	276.206	203.109	1646.86	2194.47
Volume	641.305	641.507	681.157	681.516	736.471	707.775
Max dev perturbed	0.145584	0.144276	0.152499	0.152054	0.0	0.156123
Max dev Original	0.211	0.211488	0.202493	0.203993	0.156123	0.0

Table 4.13: Smoothing/Fairness measures and values for totally perturbed volume section smoothed with factor 0.1 over complete volume

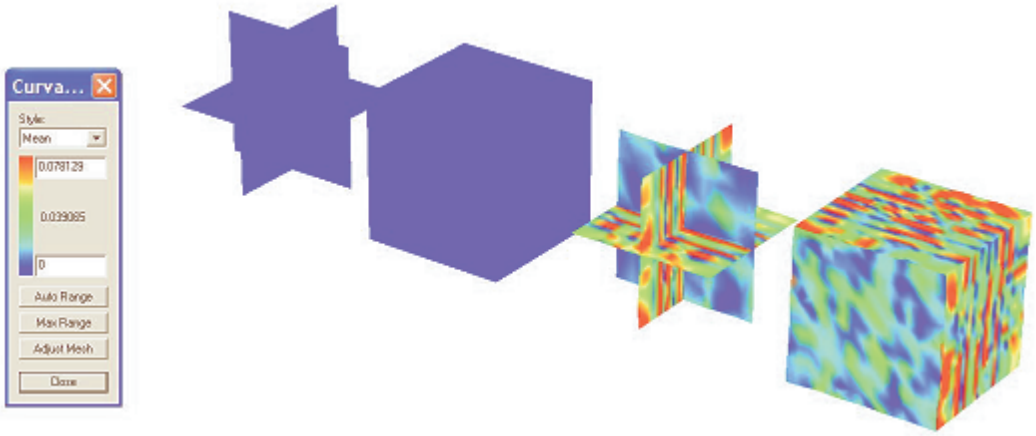


Figure 4.69: Original and perturbed cube, sectional view with mean curvature map

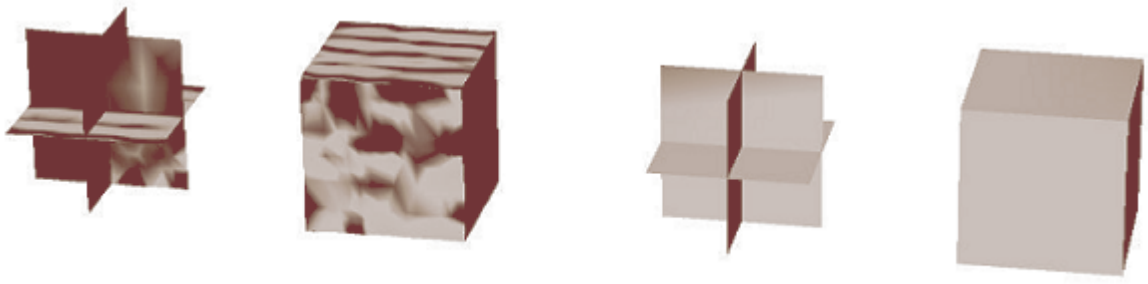


Figure 4.70: Original and perturbed cube, sectional view with environment mapped image

Evaluation Measure	*Optimisation Criteria, Smoothing factor 0.2*				*Perturbed Volume*	*Original Volume*
	$min \longrightarrow J^2_{uvw}$	$min \longrightarrow J^2_{pas}$	$min \longrightarrow J^3_{uvw}$	$min \longrightarrow J^3_{pas}$		
J^2_{uvw}	81.233	82.9163	167.86	172.386	1175.86	1059.88
J^2_{pas}	96.8303	93.9574	180.636	179.314	1175.86	1066.08
J^3_{uvw}	299.438	304.06	88.134	89.6675	1646.86	1813.17
J^3_{pas}	442.216	353.487	190.978	107.504	1646.86	2194.47
Volume	636.517	636.883	676.458	677.059	736.471	707.775
Max dev perturbed	0.191462	0.188791	0.184822	0.181374	0.0	0.156123
Max dev Original	0.240045	0.240354	0.223215	0.223896	0.156123	0.0

Table 4.14: Smoothing/Fairness measures and values for totally perturbed volume section smoothed with factor 0.2 over complete volume

Evaluation Measure	*Optimisation Criteria, Smoothing factor 0.5*				*Perturbed Volume*	*Original Volume*
	$min \longrightarrow J^2_{uvw}$	$min \longrightarrow J^2_{pas}$	$min \longrightarrow J^3_{uvw}$	$min \longrightarrow J^3_{pas}$		
J^2_{uvw}	47.6396	50.4563	121.51	127.135	1175.86	1059.88
J^2_{pas}	68.4949	59.0382	132.262	130.768	1175.86	1066.08
J^3_{uvw}	136.448	139.71	39.3988	39.1131	1646.86	1813.17
J^3_{pas}	156.855	112.242	119.045	47.564	1646.86	2194.47
Volume	626.206	627.392	666.783	667.885	736.471	707.775
Max dev perturbed	0.259709	0.258397	0.22508	0.22273	0.0	0.156123
Max dev Original	0.298455	0.299063	0.257298	0.262294	0.156123	0.0

Table 4.15: Smoothing/Fairness measures and values for totally perturbed volume section smoothed with factor 0.5 over complete volume

Evaluation Measure	*Optimisation Criteria, Smoothing factor 0.1*				*Perturbed Volume*	*Original Volume*
	$min \longrightarrow J^2_{uvw}$	$min \longrightarrow J^2_{pas}$	$min \longrightarrow J^3_{uvw}$	$min \longrightarrow J^3_{pas}$		
J^2_{uvw}	28.6422	32.9368	92.788	100.325	1175.86	1059.88
J^2_{pas}	47.4557	36.883	101.156	102.349	1175.86	1066.08
J^3_{uvw}	65.6644	65.3551	23.2107	22.8942	1646.86	1813.17
J^3_{pas}	74.1624	54.6438	84.6556	27.7566	1646.86	2194.47
Volume	612.878	615.457	656.342	658.209	736.471	707.775
Max dev perturbed	0.312025	0.321256	0.258374	0.261906	0.0	0.156123
Max dev Original	0.352291	0.361854	0.291048	0.290904	0.156123	0.0

Table 4.16: Smoothing/Fairness measures and values for totally perturbed volume section smoothed with factor 1.0 over complete volume

Evaluation Measure	*Optimisation Criteria, Smoothing factor 10.0*				*Perturbed Volume*	*Original Volume*
	$min \longrightarrow J^2_{uvw}$	$min \longrightarrow J^2_{pas}$	$min \longrightarrow J^3_{uvw}$	$min \longrightarrow J^3_{pas}$		
J^2_{uvw}	1.90455	2.59005	16.9353	23.5119	1175.86	1059.88
J^2_{pas}	6.65001	2.60717	29.2528	23.8873	1175.86	1066.08
J^3_{uvw}	3.20003	2.64712	4.05336	4.48858	1646.86	1813.17
J^3_{pas}	12.6035	2.59015	4.96622	4.66045	1646.86	2194.47
Volume	562.144	565.012	596.811	599.861	736.471	707.775
Max dev perturbed	0.539557	0.653563	0.39657	0.504856	0.0	0.156123
Max dev Original	0.557122	0.650506	0.418407	0.502111	0.156123	0.0

Table 4.17: Smoothing/Fairness measures and values for totally perturbed volume section smoothed with factor 10.0 over complete volume

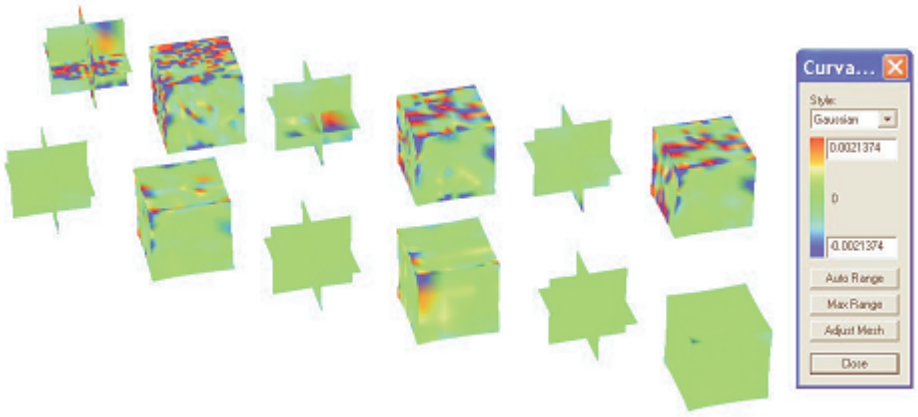


Figure 4.71: Perturbed and smoothed cube using J^2_{uvw} functional, smoothing factors 0.1,0.2,0.5,1,10, sectional view with guassian curvature map

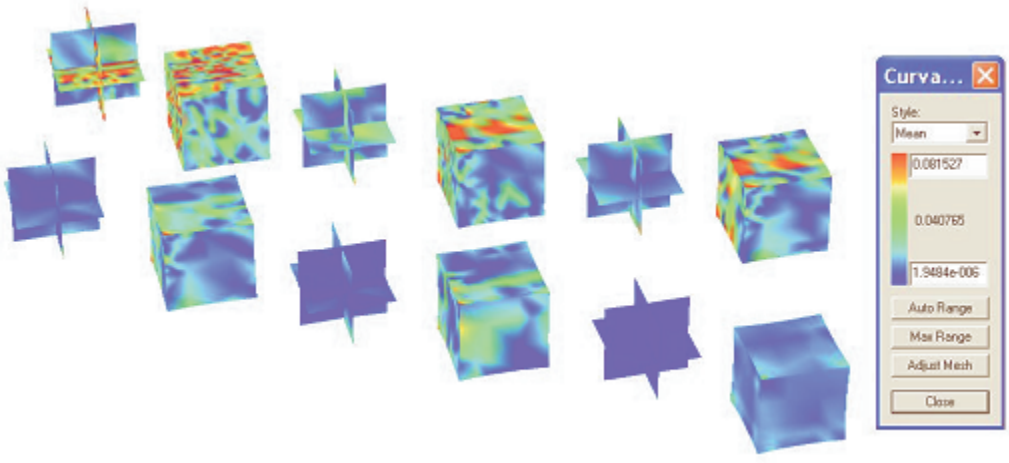


Figure 4.72: Perturbed and smoothed cube using J^2_{uvw} functional, smoothing factors 0.1,0.2,0.5,1,10, sectional view with mean curvature map

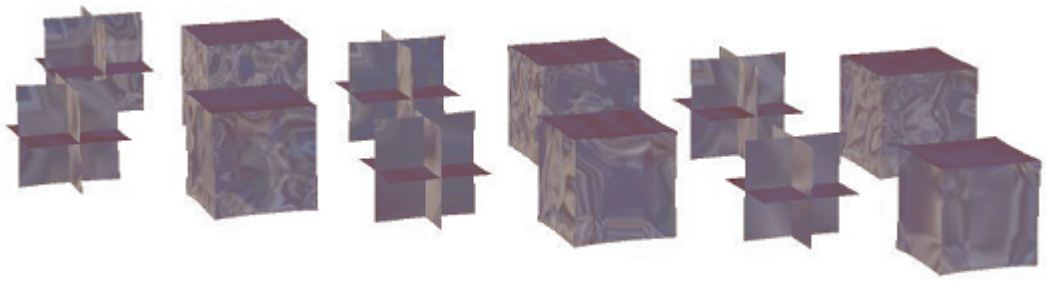


Figure 4.73: Perturbed and smoothed cube using J^2_{uvw} functional, smoothing factors 0.1,0.2,0.5,1,10, sectional view with environment mapped image

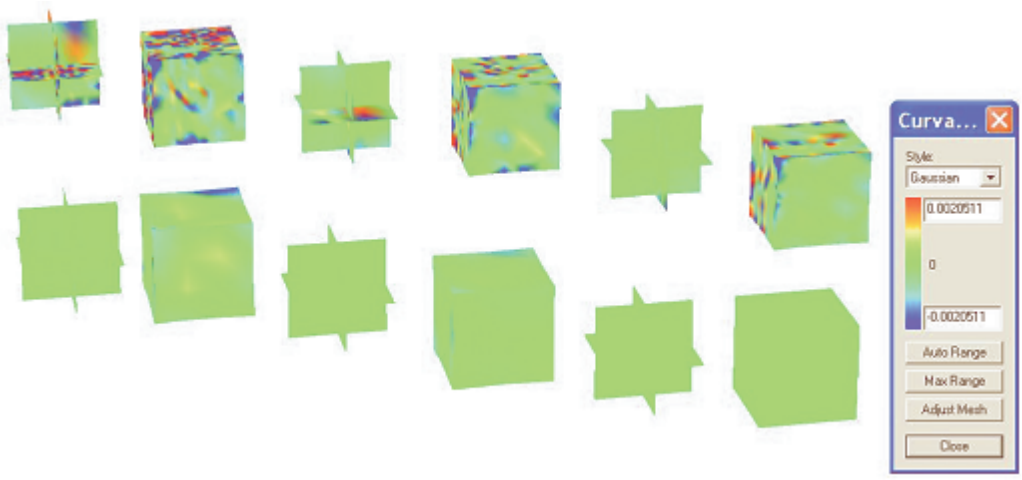


Figure 4.74: Perturbed and smoothed cube using J^2_{pas} functional, smoothing factors 0.1,0.2,0.5,1,10, sectional view with Gaussian curvature map

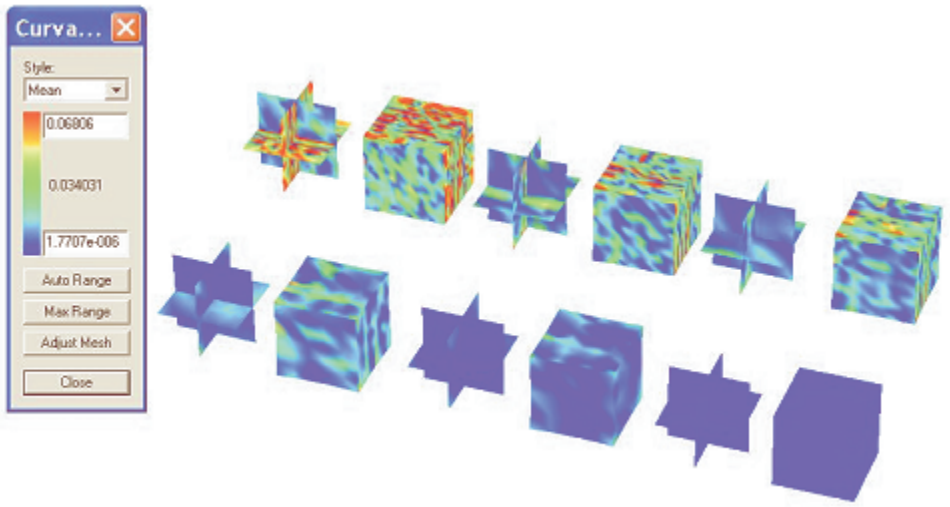


Figure 4.75: Perturbed and smoothed cube using J^2_{pas} functional, smoothing factors 0.1,0.2,0.5,1,10, sectional view with mean curvature map

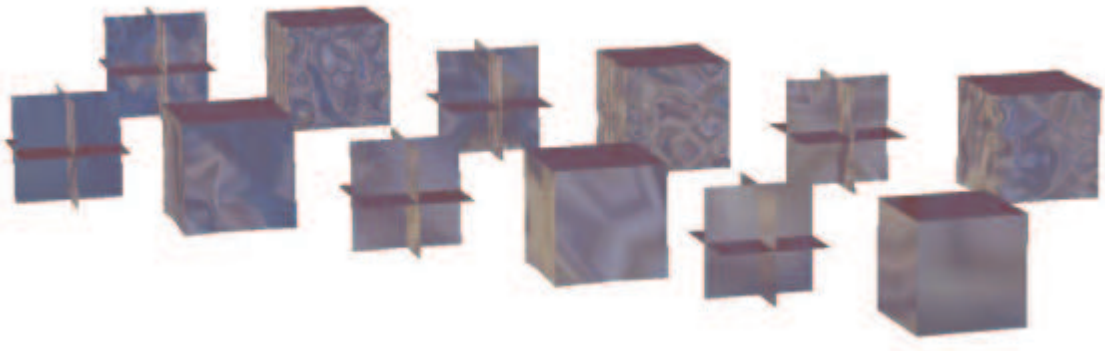


Figure 4.76: Perturbed and smoothed cube using J^2_{pas} functional, smoothing factors 0.1,0.2,0.5,1,10, sectional view with environment mapped image

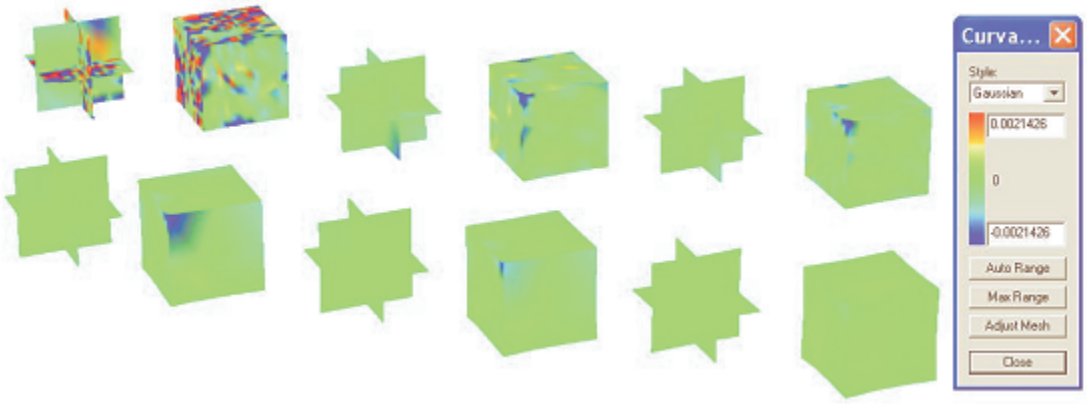


Figure 4.77: Perturbed and smoothed cube using J^3_{uvw} functional, smoothing factors 0.1,0.2,0.5,1,10, sectional view with Gaussian curvature map

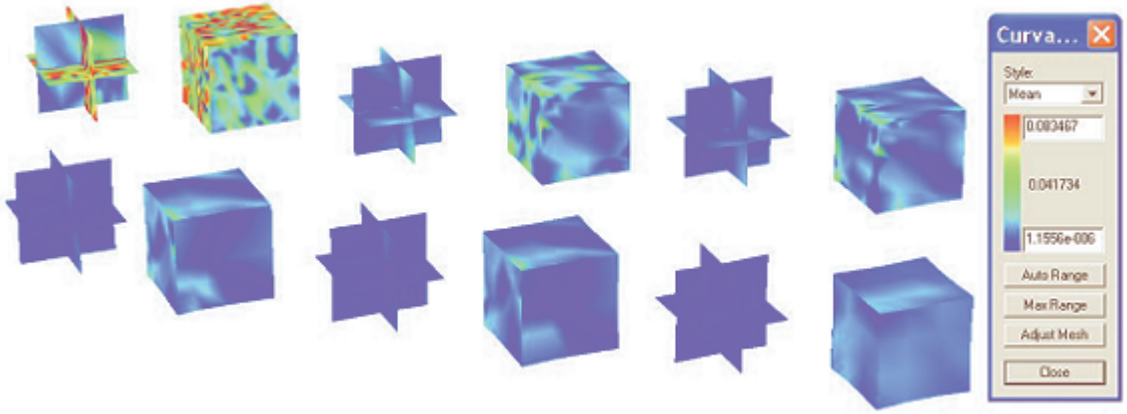


Figure 4.78: Perturbed and smoothed cube using J^3_{uvw} functional, smoothing factors 0.1,0.2,0.5,1,10, sectional view with mean curvature map

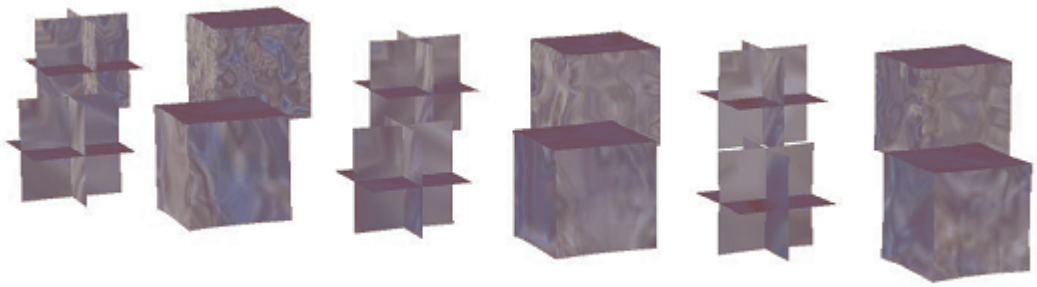


Figure 4.79: Perturbed and smoothed cube using J^3_{uvw} functional, smoothing factors 0.1,0.2,0.5,1,10, sectional view with environment mapped image

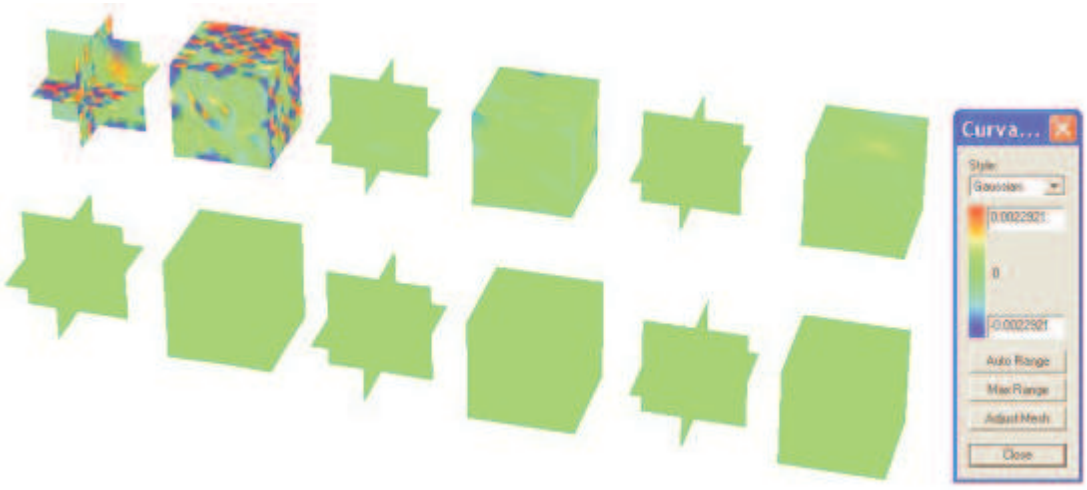


Figure 4.80: Perturbed and smoothed cube using J^3_{pas} functional, smoothing factors 0.1,0.2,0.5,1,10, sectional view with Gaussian curvature map

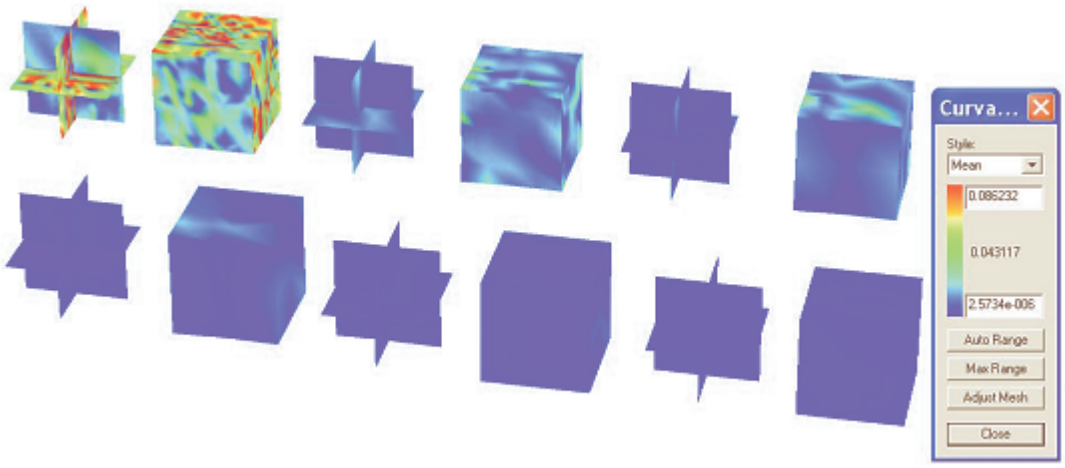


Figure 4.81: Perturbed and smoothed cube using J^3_{pas} functional, smoothing factors 0.1,0.2,0.5,1,10, sectional view with mean curvature map

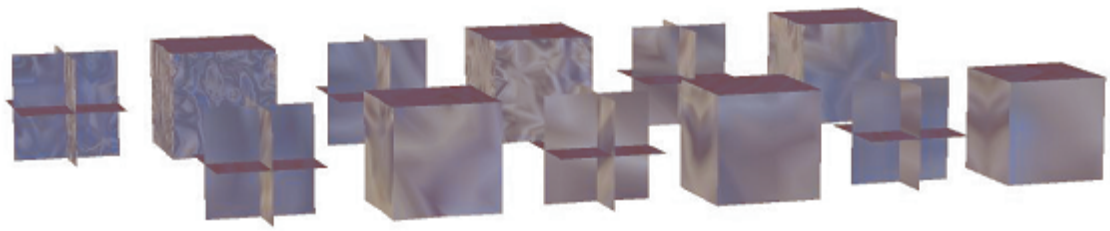


Figure 4.82: Perturbed and smoothed cube using J^3_{pas} functional, smoothing factors 0.1,0.2,0.5,1,10, sectional view with environment mapped image

4.8 Summary

In this chapter we have presented a number of algorithms for smoothing B-spline curves, surfaces and volumes both integrated with and separate from least squares data approximation. The algorithms are simple to implement, flexible, can deal with local smoothing and, in principle, linear geometric constraints. In addition they generalise well from curves and surfaces to volumes. The algorithms have been based on the exact computation of the matrices arising from the minimisation of energy based functionals developed in chapters 2 and 3. We have tested their effectiveness on a number of examples and demonstrated the quality of smoothing for a range of functionals using a selection of fairness measures. For the surface case we have examined the use of a computationally efficient technique for solving the Sylvester type equation which arises when smoothing existing B-spline surfaces. The methods implemented for curves and surfaces have been generalised to the volumetric case and, using suitable generalisations of the energy based functionals, algorithms for smoothing the functional and parametric forms presented with examples to illustrate their utility.

In the following chapter we consider one other area where the results of chapter 3 can be applied, that of the variational formulation of the finite element technique using the Rayleigh-Ritz method.

Chapter 5

Applications to Finite Element Problems using the Rayleigh-Ritz Method

5.1 Introduction

The finite element method is applicable to a wide variety of boundary problems in engineering (Rao, [70]). A solution is sought in the region of a body subject to the interpolation of prescribed boundary (edge) conditions on the dependent variables and their derivatives. Examples of typical problems include static analysis of beams, plates and shells, steady state temperature distributions in solids and fluids, analysis of potential flows, free surface flows, boundary layer flows.

Although B-splines have become a standard for both numerical approximation schemes and free-form design and geometrical processing, mechanical analysis is normally operated as a separate module. However, interaction between these stages is desirable due to the iterative nature of a product development process. For example, a given shape may have to be redesigned to satisfy certain special constraints resulting from analysis. Then conversion to the original design geometry representation is needed before the results of the analysis can be implemented. Hence using the same representation and technique for design, analysis and optimisation can be advantageous. The close link between geometric modelling and numerical simulation in engineering applications suggests the use of B-splines as finite element basis functions.

Although not widespread, B-spline functions have in fact been employed to solve problems

in structural analysis for some time now. In particular, research has been carried out on static, vibration, dynamic and stability analysis of plates and shells using B-splines as coordinate functions based on energy principles. Antes, [2], investigates the use of bicubic splines for plate bending on a rectangular topology. He uses a 2D variational formulation obtained by the principle of minimum potential energy of a thin plate in the small-deflection theory. The author presents some plate bending examples with differing boundary conditions and compares the results with known analytical solutions. Wang, [88], uses B-spline functions and a Rayleigh-Ritz variational approach to analyse the vibration and buckling of beams and plates. Cheng and Dade, [15], use splines in an energy based approach for modelling the dynamic response of stiffened plates and shells and Shen and Wang, [89], use B-spline functions to obtain approximate solutions for vibration of shells. Recent, more general accounts of the use of B-splines in finite elements and not restricted to rectangular domains are given in Hollig, [37], and Sabin, [75].

In this chapter we combine the results of chapter 3 with the use of the Rayleigh-Ritz variational formulation of a continuum problem, to seek B-spline solutions to a selection of finite element problems in one, two and three dimensions. To keep within the original scope of the thesis we focus on the restricted class of problems enforced by topologically rectangular domains. Within this class we examine the following cases: static analysis of beams, plate bending and the elastic deformation of isotropic solids. Using the reduced transformation technique we deal with a variety of geometric constraints and we use a number of examples in the above fields to demonstrate the accuracy of the resulting solutions for varying orders and segment numbers. The results presented for the plate bending examples can be considered to be a generalisation of the analysis in [2]. We begin by reviewing briefly the essential elements of the Rayleigh-Ritz method focusing on the one dimensional case.

5.1.1 The Raleigh-Ritz Method

In the Raleigh-Ritz method the physical problem is specified as a variational problem on the extremum of a functional $I(x)$ for which the given equation is the Euler equation (see section 3.2). The values of a functional $I(x)$ are considered not on arbitrary admissible curves of a given variational problem but only on all possible linear combinations

$$x_n = \sum_{i=1}^{n} \alpha_i W_i$$

with constant coefficients composed of n first functions of some chosen sequence of functions

$$W_1, \ldots, W_n, \ldots$$

The functions x_n must be admissible in the problem at hand which imposes certain restrictions on the choice of sequence of W_i (see below). On such linear combinations, the functional $I(x)$ is transformed into a function $\psi(\alpha_1, \ldots, \alpha_n)$ of the coefficients α_i. These coefficients are chosen so that the function ψ is extremeised and hence are determined from the conditions:

$$\frac{\partial \psi}{\partial \alpha_i} = 0, \quad i = 1, 2, \ldots, n$$

Passing to the limit as $n \to \infty$, if the limit exists, we get the function

$$y = \sum_{i=1}^{\infty} \alpha_i W_i(x)$$

which for certain restrictions imposed on the functional $I(x)$ and on the sequence $W_1(x), \ldots, W_n(x), \ldots$ is the exact solution of the variational problem at hand. By confining ourselves to the first n terms we obtain an approximate solution of the variational problem. We search for a solution not in the set of all admissible functions but in a finite sub space of admissible functions as the set of polynomials of given degree. In this chapter we approximate the function x using so-called trial functions of B-spline form which satisfy the given boundary conditions but contain undetermined parameters to be optimised (the control points of the B-spline). By substituting this approximation into the functional, we integrate and minimise with respect to the unknown parameters.

For the trial functions to be admissible in the variational form of a problem of order m, the following conditions should be met (see for example Strang, [80]):

1. the trial functions posses m derivatives and hence be of class C^{m-1} across element boundaries
2. essential boundary conditions are met
3. the functional of the problem is computed exactly

Using the B-spline theory from chapter 3 and the reduced transformation technique to satisfy the essential boundary conditions we can ensure that the trial functions used for solution to our finite element problems are admissible in the variational principle.

5.2 B-splines and Finite Elements

The basic idea of finite element modelling is piecewise approximation, that is, the solution of a complicated problem is obtained by dividing the region of interest into small regions (finite elements) and approximating the solution over each subregion by a simple function. The functions

used to represent the behavior of the solution within an element, the interpolation functions, are most often polynomial in nature. The principal reasons for this are:

1. It is easier to formulate and code the finite element equations with polynomial type of interpolation functions and it is easier to perform integration and differentiation with polynomials.

2. It is possible to improve the accuracy of results by increasing the order of the polynomial. Theoretically a polynomial of infinite order corresponds to exact solution. The use of higher order elements is especially important when the gradient of the field variable is expected to vary rapidly.

B-splines, with their built-in continuity conditions, local control, efficient evaluation, fast subdivision and refinement lend themselves to problems where piecewise polynomial approximations to problems are desired. In addition to their geometric modelling properties there are a number of algorithmic advantages of B-spline bases which can lead to efficient simulation techniques:

- no mesh generation is required
- accurate approximations are possible with relatively low dimensional spaces
- smoothness and approximation order can be chosen arbitrarily
- integral polynomial form of arbitrary degree
- analytical derivatives and integrals

5.2.1 Accuracy

Considering the B-spline solution to the variational formulation of a given finite element problem, there are three possible ways to increase the accuracy of the solution:

1. Increase the number of B-spline basis functions in the Rayleigh-Ritz expansion. This is so called n-refinement where n refers to the upper index on the B-spline summation.

2. Elevate the degree of the B-spline used in the expansion, so called p-refinement.

3. Modify the knot vector without increasing the degrees of freedom.

In the examples considered in this chapter we focus on the first two of these noting that knot vector manipulation (in particular locating them at specific points) can be an important element in adaptive schemes. For 1 and 2 it is known in standard finite element theory that (see for example, Fischer et al, [22] or Strang, [80])

- as the number of B-spline functions in the expansion increases, the solution converges algebraically at a rate proportional to the expansion degree. This means that increasing the number of B-spline functions n in the approximation, the maximum error asymptotically decreases in proportion to n^{α} with slope α depending on the degree of the B-spline and the order of the variational problem.

- as the approximation degree p increases, the maximum error asymptotically decreases in proportion to $e^{\beta p}$, with slope β being a scheme and problem dependent parameter, and the rate of convergence is asymptotically independent of the the number of basis functions used.

In addition convergence of the sth derivative where s may be smaller or larger than m is proportional to n^{p-s}.

Roundoff also presents a potential source of error and is proportional to a positive power of n. The competition between the two opposing effects on accuracy generated by increasing the value of n becomes significant when the element mesh size reduces below a certain point. As shown in Strang, [80] this is dominated by a factor h^{-2m} where h is the mesh size. It does not depend strongly on the degree so where roundoff becomes a serious issue the normal course of action is to increase the degree of the expansion.

In the following sections we present algorithms for solving in B-spline form finite element type problems using the minimum energy variational formulation. For the curve case we graph a selection of static beam bending problems under various load and boundary conditions. For these examples, where analytic polynomial solutions are known, we fix the degree and segment number. Algorithms are presented for solving a range of plate bending problems and accuracy comparisons for varying degrees and segment numbers made with results based on analytic series solutions taken from Timoshenko, [85]. We then look at generalisations of the curve and surface algorithms to the solution of problems in the elastic deformation of isotropic solids. Finally, an example of heat flow through a cube is presented for which an analytic solution is also known.

We begin with a look at the one dimensional case.

5.3 1D Energy Minimisation Problems

5.3.1 Deflection of an elastic string

For a stretched string of length L at rest the potential energy is proportional to the change of length with the proportionality factor being the tension T. If $x(t)$ describes the deflection then the change of length assuming small deflections is given by

$$\sqrt{1+(\frac{dx}{dt})^2}\delta t-\delta t\approx\Big(1+\frac{1}{2}(\frac{dx}{dt})^2\Big)\delta t-\delta t=\frac{1}{2}(\frac{dx}{dt})^2\delta t.$$

The internal energy E_{int} is due to tension T:

$$E_{int}=\int_0^L\frac{1}{2}T(\frac{dx}{dt})^2dt.$$

External energy E_{ext} is due to the applied load conditions. Assuming distributed load functions given by $(F_i(t))_{i=1}^r$ defined over regions $[a_i,b_i]\in[0,L]$, and point forces $(P_i)_{i=1}^s$ at points (t_i^*), the external energy is

$$E_{ext}=\sum_{i=1}^r\int_{a_i}^{b_i}x(t)F_i(t)dt+\sum_{i=1}^sP_ix(t_i^*).$$

The total potential energy, $W(x)$, of the system is then given by:

$$W(x)=E_{int}-E_{ext}=\int_0^L\Big(\frac{1}{2}T(\frac{dx}{dt})^2-\sum_{i=1}^rF_i(t)x(t)\Big)dt-\sum_{i=1}^sP_ix(t_i^*).\tag{5.1}$$

Assuming a B-spline solution

$$x(t)=\sum_{i=1}^nd_iN_{i,k}(t),$$

and using 3.6 we minimise the internal energy expression to get the matrix equation

$$\frac{1}{2}T\mathbf{M}_1\mathbf{d}=\Big(\sum_{i=1}^p\int_{a_i}^{b_i}F_i(t)N_{j,k}(t)dx+\sum_{i=1}^qP_iN_{j,k}(t_i^*)\Big)_{j=1}^n.\tag{5.2}$$

5.3.2 The loaded beam problem

We take the case of a beam of length L and flexural rigidity $N = EI$ (also called the bending stiffness), where E is Young's modulus and I the moment of inertia of the beam, under a series of point and distributed loads. In particular, we seek to minimise the potential energy of the beam with deflection $x(t)$ under a series of applied distributed loads $(F_i(t))_{i=1}^{p}$ over $[a_i, b_i] \in [0, L]$ and point forces $(P_i)_{i=1}^{q}$ at points $t_i^* \in [0, L]$. The total energy of the system which represents a combination of the strain energy due to bending and the work done by the applied forces over the displacement x is given by

$$W(x) = \frac{1}{2}N \int_0^L (x''(t))^2 dt - \sum_{i=1}^{p} \int_{a_i}^{b_i} F_i(t)x(t)dt - \sum_{i=1}^{q} P_i x(t_i^*). \tag{5.3}$$

We treat two types of end condition, a simply supported beam and a built-in (cantilever) beam. The boundary conditions for these are are such that

- if an end is simply supported then the deflection and the bending moment $M = -EIx''$ must vanish. This leads to the requirements $x = x'' = 0$.
- if an end is built-in, then the deflection and slope must vanish, leading to the requirement $x = x' = 0$.

If the beam ends are free then $x''(t) = 0$ and $x'''(t) = 0$ at the ends. The variational functional takes care of this condition automatically.

We assume a form for the deflection $x(t)$ for the beam which satisfies all the geometric boundary conditions i.e. is a so-called admissible function. Using the principle of stationary potential energy and the Raleigh-Ritz technique we express the unknown function as a linear combination of known functions, in our case B-spline basis functions, with unknown coefficients:

$$x(t) = \sum_{i=1}^{n} d_i N_{i,k}(t).$$

Using 3.6 we minimise the strain energy functional given by equation 5.3 to get the matrix equation

$$\frac{1}{2}N\mathbf{M_2}\mathbf{d} = \Big(\sum_{i=1}^{p} \int_{a_i}^{b_i} F_i(t)N_{j,k}(t)dx + \sum_{i=1}^{q} P_i N_{j,k}(t_i^*)\Big)_{j=1}^{n}, \tag{5.4}$$

or, writing $\mathbf{M} = (m_{ij}^{(2)})_{i,j=1}^{n}$,

$$\frac{1}{2}N \begin{pmatrix} m_{11}^{(2)} & \dots & \dots & m_{1n}^{(2)} \\ \vdots & \ddots & & \vdots \\ \vdots & & \ddots & \vdots \\ m_{n1}^{(2)} & \dots & \dots & m_{nn}^{(2)} \end{pmatrix} \begin{pmatrix} d_1 \\ \vdots \\ \vdots \\ d_n \end{pmatrix} = \begin{pmatrix} \sum_{i=1}^{p} \int_{a_i}^{b_i} F_i(t) N_{1,k}(t) dt + \sum_{i=1}^{q} P_i N_{1,k}(t_i^*) \\ \vdots \\ \vdots \\ \sum_{i=1}^{p} \int_{a_i}^{b_i} F_i(t) N_{n,k}(t) dt + \sum_{i=1}^{q} P_i N_{n,k}(t_i^*) \end{pmatrix}. \quad (5.5)$$

System 5.5 is solved subject to the boundary conditions. Using the Ritz method, it is not necessary to satisfy all the boundary conditions. The procedure verifies the so-called natural boundary conditions automatically. In the case of beam bending all static conditions are natural, which means for example in simply supported ends the condition of disappearing bending moments, and for free ends the condition of disappearing second and third derivatives.

To fit the essential (geometric) boundary constraints we use the boundary point and derivative property of B-spline curves and proceed as follows. For boundary point interpolation to a value b at 0 we use algorithm 2.1 to generate the constraint equation

$$(\mathbf{N}(0))^T \cdot \mathbf{d} = b.$$

For boundary derivative interpolation to a value b^* say, we use algorithm 2.5 to generate the equation

$$(\mathbf{N}^1(0))^T \mathbf{D}_0^1 \cdot \mathbf{d} = b^*,$$

where $\mathbf{N}^1(0)$ represents the vector basis function set formed from the derivative knot set evaluated at 0. Having set up the system of equations corresponding to the geometric boundary conditions we use the reduced transformation technique presented in section 3.4 to obtain an admissible solution. From the B-spline solution we can then compute the bending moment ($= EIx^{(2)}(t)$) and reactive force profiles ($= EIx^{(3)}(t)$). The complete algorithm for a B-spline solution to the elastic beam bending problem (and string stretching) under a combination of point and distributed forces is presented in algorithm 5.1.

5.3.3 Examples

Algorithm 5.1 will produce exact solutions to minimisation problems for which exact polynomial solutions exist. For example, a stretched string of length 4 under a load function given by

Algorithm 5.1: B-spline solution to 1D beam bending/string stretching problem
given k and n, *solution* $x(t) = \sum_{i=1}^{n} d_i N_{i,k}(t)$

1. *create the basis function set* $(\mathtt{N_{i,k}(t)})_{\mathtt{i=1}}^{\mathtt{n}}$
2. *create the minimisation matrix* $\mathtt{M_2(M_1)}$ *from the basis function set, alg. 3.1*
3. *create a vector,* `loads`, *size* `n,` *to store point and distributed forces*
   all of which appear on rhs of final linear system
4. *for each point force at value* $\mathtt{t_i^*}$ *with magnitude* $\mathtt{P}_i$
   4.1 calculate value of curve basis function set, `b,` *at* $\mathtt{t_i^*}$, *that is* $\mathtt{b(t_i^*)} = \left(\mathtt{N_{j,k}(t_i^*)}\right)_{\mathtt{j=1}}^{\mathtt{n}}$
   4.2 then $\mathtt{loads+=P_i b(t_i^*)}$
5. *for each distributed force over given limits* $(\mathtt{a_i, b_i})$ *with magnitude* $\mathtt{F_i(t)}$
   5.1 calculate integral product of $\mathtt{F_i(t)}$ *with curve basis function set* `b` *over* $(\mathtt{a_i, b_i})$`,` *alg. 3.5*
   $$\mathtt{Integral(a_i, b_i)} = \left(\int_{\mathtt{a_i}}^{\mathtt{b_i}} \mathtt{F_i(t) N_{j,k}(t) dt}\right)_{\mathtt{j=1}}^{\mathtt{n}}$$
   5.2 then $\mathtt{loads+=Integral(a_i, b_i)}$
6. *determine the number of geometric point and derivative boundary conditions,* `num_bound`
7. *create a matrix* `(g_bound)` *size* `num_bound*n` *to store these conditions*
8. *for the left hand boundary*
   8.1 if there is a boundary point interpolation
      8.1.1 create the constraint equation for end control point, alg. 2.1, insert into matrix `g_bound`
   8.2 if there is boundary end derivative interpolation
      8.2.1 create the constraint equation for control point, alg. 2.5, insert into matrix `g_bound`
   8.3 if there is boundary end second derivative interpolation
      8.3.1 create the constraint equation for control point, alg. 2.5, insert into matrix `g_bound`
9. *repeat step 8 for the right hand boundary*
10. *add in any internal point interpolation conditions (supports), alg. 2.1, to the* `g_bound` *matrix*
11. *multiply the* `loads` *matrix by -1*
12. *find a set of elimination indices from the geometric boundary constraints matrix,* `g_bound`
13. *eliminate corresponding rows of the minimisation matrix* $\mathtt{M_2}$ *and rhs vector,* `loads`
14. *solve the resulting reduced system of equations*
15. *reconstitute the complete solution using the elimination information*
16. *build the B-spline curve from the control points found from 15*

Figure 5.1: Algorithm 5.1: B-spline solution to 1D beam bending problem

$F(t) = 1 + t^2/4$ over $[0, 4]$ and subject to the geometric boundary conditions $x(0) = x(4) = 0$, produces the exact solution

$$x(t) = \frac{1}{T}\Big(\frac{t^4}{48} + \frac{t^2}{2} - \frac{10t}{3}\Big).$$

For more extensive tests of the algorithm we consider a range of beam bending problems using simple and cantilever supports and under various point and distributed load conditions. In each case we compute a B-spline solution of order 4 with the number of segments set to be 10. Although most of the distributed loads are constant, in examples 4 and 5 we allow the load to vary according to a simple function. In example 8 we use algorithm 2.6 to compute the bending moment and reactive force profiles for a simply supported beam under both point and distributed loads. The oscillations that appear in the reactive force curve are due to the functional nature of the B-splines that are being used here (they represent a step change in the value of the reactive force). In the examples listed below the theoretically exact solutions (which are polynomial in nature), maximum displacements and reactive forces are given. Depending on the degree, the B-spline representation generated by algorithm 5.1 matches exactly the theoretical solution or approximates it. Figures 5.2 to 5.11 illustrate the beam configurations, the computed displacement functions, and, in example 7, the bending moment and reactive force profile curves.

Beam bending cases

1. A simply supported beam under a point load, P_1:

$$EIx(t) = \frac{P_1}{12}t^3 - \frac{P_1L^2}{16}t, \quad 0 \le t \le \frac{L}{2}, \quad x_{max} = -\frac{P_1L^3}{48EI}.$$

2. A simply supported beam under a uniform distributed load, F_1:

$$EIx(t) = \frac{F_1L}{12}t^3 - \frac{F_1}{24}t^4 - \frac{F_1L^3}{24}t, \quad x_{max} = -\frac{5}{384}\frac{F_1L^4}{EI}.$$

3. An overhanging simply supported beam under a uniform load, F_1:

$$EIx(t) = -\frac{F_1}{24}t^4 + \frac{F_1(L^4-1)}{96}t - \frac{F_1L^2}{3}t + \frac{F_1}{24} - \frac{F_1(L^4-1)}{96} + \frac{F_1L^2}{3}, \quad 0 < t < 1,$$
$$EIx(t) = -\frac{F_1}{24}t^4 + \frac{F_1L^2}{48}(t-1)^3 + \frac{F_1(L^4-1)}{96}t - \frac{F_1L^2}{3}t + \frac{F_1}{24} - \frac{F_1(L^4-1)}{96} + \frac{F_1L^2}{3},$$
$$1 < t < L$$

$$x_{max} = x_{t=3} = 3.4mm.$$

4. A simply supported beam under a linearly increasing distributed load with maximum value F_{max}:
$$EIx(t) = \frac{F_{max}L}{2}\Big(-\frac{1}{60L^2}t^5 + \frac{1}{18}t^3 - \frac{7L^2}{180}t\Big).$$

5. A simply supported beam under a triangular profile distributed load with maximum value F_{max}:
$$EIx(t) = \frac{23F_{max}L^3}{1536}t - \frac{F_{max}L}{48}t^3, \quad 0 < t < \frac{L}{4},$$
$$EIx(t) = \frac{F_{max}}{30L}\Big(t - \frac{L}{4}\Big)^5 - \frac{F_{max}L}{48}t^3 + \frac{23F_{max}L^3}{1536}t, \quad \frac{L}{4} < t < \frac{L}{2}.$$

6. A simply supported beam with multiple supports under point and distributed loads. Reaction forces computed at $t = 0, 4, 12, 17$:
$$R_{t=0m} = 1.68kN, \quad R_{t=4m} = 87.2kN, \quad R_{t=12m} = 84.5kN, \quad R_{t=17m} = 0kN.$$

7. A simply supported beam with multiple supports under point and distributed loads. Reaction forces computed at $t = 0, 6, 12, 18$:
$$R_{t=0m} = 26.9375kN, \quad R_{t=6m} = 33.375kN, \quad R_{t=12m} = 14.4375kN, \quad R_{t=18m} = 42.25kN.$$

8. Bending moment and reactive force profile curves for example 7:

9. Cantilever beam under point load P_1:
$$EIx(t) = -\frac{P_1L}{2}t^2 + \frac{P_1}{6}t^3, \quad x_{max} = -\frac{P_1L^3}{3EI}.$$

10. Cantilever beam under partial distributed load F_1:
$$EIx(t) = \frac{F_1L}{12}t^3 - \frac{3F_1L^2}{16}t^2, \quad 0 < t < \frac{L}{2},$$
$$EIx(t) = -\frac{F_1(L-t)^4}{24} - \frac{7F_1L^3}{48}t + \frac{15F_1L^4}{384}, \quad \frac{L}{2} < t < L, \quad x_{max} = \frac{41}{384}\frac{F_1L^4}{EI}.$$

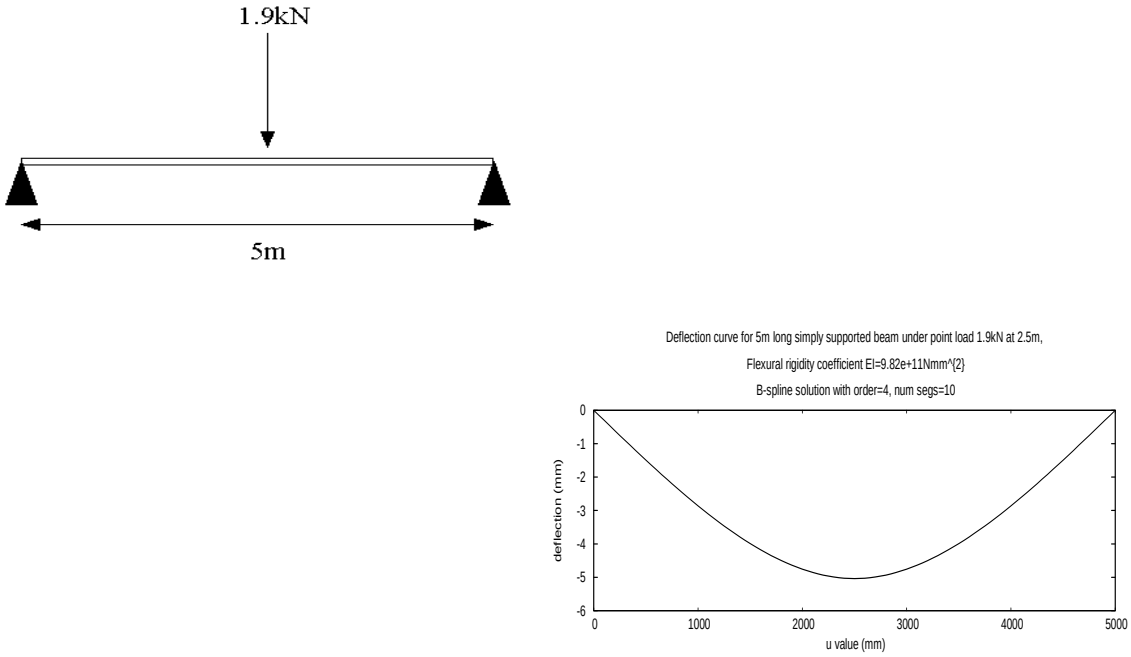


Figure 5.2: **Example 1:** Simply supported beam under a point load

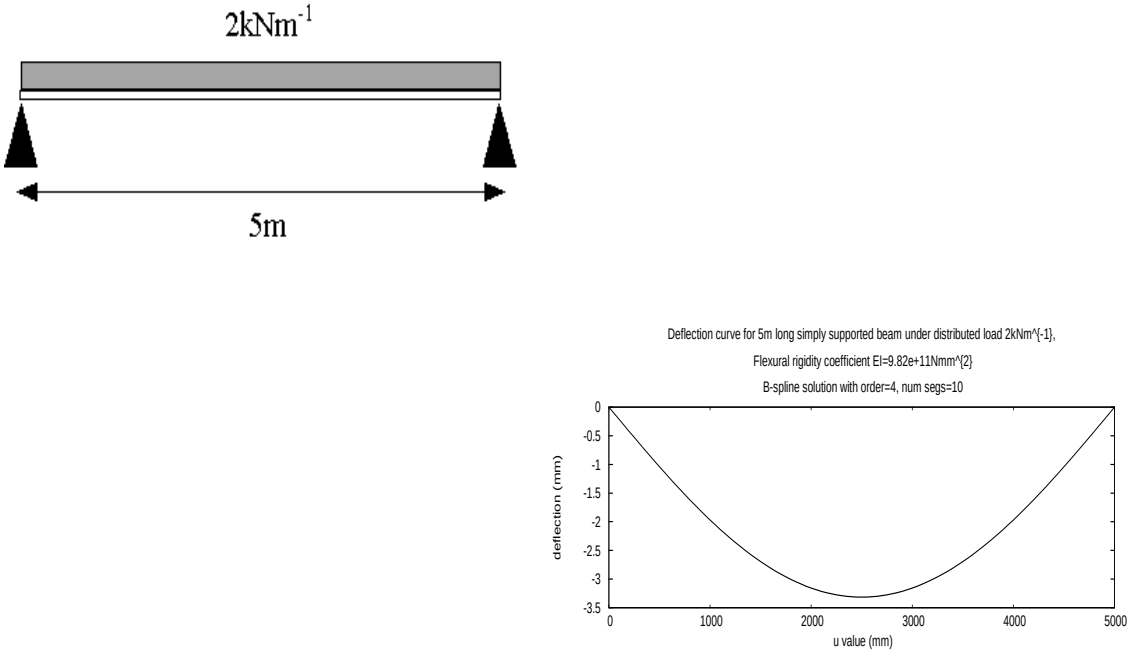


Figure 5.3: **Example 2:** Simply supported uniformly loaded beam

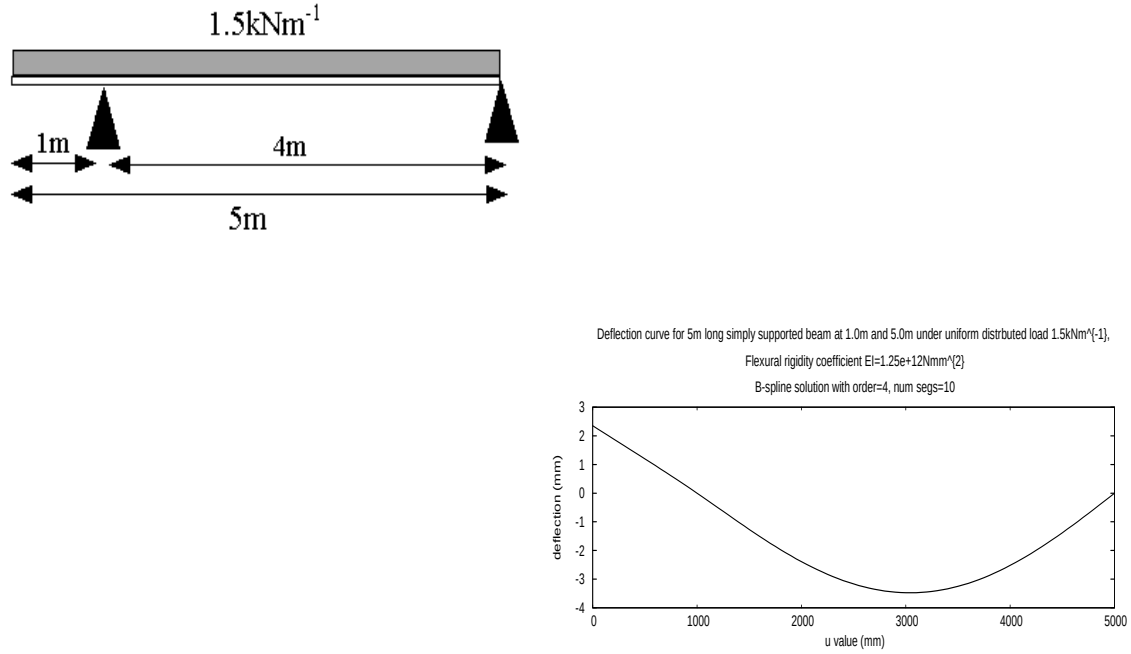


Figure 5.4: **Example 3:** Overhanging simply supported uniformly loaded beam

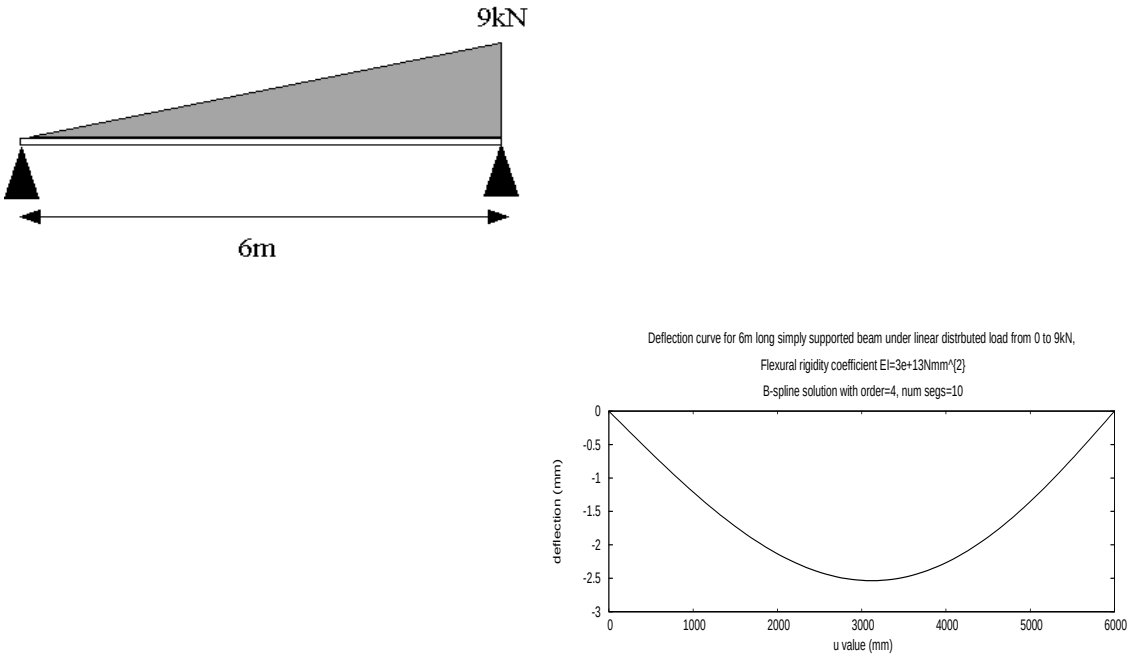
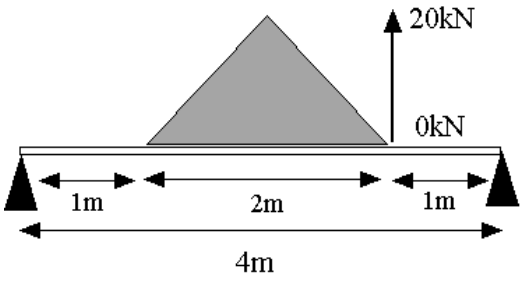


Figure 5.5: **Example 4:** Simply supported beam under ramp load



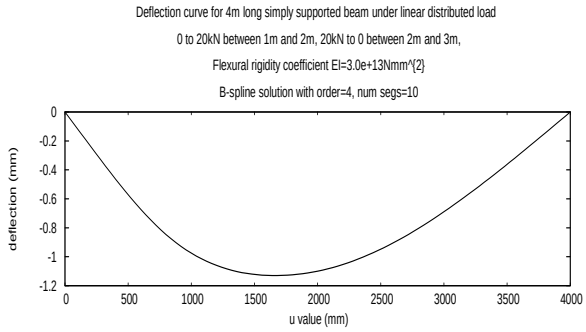
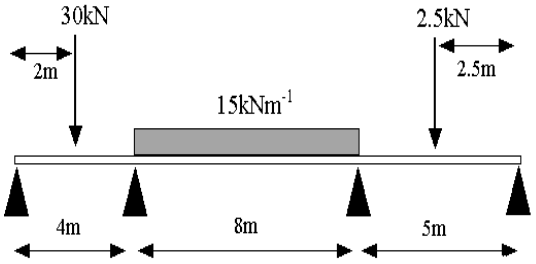


Figure 5.6: **Example 5:** Simply supported beam under triangular profile distributed load



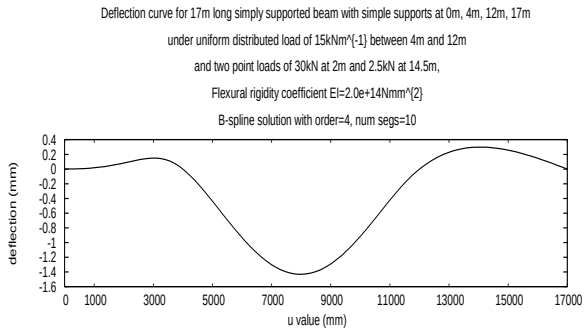


Figure 5.7: **Example 6:** Multiple simply supported beam under point and distributed loads

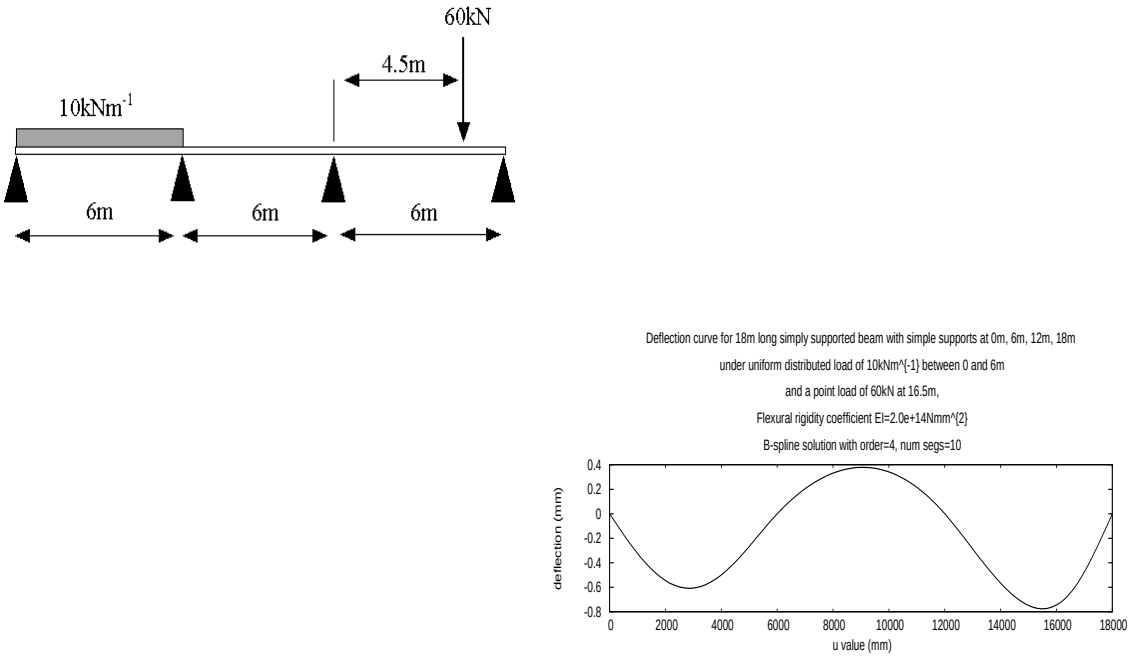


Figure 5.8: **Example 7:** Multiple simply supported beam under point and distributed loads

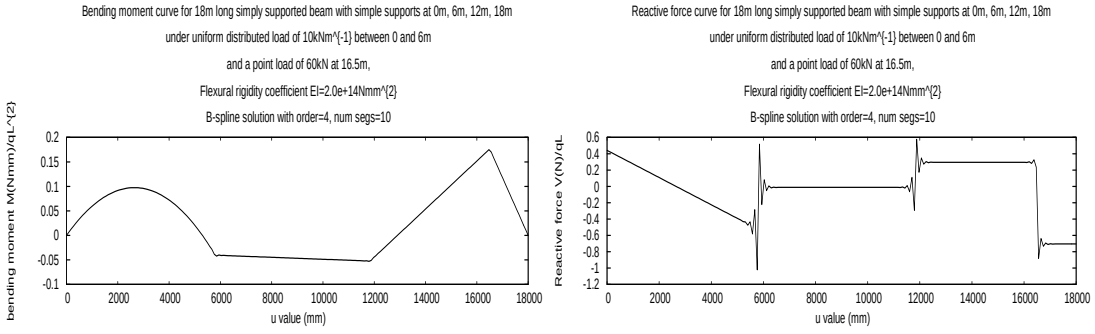


Figure 5.9: **Example 8:** Bending moment and reactive force curves for simply supported beam under point and distributed loads

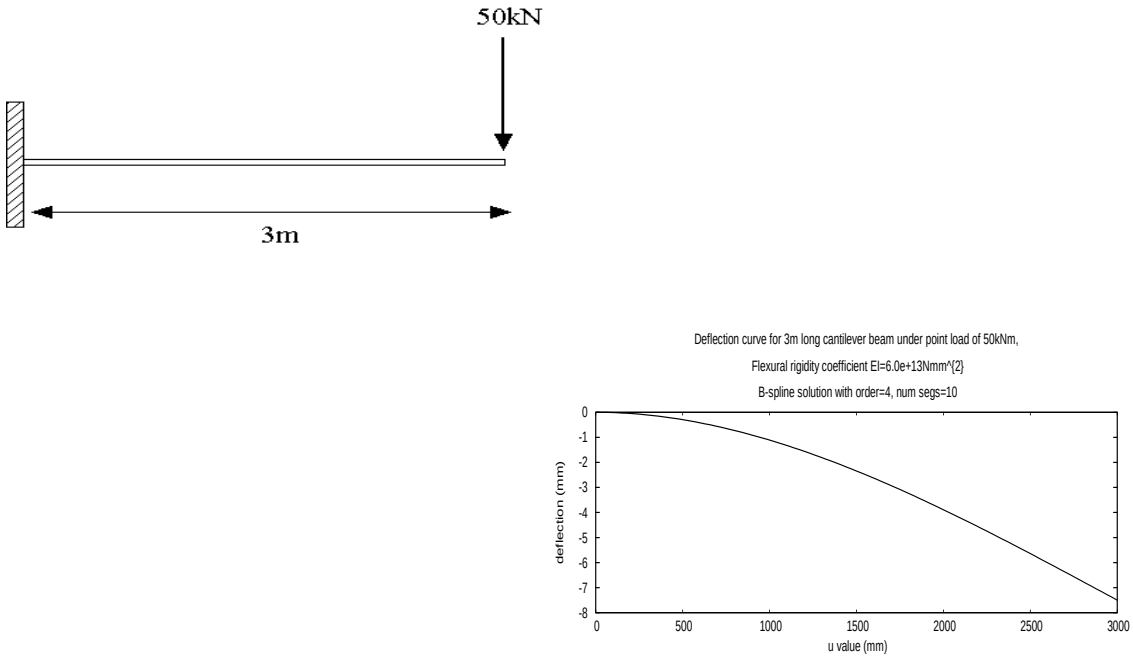


Figure 5.10: **Example 9:** Cantilever beam under point load

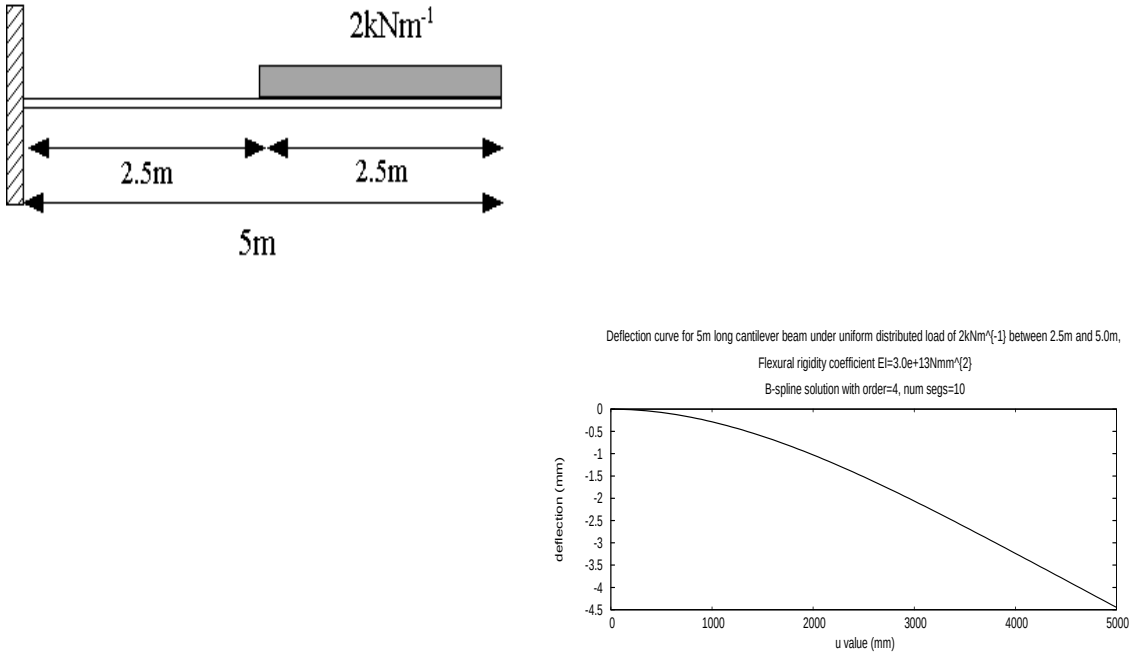


Figure 5.11: **Example 10:** Cantilever beam under partial distributed load

5.4 2D Energy Minimisation Problems

5.4.1 Membrane deflection

For a membrane at rest the potential energy is proportional to the change of area with the proportionality factor being the tension T. If $x(u,v)$ describes the deflection then the change of area for small deflections is given by

$$\sqrt{1+(\frac{\partial x}{\partial u})^2+(\frac{\partial x}{\partial v})^2}\delta u\,\delta v-\delta u\,\delta v\approx\frac{1}{2}\Big((\frac{\partial x}{\partial u})^2+(\frac{\partial x}{\partial v})^2\Big)\delta u\,\delta v.$$

The internal energy E_{int} is due to tension T and is given by

$$E_{int}^{mem}=\frac{1}{2}T\int_R\Big((\frac{\partial x}{\partial u})^2+(\frac{\partial x}{\partial v})^2\Big)du\,dv. \tag{5.6}$$

The external energy E_{ext} is due to the applied load conditions. Assuming distributed load functions given by $(F_i(u,v))_{i=1}^r$ defined over regions $[a_i,b_i]\times[c_i,d_i]$, and point forces $(P_i)_{i=1}^s$ at points (u_i^*,v_i^*), the external energy is

$$E_{ext}^{mem}=\sum_{i=1}^r\int_{a_i}^{b_i}\int_{c_i}^{d_i}x(u,v)F_i(u,v)du\,dv+\sum_{i=1}^s P_i x(u_i^*,v_i^*).$$

The total potential energy, $W(x)$, is then given by:

$$W(x)=E_{int}^{mem}-E_{ext}^{mem}=\int_R\Big(\frac{1}{2}T\Big((\frac{\partial x}{\partial u})^2+(\frac{\partial x}{\partial v})^2\Big)-\sum_i F_i(u,v)x(u,v)\Big)du\,dv-\sum_{i=1}^s P_i x(u_i^*,v_i^*). \tag{5.7}$$

Assuming a functional B-spline form for the displacement

$$x(u,v)=\sum_{i=1}^p\sum_{j=1}^q d_{ij}N_{i,k}(u)N_{j,l}(v),\quad\text{knot set}\quad(u_i)_{i=1}^{p+k}\times(v_j)_{j=1}^{q+l},$$

and by minimising the energy functional using 3.9 we have the following matrix system for the unknown control points $\mathbf{d}$:

$$\frac{T}{2}\Big(\mathbf{M}_1^u\mathbf{d}\mathbf{M}_0^v+\mathbf{M}_0^u\mathbf{d}\mathbf{M}_1^v\Big)=\left(\sum_{m=1}^r\int_{a_m}^{b_m}\int_{c_m}^{d_m}F_m(u,v)N_{i,k}(u)N_{j,l}(v)du\,dv+\sum_{m=1}^s P_m x(u_m^*,v_m^*)\right)_{i,j=1}^{p,q}. \tag{5.8}$$

5.4.2 The loaded plate problem

We consider the case of a rectangular plate of dimensions a, b under bending and under the following 'thin-plate' assumptions:

1. the thickness of the plate is small compared to the other dimensions
2. the deflections are small
3. the transverse shear deformation is zero

Under these conditions the bending energy per unit volume of such a plate is given by the following expression (see for example, Timoshenko [85], Parnes [66]):

$$\frac{E}{1+\nu}\left(\frac{1}{2(1-\nu)}\big(\frac{\partial^2 x}{\partial u^2}+\frac{\partial^2 x}{\partial v^2}\big)^2+\big(\frac{\partial^2 x}{\partial u \partial v}-\frac{\partial^2 x}{\partial u^2}\frac{\partial^2 x}{\partial v^2}\big)\right),$$

where E is Young's modulus relating the tension of the object to its stretch in the same direction (dimension N/m^2) and ν Poisson's ratio, the ratio between lateral contraction and axial extension. Here the uv plane is the undeformed plate (so-called 'neutral surface'), the w-axis is normal to the surface, and x is the displacement of a point on the neutral surface (the deflection function). The total bending energy of a deformed plate of thickness h is then given by

$$\frac{Eh^3}{24(1-\nu^2)}\int_R \Big[\big(\frac{\partial^2 x}{\partial u^2}+\frac{\partial^2 x}{\partial v^2}\big)^2+2(1-\nu)\big(\frac{\partial^2 x}{\partial u \partial v}-\frac{\partial^2 x}{\partial u^2}\frac{\partial^2 x}{\partial v^2}\big)\Big]\,du\,dv, \tag{5.9}$$

where $R=[0,a]\times[0,b]$. The total energy in plate bending consists of two parts, the strain energy of bending given by 5.9 and the work done by the applied point and distributed load conditions over the plate. Defining the flexural rigidity as

$$N=\frac{Eh^3}{12(1-\nu^2)}, \tag{5.10}$$

and assuming distributed load functions given by $(F_i(u,v))_{i=1}^r$ defined over regions $[a_i,b_i]\times[c_i,d_i]$, and point forces $(P_i)_{i=1}^s$ at points (u_i^*,v_i^*), the total energy, $W(x)$, is given by:

$$W(x)=\int_R \frac{N}{2}\Big[\big(\frac{\partial^2 x}{\partial u^2}\big)^2+\big(\frac{\partial^2 x}{\partial v^2}\big)^2+2\nu\frac{\partial^2 x}{\partial u^2}\frac{\partial^2 x}{\partial v^2}+2(1-\nu)\big(\frac{\partial^2 x}{\partial u \partial v}\big)^2\Big]\,du\,dv-$$
$$\sum_{i=1}^{r}\int_{a_i}^{b_i}\int_{c_i}^{d_i} x(u,v)F_i(u,v)\,du\,dv-\sum_{i=1}^{s}P_i x(u_i^*,v_i^*). \tag{5.11}$$

The first integral term in this expression is the strain energy functional. If a system is in a position of stable equilibrium, its total energy is a minimum, therefore the problem of bending of plates reduces to that of finding a function x of u and v that satisfies the given boundary conditions and makes the above integral a minimum. On computing a solution, the following expressions are used (see Timoshenko [85]) to calculate the bending/twisting moments and the shearing/reactive forces on the plate:

- bending moments in u and v, M_u, M_v:

$$M_u = -N\Big(\frac{\partial^2 x}{\partial u^2} + \nu\frac{\partial^2 x}{\partial v^2}\Big), \qquad M_v = -N\Big(\frac{\partial^2 x}{\partial v^2} + \nu\frac{\partial^2 x}{\partial u^2}\Big). \tag{5.12}$$

- twisting moment

$$M_{uv} = -M_{vu} = N(1-\nu)\frac{\partial^2 x}{\partial u \partial v}. \tag{5.13}$$

- shearing forces in u and v, Q_u, Q_v:

$$Q_u = \frac{\partial M_{uv}}{\partial v} + \frac{\partial M_u}{\partial u} = -N\frac{\partial}{\partial u}\Big(\frac{\partial^2 x}{\partial u^2} + \frac{\partial^2 x}{\partial v^2}\Big),$$

$$Q_v = -\frac{\partial M_{uv}}{\partial u} + \frac{\partial M_v}{\partial v} = -N\frac{\partial}{\partial v}\Big(\frac{\partial^2 x}{\partial u^2} + \frac{\partial^2 x}{\partial v^2}\Big). \tag{5.14}$$

- reactive forces in u and v, V_u, V_v:

$$V_u = \Big(Q_u - \frac{\partial M_{uv}}{\partial v}\Big), \qquad V_v = \Big(Q_v - \frac{\partial M_{uv}}{\partial u}\Big). \tag{5.15}$$

Assuming a functional B-spline form for the displacement

$$x(u,v) = \sum_{i=1}^{p}\sum_{j=1}^{q} d_{ij}N_{i,k}(u)N_{j,l}(v), \quad \text{knot set} \quad (u_i)_{i=1}^{p+k} \times (v_j)_{j=1}^{q+l},$$

and by minimising the strain energy functional from 5.11 using 3.9 and 3.11 we have the following matrix system for the unknown control points $\mathbf{d}$:

$$\begin{aligned}
&\frac{N}{2}\left(\mathbf{M}_2^u\mathbf{d}\mathbf{M}_0^v + \mathbf{M}_0^u\mathbf{d}\mathbf{M}_2^v + 2\nu\Big(\mathbf{M}_2^{0u}\mathbf{d}\mathbf{M}_2^{0v} + (\mathbf{M}_2^{0u})^T\mathbf{d}(\mathbf{M}_2^{0v})^T\Big) + 2(1-\nu)\mathbf{M}_1^u\mathbf{d}\mathbf{M}_1^v\right) \\
&= \left(\sum_{m=1}^{r}\int_{a_m}^{b_m}\int_{c_m}^{d_m} F_m(u,v)N_{i,k}(u)N_{j,l}(v)du\,dv + \sum_{m=1}^{s} P_m x(u_m^*, v_m^*)\right)_{i,j=1}^{p,q}.
\end{aligned} \tag{5.16}$$

5.4.3 Boundary conditions

The three possible types of boundary condition dealt with for an edge (taken as example $v = 0$) are the following:

1. Built-in edge: implies the deflection and slope along the edge is zero. The boundary conditions are
$$x(u,v)\Big|_{v=0} = 0, \quad \frac{\partial x}{\partial v}\Big|_{v=0} = 0.$$

2. Simply supported edge: this means that the deflection along the edge is zero, $x(u,0) = 0$, but that it can rotate freely with respect to this axis, which means there are no bending moments M_v along this edge
$$x(u,v)\Big|_{v=0} = 0, \qquad \Big(\frac{\partial^2 x}{\partial v^2} + \nu\frac{\partial^2 x}{\partial u^2}\Big)\Big|_{v=0} = 0.$$

3. Free-edge: implies no bending or twisting moments and no vertical shearing forces
$$M_u\Big|_{u=0} = 0, \quad M_{uv}\Big|_{u=0} = 0, \quad Q_u\Big|_{u=0} = 0.$$
These reduce to the following two conditions along the free edge:
$$\frac{\partial^3 x}{\partial u^3} + (2-\nu)\frac{\partial^3 x}{\partial u \partial v^2}\Big|_{u=a} = 0, \quad \frac{\partial^2 x}{\partial u^2} + \nu\frac{\partial^2 x}{\partial v^2}\Big|_{u=0} = 0.$$

As in the curve case the boundary conditions are simplified when using the Ritz method and the natural boundary conditions are verified automatically. In the case of plate bending all static conditions are natural, which means in free edges the conditions of disappearing moments (second derivatives) and shearing forces (third derivatives) and in simply supported edges the disappearing of the bending moments.

To fit the essential (geometric) boundary constraints we use the boundary curve and derivative property of B-spline surfaces, (figure 2.13), and proceed as follows. For boundary edge interpolation we create the constraint equations for an edge by setting up the system as follows: assume we wish to fit the function
$$\sum_{j=1}^{q} b_j N_{j,l}(v)$$
along the edge $u = 0$, for example. The set of q constraint equations is then given by
$$vec\Big(\mathbf{N}_u(0) \otimes \mathbf{e}_j\Big) \cdot vec(\mathbf{d}) = b_j, \quad \mathbf{e}_j = (0,\ldots,0,1,0,\ldots,0)^T,$$

$\mathbf{e}_j$ the jth unit vector of R^q, $j = 1, \ldots, q$. For boundary derivative interpolation along $u = 0$ to a function given by, for example,

$$\sum_{j=1}^{q} b_j^* N_{j,l}(v),$$

the set of q equations is

$$vec\Big(\big(\mathbf{N}_u^1(0)\mathbf{D}_u^1\big) \otimes \mathbf{e}_j\Big) \cdot vec(\mathbf{d}) = b_j^*, \quad j = 1, \ldots, q,$$

where $\mathbf{N}_u^1(0)$ represents the vector basis function set formed from the derivative knot set in u evaluated at 0 (see section 2.4.2). Particular care has to be taken when setting up these constraint equations so as not to introduce redundancy into the system. For example, fitting a boundary function along $u = 0$ reduces the number of constraint equations when fitting a derivative function along $v = 0$ because of the boundary/derivative control point overlap at the intersecting corner. When all constraints have been inserted into the constraint matrix the reduced transformation technique described in section 3.6 is then used to furnish an admissible solution. Based on equation 5.16, algorithm 5.2 presents a general B-spline solution to the 2D plate bending problem.

Algorithm 5.2: B-spline solution to 2D plate bending problem

Loads $(F_i(u,v)_{i=1}^r$, *point forces* $(P_i)_{i=1}^s$, *Poisson ratio* ν, *flexural rigidity* N
Solution $\sum_{i=1}^{p}\sum_{j=1}^{q} d_{ij}N_{i,k}(u)N_{j,l}(v)$, *knot set* $(u_i)_{i=1}^{p+k} \times (v_j)_{j=1}^{q+l}$

1. *create the surface basis function set* $(\mathtt{N_{i,k}(u)N_{j,l}(v)})_{\mathtt{i,j=1}}^{\mathtt{p,q}}$
2. *create the minimisation matrices from the basis function set for the 0, 1st and 2nd derivatives* $\mathtt{M_0^u, M_0^v, M_1^u, M_1^v, M_2^u, M_2^v}$, *alg. 3.1*
3. *create the minimisation matrices from the basis function set based on the (2,0), (0,2)* $(\mathtt{u}, \mathtt{v})$ *derivatives,* $\mathtt{M_2^{0u}, M_2^{0v}}$ *respectively, alg. 3.4*
4. *form the following kronecker products from these matrices* $\mathtt{vm1=M_0^v \otimes M_2^u, vm2=M_2^v \otimes M_0^u, vm3=2\nu(M_2^{0v} \otimes M_2^{0u})}$ $\mathtt{vm4=2\nu((M_2^{0v})^T \otimes (M_2^{0u})^T), vm5=2(1-\nu)(M_1^v \otimes M_1^u)}$
5. *form the minimisation matrix,* $\mathtt{mat=\frac{N}{2}(vm1+vm2+vm3+vm4+vm5)}$
6. *create a* `loads` *matrix size* `(p,q)` *to store the point and distributed forces and natural boundary conditions which all appear on rhs of final linear system*
7. *for each point force at given* $\mathtt{(u_i^*, v_i^*)}$ *value with magnitude* $\mathtt{P_i}$

7.1 calculate value of surface basis function set, `b,` *at* (u_i^*, v_i^*)`,` $b(u_i^*, v_i^*)=\left(N_{m,k}(u_i^*)N_{n,l}(v_i^*)\right)_{m,n=1}^{p,q}$

7.2 then `loads+=`$P_i b(u_i^*, v_i^*)$

8. for each distributed force over given region $R_i=[a_i, b_i]\times[c_i, d_i]$ *with magnitude* $F_i(u, v)$

8.1 calculate the integral product of the surface basis function set `b` *over* R_i`,` *alg. 3.6*

$$\texttt{Integral}(R_i)=\left(\int_{R_i} F_i(u,v)N_{m,k}(u)N_{n,l}(v)\,du\,dv\right)_{m,n=1}^{p,q}$$

8.2 then `loads+=Integral(`R_i`)`

9. Compute the number of boundary conditions, `num_bound`

10. Create a matrix `(g_bound)` *to store the geometric boundary conditions of size*`=num_bound*pq`

11. for the boundary edge `u=0`

11.1 if there is a boundary edge interpolation

11.1.1 convert boundary data to B-spline curve form

11.1.2 for each control point in `v`

11.1.2.1 create the constraint equation for the control point, insert into matrix `g_bound`

11.2 if there is boundary edge derivative interpolation

11.2.1 convert boundary data to B-spline curve form

11.2.2 for each control point in `v`

11.2.2.1 create the constraint equation for the control point, insert into matrix `g_bound`

12. repeat step 11 for the remaining three edges

13. add in any internal point interpolation conditions (supports) to the `g_bound` *matrix*

14. create an empty matrix `bound` *size* p, q *to store natural boundary conditions*

15. create four vectors `(n_bound1-4)` *for temporary storage*

16. for the boundary edge `u=0`

16.1 if there is a natural boundary condition

16.1.1 convert the interpolation data into B-spline curve form, `curv`

16.1.2 calculate the `v` *curve integral using the 1D product rule and* `v` *surface limits*

`vmin,vmax` *alg 3.5,* `n_bound1 += IntegralProd(curv,vmin,vmax)`

16.1.3 insert `n_bound1` *into* `bound` *along the bottom edge*

17. repeat 16 for the remaining 3 edges, inserting into appropriate boundary row/column of `bound`

18. subtract matrix 16 from the matrix of loads calculated in 6

19. create a Kronecker vector from the resulting matrix and multiply by -1

(this represents the rhs vector for the linear system)

20. find a set of elimination indices from the geometric boundary constraints

21. eliminate corresponding rows of the minimisation matrix, `mat` *(step 4), and rhs vector, 19*

22. solve the resulting reduced system of equations

23. reconstitute the complete solution using the elimination information

24. build the B-spline surface from the control points found

Figure 5.12: Algorithm 5.2: B-spline solution to 2D plate bending problem

5.5 Surface FEA Examples

Algorithm 5.2 will produce exact solutions to 2D minimisation problems for which exact polynomial solutions exist. For example, running the algorithm on the following 2D Poisson problem:

$$\frac{\partial^2 x}{\partial u^2} + \frac{\partial^2 x}{\partial v^2} = 2$$

in the region $0 \leq u \leq 1.5, 0 \leq v \leq 1$, with boundary conditions, $x = 3$ on $u = 0$ and with normal derivatives $\frac{\partial x}{\partial n}$, as shown in figure 5.13, returns the exact solution

$$x(u, v) = 3 + 2uv + v^2.$$

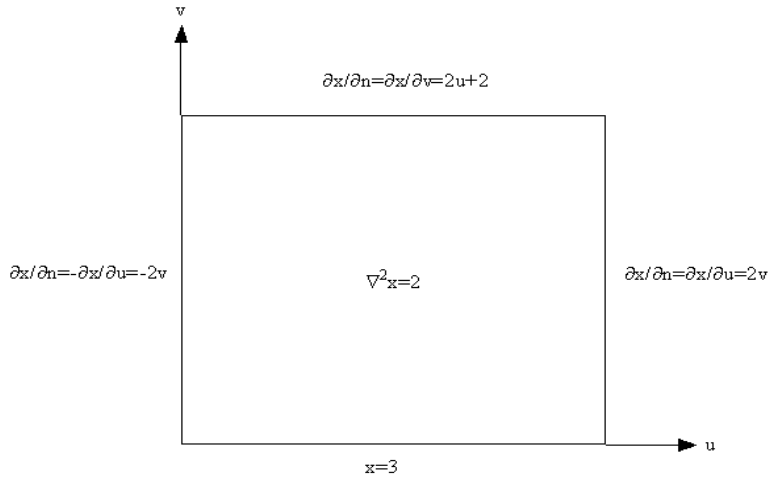


Figure 5.13: Region and boundary conditions for Poisson problem

We test algorithm 5.2 more extensively by finding B-spline solutions of varying orders and segment numbers to a selection of plate bending problems with load and boundary conditions taken from Timoshenko [85]. In particular we look at the following four cases:

1. Uniformly loaded, simply supported rectangular plate
2. Simply supported partially loaded square plate
3. Centrally point loaded, simply supported rectangular plate

4. Uniformly loaded square plate with two edges simply supported and two edges clamped

In addition we present numerical results for three more cases in appendix A:

- Uniformly loaded rectangular plate with two opposite edges simply supported, the third edge free and the fourth clamped
- Uniformly loaded rectangular plate with three edges simply supported and the fourth edge free
- Uniformly loaded rectangular plate with all edges clamped

In each of the above cases the theoretically exact solutions are known (for example, Timoshenko, [85]) and expressed in the form of trigonometric infinite series which allows us to examine the accuracy of the results.

5.5.1 Example 1: Uniformly loaded and simply supported rectangular plate

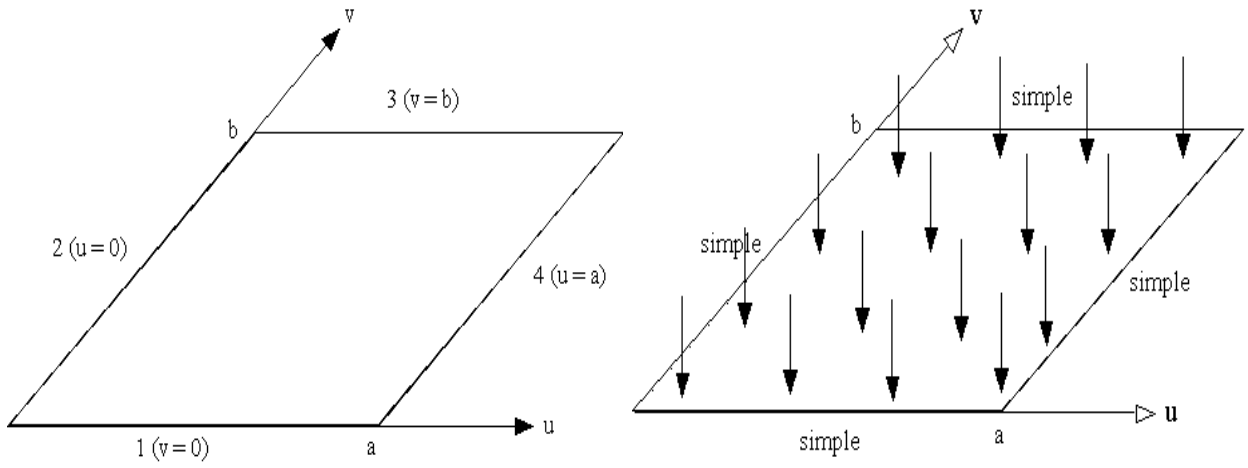


Figure 5.14: Rectangular plate with corresponding isoparametric lines and load/boundary conditions

For the case of a uniformly loaded, simply supported plate as shown in 5.14, the theoretically exact solution can be expressed in terms of a trigonometric series. For a uniform load $F_1 = F$,

the deflection $x(u,v)$ is given by:

$$x(u,v) = \frac{16F}{\pi^6 N} \sum_{m=1,3,5\ldots}^{\infty} \sum_{n=1,3,5\ldots}^{\infty} \frac{sin\frac{m\pi u}{a} sin\frac{n\pi v}{b}}{mn(\frac{m^2}{a^2} + \frac{n^2}{b^2})^2},$$

from which the maximum deflection of the plate (at the center) is given by

$$x_{max} = x(\frac{a}{2}, \frac{b}{2}) = \frac{16F}{\pi^6 N} \sum_{m=1,3,5\ldots}^{\infty} \sum_{n=1,3,5\ldots}^{\infty} \frac{-1^{\frac{m+n}{2}-1}}{mn\left(\frac{m^2}{a^2} + \frac{n^2}{b^2}\right)^2}.$$

The series expansions for bending and twisting moments can be obtained from 5.13 and 5.14.

Numerical results

Timoshenko uses the following forms to express the maximum deflection, maximum bending moments, shearing and reactive forces along the middle of the sides $u = a, v = b$:

$$\begin{aligned}
&x_{max} = \alpha \frac{Fa^4}{Eh^3}, \\
&(M_u)_{max} = \beta F a^2, \quad (M_v)_{max} = \beta_1 F a^2, \\
&(Q_u)_{u=a,v=b/2} = -\gamma F a, \quad (Q_v)_{u=a/2,v=b} = -\gamma_1 F a, \\
&(V_u)_{u=a,v=b/2} = -\delta F a, \quad (V_v)_{u=a/2,v=b} = -\delta_1 F a,
\end{aligned} \tag{5.17}$$

where $\alpha, \beta, \beta_1, \gamma, \gamma_1, \delta, \delta_1$ are numerical factors depending upon the ratio b/a of the sides of the plate and Poisson's ratio ν. We take the numerical results from the tables presented in [85] (the values are tabulated to 4dp) which are based on the theoretically exact values, and, for the case of a square plate, compute percentage absolute relative errors from the B-spline solution for the maximum values of the deflection, bending moments, shearing and reactive forces in u (the v values are identical due to symmetry). These value are computed using equations 5.12 to 5.15 and the B-spline surface generalisation of algorithm 2.6. We give results for a Poisson ratio of 0.3 noting that results for a different value of ν can be obtained by using 5.10 and multiplying by $(1-\nu^2)/0.91$. Figures 5.15 to 5.18 present the graphs of these errors for both fixed segment number and varying order and vice versa. Figure 5.19 gives a log-log plot of the error graphs for bending moment and shearing force. Together these graphs show the approximate relationships predicted from the theory:

- From the displacement error graph, 5.15, doubling the number of segments decrease the error by a factor of 2 for order 4

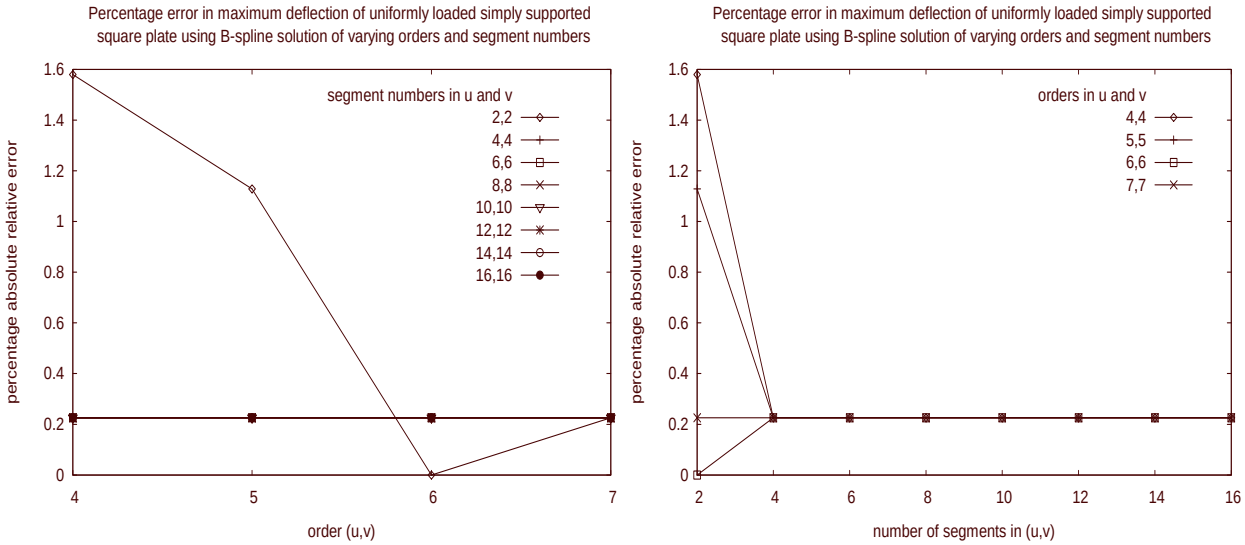


Figure 5.15: Percentage error in max deflection for uniformly loaded simply supported square plate

- From the shearing force error graph, 5.19, doubling the number of segments reduces the error by a factor of approximately two for order 4 and four for order 5.
- From figure 5.19, the rate of convergence when increasing the degree is asymptotically independent of the number of basis functions.

Appendix A.1 gives the numerical results in tabular form according to the parameters $\alpha, \beta, \beta_1, \gamma, \gamma_1, \delta, \delta_1$ for this case and for a selection of plates of varying ratios of b/a.

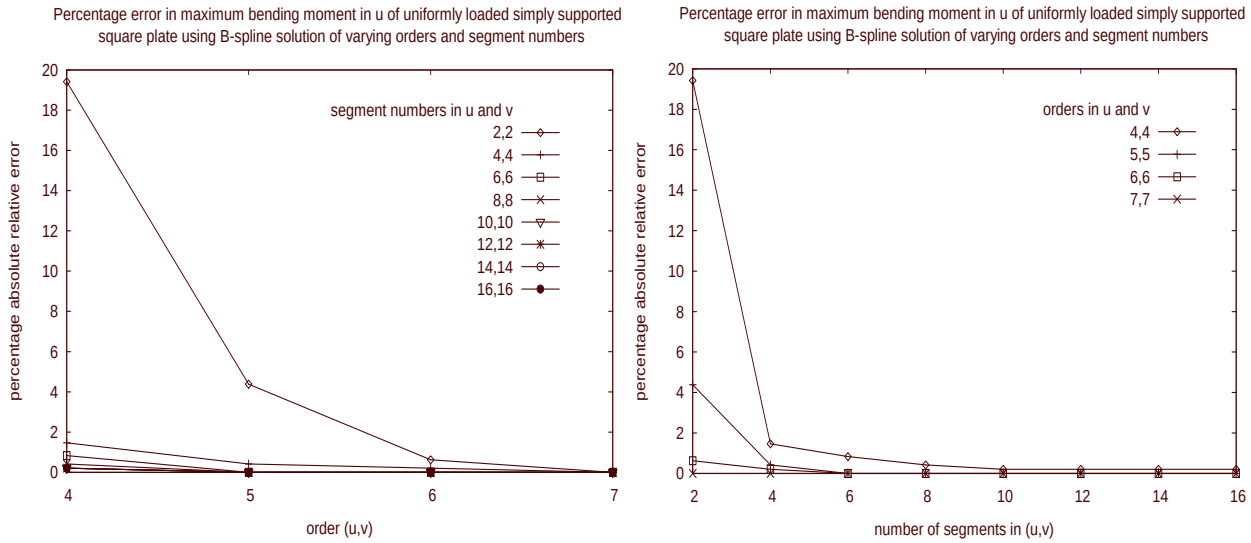


Figure 5.16: Percentage error in max bending moment in u for uniformly loaded simply supported square plate

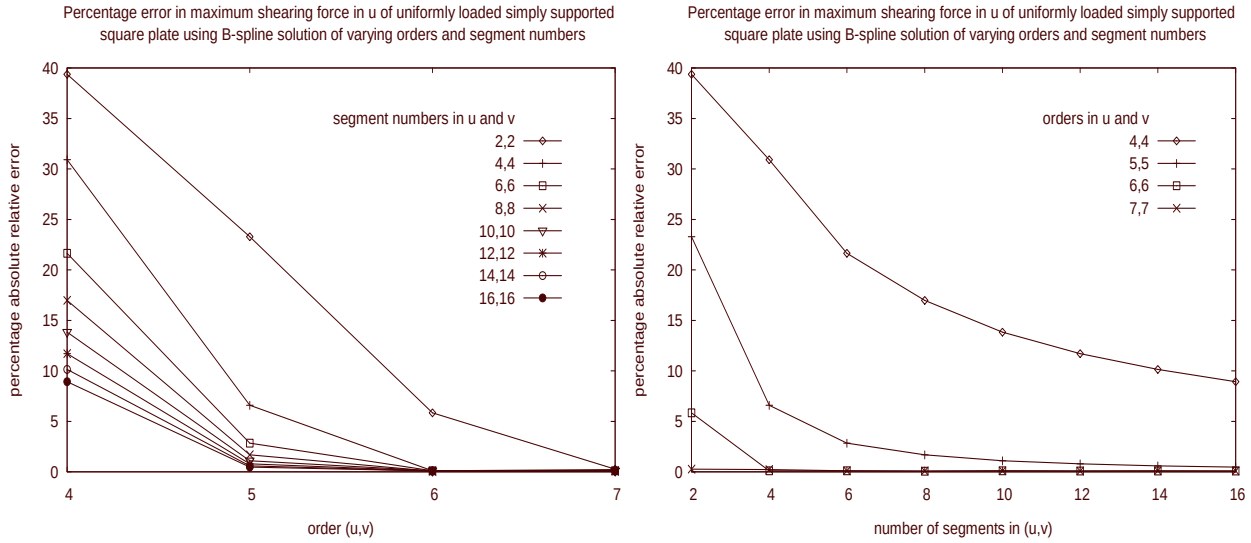


Figure 5.17: Percentage error in max shearing force in u for uniformly loaded simply supported square plate

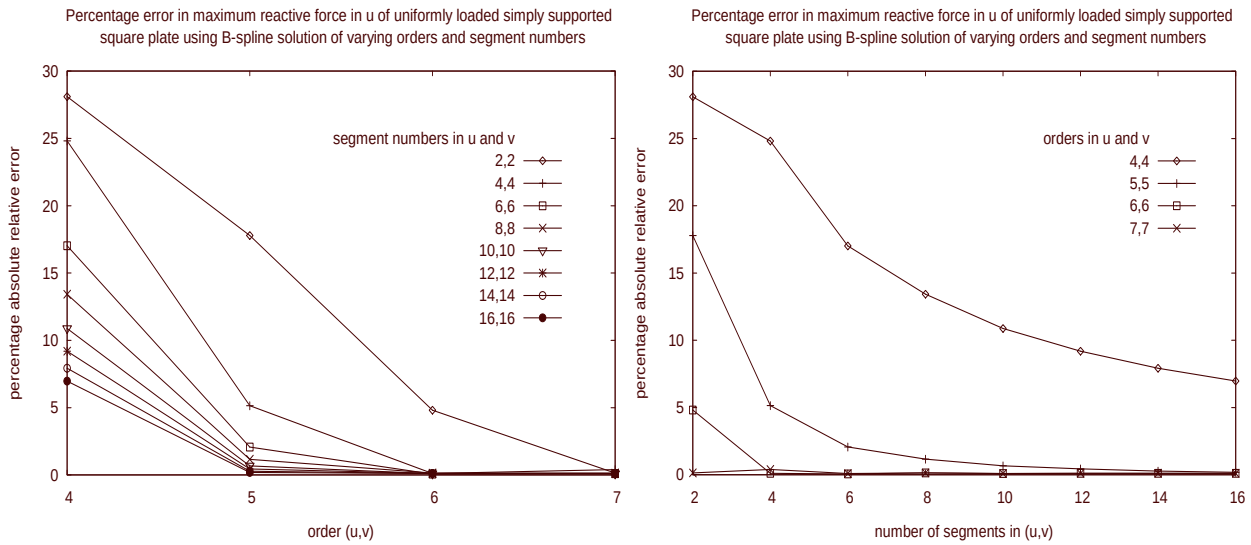


Figure 5.18: Percentage error in max reactive force in u for uniformly loaded simply supported square plate

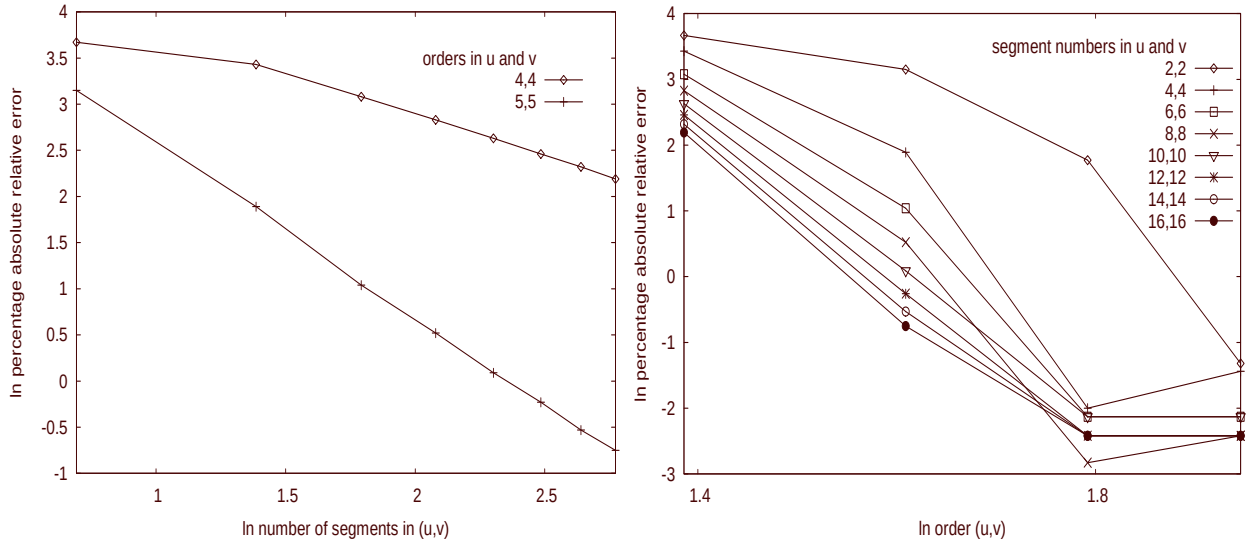


Figure 5.19: Log-log plot percentage error in max shearing force in u vs segment number/order for uniformly loaded simply supported square plate

5.5.2 Example 2: Centrally point loaded and simply supported rectangular plate

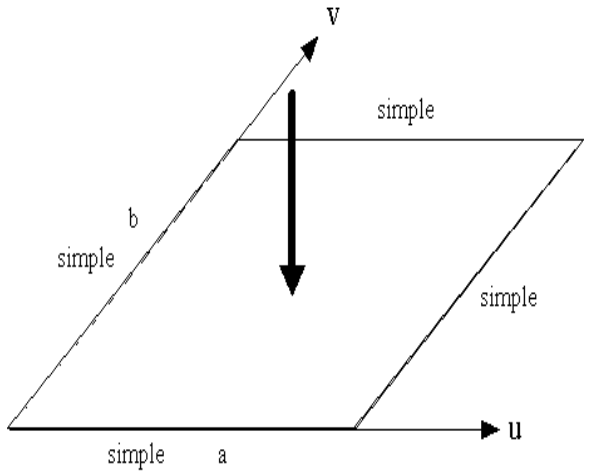


Figure 5.20: Simply supported centrally point loaded rectangular plate

For the second example we take the case of a centrally point loaded and simply supported rectangular plate as shown in 5.20. The solution for the maximum deflection under a concentrated force $P_1 = P$ perpendicular to the plate is given in [85] as

$$x(u,v) = \frac{Pab}{2N\pi^3} \sum_{m=1,3,5,\ldots}^{\infty} \frac{1}{m^3}\Big(tanh(\alpha_m) - \frac{\alpha_m}{cosh^2\alpha_m}\Big) = \alpha\frac{Pa^2}{Eh^3}.$$

We take the values of α corresponding to the maximum deflection from [85]. Figure 5.21 presents the percentage error graphs for varying orders and segment numbers for the case $b/a = 2$, and figure 5.22 shows the log-log plot of error against segment number. Appendix A.2 gives the numerical results for this and a selection of rectangular plates with varying ratio b/a under the same load and boundary conditions.

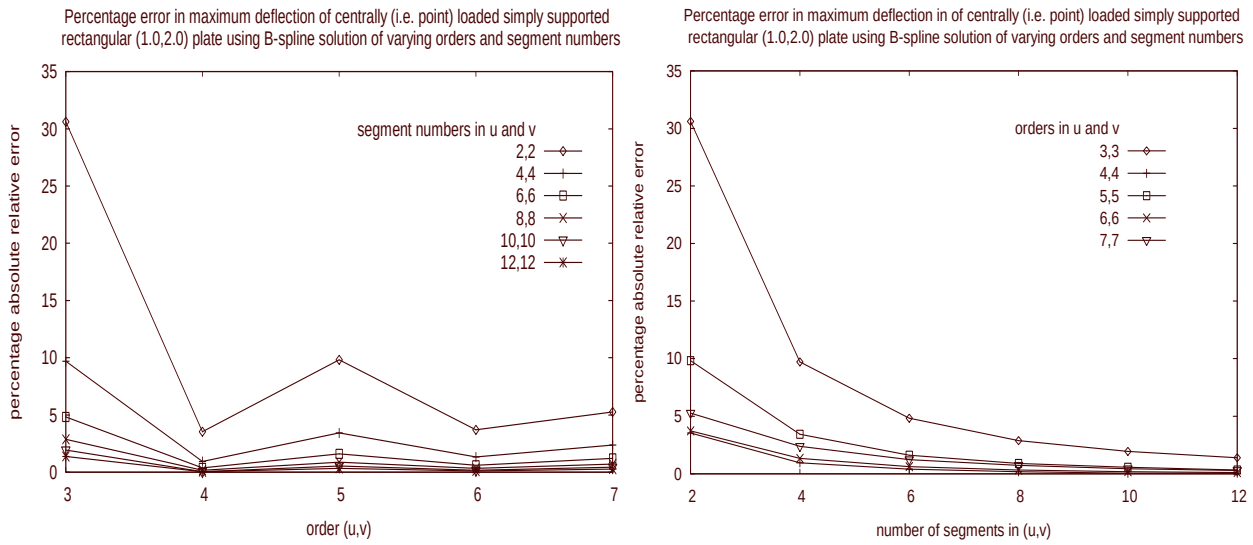


Figure 5.21: Percentage error in max deflection for centrally point loaded simply supported rectangular plate (1,2)

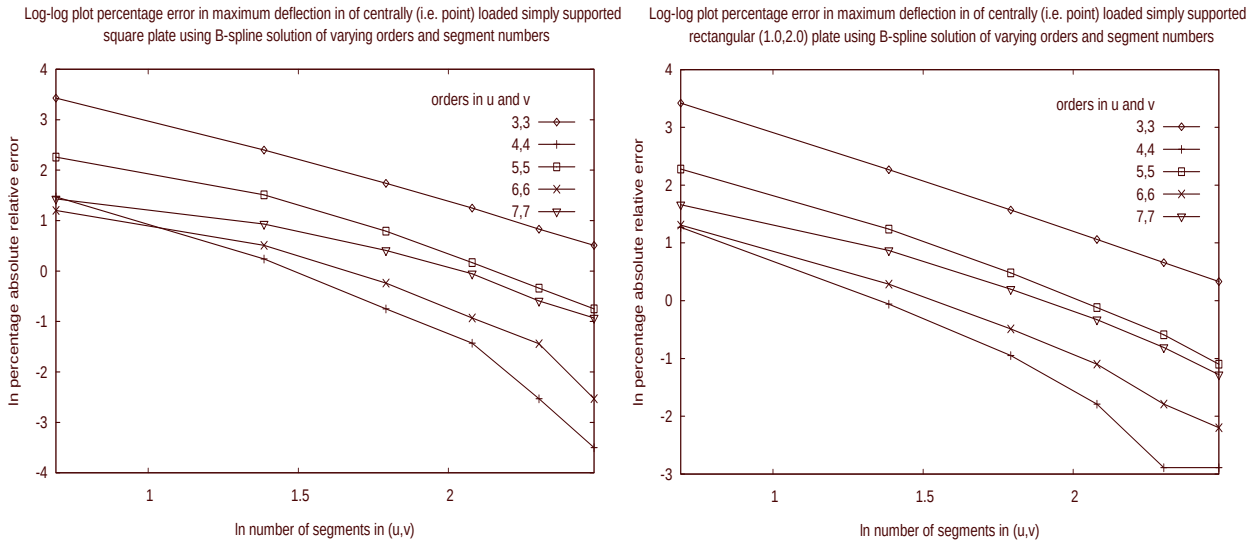


Figure 5.22: Log-log plot percentage error in max deflection vs segment number for centrally point loaded simply supported square/rectangular plate (1,2)

5.5.3 Example 3: Simply supported and partially loaded square plate

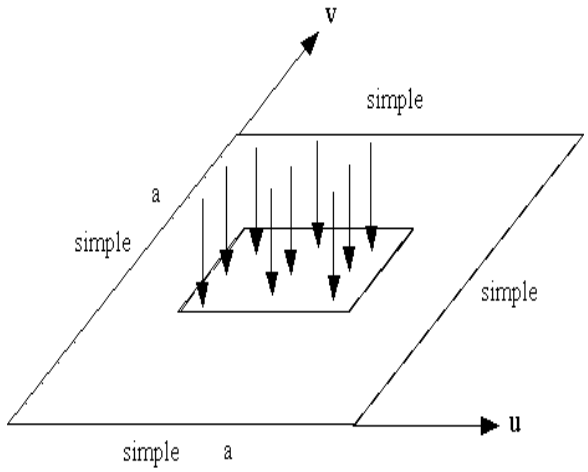


Figure 5.23: Simply supported partially loaded square plate

For the third example we take the case of a simply supported, partially loaded square plate, figure 5.23. We assume the load, $F_1 = F$, is centered on the plate and distributed uniformly over a rectangular region of side a_1, b_1. The B-spline solution is obtained using algorithm 5.2 and the factor β in the maximum bending moment in u (which occurs at the center) is computed using 5.12, where

$$(M_u)_{max} = \beta a_1^2 F = \beta P,$$

and $P = a_1^2 F$ is the total load. This is then compared with the value given in [85]. Figures 5.24 and 5.26 show error graphs for this bending moment factor in u for three values of a_1, $(0.1, 0.2, 0.5)$. The graphs display the absolute relative error between the value of β calculated from the theoretical result and that obtained from the B-spline solution for varying orders and segment numbers. The trend in increasing accuracy with increasing size of area bearing the load for a given segment number can be understood in the sense that the mesh of basis functions is better able to cover the defined region. For smaller load regions $a_1 \times b_1$ the smaller the coverage of the basis functions over that region. To achieve a target accuracy with smaller a_1, b_1 we require increased segment numbers.

The numerical results for the bending moment are presented in tabular form in Appendix A.3 for these cases and others with values of a_1, b_1 ranging from 0.1 to 1.0 in steps of 0.1.

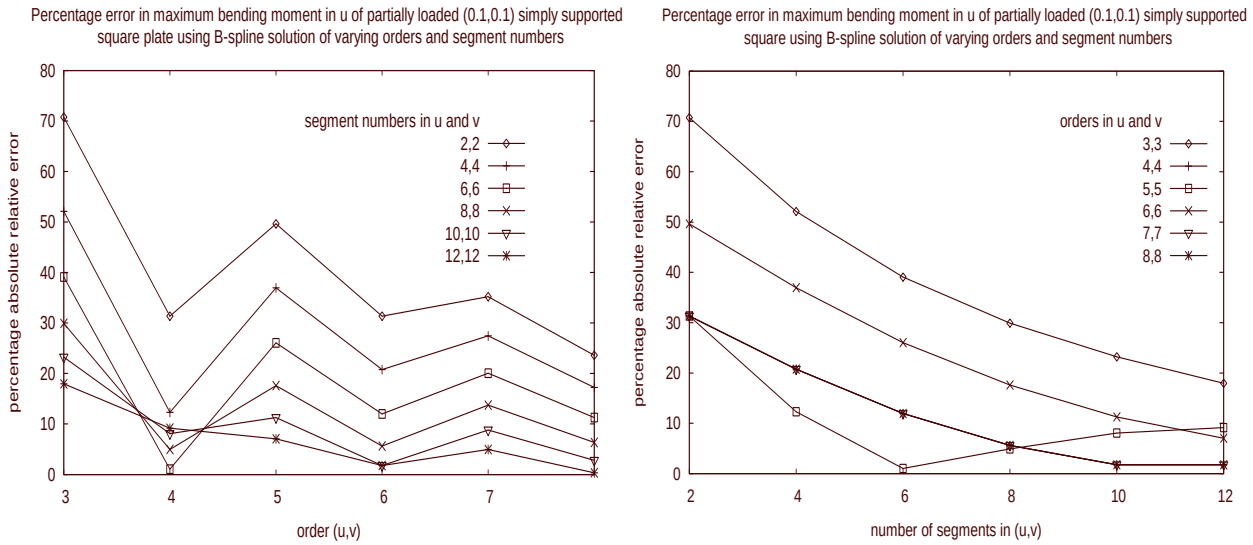


Figure 5.24: Percentage error in max bending moment in u for partially loaded (0.1,0.1) simply supported square plates

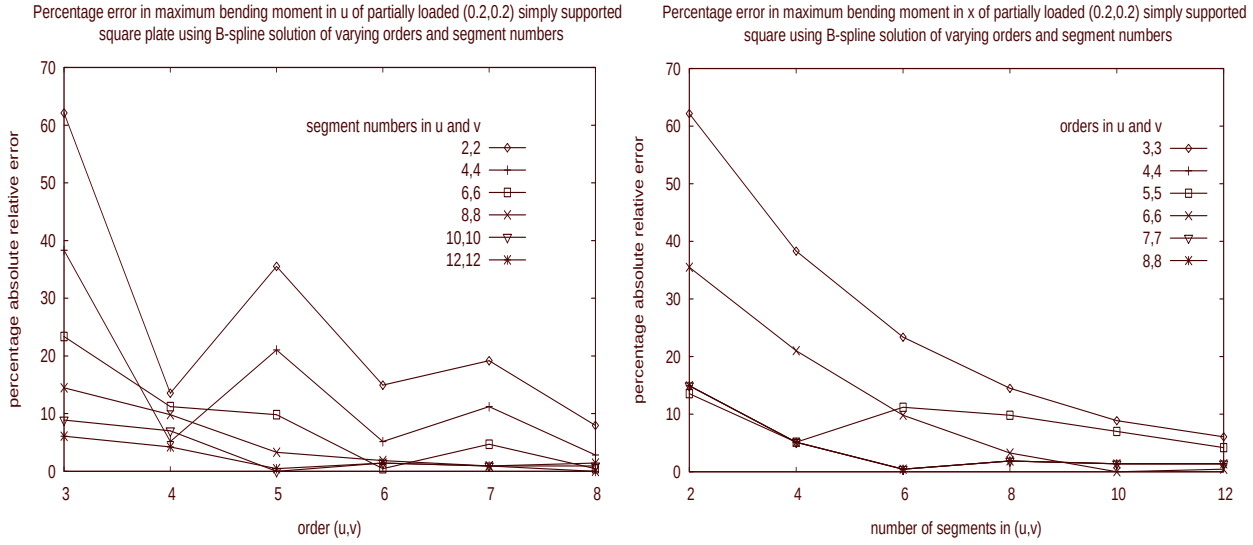


Figure 5.25: Percentage error in max bending moment in u for partially loaded (0.2,0.2) simply supported square plate

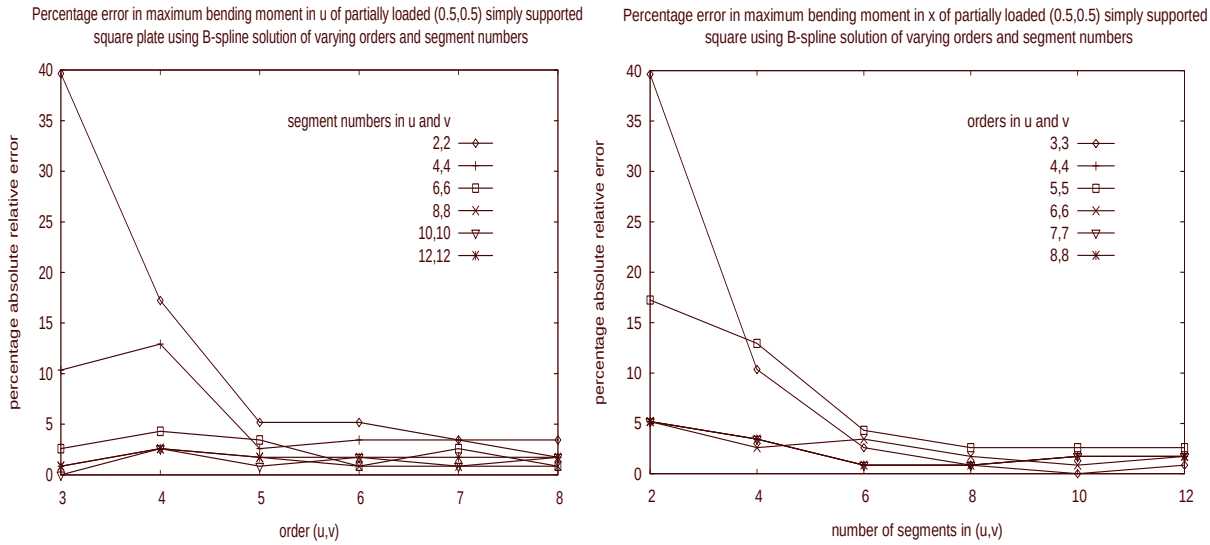


Figure 5.26: Percentage error in max bending moment in u for partially loaded (0.5,0.5) simply supported square plate

5.5.4 Example 4: Uniformly loaded rectangular plate with two edges simply supported and two edges clamped

For the final example we take the uniformly loaded rectangular plate but this time with mixed boundary conditions: two edges are clamped $u = 0, u = a$ and the remaining two simply supported, see figure 5.27. We use the tables provided in [85] and, for the case of a square plate, graph absolute relative percentage errors in the maximum deflection (which occurs at the center of the plate $(a/2, a/2)$), the bending moment in v along the side $v = a$, at $(a/2, a)$, and the bending moment of u at the center of the plate $(a/2, a/2)$. Figures 5.28, 5.30 and 5.29 display the error graphs based on the difference between the factors α, β, β_1 computed for the exact solution and the values obtained from the B-spline approximation for varying orders and segment numbers. Figure 5.31 shows a log-log plot of bending moment error against segment number and order for a restricted set of the data. The small number of visible points on figure 5.28 is due to the error reducing to zero within the 4dp accuracy we are using with the comparison data from [85]. Appendix A.4 gives the numerical results for α, β, β_1 in tabular form for this case and for a further selection of plates of varying ratio $a : b$ under the same conditions.

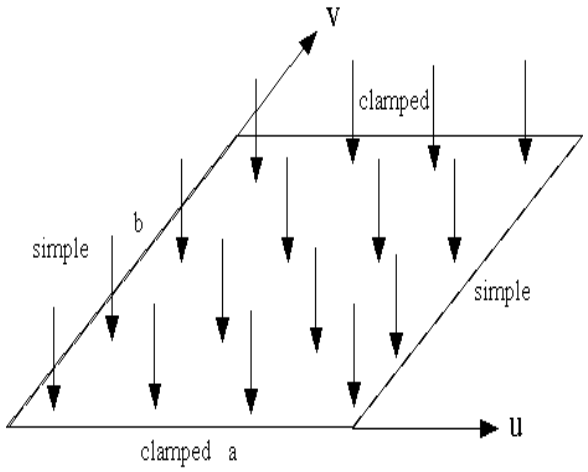


Figure 5.27: Uniformly loaded rectangular plate with two edges simply supported and two edges clamped

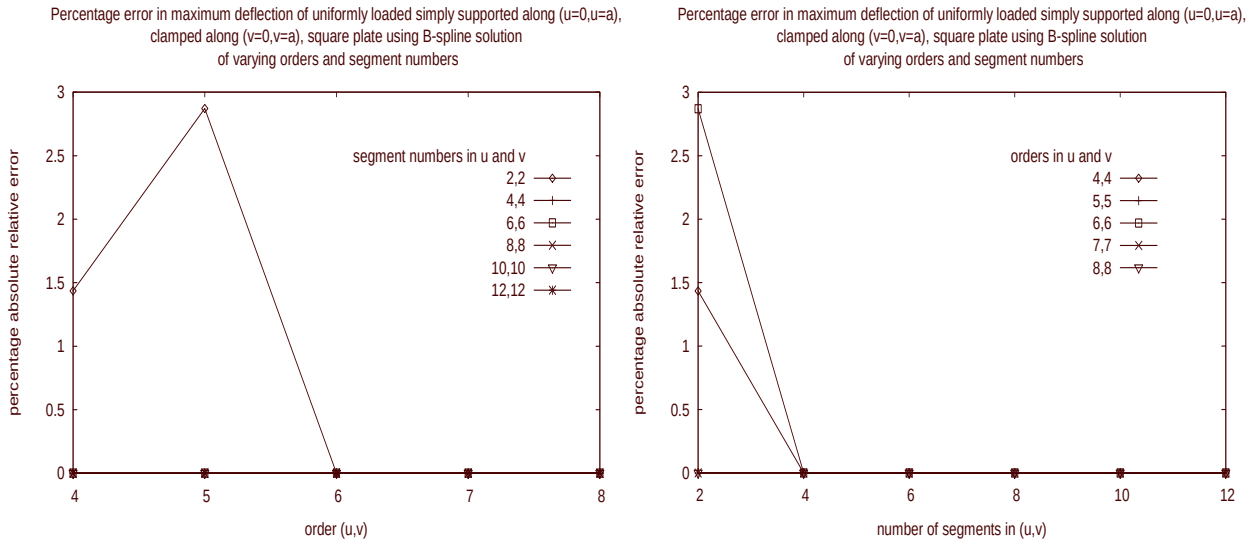


Figure 5.28: Percentage error in max deflection for uniformly loaded square plate simply supported on two edges and clamped on two edges

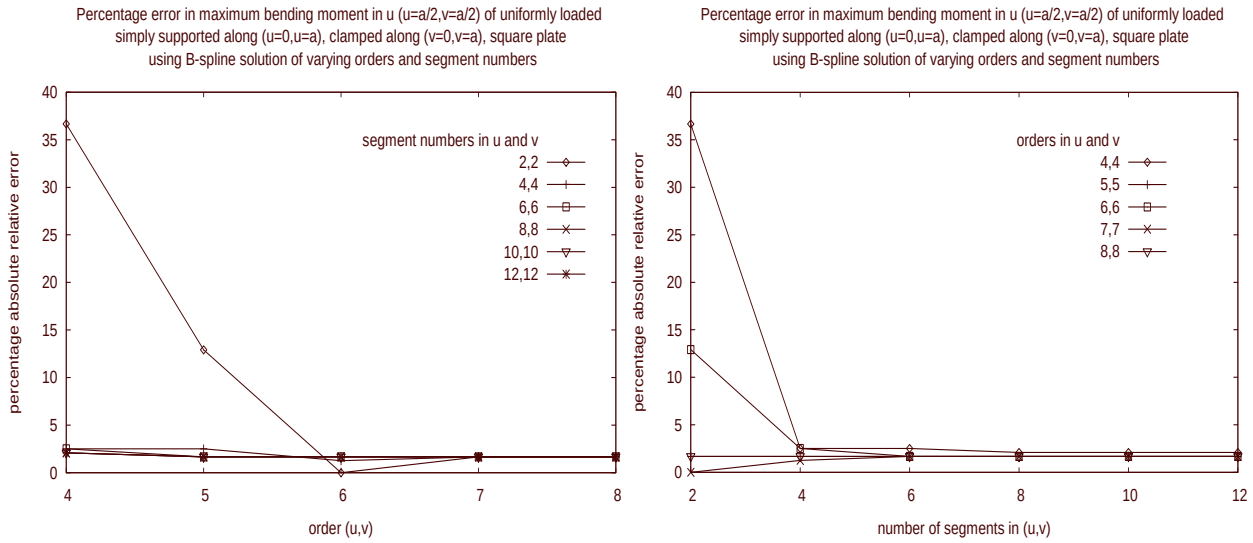


Figure 5.29: Percentage error in max bending moment in u for uniformly loaded square plate simply supported on two edges and clamped on two edges

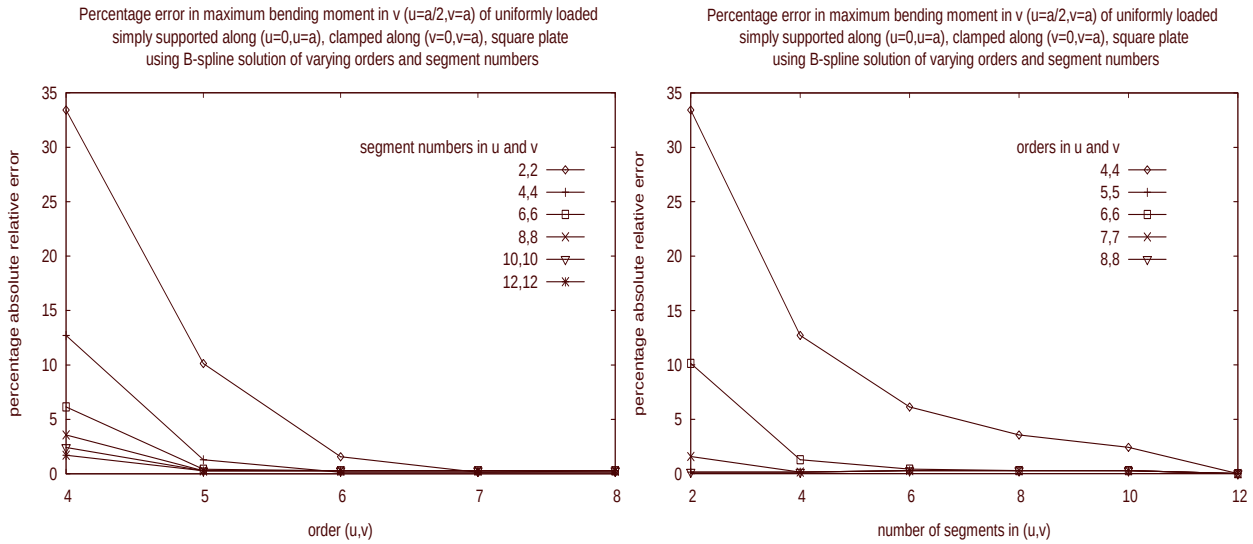


Figure 5.30: Percentage error in max bending moment in v for uniformly loaded square plate simply supported on two edges and clamped on two edges

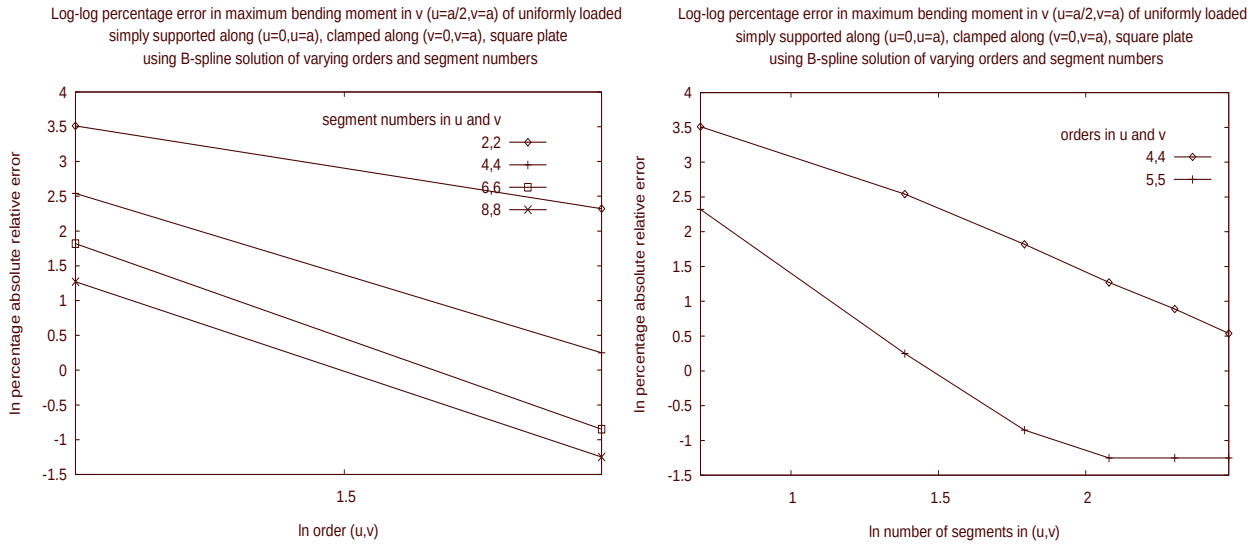


Figure 5.31: Log-log plot percentage error in max bending moment in v for uniformly loaded square plate simply supported on two edges and clamped on two edges

5.6 3D Energy Minimisation Problems

5.6.1 Deformation of isotropic elastic solids

In this final section we consider the possible generalisations of the energy minimisation expressions given by 5.4 and 5.16 for the one and two dimensional cases respectively, to solids. We consider linear elastic solids, that is materials that satisfy the following conditions:

1. they deform reversibly, if the loads are removed the solid returns to its original shape,
2. the strain in the solid depends only on the stress applied to it, not on the rate of loading or the history of loading,
3. the stress is a linear function of the strain,
4. the solid has no characteristic orientation (isotropic), i.e. the stress/strain curve is independent of orientation.

Membrane generalisation

An extension of the internal energy membrane term, 5.6, for an isotropic solid described by a parametric function

$$\mathbf{x}(u,v,w) = \begin{pmatrix} x(u,v,w) \\ y(u,v,w) \\ z(u,v,w) \end{pmatrix}$$

is given by

$$E_{int}(\mathbf{x}) = \int_V \Big((\frac{\partial x}{\partial u})^2 + (\frac{\partial x}{\partial v})^2 + (\frac{\partial x}{\partial w})^2 + (\frac{\partial y}{\partial u})^2 + (\frac{\partial y}{\partial v})^2 + (\frac{\partial y}{\partial w})^2 + (\frac{\partial z}{\partial u})^2 + (\frac{\partial z}{\partial v})^2 + (\frac{\partial z}{\partial w})^2 \Big) du\, dv\, dw,$$

where V represents the parametric domain of the solid.

Thin-plate energy generalisation

The thin plate internal strain energy component, 5.9, can be generalised to a volume in 3D in more than one way. One possible formulation is given by

$$\begin{aligned} E_{int}(\mathbf{x}) = \int_V \Bigg(& (\frac{\partial^2 x}{\partial u^2})^2 + (\frac{\partial^2 x}{\partial v^2})^2 + (\frac{\partial^2 x}{\partial w^2})^2 + 2\Big((\frac{\partial^2 x}{\partial u \partial v})^2 + (\frac{\partial^2 x}{\partial v \partial w})^2 + (\frac{\partial^2 x}{\partial u \partial w})^2 \Big) \\ & + (\frac{\partial^2 y}{\partial u^2})^2 + (\frac{\partial^2 y}{\partial v^2})^2 + (\frac{\partial^2 y}{\partial w^2})^2 + 2\Big((\frac{\partial^2 y}{\partial u \partial v})^2 + (\frac{\partial^2 y}{\partial v \partial w})^2 + (\frac{\partial^2 y}{\partial u \partial w})^2 \Big) \\ & + (\frac{\partial^2 z}{\partial u^2})^2 + (\frac{\partial^2 z}{\partial v^2})^2 + (\frac{\partial^2 z}{\partial w^2})^2 + 2((\frac{\partial^2 z}{\partial u \partial v})^2 + (\frac{\partial^2 z}{\partial v \partial w})^2 + (\frac{\partial^2 z}{\partial u \partial w})^2) \Big) du\, dv\, dw. \end{aligned} \tag{5.18}$$

Another more intuitive formulation based on solid mechanics and called the linear elastic model can be described as the following problem (see for example Parnes [65]) - given the geometry of a body, the loads, the boundary or support conditions and the stress-strain relation of the material, calculate the displacement field in the body.

We recall the stress (σ_{ij})- strain(ϵ_{ij}) rank 2 tensor relation from linear elasticity (using the summation convention):

$$\sigma_{ij} = \frac{E}{1+\nu}\Big(\epsilon_{ij} + \frac{\nu}{1-2\nu}\epsilon_{kk}\delta_{ij} \Big) - \frac{E\alpha\Delta T}{1-2\nu}\delta_{ij},$$

where E and ν are Young's modulus and poisson's ratio, α is the coefficient of thermal expansion, and ΔT is the increase in temperature of the solid. The strain field ϵ_{ij} is related to the

displacement by the following relations:

$$\epsilon_{11} = \frac{\partial x}{\partial u},\ \epsilon_{22} = \frac{\partial y}{\partial v},\quad \epsilon_{33} = \frac{\partial z}{\partial w},\ \epsilon_{12} = \frac{1}{2}\Big(\frac{\partial x}{\partial v} + \frac{\partial y}{\partial u}\Big),\quad \epsilon_{13} = \frac{1}{2}\Big(\frac{\partial x}{\partial w} + \frac{\partial z}{\partial u}\Big),\quad \epsilon_{23} = \frac{1}{2}\Big(\frac{\partial y}{\partial w} + \frac{\partial z}{\partial v}\Big), \tag{5.19}$$

with $\epsilon_{ij} = \epsilon_{ji}$. Assuming $\Delta T = 0$, the stored elastic energy in the solid is given by

$$E_{int} = \int\limits_V \frac{E}{2(1+\nu)}\Big(\epsilon_{ij}\epsilon_{ij} + \frac{\nu}{1-2\nu}\epsilon_{kk}\epsilon_{mm}\Big)du\,dv\,dw =$$
$$\int\limits_V \left(\frac{E}{2(1+\nu)}\Big(\epsilon_{11}^2 + \epsilon_{22}^2 + \epsilon_{33}^2 + 2\epsilon_{12}^2 + 2\epsilon_{13}^2 + 2\epsilon_{23}^2\Big) + \frac{E\nu}{2(1-2\nu)(1+\nu)}\big(\epsilon_{11} + \epsilon_{22} + \epsilon_{33}\big)^2\right)du\,dv\,dw \tag{5.20}$$

Using 5.19 we can write 5.20 as

$$E_{int}(\mathbf{x}) = \int\limits_V \left(\frac{\lambda}{2}\Big(\frac{\partial x}{\partial u} + \frac{\partial y}{\partial v} + \frac{\partial z}{\partial w}\Big)^2 + \mu\Big((\frac{\partial x}{\partial u})^2 + (\frac{\partial y}{\partial v})^2 + (\frac{\partial z}{\partial w})^2\Big) + \right.$$
$$\left.\frac{\mu}{2}\Big((\frac{\partial x}{\partial v})^2 + (\frac{\partial x}{\partial w})^2 + (\frac{\partial y}{\partial u})^2 + (\frac{\partial y}{\partial w})^2 + (\frac{\partial z}{\partial u})^2 + (\frac{\partial z}{\partial v})^2 + 2\frac{\partial x}{\partial v}\frac{\partial y}{\partial u} + 2\frac{\partial x}{\partial w}\frac{\partial z}{\partial u} + 2\frac{\partial y}{\partial w}\frac{\partial z}{\partial v}\Big)\right)du\,dv\,dw, \tag{5.21}$$

where λ and μ are elastic weighting terms called the Lame constants. These can be written in terms of of the more intuitive constants E, Young's modulus and ν Poisson's ratio:

$$E = \mu\frac{(3\lambda + 2\mu)}{\lambda + \mu},\quad \nu = \frac{\lambda}{2(\lambda + \mu)}.$$

For our purposes we use a similar but slightly simplified form of the linear elastic model using the following energy form taken from Rappoport et al [75] and Terzopoulos & Qin, [82]:

$$E_{int} = \alpha\int\limits_V \left(\Big(\frac{\partial x}{\partial u}\Big)^2 + \Big(\frac{\partial x}{\partial v}\Big)^2 + \Big(\frac{\partial x}{\partial w}\Big)^2 + \Big(\frac{\partial y}{\partial u}\Big)^2 + \Big(\frac{\partial y}{\partial v}\Big)^2 + \Big(\frac{\partial y}{\partial w}\Big)^2 + \right.$$
$$\left.\Big(\frac{\partial z}{\partial u}\Big)^2 + \Big(\frac{\partial z}{\partial v}\Big)^2 + \Big(\frac{\partial z}{\partial w}\Big)^2\right)du\,dv\,dw + \beta\int\limits_V \left(\frac{\partial x}{\partial v}\frac{\partial y}{\partial u} + \frac{\partial x}{\partial w}\frac{\partial z}{\partial u} + \frac{\partial y}{\partial w}\frac{\partial z}{\partial v}\right)du\,dv\,dw, \tag{5.22}$$

where α and β are material property constants. Using the two internal energy forms, 5.18 and 5.22, we form total energy expressions suitable for minimisation.

Total energy expressions

Using the internal energy form given by 5.18 and assuming distributed load functions given by $(\mathbf{F}_i(u,v,w))_{i=1}^{r_1}$ defined over regions $[a_i,b_i]\times[c_i,d_i]\times[e_i,f_i]$ in (u,v,w) space, and point forces $(\mathbf{P}_i)_{i=1}^{s_1}$ applied at points (u_i^*,v_i^*,w_i^*), the total potential energy is given by

$$\begin{aligned}W(\mathbf{x}) = \int_V \Bigg(&\big(\frac{\partial^2 x}{\partial u^2}\big)^2 + \big(\frac{\partial^2 x}{\partial v^2}\big)^2 + \big(\frac{\partial^2 x}{\partial w^2}\big)^2 + 2\Big(\frac{\partial^2 x}{\partial u\partial v} + \frac{\partial^2 x}{\partial v\partial w} + \frac{\partial^2 x}{\partial u\partial w}\Big) + \\ &\big(\frac{\partial^2 y}{\partial u^2}\big)^2 + \big(\frac{\partial^2 y}{\partial v^2}\big)^2 + \big(\frac{\partial^2 y}{\partial w^2}\big)^2 + 2\Big(\frac{\partial^2 y}{\partial u\partial v} + \frac{\partial^2 y}{\partial v\partial w} + \frac{\partial^2 y}{\partial u\partial w}\Big) + \\ &\big(\frac{\partial^2 z}{\partial u^2}\big)^2 + \big(\frac{\partial^2 z}{\partial v^2}\big)^2 + \big(\frac{\partial^2 z}{\partial w^2}\big)^2 + 2\Big(\frac{\partial^2 z}{\partial u\partial v} + \frac{\partial^2 z}{\partial v\partial w} + \frac{\partial^2 z}{\partial u\partial w}\Big)\Bigg) du\,dv\,dw - \\ &\sum_{i=1}^{r_1}\int_{a_i}^{b_i}\int_{c_i}^{d_i}\int_{e_i}^{f_i} \mathbf{x}(u,v,v)\mathbf{F}_i(u,v,w)du\,dv\,dw - \sum_{i=1}^{s_1}\mathbf{P}_i\mathbf{x}(u_i^*,v_i^*,w_i^*).\end{aligned} \tag{5.23}$$

The first integral term in this expression is the solid strain energy functional and the other two terms represent the work done by the applied point forces and distributed loads. We seek a parametric function $\mathbf{x} = (x(u,v,w), y(u,v,w), z(u,v,w))$ of u,v,w that satisfies the given boundary conditions and makes the above integral a minimum. Assuming a B-spline solution to the displacement function

$$\mathbf{x}(u,v,w) = \sum_{i=1}^{p}\sum_{j=1}^{q}\sum_{k=1}^{r}\mathbf{d}_{ijk}N_{i,l}(u)N_{j,m}(v)N_{k,n}(w),\ \text{knot set}\ (u_i)_{i=1}^{p+l}\times(v_j)_{j=1}^{q+m}\times(w_k)_{k=1}^{r+n},$$

and by minimising the strain energy functional using 3.13 and 5.4 we obtain the following matrix system for the unknown control points:

$$\begin{aligned}&\mathbf{M}_2^u\otimes_i\mathbf{d}\otimes_j\mathbf{M}_0^v\otimes_k\mathbf{M}_0^w + \mathbf{M}_0^u\otimes_i\mathbf{d}\otimes_j\mathbf{M}_2^v\otimes_k\mathbf{M}_0^w + \mathbf{M}_0^u\otimes_i\mathbf{d}\otimes_j\mathbf{M}_0^v\otimes_k\mathbf{M}_2^w + \\ &2\Big(\mathbf{M}_1^u\otimes_i\mathbf{d}\otimes_j\mathbf{M}_1^v\otimes_k\mathbf{M}_0^w + \mathbf{M}_0^u\otimes_i\mathbf{d}\otimes_j\mathbf{M}_1^v\otimes_k\mathbf{M}_1^w + \mathbf{M}_1^u\otimes_i\mathbf{d}\otimes_j\mathbf{M}_0^v\otimes_k\mathbf{M}_1^w\Big) = \\ &\left(\sum_{t=1}^{r_1}\int_{a_t}^{b_t}\int_{c_t}^{d_t}\int_{e_t}^{f_t}\mathbf{F}_t(u,v,w)N_{i,l}(u)N_{j,m}(v)N_{k,n}(w)du\,dv\,dw + \sum_{t=1}^{s_1}\mathbf{P}_tN_{i,l}(u_t^*)N_{j,m}(v_t^*)N_{k,n}(w_t^*)\right)_{i,j,k=1}^{p,q,r}.\end{aligned} \tag{5.24}$$

Using the simplified linear elastic model, 5.22, as the energy functional and using the volume minimisation results, 3.13, 3.14, 3.15, 3.16, we obtain the matrix3D equation:

$$\alpha\Big(\mathbf{M}_2^u\otimes_i\mathbf{d}\otimes_j\mathbf{M}_0^v\otimes_k\mathbf{M}_0^w + \mathbf{M}_0^u\otimes_i\mathbf{d}\otimes_j\mathbf{M}_2^v\otimes_k\mathbf{M}_0^w + \mathbf{M}_0^u\otimes_i\mathbf{d}\otimes_j\mathbf{M}_0^v\otimes_k\mathbf{M}_2^w\Big) +$$

$$\beta\Bigg(\mathbf{M}_1^{0u}\otimes_i\mathbf{d}\otimes_j\mathbf{M}_1^{0v}\otimes_k\mathbf{M}_0^{w}+(\mathbf{M}_1^{0u})^T\otimes_i\mathbf{d}\otimes_j(\mathbf{M}_1^{0v})^T\otimes_k\mathbf{M}_0^{w}+$$
$$\mathbf{M}_1^{0u}\otimes_i\mathbf{d}\otimes_j\mathbf{M}_0^{v}\otimes_k\mathbf{M}_1^{0w}+(\mathbf{M}_1^{0u})^T\otimes_i\mathbf{d}\otimes_j\mathbf{M}_0^{v}\otimes_k(\mathbf{M}_1^{0w})^T+$$
$$\mathbf{M}_0^{u}\otimes_i\mathbf{d}\otimes_j\mathbf{M}_1^{0v}\otimes_k\mathbf{M}_1^{0w}+\mathbf{M}_0^{u}\otimes_i\mathbf{d}\otimes_j(\mathbf{M}_1^{0v})^T\otimes_k(\mathbf{M}_1^{0w})^T\Bigg)=$$

$$\left(\sum_{t=1}^{r_1}\int_{a_t}^{b_t}\int_{c_t}^{d_t}\int_{e_t}^{f_t}\mathbf{F}_t(u,v,w)N_{i,l}(u)N_{j,m}(v)N_{k,n}(w)du\,dv\,dw+\sum_{t=1}^{s_1}\mathbf{P}_tN_{i,l}(u_t^*)N_{j,m}(v_t^*)N_{k,n}(w_t^*)\right)_{i,j,k=1}^{p,q,r}. \tag{5.25}$$

Boundary conditions

The boundary conditions are simplified when using the Ritz method, as in the curve/surface case the natural boundary conditions are verified automatically. To fit the geometric boundary constraints we use the boundary surface and derivative property of B-spline volumes, figure 2.14. For boundary surface interpolation we create the constraint equations for a surface by setting up the system as follows: assume we wish to fit the function

$$\sum_{i=1}^{p}\sum_{j=1}^{q}b_{ij}N_{i,l}(u)N_{j,m}(v)$$

along the boundary $w=0$, for example. The set of $p*q$ constraint equations is then given by

$$vec\Big(\mathbf{N}_w(0)\otimes\mathbf{f}_j\otimes\mathbf{e}_i\Big)\cdot vec(\mathbf{d})=b_{ij},\quad i,j=1,\ldots,p,q,$$

where

$$\mathbf{e}_i=(0,\ldots,0,1,0,\ldots,0)^T\ \text{ the } i\text{th unit vector of } R^p,\ \ i=1,\ldots,p.$$

$$\mathbf{f}_j=(0,\ldots,0,1,0,\ldots,0)^T\ \text{ the } j\text{th unit vector of } R^q,\ \ j=1,\ldots,q.$$

For boundary derivative interpolation along $w=0$ to a function given by

$$\sum_{i=1}^{p}\sum_{j=1}^{q}b_{ij}^*N_{i,l}(u)N_{j,m}(v),$$

the set of $p * q$ equations is

$$vec\Big(\big(\mathbf{N}_w^1(0)\mathbf{D}_w^1\big) \otimes \mathbf{f}_j \otimes \mathbf{e}_i\Big) \cdot vec(\mathbf{d}) = b_{ij}^*, \qquad i,j = 1,\ldots,p,q,$$

where $\mathbf{N}_w^1(0)$ represents the vector basis function set formed from the derivative knot set in w evaluated at 0.

As in the surface case, particular care has to be taken when setting up these constraint equations so as not to introduce redundancy into the system. For example, fitting a boundary surface function along $w = 0$ reduces the number of constraint equations when fitting a derivative function along $v = 0$ because of the boundary/derivative control point overlap at the intersecting edge. When all constraints have been inserted into the constraint matrix the reduced transformation technique described in section 3.6 is then used to furnish an admissible solution.

Algorithm 5.3 computes a B-spline solution to the solid deformation problem based on the total energy form 5.23 and the minimisation 5.24, and algorithm 5.4 a B-spline solution based on the total energy expression with internal energy given by 5.22 and minimisation equation 5.25. They will reproduce exact solutions where exact polynomial solutions exist.

Algorithm 5.3: Computes a B-spline solution to elastic solid deformation

given orders l,m,n, *dimensions* p,q,r, knot sets in (u,v,w)
internal energy functional given by 5.18
solution $\mathbf{x}(u,v,w) = \sum_{i=1}^{p}\sum_{j=1}^{q}\sum_{k=1}^{r} d_{ijk}N_{i,l}(u)N_{j,m}(v)N_{k,n}(v)$

1. *create the basis function set* $(\mathtt{N_{i,l}(u)N_{j,m}(v)N_{k,n}(w)})_{\mathtt{i,j,k=1}}^{\mathtt{p,q,r}}$
2. *create the minimisation matrices from the basis function set for the* $\mathtt{u,v,w}$ *0,1,2 derivatives*
   $\mathtt{M_0^u, M_0^v, M_0^w, M_1^u, M_1^v, M_1^w, M_2^u, M_2^v, M_2^w}$
3. *form the following Kronecker products from these matrices*
   $\mathtt{vm1} = \mathtt{M_0^w} \otimes (\mathtt{M_0^v} \otimes \mathtt{M_2^u}), \quad \mathtt{vm2} = \mathtt{M_0^w} \otimes (\mathtt{M_2^v} \otimes \mathtt{M_0^u})$
   $\mathtt{vm3} = \mathtt{M_2^w} \otimes (\mathtt{M_0^v} \otimes \mathtt{M_0^u}), \quad \mathtt{vm4} = \mathtt{M_0^w} \otimes (\mathtt{M_1^v} \otimes \mathtt{M_1^u})$
   $\mathtt{vm5} = \mathtt{M_1^w} \otimes (\mathtt{M_1^v} \otimes \mathtt{M_0^u}), \quad \mathtt{vm6} = \mathtt{M_1^w} \otimes (\mathtt{M_0^v} \otimes \mathtt{M_1^u})$
4. *form the minimisation matrix,* `mat=vm1+vm2+vm3+2(vm4+vm5+vm6)`
5. *create a* `loads` *matrix3D size* `(p,q,r)` *to store point and distributed forces and natural boundary conditions all of which appear on rhs of final linear system*

Algorithm 5.3 cont: Computes a B-spline solution to elastic solid deformation

6. for each point force $\mathtt{P_t}$ *at given value* $(\mathtt{u_t^*}, \mathtt{v_t^*}, \mathtt{w_t^*})$

6.1 calculate value of volume basis function set, $\mathtt{b}$ *at* $(\mathtt{u_t^*}, \mathtt{v_t^*}, \mathtt{w_t^*})$

$$\mathtt{b}(\mathtt{u_t^*}, \mathtt{v_t^*}, \mathtt{w_t^*}) = \left(\mathtt{N_{i,l}}(\mathtt{u_t^*})\mathtt{N_{j,m}}(\mathtt{v_t^*})\mathtt{N_{k,n}}(\mathtt{w_t^*})\right)_{\mathtt{i,j,k=1}}^{\mathtt{pqr}}$$

6.2 then $\mathtt{loads+=P_t} * \mathtt{b}(\mathtt{u_t^*}, \mathtt{v_t^*}, \mathtt{w_t^*})$

7. for each distributed force $\mathtt{F_t(u,v,w)}$ *over given* $\mathtt{u,v,w}$ *range* $[\mathtt{a_t}, \mathtt{b_t}] \times [\mathtt{c_t}, \mathtt{d_t}] \times [\mathtt{e_t}, \mathtt{f_t}]$

7.1 calculate integral of volume basis function set $\mathtt{b}$ *over* $(\mathtt{a_t}, \mathtt{b_t}, \mathtt{c_t}, \mathtt{d_t}, \mathtt{e_t}, \mathtt{f_t})$, *alg 3.7*

$$\mathtt{Integral}(\mathtt{a_t}, \mathtt{b_t}, \mathtt{c_t}, \mathtt{d_t}, \mathtt{e_t}, \mathtt{f_t}) = \left(\int_{\mathtt{a_t}}^{\mathtt{b_t}} \int_{\mathtt{c_t}}^{\mathtt{d_t}} \int_{\mathtt{e_t}}^{\mathtt{f_t}} \mathtt{F_t}(\mathtt{u},\mathtt{v},\mathtt{w})\mathtt{N_{i,l}}(\mathtt{u})\mathtt{N_{j,m}}(\mathtt{v})\mathtt{N_{k,n}}(\mathtt{w})\,\mathtt{du}\,\mathtt{dv}\,\mathtt{dw}\right)_{\mathtt{i,j,k=1}}^{\mathtt{pqr}}$$

7.2 then $\mathtt{loads+=Integral}(\mathtt{a_t}, \mathtt{b_t}, \mathtt{c_t}, \mathtt{d_t}, \mathtt{e_t}, \mathtt{f_t})$

8. determine the number of geometric boundary conditions, `num_bound`

9. create a matrix `(g_bound)` *size* `num_bound*pqr` *to store the geometric boundary conditions*

10. for the boundary face $\mathtt{w} = 0$

10.1 if there is a boundary face interpolation

10.1.1 convert boundary data to B-spline surface form

10.1.2 for each control point in `u`

10.1.2.1 for each control point in `v`

10.1.2.1.1 create the constraint equation and insert into matrix `g_bound`

10.2 if there is boundary face derivative interpolation

10.2.1 convert boundary data to B-spline surface form

10.2.2 for each control point in `u`

10.2.2.1 for each control point in `v`

10.2.2.1.1 create the constraint equation and insert into matrix `g_bound`

11. repeat step 10 for the remaining five faces

12. add in any internal point interpolation conditions (supports) to the `g_bound` *matrix*

13. Create six matrices `(n_bound1-6)` *to store the natural boundary conditions*

14. for the boundary face `w=0`

14.1 if there is a natural boundary condition

14.1.1 convert the interpolation data into B-spline surface form, `surf`

14.1.2 calculate the `uv` *surface integral using the 2D product rule and* `uv` *surface limits of volume, alg 3.6*

14.1.3 then `n_bound1 += IntegralProd(surf,umin,umax,vmin,vmax)`

15. repeat 14 for the remaining 5 faces

16. insert the 6 matrices, `n_bound1-6,` *into the boundary faces of a empty matrix3D*

Algorithm 5.3 cont: Computes a B-spline solution to elastic solid deformation

17. subtract the matrix3D from 13 from the matrix of loads from 4
18. create a Kronecker vector from the resulting matrix3D and multiply by -1
(this represents the rhs vector for the linear system)
19. find a set of elimination indices from the geometric boundary constraints matrix `g_bound`
20. eliminate corresponding rows of the minimisation matrix, `mat,` *and rhs vector, 18*
21. solve the resulting reduced system of equations
22. reconstitute the complete solution using the elimination information
23. build the B-spline volume from the control points found

Figure 5.32: Algorithm 5.3: B-spline solution to elastic solid deformation

5.7 Example: The 3D Laplace equation for heat flow through a cube

To test algorithm 5.3/5.4 we take a slightly different problem, that of heat transfer through a cube. We examine the B-spline functional obtained by solving Laplace's equation

$$\nabla^2 x = \frac{\partial^2 x}{\partial u^2} + \frac{\partial^2 x}{\partial v^2} + \frac{\partial^2 x}{\partial w^2} = 0,$$

for the steady state temperature distribution on a parallelepiped (figure 5.34), for which the analytical solution (non-polynomial) is known. The corresponding variational functional is given by (see 3.3):

$$I(x(u,v,w)) = \frac{1}{2}\int_V \Big[\Big(\frac{\partial x}{\partial u}\Big)^2 + \Big(\frac{\partial x}{\partial v}\Big)^2 + \Big(\frac{\partial x}{\partial w}\Big)^2\Big]\,du\,dv\,dw.$$

We assume the volume is a homogeneous parallelepiped with a thermal conductivity of 1.0. The temperature at the surface $w = L_w$ is taken to be 1. On the other five sides the temperature is taken to be zero. The domain V of the parallelepiped is limited by

$$0 \le u \le L_u, \quad 0 \le v \le L_v, \quad 0 \le w \le L_w,$$

Algorithm 5.4: Computes a B-spline solution to elastic solid deformation
internal energy functional given by 5.22

1. *create the basis function set* $(\mathtt{N_{i,l}(u)N_{j,m}(v)N_{k,n}(w)})_{\mathtt{i,j,k=1}}^{\mathtt{p,q,r}}$
2. *create the minimisation matrices from the basis function set for the 0,1,2* $\mathtt{u,v,w}$ *derivatives*
   $\mathtt{M_0^u, M_0^v, M_0^w, M_1^u, M_1^v, M_1^w, M_2^u, M_2^v, M_2^w}$
3. *create the minimisation matrices from the basis function set based on the (1,1,0), (1,0,1), (0,1,1) derivatives* $\mathtt{M_1^{0u}, M_1^{0v}, M_1^{0w}}$
4. *form the following Kronecker products from these matrices*
   $\mathtt{vm1} = \mathtt{M_0^w} \otimes (\mathtt{M_0^v} \otimes \mathtt{M_2^u})$, $\mathtt{vm2} = \mathtt{M_0^w} \otimes (\mathtt{M_2^v} \otimes \mathtt{M_0^u})$
   $\mathtt{vm3} = \mathtt{M_2^w} \otimes (\mathtt{M_0^v} \otimes \mathtt{M_0^u})$, $\mathtt{vm4} = \mathtt{M_0^w} \otimes (\mathtt{M_1^{0v}} \otimes \mathtt{M_1^{0u}})$
   $\mathtt{vm5} = \mathtt{M_0^w} \otimes ((\mathtt{M_1^{0v}})^\mathtt{T} \otimes (\mathtt{M_1^{0u}})^\mathtt{T})$, $\mathtt{vm6} = \mathtt{M_1^{0w}} \otimes (\mathtt{M_0^v} \otimes \mathtt{M_1^{0u}})$
   $\mathtt{vm7} = (\mathtt{M_1^{0w}})^\mathtt{T} \otimes \mathtt{M_0^v} \otimes (\mathtt{M_1^{0u}})^\mathtt{T})$, $\mathtt{vm8} = \mathtt{M_1^{0w}} \otimes (\mathtt{M_1^{0v}} \otimes \mathtt{M_0^u})$
   $\mathtt{vm9} = (\mathtt{M_1^{0w}})^\mathtt{T} \otimes ((\mathtt{M_1^{0v}})^\mathtt{T} \otimes \mathtt{M_0^u})$
4. *form the minimisation matrix,* $\mathtt{mat=}\ \alpha(\mathtt{vm1}+\mathtt{vm2}+\mathtt{vm3}) + \beta(\mathtt{vm4}+\mathtt{vm5}+\mathtt{vm6}+\mathtt{vm7}+\mathtt{vm8}+\mathtt{vm9})$
5. *to 23. as in algorithm 5.3*

Figure 5.33: Algorithm 5.4: B-spline solution to elastic solid deformation, version 2

and for the steady state solution the Laplacian operator is zero:

$$\nabla^2 x = 0 \quad \text{inside } V$$

For the boundary conditions we set the Dirichlet conditions $T = 0$ on all boundaries except $w = L_w$ and then examine the cases for two different boundary conditions on $w = L_w$, figure 5.35 (shown for $L_u = L_v = L_w = 1$):

1. $T(u, v, L_w) = 1, \quad 0 < u < L_u, \quad 0 < v < L_v$
2. $\left.\frac{\partial T}{\partial w}\right|_{u,v,L_w} = 1, \quad 0 < u < L_u, \quad 0 < v < L_v$

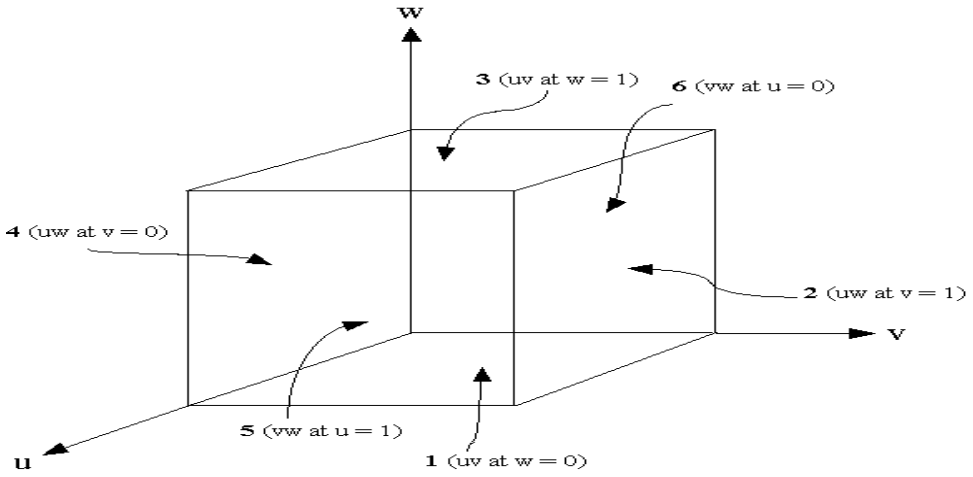


Figure 5.34: Volume model showing isoparametric faces

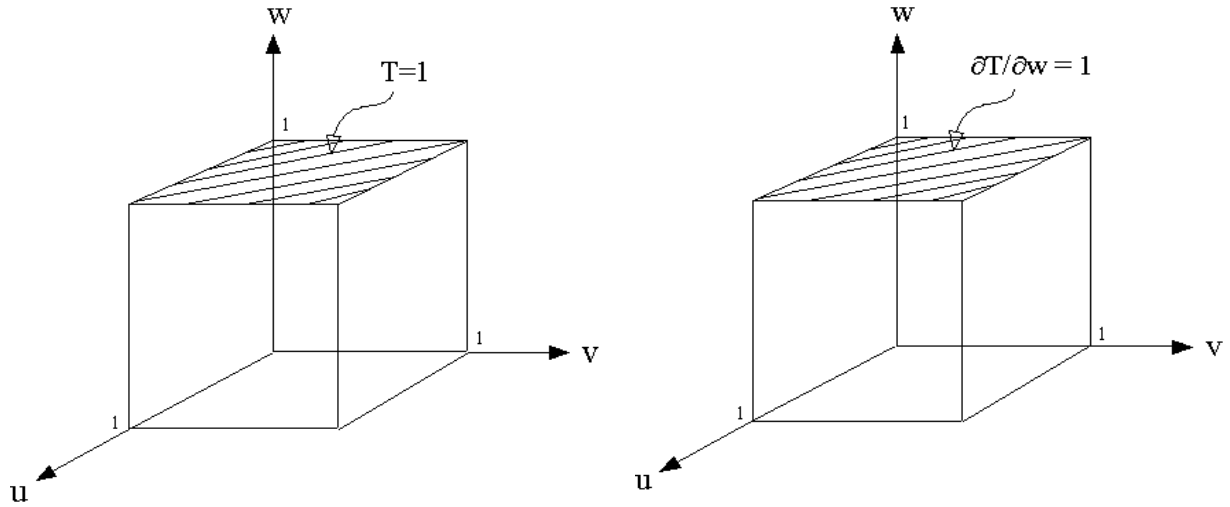


Figure 5.35: Volume element with boundary conditions for 3D Poisson equation for heat flow in a cube, temperature = 0 on all unmarked faces

5.7.1 Analytical solution for case 1

The analytical solution for boundary condition 1 is given by

$$T(u,v,w) = \frac{16}{\pi^2}\sum_{m=0}^{\infty}\sum_{n=0}^{\infty}\frac{sin\big(\frac{(2m+1)\pi u}{L_u}\big)}{2m+1}\frac{sin\big(\frac{(2n+1)\pi v}{L_v}\big)}{2n+1}\frac{sinh\big[\pi w\sqrt{\frac{(2m+1)^2}{L_u^2}+\frac{(2n+1)^2}{L_v^2}}\big]}{sinh\big[\pi L_w\sqrt{\frac{(2m+1)^2}{L_u^2}+\frac{(2n+1)^2}{L_v^2}}\big]}.$$

From this the flow through the bottom side $w = 0$ can be calculated to be:

$$T_0^{exact} = \int_0^{L_u} du \int_0^{L_v} dv \frac{\partial T}{\partial w}\Big|_{(u,v,0)}$$

$$= \frac{64}{\pi^3}L_uL_v\sum_{m=0}^{\infty}\sum_{n=0}^{\infty}\frac{1}{(2m+1)^2}\frac{1}{(2n+1)^2}\frac{\sqrt{\frac{(2m+1)^2}{L_u^2}+\frac{(2n+1)^2}{L_v^2}}}{sinh\big(\pi L_w\sqrt{\frac{(2m+1)^2}{L_u^2}+\frac{(2n+1)^2}{L_v^2}}\big)}.$$

We use algorithm 5.3 adapted to functional volumes to solve for a B-spline solution to the heat transfer problem using varying orders and segment numbers. In particular we calculate solutions for orders $l, m, n = 3, 4, 5$ and for segment numbers $(4, 6, 8, 10, 12)$ in (u, v, w). Figures 5.36 to 5.38 show the percentage absolute relative error expressed as

$$\epsilon = \Big|\frac{Q - Q^{exact}}{Q^{exact}}\Big|$$

where Q represents either the heat flow through the bottom of the cube or the steady state temperature distribution for the varying orders and segment numbers and Q_{exact} the corresponding theoretical value. Figure 5.38 shows log-log plots of the relative errors against segment number for the heat flow across the bottom side and the solution itself. With the small range of orders and segment numbers tested the individual results show some variation but the trends are apparent.

5.7.2 Analytical solution for case 2

For the second boundary condition the analytic solution is given by

$$T(u,v,w) = \frac{16}{\pi^3}\sum_{m=0}^{\infty}\sum_{n=0}^{\infty}\frac{sin\big(\frac{(2m+1)\pi u}{L_u}\big)}{2m+1}\frac{sin\big(\frac{(2n+1)\pi v}{L_v}\big)}{2n+1}\frac{sinh\big[\pi w\sqrt{\frac{(2m+1)^2}{L_u^2}+\frac{(2n+1)^2}{L_v^2}}\big]}{\sqrt{\frac{(2m+1)^2}{L_u^2}+\frac{(2n+1)^2}{L_v^2}}cosh\big[\pi L_w\sqrt{\frac{(2m+1)^2}{L_u^2}+\frac{(2n+1)^2}{L_v^2}}\big]},$$

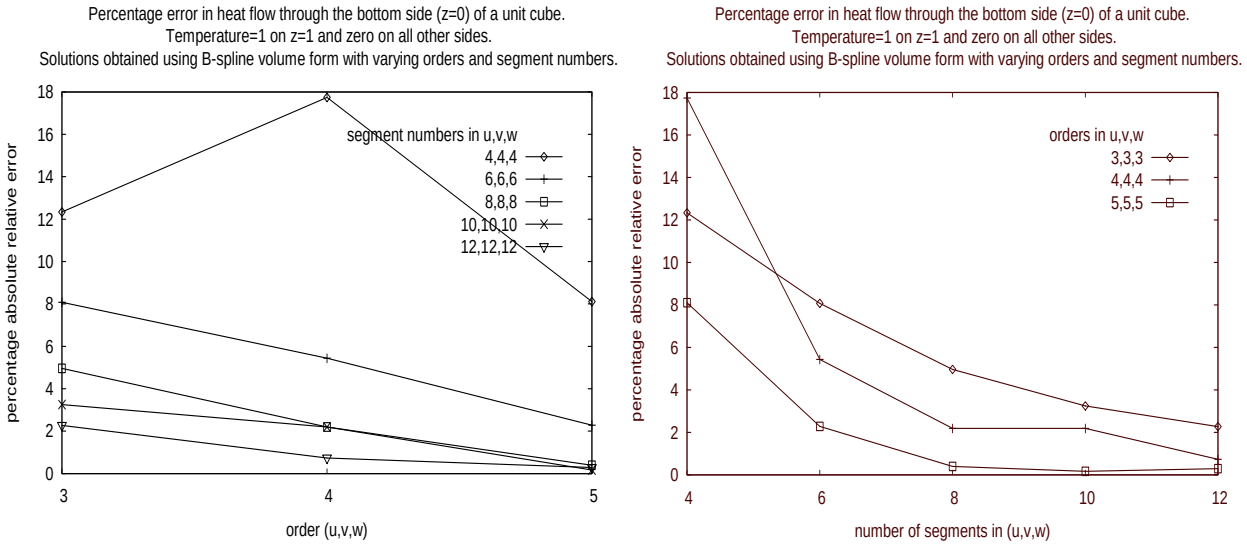


Figure 5.36: Percentage error in heat flow through bottom side of cube

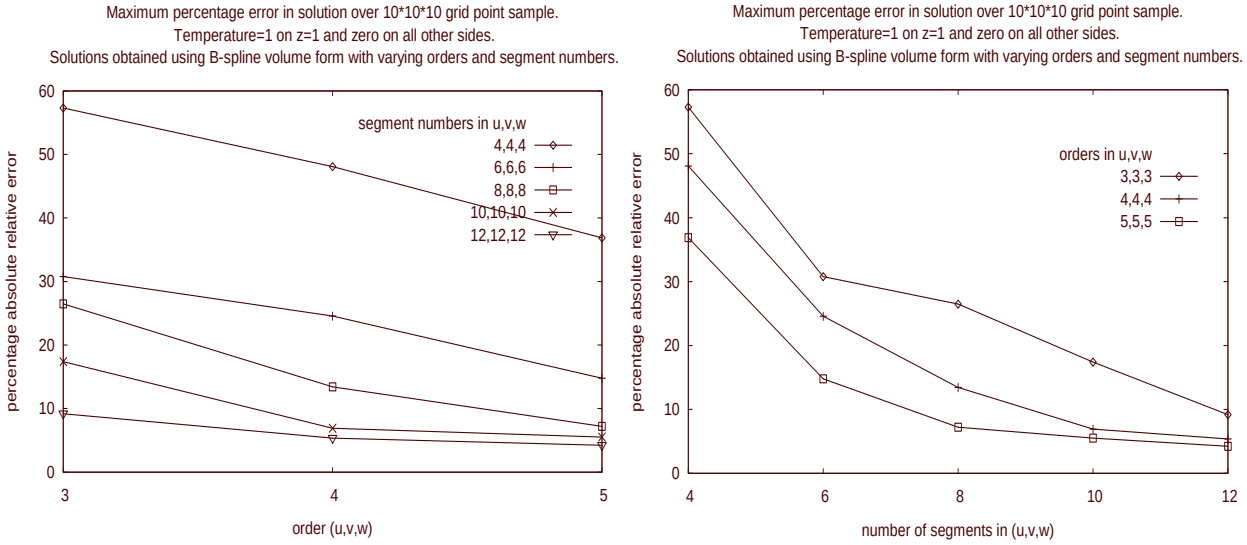


Figure 5.37: Maximum percentage error in solution for varying segment numbers

from which the heat flow through the bottom side $w = 0$ can be computed to be:

$$T_0^{exact} = \int_0^{L_u} du \int_0^{L_v} dv \frac{\partial T}{\partial w}\Big|_{(u,v,0)} = \frac{64}{\pi^4} L_u L_v \sum_{m=0}^{\infty} \sum_{n=0}^{\infty} \frac{1}{(2m+1)^2} \frac{1}{(2n+1)^2} \frac{1}{cosh\left[\pi L_w \sqrt{\frac{(2m+1)^2}{L_u^2} + \frac{(2n+1)^2}{L_v^2}}\right]}.$$

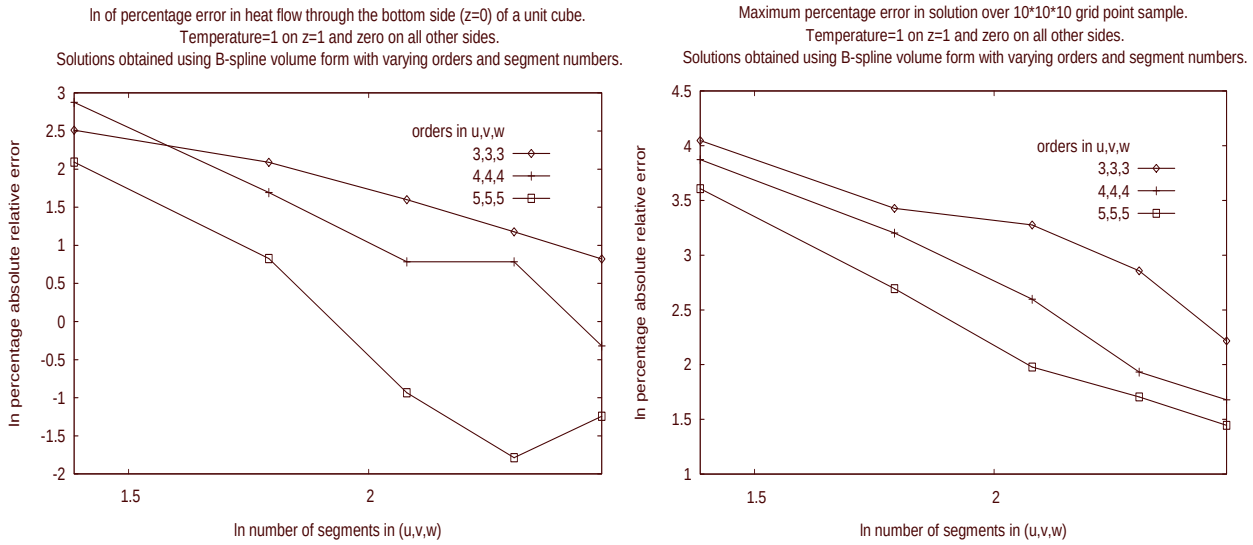


Figure 5.38: Log-log plot, maximum percentage error in heat flow and solution for varying segment numbers

Figures 5.39 and 5.40 show percentage absolute relative error graphs for the steady state temperature distribution and the heat flow through the bottom of the cube for the same set of orders and segment numbers, and figure 5.41 shows the corresponding log-log plots.

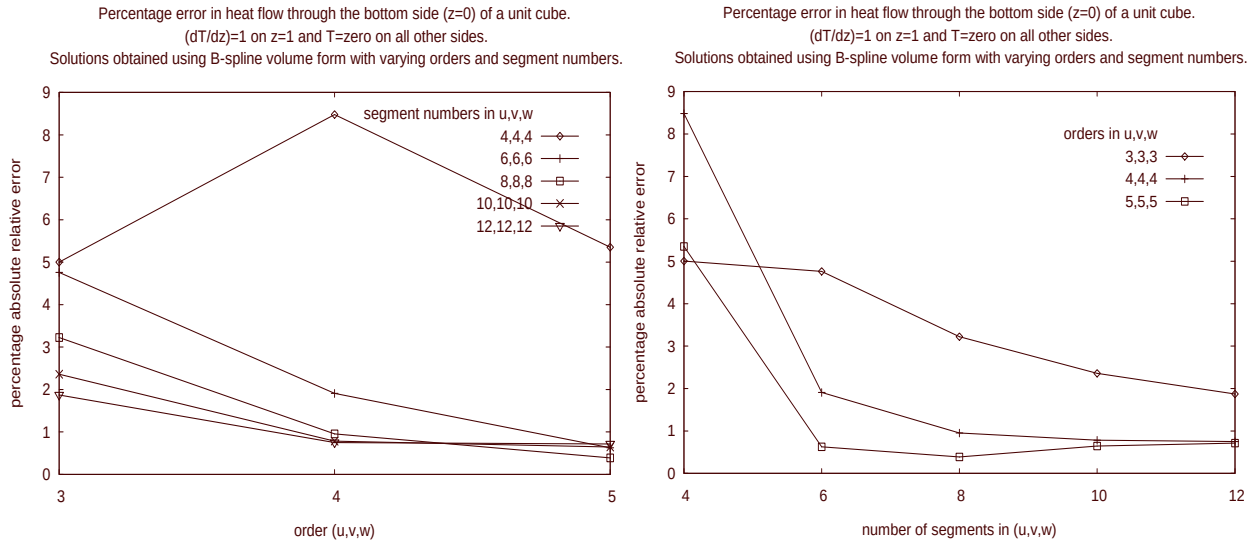


Figure 5.39: Percentage error in heat flow through bottom side of cube

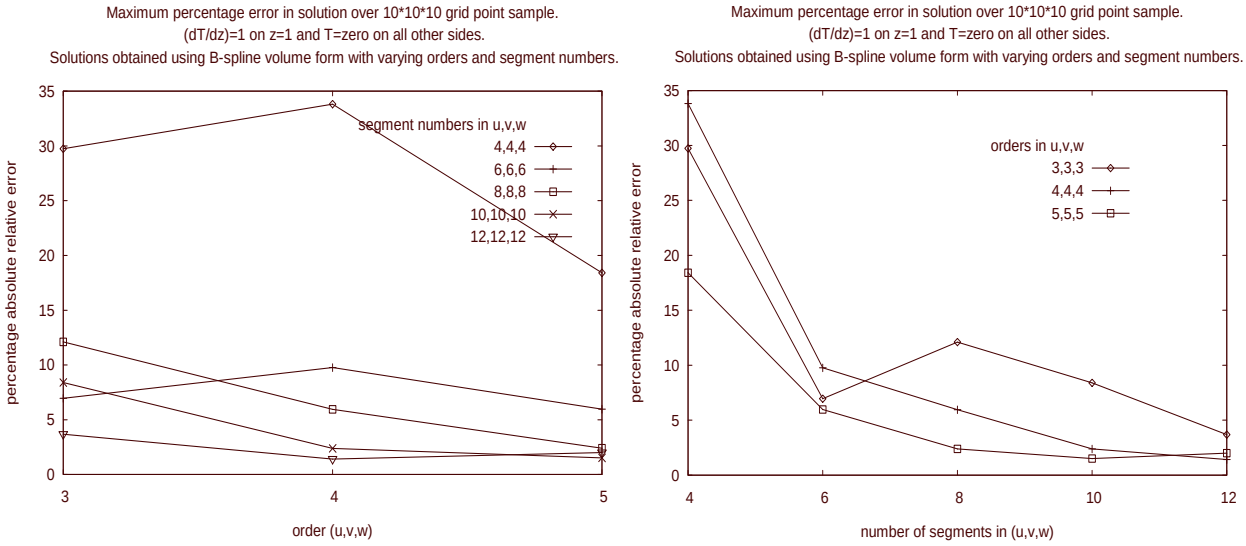


Figure 5.40: Maximum percentage error in solution over a sampled grid

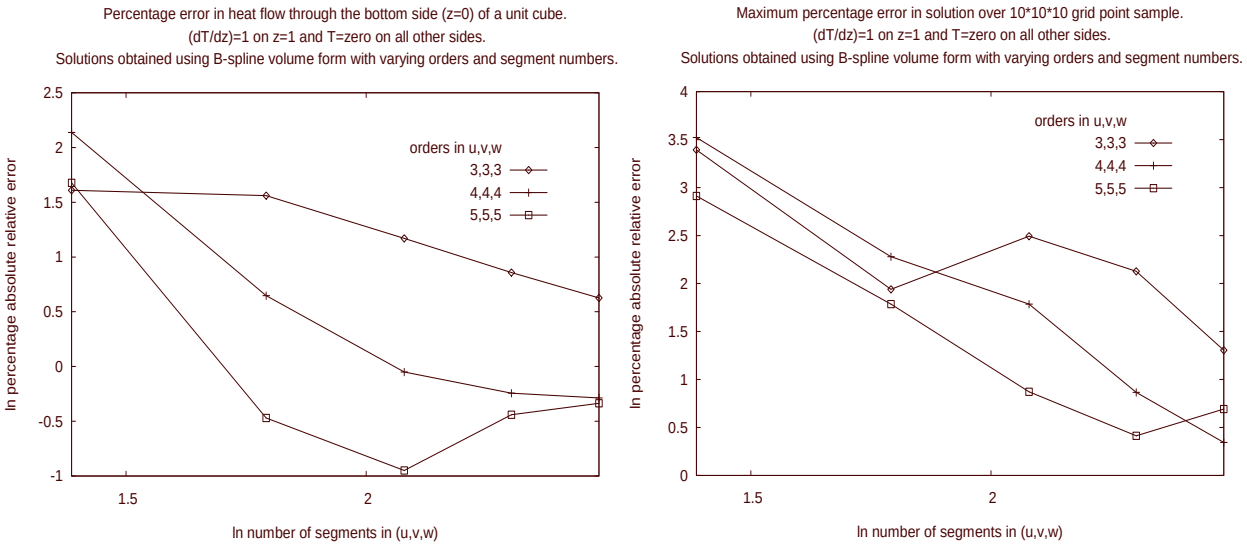


Figure 5.41: Log-log plot, maximum percentage error in heat flow and solution for varying segment numbers

5.8 Summary

By using the variational form of the governing differential equation for certain types of finite element problem and minimising the resulting energy based functional using the formulae from chapter 3, we have presented algorithms for computing a B-spline solution for one, two and three dimensional cases which reproduce exact polynomial solutions where they exist. A number of examples have been tested ranging from the static behaviour of beams under small deflection to the elastic bending of plates under various boundary conditions and the flow of heat through a solid. Comparisons have been made with known analytical solutions to these problems and, within the limits of the results generated, convergence behaviour has been compared and verified with theoretical predictions. Assuming a rectangular topology, the algorithm presented for the case of the deflection of a beam under varying load and boundary conditions, catered for using the reduced transformation technique, has been generalised to cope with two and three dimensional problems. Finally, the computation of the derivative B-spline curve/surface/volume form as an entity of the same type has enabled us to represent bending moments and reactive/shearing forces and other derived properties in closed form.

Chapter 6

Conclusions

The principal theme of this work has been B-spline based functional minimisation and the derivation of a unified description of the equations for curves, surfaces and volumes. To support this the report has focused in four main areas;

1. the derivation of the derivative matrix for a B-spline curve, expressing derivative control points in terms of the original, and its generalisation to the surface and volume forms

2. using 1, derivation of the minimisation equations for B-spline curve, surface and volume functionals based on squares and products of derivatives and an algorithm for the exact computation of the minimisation matrices that arise

3. applications of 2 to geometric smoothing and

4. to solving finite element type problems using the principle of minimum energy.

Concerning 3, we have presented some new methods for pre- and post-construction smoothing of B-splines which are simple to implement, generalise from curves to surfaces, and perform well as measured by standard smoothing criteria. For the surface case we have investigated two techniques for implementation, one based on a standard LU factorisation and the other on an efficient solution of Sylvester's equation. We have generalised the curve/surface algorithms to the volume case with new smoothing algorithms and demonstrated the capability they have to remove unwanted curvature variations as measured by appropriate volume smoothing criteria.

Concerning 4, we have used the minimisation equations to derive algorithms for solving finite element type problems expressed in terms of energy based functionals by applying the minimum energy criteria and variational principles to find B-spline solutions. We have considered problems

in one, two and three dimensions and the algorithms developed have been verified for accuracy by testing them with a number of cases for which analytic solutions are known. For the 2D case we have taken a number of examples from Timoshenko's book and reproduced B-spline solutions. The generalisation to the volume case represents a new approach for solving relatively small problems in elastic solid deformation, heat conduction and others.

There are a number of issues and limitations arising from the work. Firstly, the nature of the development with tensor product functions has imposed the restriction of rectangular topologies. Although using the Rayleigh-Ritz method normally restricts the domain to relatively simple shapes this factor has more of an effect on the scope of the work for treating finite element problems than it does for the geometric smoothing application where tensor product functions are already heavily used. It is well known that the computational advantages of bringing a problem into rectangular form are great even if this can only be done locally. The finite element problems we have dealt with here have been strictly rectangular but by suitably breaking the domain down, more complicated shapes (for example L shapes) could be dealt with.

Secondly, we have attempted to maintain from the outset the principle that we use exact methods. There are obviously pros and cons with this approach. The exact approach is theoretically interesting and will reproduce polynomial solutions where they exist. However, although errors are kept to a minimum this is often at the expense of execution speed. With the computation of the minimisation matrices in algorithms 3.2 and 3.3, the bottleneck is the product calculation and of course it grows considerably as we move up from curves to surfaces and volumes. How significant the factor of execution speed is depends partly on whether computation of the relevant (i.e. the time consuming) terms can be performed prior to the running of the critical user code. For example, in geometric smoothing we can in principle pre-compute the minimisation matrices since they are fixed for a given knot set. This can provide the user with real-time manipulation of the smoothing parameter, behaving much like a tuning knob on a receiver. It is also possible that minimisation matrices for a number of 'typical' (often used) knot sets might be stored in memory and called up when required. For the finite element algorithms we are often in the position where we may want to refine the mesh and this of course requires a new computation of the matrices. However, it may still be possible to pre-compute them in certain circumstances. Another significant factor, again due to the use of the product algorithm, is the computation of the source/boundary term as a B-spline in the variational form of the problem. This factor occurs on the right hand side of the resulting system and it seems sensible to compute it numerically.

Another factor that needs to be mentioned is that of the so called 'curse of dimensionality'[1]. As we go up in dimension we run the risk of rapidly outstripping the computational and memory storage capabilities. This is of particular importance for the volume methods. As discussed in Chapter 5 the increase in popularity of volume modelling is due to the rapid increase of computing memory and power. However the computational burden for large problems still represents a serious challenge. In this thesis we have used the Kronecker product 'brute force' method to solve the systems of equations arising from the smoothing and the finite element techniques and this method suffers from the curse. Of course with tensor product algorithms (as described in Chapter 2) we gain considerably over the brute force approach to solving the system of equations. For example, a standard least squares algorithm operated in tensor product form has an operation count $O(p^3+p^2q+pq^2+q^3)$ whereas the brute force approach is $O((pq)^3)$ (where p and q are the dimensions of B-spline). The methods that we have implemented in geometric smoothing and finite elements do not have a tensor product structure (but are more general and flexible because of this fact!) and therefore cannot be brought into this form. There are two possible routes to improve the situation, i/ find more efficient 'standard' methods to solve the systems or ii/ accept the fact of the size of the system and investigate more advanced techniques for speed optimisation such as paralleisation of the algorithms. The Sylvester type solution to the surface smoothing problem implemented in Chapter 4 and the use of banded factorisations where applicable are examples of i/. However, not even going this far we can in principle improve the execution speed by using optimised tensor operator functions implemented in libraries such as *blitz++*, which is written in C++ and is template based.

[1] the size/complexity of a data set grows exponentially with its dimension

Appendix A

Numerical Results for Surface FEA Examples

The numerical results obtained by applying algorithm 5.2 to problems involving the bending of loaded plates with the following boundary conditions imposed are presented. In each case the first table gives the numerical results from [85] for comparison.

1. Uniformly loaded and simply supported rectangular plate.
2. Simply supported and centrally point loaded rectangular plate.
3. Simply supported and partially loaded square plate.
4. Uniformly loaded rectangular plate with two edges simply supported and two edges clamped.
5. Uniformly loaded rectangular plate with two opposite edges simply supported, the third edge free and the fourth clamped.
6. Uniformly loaded rectangular plate with three edges simply supported and the fourth edge free.
7. Uniformly loaded rectangular plate with all edges built in.

A.1 Uniformly loaded and simply supported rectangular plate

Coefficients $\alpha, \beta, \beta_1, \gamma, \gamma_1, \delta, \delta_1$ (section 5.5) for B-spline solution to uniformly loaded (load F_1) simply supported rectangular plate of length a, width b, figure A.1.

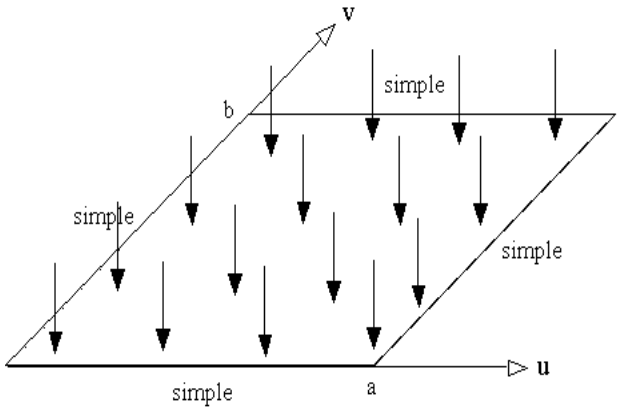


Figure A.1: Simply supported rectangular plate with uniform load

$\frac{b}{a}$	$x_{max} = \alpha \frac{F_1 a^4}{Eh^3}$	$(M_u)_{max} = \beta F_1 a^2$	$(M_v)_{max} = \beta_1 F_1 a^2$	$(Q_u)_{max} = \gamma F_1 a$	$(Q_v)_{max} = \gamma_1 F_1 a$	$(V_u)_{max} = \delta F_1 a$	$(V_v)_{max} = \delta_1 F_1 a$
	α	β	β_1	γ	γ_1	δ	δ_1
1.0000	0.0443	0.0479	0.0479	0.338	0.338	0.420	0.420
1.2000	0.0616	0.0626	0.0501	0.380	0.353	0.455	0.453
1.4000	0.0770	0.0753	0.506	0.411	0.361	0.478	0.471
1.6000	0.0906	0.0862	0.0493	0.435	0.365	0.491	0.485
1.8000	0.1017	0.0948	0.0479	0.452	0.368	0.499	0.491
2.0000	0.1106	0.1017	0.0464	0.465	0.370	0.503	0.496
3.0000	0.1336	0.1189	0.0404	0.493	0.372	0.505	0.498
4.0000	0.1400	0.1235	0.0384	0.498	0.372	0.502	0.500
5.0000	0.1416	0.1246	0.0375	0.500	0.372	0.501	0.500

Table A.1: Numerical results from [85], simply supported rectangular plate, uniform load

ordu = 4, ordv = 4, segu = 2, segv = 2, 3, 4, 5, 6, 4, 6, 8, 10							
$\frac{b}{a}$	$x_{max} = \alpha \frac{F_1 a^4}{Eh^3}$	$(M_u)_{max} = \beta F_1 a^2$	$(M_v)_{max} = \beta_1 F_1 a^2$	$(Q_u)_{max} = \gamma F_1 a$	$(Q_v)_{max} = \gamma_1 F_1 a$	$(V_u)_{max} = \delta F_1 a$	$(V_v)_{max} = \delta_1 F_1 a$
	α	β	β_1	γ	γ_1	δ	δ_1
1.0000	0.0450	0.0572	0.0572	0.2049	0.2049	0.3020	0.3020
1.2000	0.0624	0.0728	0.0463	0.2120	0.2311	0.2916	0.3472
1.4000	0.0777	0.0871	0.0382	0.2188	0.2561	0.2824	0.3855
1.6000	0.0911	0.1000	0.0337	0.2340	0.2695	0.2906	0.4072
1.8000	0.1021	0.1100	0.0273	0.2402	0.2786	0.2869	0.4216
2.0000	0.1107	0.1171	0.0187	0.2353	0.2380	0.2669	0.3856
3.0000	0.1337	0.1385	0.0069	0.2506	0.2388	0.2623	0.3906
4.0000	0.1400	0.1439	0.0015	0.2494	0.2388	0.2518	0.3908
5.0000	0.1417	0.1454	0.0005	0.2502	0.2388	0.2511	0.3908

ordu = 4, ordv = 4, segu = 4, segv = 4, 5, 6, 7, 8, 8, 12, 16, 20							
$\frac{b}{a}$	$x_{max} = \alpha \frac{F_1 a^4}{Eh^3}$	$(M_u)_{max} = \beta F_1 a^2$	$(M_v)_{max} = \beta_1 F_1 a^2$	$(Q_u)_{max} = \gamma F_1 a$	$(Q_v)_{max} = \gamma_1 F_1 a$	$(V_u)_{max} = \delta F_1 a$	$(V_v)_{max} = \delta_1 F_1 a$
	α	β	β_1	γ	γ_1	δ	δ_1
1.0000	0.0444	0.0486	0.0486	0.2335	0.2335	0.3158	0.3158
1.2000	0.0617	0.0640	0.0446	0.2705	0.2536	0.3464	0.3541
1.4000	0.0774	0.0776	0.0393	0.2995	0.2652	0.3666	0.3779
1.6000	0.0908	0.0890	0.0336	0.3207	0.2719	0.3778	0.3923
1.8000	0.1018	0.0980	0.0274	0.3341	0.2758	0.3804	0.4010
2.0000	0.1106	0.1053	0.0222	0.3450	0.2688	0.3824	0.3967
3.0000	0.1336	0.1236	0.0069	0.3689	0.2704	0.3805	0.4020
4.0000	0.1400	0.1285	0.0019	0.3738	0.2704	0.3770	0.4023
5.0000	0.1416	0.1298	0.0005	0.3747	0.2704	0.3756	0.4024

ordu = 4, ordv = 4, segu = 6, segv = 6, 7, 8, 9, 10, 12, 18, 24, 30							
$\frac{b}{a}$	$x_{max} = \alpha \frac{F_1 a^4}{Eh^3}$	$(M_u)_{max} = \beta F_1 a^2$	$(M_v)_{max} = \beta_1 F_1 a^2$	$(Q_u)_{max} = \gamma F_1 a$	$(Q_v)_{max} = \gamma_1 F_1 a$	$(V_u)_{max} = \delta F_1 a$	$(V_v)_{max} = \delta_1 F_1 a$
	α	β	β_1	γ	γ_1	δ	δ_1
1.0000	0.0444	0.0483	0.0483	0.2648	0.2648	0.3485	0.3485
1.2000	0.0617	0.0633	0.0446	0.3034	0.2790	0.3799	0.3803
1.4000	0.0774	0.0765	0.0393	0.3330	0.2865	0.3997	0.3995
1.6000	0.0907	0.0875	0.0335	0.3560	0.2902	0.4129	0.4107
1.8000	0.1017	0.0963	0.0275	0.3719	0.2919	0.4186	0.4171
2.0000	0.1106	0.1033	0.0224	0.3839	0.2983	0.4219	0.4263
3.0000	0.1336	0.1210	0.0070	0.4099	0.2998	0.4217	0.4314
4.0000	0.1400	0.1257	0.0019	0.4153	0.2999	0.4185	0.4317
5.0000	0.1416	0.1269	0.0005	0.4164	0.2999	0.4172	0.4317

ordu = 4, ordv = 4, segu = 8, segv = 8, 9, 10, 11, 12, 16, 24, 32, 40							
$\frac{b}{a}$	$x_{max} = \alpha \frac{F_1 a^4}{Eh^3}$	$(M_u)_{max} = \beta F_1 a^2$	$(M_v)_{max} = \beta_1 F_1 a^2$	$(Q_u)_{max} = \gamma F_1 a$	$(Q_v)_{max} = \gamma_1 F_1 a$	$(V_u)_{max} = \delta F_1 a$	$(V_v)_{max} = \delta_1 F_1 a$
	α	β	β_1	γ	γ_1	δ	δ_1
1.0000	0.0444	0.0481	0.0481	0.2806	0.2806	0.3636	0.3636
1.2000	0.0617	0.0631	0.0446	0.3209	0.2929	0.3972	0.3933
1.4000	0.0774	0.0761	0.0393	0.3516	0.2988	0.4185	0.4109
1.6000	0.0907	0.0869	0.0335	0.3749	0.3013	0.4318	0.4208
1.8000	0.1017	0.0956	0.0276	0.3912	0.3020	0.4380	0.4260
2.0000	0.1106	0.1026	0.0225	0.4038	0.3135	0.4419	0.4403
3.0000	0.1336	0.1201	0.0070	0.4305	0.3150	0.4424	0.4455
4.0000	0.1400	0.1247	0.0020	0.4361	0.3150	0.4393	0.4458
5.0000	0.1416	0.1259	0.0005	0.4372	0.3150	0.4381	0.4458

ordu = 4, ordv = 4, segu = 10, segv = 10, 11, 12, 13, 14, 20, 30, 40, 50							
$\frac{b}{a}$	$x_{max} = \alpha \frac{F_1 a^4}{Eh^3}$	$(M_u)_{max} = \beta F_1 a^2$	$(M_v)_{max} = \beta_1 F_1 a^2$	$(Q_u)_{max} = \gamma F_1 a$	$(Q_v)_{max} = \gamma_1 F_1 a$	$(V_u)_{max} = \delta F_1 a$	$(V_v)_{max} = \delta_1 F_1 a$
	α	β	β_1	γ	γ_1	δ	δ_1
1.0000	0.0444	0.0480	0.0480	0.2912	0.2912	0.3743	0.3743
1.2000	0.0617	0.0629	0.0447	0.3319	0.3029	0.4084	0.4033
1.4000	0.0774	0.0759	0.0393	0.3630	0.3083	0.4299	0.4202
1.6000	0.0907	0.0867	0.0335	0.3864	0.3102	0.4432	0.4296
1.8000	0.1017	0.0954	0.0276	0.4032	0.3104	0.4500	0.4343
2.0000	0.1106	0.1023	0.0226	0.4158	0.3238	0.4540	0.4505
3.0000	0.1336	0.1196	0.0071	0.4429	0.3253	0.4548	0.4556
4.0000	0.1400	0.1243	0.0020	0.4485	0.3254	0.4518	0.4559
5.0000	0.1416	0.1255	0.0005	0.4497	0.3254	0.4506	0.4559

ordu = 4, ordv = 4, segu = 12, segv = 12, 13, 14, 15, 16, 24, 36, 48, 60							
$\frac{b}{a}$	$x_{max} = \alpha \frac{F_1 a^4}{Eh^3}$	$(M_u)_{max} = \beta F_1 a^2$	$(M_v)_{max} = \beta_1 F_1 a^2$	$(Q_u)_{max} = \gamma F_1 a$	$(Q_v)_{max} = \gamma_1 F_1 a$	$(V_u)_{max} = \delta F_1 a$	$(V_v)_{max} = \delta_1 F_1 a$
	α	β	β_1	γ	γ_1	δ	δ_1
1.0000	0.0444	0.0480	0.0480	0.2984	0.2984	0.3814	0.3814
1.2000	0.0617	0.0628	0.0447	0.3395	0.3100	0.4159	0.4102
1.4000	0.0774	0.0758	0.0393	0.3707	0.3151	0.4377	0.4269
1.6000	0.0907	0.0865	0.0335	0.3942	0.3169	0.4511	0.4360
1.8000	0.1017	0.0952	0.0277	0.4112	0.3169	0.4580	0.4406
2.0000	0.1106	0.1021	0.0226	0.4239	0.3309	0.4622	0.4573
3.0000	0.1336	0.1194	0.0071	0.4512	0.3323	0.4631	0.4624
4.0000	0.1400	0.1240	0.0020	0.4568	0.3324	0.4602	0.4627
5.0000	0.1416	0.1252	0.0005	0.4580	0.3324	0.4589	0.4627

ordu = 5, ordv = 5, segu = 2, segv = 2, 3, 4, 5, 6, 4, 6, 8, 10							
$\frac{b}{a}$	$x_{max} = \alpha \frac{F_1 a^4}{Eh^3}$	$(M_u)_{max} = \beta F_1 a^2$	$(M_v)_{max} = \beta_1 F_1 a^2$	$(Q_u)_{max} = \gamma F_1 a$	$(Q_v)_{max} = \gamma_1 F_1 a$	$(V_u)_{max} = \delta F_1 a$	$(V_v)_{max} = \delta_1 F_1 a$
	α	β	β_1	γ	γ_1	δ	δ_1
1.0000	0.0448	0.0500	0.0500	0.2593	0.2593	0.3453	0.3453
1.2000	0.0618	0.0639	0.0441	0.3070	0.3048	0.3812	0.4079
1.4000	0.0775	0.0767	0.0395	0.3543	0.3250	0.4207	0.4393
1.6000	0.0909	0.0871	0.0333	0.3901	0.3365	0.4473	0.4581
1.8000	0.1018	0.0956	0.0277	0.4171	0.3437	0.4644	0.4698
2.0000	0.1106	0.1021	0.0224	0.4366	0.2940	0.4744	0.4225
3.0000	0.1337	0.1191	0.0072	0.4856	0.2968	0.4979	0.4288
4.0000	0.1400	0.1235	0.0019	0.4963	0.2971	0.4994	0.4293
5.0000	0.1417	0.1246	0.0005	0.4993	0.2972	0.5002	0.4294

ordu = 5, ordv = 5, segu = 4, segv = 4, 5, 6, 7, 8, 8, 12, 16, 20							
$\frac{b}{a}$	$x_{max} = \alpha \frac{F_1 a^4}{Eh^3}$	$(M_u)_{max} = \beta F_1 a^2$	$(M_v)_{max} = \beta_1 F_1 a^2$	$(Q_u)_{max} = \gamma F_1 a$	$(Q_v)_{max} = \gamma_1 F_1 a$	$(V_u)_{max} = \delta F_1 a$	$(V_v)_{max} = \delta_1 F_1 a$
	α	β	β_1	γ	γ_1	δ	δ_1
1.0000	0.0444	0.0481	0.0481	0.3157	0.3157	0.3984	0.3984
1.2000	0.0617	0.0628	0.0446	0.3626	0.3319	0.4393	0.4317
1.4000	0.0774	0.0757	0.0394	0.3979	0.3416	0.4651	0.4529
1.6000	0.0907	0.0863	0.0334	0.4240	0.3469	0.4806	0.4655
1.8000	0.1017	0.0949	0.0278	0.4439	0.3500	0.4909	0.4732
2.0000	0.1106	0.1018	0.0227	0.4585	0.3473	0.4968	0.4732
3.0000	0.1336	0.1189	0.0071	0.4910	0.3489	0.5029	0.4784
4.0000	0.1400	0.1235	0.0020	0.4980	0.3490	0.5014	0.4787
5.0000	0.1416	0.1246	0.0005	0.4996	0.3490	0.5004	0.4787

ordu = 5, ordv = 5, segu = 6, segv = 6, 7, 8, 9, 10, 12, 18, 24, 30							
$\frac{b}{a}$	$x_{max} = \alpha \frac{F_1 a^4}{Eh^3}$	$(M_u)_{max} = \beta F_1 a^2$	$(M_v)_{max} = \beta_1 F_1 a^2$	$(Q_u)_{max} = \gamma F_1 a$	$(Q_v)_{max} = \gamma_1 F_1 a$	$(V_u)_{max} = \delta F_1 a$	$(V_v)_{max} = \delta_1 F_1 a$
	α	β	β_1	γ	γ_1	δ	δ_1
1.0000	0.0444	0.0479	0.0479	0.3284	0.3284	0.4113	0.4113
1.2000	0.0617	0.0627	0.0447	0.3717	0.3429	0.4480	0.4430
1.4000	0.0774	0.0756	0.0394	0.4051	0.3509	0.4721	0.4625
1.6000	0.0907	0.0862	0.0334	0.4302	0.3551	0.4870	0.4741
1.8000	0.1017	0.0949	0.0278	0.4487	0.3573	0.4957	0.4808
2.0000	0.1106	0.1017	0.0226	0.4622	0.3604	0.5005	0.4866
3.0000	0.1336	0.1189	0.0071	0.4920	0.3619	0.5040	0.4917
4.0000	0.1400	0.1235	0.0020	0.4983	0.3619	0.5016	0.4920
5.0000	0.1416	0.1246	0.0005	0.4996	0.3619	0.5005	0.4920

ordu = 5, ordv = 5, segu = 8, segv = 8, 9, 10, 11, 12, 16, 24, 32, 40							
$\frac{b}{a}$	$x_{max} = \alpha \frac{F_1 a^4}{Eh^3}$	$(M_u)_{max} = \beta F_1 a^2$	$(M_v)_{max} = \beta_1 F_1 a^2$	$(Q_u)_{max} = \gamma F_1 a$	$(Q_v)_{max} = \gamma_1 F_1 a$	$(V_u)_{max} = \delta F_1 a$	$(V_v)_{max} = \delta_1 F_1 a$
	α	β	β_1	γ	γ_1	δ	δ_1
1.0000	0.0444	0.0479	0.0479	0.3323	0.3323	0.4151	0.4151
1.2000	0.0617	0.0627	0.0447	0.3753	0.3468	0.4517	0.4467
1.4000	0.0774	0.0756	0.0394	0.4079	0.3545	0.4749	0.4660
1.6000	0.0907	0.0862	0.0335	0.4321	0.3586	0.4888	0.4774
1.8000	0.1017	0.0949	0.0278	0.4502	0.3606	0.4972	0.4839
2.0000	0.1106	0.1017	0.0226	0.4635	0.3643	0.5018	0.4904
3.0000	0.1336	0.1189	0.0071	0.4923	0.3658	0.5043	0.4955
4.0000	0.1400	0.1235	0.0020	0.4984	0.3659	0.5017	0.4958
5.0000	0.1416	0.1246	0.0005	0.4997	0.3659	0.5005	0.4958

ordu = 5, ordv = 5, segu = 10, segv = 10, 11, 12, 13, 14, 20, 30, 40, 50							
$\frac{b}{a}$	$x_{max} = \alpha \frac{F_1 a^4}{Eh^3}$	$(M_u)_{max} = \beta F_1 a^2$	$(M_v)_{max} = \beta_1 F_1 a^2$	$(Q_u)_{max} = \gamma F_1 a$	$(Q_v)_{max} = \gamma_1 F_1 a$	$(V_u)_{max} = \delta F_1 a$	$(V_v)_{max} = \delta_1 F_1 a$
	α	β	β_1	γ	γ_1	δ	δ_1
1.0000	0.0444	0.0479	0.0479	0.3343	0.3343	0.4172	0.4172
1.2000	0.0617	0.0627	0.0447	0.3767	0.3489	0.4531	0.4489
1.4000	0.0774	0.0756	0.0394	0.4090	0.3567	0.4759	0.4682
1.6000	0.0907	0.0862	0.0335	0.4331	0.3608	0.4899	0.4796
1.8000	0.1017	0.0948	0.0278	0.4509	0.3628	0.4980	0.4861
2.0000	0.1106	0.1017	0.0226	0.4640	0.3664	0.5024	0.4925
3.0000	0.1336	0.1189	0.0071	0.4925	0.3678	0.5045	0.4975
4.0000	0.1400	0.1235	0.0020	0.4984	0.3679	0.5017	0.4978
5.0000	0.1416	0.1246	0.0005	0.4997	0.3679	0.5005	0.4979

ordu = 5, ordv = 5, segu = 12, segv = 12, 13, 14, 15, 16, 24, 36, 48, 60							
$\frac{b}{a}$	$x_{max} = \alpha \frac{F_1 a^4}{Eh^3}$	$(M_u)_{max} = \beta F_1 a^2$	$(M_v)_{max} = \beta_1 F_1 a^2$	$(Q_u)_{max} = \gamma F_1 a$	$(Q_v)_{max} = \gamma_1 F_1 a$	$(V_u)_{max} = \delta F_1 a$	$(V_v)_{max} = \delta_1 F_1 a$
	α	β	β_1	γ	γ_1	δ	δ_1
1.0000	0.0444	0.0479	0.0479	0.3353	0.3353	0.4181	0.4181
1.2000	0.0617	0.0627	0.0447	0.3776	0.3500	0.4540	0.4500
1.4000	0.0774	0.0756	0.0394	0.4097	0.3579	0.4766	0.4694
1.6000	0.0907	0.0862	0.0335	0.4336	0.3620	0.4903	0.4809
1.8000	0.1017	0.0948	0.0278	0.4513	0.3641	0.4983	0.4874
2.0000	0.1106	0.1017	0.0226	0.4643	0.3674	0.5027	0.4934
3.0000	0.1336	0.1189	0.0071	0.4925	0.3688	0.5045	0.4985
4.0000	0.1400	0.1235	0.0020	0.4984	0.3689	0.5018	0.4988
5.0000	0.1416	0.1246	0.0005	0.4997	0.3689	0.5005	0.4988

ordu = 6, ordv = 6, segu = 2, segv = 2, 3, 4, 5, 6, 4, 6, 8, 10							
$\frac{b}{a}$	$x_{max} = \alpha \frac{F_1 a^4}{Eh^3}$	$(M_u)_{max} = \beta F_1 a^2$	$(M_v)_{max} = \beta_1 F_1 a^2$	$(Q_u)_{max} = \gamma F_1 a$	$(Q_v)_{max} = \gamma_1 F_1 a$	$(V_u)_{max} = \delta F_1 a$	$(V_v)_{max} = \delta_1 F_1 a$
	α	β	β_1	γ	γ_1	δ	δ_1
1.0000	0.0443	0.0476	0.0476	0.3182	0.3182	0.3998	0.3998
1.2000	0.0617	0.0625	0.0449	0.3655	0.3451	0.4418	0.4439
1.4000	0.0774	0.0753	0.0393	0.4015	0.3546	0.4687	0.4650
1.6000	0.0907	0.0861	0.0334	0.4272	0.3618	0.4841	0.4796
1.8000	0.1017	0.0947	0.0278	0.4460	0.3642	0.4929	0.4865
2.0000	0.1106	0.1016	0.0226	0.4617	0.3497	0.5004	0.4744
3.0000	0.1336	0.1189	0.0071	0.4906	0.3518	0.5025	0.4801
4.0000	0.1400	0.1234	0.0020	0.4987	0.3520	0.5022	0.4805
5.0000	0.1416	0.1246	0.0005	0.4992	0.3521	0.5001	0.4806

ordu = 6, ordv = 6, segu = 4, segv = 4, 5, 6, 7, 8, 8, 12, 16, 20							
$\frac{b}{a}$	$x_{max} = \alpha \frac{F_1 a^4}{Eh^3}$	$(M_u)_{max} = \beta F_1 a^2$	$(M_v)_{max} = \beta_1 F_1 a^2$	$(Q_u)_{max} = \gamma F_1 a$	$(Q_v)_{max} = \gamma_1 F_1 a$	$(V_u)_{max} = \delta F_1 a$	$(V_v)_{max} = \delta_1 F_1 a$
	α	β	β_1	γ	γ_1	δ	δ_1
1.0000	0.0444	0.0478	0.0478	0.3376	0.3376	0.4205	0.4205
1.2000	0.0617	0.0626	0.0447	0.3798	0.3534	0.4563	0.4535
1.4000	0.0774	0.0755	0.0394	0.4110	0.3618	0.4779	0.4734
1.6000	0.0907	0.0862	0.0335	0.4348	0.3664	0.4915	0.4855
1.8000	0.1017	0.0948	0.0277	0.4525	0.3689	0.4996	0.4924
2.0000	0.1106	0.1017	0.0226	0.4654	0.3699	0.5038	0.4961
3.0000	0.1336	0.1189	0.0071	0.4928	0.3713	0.5049	0.5012
4.0000	0.1400	0.1235	0.0020	0.4985	0.3714	0.5019	0.5015
5.0000	0.1416	0.1246	0.0005	0.4997	0.3714	0.5006	0.5015

ordu = 6, ordv = 6, segu = 6, segv = 6, 7, 8, 9, 10, 12, 18, 24, 30							
$\frac{b}{a}$	$x_{max} = \alpha \frac{F_1 a^4}{Eh^3}$	$(M_u)_{max} = \beta F_1 a^2$	$(M_v)_{max} = \beta_1 F_1 a^2$	$(Q_u)_{max} = \gamma F_1 a$	$(Q_v)_{max} = \gamma_1 F_1 a$	$(V_u)_{max} = \delta F_1 a$	$(V_v)_{max} = \delta_1 F_1 a$
	α	β	β_1	γ	γ_1	δ	δ_1
1.0000	0.0444	0.0479	0.0479	0.3376	0.3376	0.4203	0.4203
1.2000	0.0617	0.0627	0.0447	0.3793	0.3529	0.4557	0.4528
1.4000	0.0774	0.0755	0.0394	0.4113	0.3614	0.4783	0.4728
1.6000	0.0907	0.0862	0.0335	0.4349	0.3659	0.4917	0.4847
1.8000	0.1017	0.0948	0.0278	0.4521	0.3684	0.4991	0.4917
2.0000	0.1106	0.1017	0.0226	0.4651	0.3697	0.5034	0.4957
3.0000	0.1336	0.1189	0.0071	0.4927	0.3711	0.5047	0.5007
4.0000	0.1400	0.1235	0.0020	0.4985	0.3712	0.5018	0.5010
5.0000	0.1416	0.1246	0.0005	0.4997	0.3712	0.5006	0.5010

ordu = 6, ordv = 6, segu = 8, segv = 8, 9, 10, 11, 12, 16, 24, 32, 40							
$\frac{b}{a}$	$x_{max} = \alpha \frac{F_1 a^4}{Eh^3}$	$(M_u)_{max} = \beta F_1 a^2$	$(M_v)_{max} = \beta_1 F_1 a^2$	$(Q_u)_{max} = \gamma F_1 a$	$(Q_v)_{max} = \gamma_1 F_1 a$	$(V_u)_{max} = \delta F_1 a$	$(V_v)_{max} = \delta_1 F_1 a$
	α	β	β_1	γ	γ_1	δ	δ_1
1.0000	0.0444	0.0479	0.0479	0.3378	0.3378	0.4206	0.4206
1.2000	0.0617	0.0627	0.0447	0.3795	0.3531	0.4560	0.4531
1.4000	0.0774	0.0755	0.0394	0.4110	0.3615	0.4780	0.4730
1.6000	0.0907	0.0862	0.0335	0.4347	0.3661	0.4914	0.4849
1.8000	0.1017	0.0948	0.0278	0.4523	0.3685	0.4993	0.4919
2.0000	0.1106	0.1017	0.0226	0.4651	0.3698	0.5034	0.4960
3.0000	0.1336	0.1189	0.0071	0.4927	0.3713	0.5047	0.5010
4.0000	0.1400	0.1235	0.0020	0.4985	0.3714	0.5018	0.5013
5.0000	0.1416	0.1246	0.0005	0.4997	0.3714	0.5006	0.5013

ordu = 6, ordv = 6, segu = 10, segv = 10, 11, 12, 13, 14, 20, 30, 40, 50							
$\frac{b}{a}$	x_{max} $= \alpha \frac{F_1 a^4}{Eh^3}$	$(M_u)_{max}$ $= \beta F_1 a^2$	$(M_v)_{max}$ $= \beta_1 F_1 a^2$	$(Q_u)_{max}$ $= \gamma F_1 a$	$(Q_v)_{max}$ $= \gamma_1 F_1 a$	$(V_u)_{max}$ $= \delta F_1 a$	$(V_v)_{max}$ $= \delta_1 F_1 a$
	α	β	β_1	γ	γ_1	δ	δ_1
1.0000	0.0444	0.0479	0.0479	0.3376	0.3376	0.4204	0.4204
1.2000	0.0617	0.0627	0.0447	0.3794	0.3530	0.4558	0.4529
1.4000	0.0774	0.0755	0.0394	0.4111	0.3614	0.4781	0.4729
1.6000	0.0907	0.0862	0.0335	0.4348	0.3660	0.4915	0.4848
1.8000	0.1017	0.0948	0.0278	0.4522	0.3685	0.4992	0.4918
2.0000	0.1106	0.1017	0.0226	0.4650	0.3697	0.5034	0.4958
3.0000	0.1336	0.1189	0.0071	0.4927	0.3712	0.5047	0.5008
4.0000	0.1400	0.1235	0.0020	0.4985	0.3712	0.5018	0.5011
5.0000	0.1416	0.1246	0.0005	0.4997	0.3712	0.5006	0.5011

ordu = 6, ordv = 6, segu = 12, segv = 12, 13, 14, 15, 16, 24, 36, 48, 60							
$\frac{b}{a}$	x_{max} $= \alpha \frac{F_1 a^4}{Eh^3}$	$(M_u)_{max}$ $= \beta F_1 a^2$	$(M_v)_{max}$ $= \beta_1 F_1 a^2$	$(Q_u)_{max}$ $= \gamma F_1 a$	$(Q_v)_{max}$ $= \gamma_1 F_1 a$	$(V_u)_{max}$ $= \delta F_1 a$	$(V_v)_{max}$ $= \delta_1 F_1 a$
	α	β	β_1	γ	γ_1	δ	δ_1
1.0000	0.0444	0.0479	0.0479	0.3377	0.3377	0.4205	0.4205
1.2000	0.0617	0.0627	0.0447	0.3794	0.3530	0.4558	0.4530
1.4000	0.0774	0.0755	0.0394	0.4111	0.3614	0.4780	0.4729
1.6000	0.0907	0.0862	0.0335	0.4347	0.3660	0.4915	0.4848
1.8000	0.1017	0.0948	0.0278	0.4522	0.3684	0.4992	0.4918
2.0000	0.1106	0.1017	0.0226	0.4650	0.3697	0.5034	0.4958
3.0000	0.1336	0.1189	0.0071	0.4927	0.3712	0.5047	0.5009
4.0000	0.1400	0.1235	0.0020	0.4985	0.3713	0.5018	0.5012
5.0000	0.1416	0.1246	0.0005	0.4997	0.3713	0.5006	0.5012

ordu = 7, ordv = 7, segu = 2, segv = 2, 3, 4, 5, 6, 4, 6, 8, 10							
$\frac{b}{a}$	x_{max} $= \alpha \frac{F_1 a^4}{Eh^3}$	$(M_u)_{max}$ $= \beta F_1 a^2$	$(M_v)_{max}$ $= \beta_1 F_1 a^2$	$(Q_u)_{max}$ $= \gamma F_1 a$	$(Q_v)_{max}$ $= \gamma_1 F_1 a$	$(V_u)_{max}$ $= \delta F_1 a$	$(V_v)_{max}$ $= \delta_1 F_1 a$
	α	β	β_1	γ	γ_1	δ	δ_1
1.0000	0.0444	0.0479	0.0479	0.3371	0.3371	0.4194	0.4194
1.2000	0.0617	0.0627	0.0447	0.3819	0.3517	0.4587	0.4511
1.4000	0.0774	0.0755	0.0393	0.4135	0.3621	0.4806	0.4731
1.6000	0.0907	0.0862	0.0335	0.4369	0.3653	0.4936	0.4836
1.8000	0.1017	0.0949	0.0278	0.4541	0.3675	0.5011	0.4904
2.0000	0.1106	0.1016	0.0225	0.4674	0.3726	0.5057	0.4981
3.0000	0.1336	0.1189	0.0071	0.4925	0.3739	0.5045	0.5031
4.0000	0.1400	0.1234	0.0019	0.4990	0.3738	0.5024	0.5031
5.0000	0.1416	0.1246	0.0005	0.4994	0.3737	0.5003	0.5030

ordu = 7, ordv = 7, segu = 4, segv = 4, 5, 6, 7, 8, 8, 12, 16, 20							
$\frac{b}{a}$	x_{max} $= \alpha \frac{F_1 a^4}{Eh^3}$	$(M_u)_{max}$ $= \beta F_1 a^2$	$(M_v)_{max}$ $= \beta_1 F_1 a^2$	$(Q_u)_{max}$ $= \gamma F_1 a$	$(Q_v)_{max}$ $= \gamma_1 F_1 a$	$(V_u)_{max}$ $= \delta F_1 a$	$(V_v)_{max}$ $= \delta_1 F_1 a$
	α	β	β_1	γ	γ_1	δ	δ_1
1.0000	0.0444	0.0479	0.0479	0.3388	0.3388	0.4217	0.4217
1.2000	0.0617	0.0627	0.0447	0.3795	0.3541	0.4559	0.4541
1.4000	0.0774	0.0755	0.0394	0.4111	0.3624	0.4780	0.4740
1.6000	0.0907	0.0862	0.0335	0.4348	0.3670	0.4916	0.4859
1.8000	0.1017	0.0948	0.0278	0.4522	0.3694	0.4993	0.4928
2.0000	0.1106	0.1017	0.0226	0.4652	0.3708	0.5035	0.4970
3.0000	0.1336	0.1189	0.0071	0.4928	0.3723	0.5048	0.5020
4.0000	0.1400	0.1235	0.0020	0.4985	0.3724	0.5018	0.5023
5.0000	0.1416	0.1246	0.0005	0.4997	0.3724	0.5006	0.5023

ordu = 7, ordv = 7, segu = 6, segv = 6, 7, 8, 9, 10, 12, 18, 24, 30							
$\frac{b}{a}$	$x_{max} = \alpha \frac{F_1 a^4}{Eh^3}$	$(M_u)_{max} = \beta F_1 a^2$	$(M_v)_{max} = \beta_1 F_1 a^2$	$(Q_u)_{max} = \gamma F_1 a$	$(Q_v)_{max} = \gamma_1 F_1 a$	$(V_u)_{max} = \delta F_1 a$	$(V_v)_{max} = \delta_1 F_1 a$
	α	β	β_1	γ	γ_1	δ	δ_1
1.0000	0.0444	0.0479	0.0479	0.3376	0.3376	0.4204	0.4204
1.2000	0.0617	0.0627	0.0447	0.3795	0.3529	0.4560	0.4529
1.4000	0.0774	0.0755	0.0394	0.4112	0.3614	0.4782	0.4729
1.6000	0.0907	0.0862	0.0335	0.4346	0.3660	0.4913	0.4848
1.8000	0.1017	0.0948	0.0278	0.4521	0.3684	0.4991	0.4918
2.0000	0.1106	0.1017	0.0226	0.4651	0.3696	0.5034	0.4957
3.0000	0.1336	0.1189	0.0071	0.4927	0.3711	0.5047	0.5007
4.0000	0.1400	0.1235	0.0020	0.4985	0.3711	0.5018	0.5010
5.0000	0.1416	0.1246	0.0005	0.4997	0.3711	0.5005	0.5010

ordu = 7, ordv = 7, segu = 8, segv = 8, 9, 10, 11, 12, 16, 24, 32, 40							
$\frac{b}{a}$	$x_{max} = \alpha \frac{F_1 a^4}{Eh^3}$	$(M_u)_{max} = \beta F_1 a^2$	$(M_v)_{max} = \beta_1 F_1 a^2$	$(Q_u)_{max} = \gamma F_1 a$	$(Q_v)_{max} = \gamma_1 F_1 a$	$(V_u)_{max} = \delta F_1 a$	$(V_v)_{max} = \delta_1 F_1 a$
	α	β	β_1	γ	γ_1	δ	δ_1
1.0000	0.0444	0.0479	0.0479	0.3377	0.3377	0.4206	0.4206
1.2000	0.0617	0.0627	0.0447	0.3793	0.3530	0.4557	0.4530
1.4000	0.0774	0.0755	0.0394	0.4110	0.3614	0.4779	0.4729
1.6000	0.0907	0.0862	0.0335	0.4348	0.3660	0.4916	0.4848
1.8000	0.1017	0.0948	0.0278	0.4523	0.3684	0.4993	0.4918
2.0000	0.1106	0.1017	0.0226	0.4650	0.3698	0.5034	0.4959
3.0000	0.1336	0.1189	0.0071	0.4927	0.3712	0.5047	0.5010
4.0000	0.1400	0.1235	0.0020	0.4985	0.3713	0.5018	0.5012
5.0000	0.1416	0.1246	0.0005	0.4997	0.3713	0.5006	0.5013

ordu = 7, ordv = 7, segu = 10, segv = 10, 11, 12, 13, 14, 20, 30, 40, 50							
$\frac{b}{a}$	$x_{max} = \alpha \frac{F_1 a^4}{Eh^3}$	$(M_u)_{max} = \beta F_1 a^2$	$(M_v)_{max} = \beta_1 F_1 a^2$	$(Q_u)_{max} = \gamma F_1 a$	$(Q_v)_{max} = \gamma_1 F_1 a$	$(V_u)_{max} = \delta F_1 a$	$(V_v)_{max} = \delta_1 F_1 a$
	α	β	β_1	γ	γ_1	δ	δ_1
1.0000	0.0444	0.0479	0.0479	0.3376	0.3376	0.4204	0.4204
1.2000	0.0617	0.0627	0.0447	0.3794	0.3529	0.4559	0.4529
1.4000	0.0774	0.0755	0.0394	0.4111	0.3614	0.4781	0.4729
1.6000	0.0907	0.0862	0.0335	0.4347	0.3659	0.4914	0.4848
1.8000	0.1017	0.0948	0.0278	0.4521	0.3684	0.4991	0.4918
2.0000	0.1106	0.1017	0.0226	0.4650	0.3697	0.5034	0.4957
3.0000	0.1336	0.1189	0.0071	0.4927	0.3711	0.5047	0.5008
4.0000	0.1400	0.1235	0.0020	0.4985	0.3712	0.5018	0.5011
5.0000	0.1416	0.1246	0.0005	0.4997	0.3712	0.5006	0.5011

ordu = 7, ordv = 7, segu = 12, segv = 12, 13, 14, 15, 16, 24, 36, 48, 60							
$\frac{b}{a}$	$x_{max} = \alpha \frac{F_1 a^4}{Eh^3}$	$(M_u)_{max} = \beta F_1 a^2$	$(M_v)_{max} = \beta_1 F_1 a^2$	$(Q_u)_{max} = \gamma F_1 a$	$(Q_v)_{max} = \gamma_1 F_1 a$	$(V_u)_{max} = \delta F_1 a$	$(V_v)_{max} = \delta_1 F_1 a$
	α	β	β_1	γ	γ_1	δ	δ_1
1.0000	0.0444	0.0479	0.0479	0.3377	0.3377	0.4205	0.4205
1.2000	0.0617	0.0627	0.0447	0.3793	0.3530	0.4558	0.4530
1.4000	0.0774	0.0755	0.0394	0.4110	0.3614	0.4780	0.4729
1.6000	0.0907	0.0862	0.0335	0.4348	0.3659	0.4915	0.4848
1.8000	0.1017	0.0948	0.0278	0.4522	0.3684	0.4992	0.4918
2.0000	0.1106	0.1017	0.0226	0.4650	0.3697	0.5034	0.4958
3.0000	0.1336	0.1189	0.0071	0.4927	0.3712	0.5047	0.5009
4.0000	0.1400	0.1235	0.0020	0.4985	0.3712	0.5018	0.5012
5.0000	0.1416	0.1246	0.0005	0.4997	0.3713	0.5006	0.5012

A.2 Simply supported and centrally point loaded rectangular plate

Coefficient α for maximum deflection, x_{max}, for the case of a centrally point loaded, P_1, simply supported rectangular plate, figure A.2.

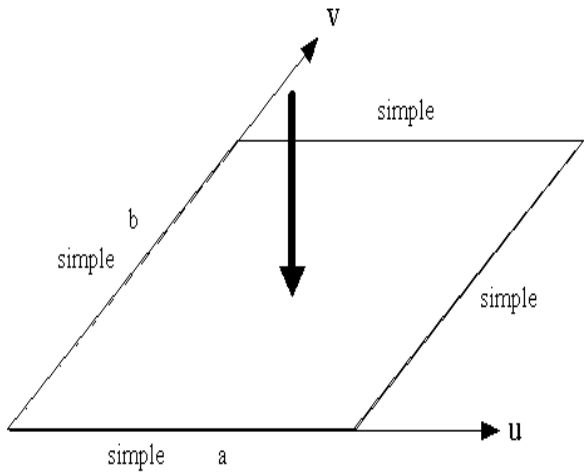


Figure A.2: Simply supported centrally point loaded rectangular plate

$\frac{b}{a}$	1.0	1.1	1.2	1.4	1.6	1.8	2.0	3.0
Coeff α *in* $x_{max} = \alpha \frac{P_1 a^3}{Eh^3}$								
	0.1265	0.1381	0.1478	0.1621	0.1714	0.1769	0.1803	0.1846

Table A.2: Numerical results from [85], simply supported rectangular plate, point load

$\mathbf{ordu = 3, ordv = 3, segu = 2, segv = 2}$								
$Coeff\ \alpha\ in\ x_{max} = \alpha \frac{P_1 a^3}{Eh^3}$								
	0.0873	0.1097	0.1137	0.1284	0.1314	0.1381	0.1251	0.1269

$\mathbf{ordu = 3, ordv = 3, segu = 4, segv = 4}$								
	0.1125	0.1276	0.1351	0.1506	0.1569	0.1640	0.1628	0.1665

$\mathbf{ordu = 3, ordv = 3, segu = 6, segv = 6}$								
	0.1193	0.1325	0.1409	0.1560	0.1633	0.1700	0.1716	0.1757

$\mathbf{ordu = 3, ordv = 3, segu = 8, segv = 8}$								
	0.1221	0.1346	0.1434	0.1583	0.1661	0.1725	0.1751	0.1793

$\mathbf{ordu = 3, ordv = 3, segu = 10, segv = 10}$								
	0.1236	0.1357	0.1448	0.1595	0.1676	0.1738	0.1768	0.1811

$\mathbf{ordu = 3, ordv = 3, segu = 12, segv = 12}$								
	0.1244	0.1364	0.1456	0.1602	0.1685	0.1746	0.1778	0.1822

$\mathbf{ordu = 4, ordv = 4, segu = 2, segv = 2}$								
	0.1210	0.1277	0.1440	0.1546	0.1677	0.1701	0.1739	0.1783

$\mathbf{ordu = 4, ordv = 4, segu = 4, segv = 4}$								
	0.1249	0.1341	0.1466	0.1587	0.1700	0.1734	0.1786	0.1831

$\mathbf{ordu = 4, ordv = 4, segu = 6, segv = 6}$								
	0.1259	0.1362	0.1473	0.1602	0.1707	0.1747	0.1796	0.1841

$\mathbf{ordu = 4, ordv = 4, segu = 8, segv = 8}$								
	0.1262	0.1370	0.1476	0.1610	0.1710	0.1754	0.1800	0.1845

$\mathbf{ordu = 4, ordv = 4, segu = 10, segv = 10}$								
	0.1264	0.1375	0.1477	0.1614	0.1711	0.1759	0.1802	0.1846

$\mathbf{ordu = 4, ordv = 4, segu = 12, segv = 12}$								
	0.1265	0.1377	0.1478	0.1616	0.1712	0.1761	0.1802	0.1847

$\mathbf{ordu = 5, ordv = 5, segu = 2, segv = 2}$								
	0.1144	0.1296	0.1375	0.1541	0.1612	0.1689	0.1626	0.1652

$\mathbf{ordu = 5, ordv = 5, segu = 4, segv = 4}$								
	0.1208	0.1343	0.1429	0.1584	0.1663	0.1731	0.1741	0.1785

ordu = 5, ordv = 5, segu = 6, segv = 6								
	0.1237	0.1363	0.1453	0.1603	0.1686	0.1750	0.1774	0.1819

ordu = 5, ordv = 5, segu = 8, segv = 8								
	0.1250	0.1371	0.1464	0.1611	0.1696	0.1758	0.1787	0.1832

ordu = 5, ordv = 5, segu = 10, segv = 10								
	0.1256	0.1375	0.1469	0.1615	0.1702	0.1762	0.1793	0.1838

ordu = 5, ordv = 5, segu = 12, segv = 12								
	0.1259	0.1378	0.1472	0.1618	0.1705	0.1764	0.1797	0.1842

ordu = 6, ordv = 6, segu = 2, segv = 2								
	0.1223	0.1325	0.1442	0.1569	0.1677	0.1717	0.1736	0.1769

ordu = 6, ordv = 6, segu = 4, segv = 4								
	0.1244	0.1352	0.1460	0.1593	0.1694	0.1739	0.1779	0.1823

ordu = 6, ordv = 6, segu = 6, segv = 6								
	0.1255	0.136	0.1469	0.1605	0.1702	0.1751	0.1792	0.1836

ordu = 6, ordv = 6, segu = 8, segv = 8								
	0.1260	0.1372	0.1473	0.1611	0.1707	0.1757	0.1797	0.1842

ordu = 6, ordv = 6, segu = 10, segv = 10								
	0.1262	0.1376	0.1476	0.1615	0.1709	0.1761	0.1800	0.1844

ordu = 6, ordv = 6, segu = 12, segv = 12								
	0.1264	0.1378	0.1477	0.1617	0.1710	0.1763	0.1801	0.1846

ordu = 7, ordv = 7, segu = 2, segv = 2								
	0.1212	0.133	0.1428	0.1580	0.1660	0.1727	0.1708	0.1723

ordu = 7, ordv = 7, segu = 4, segv = 4								
	0.1233	0.1357	0.1448	0.1597	0.1680	0.1744	0.1760	0.1802

ordu = 7, ordv = 7, segu = 6, segv = 6								
	0.1246	0.1367	0.1460	0.1607	0.1692	0.1753	0.1781	0.1825

ordu = 7, ordv = 7, segu = 8, segv = 8								
	0.1253	0.1373	0.1467	0.1612	0.1699	0.1759	0.1790	0.1835

ordu = 7, ordv = 7, segu = 10, segv = 10								
	0.1258	0.1376	0.1471	0.1616	0.1704	0.1762	0.1795	0.1840

ordu = 7, ordv = 7, segu = 12, segv = 12								
	0.1260	0.1378	0.1474	0.1618	0.1707	0.1764	0.1798	0.1843

A.3 Simply supported and partially loaded square plate

Coefficients β for maximum bending moment in u, $(M_u)_{max}$ for a simply supported partially loaded (over centrally located area size $a_1 \times b_1$) square plate, figure A.3.

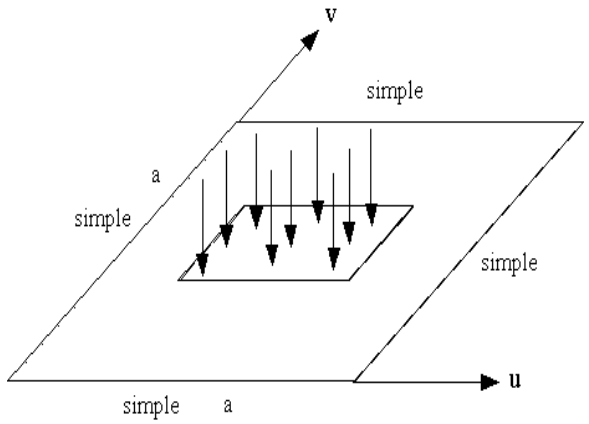


Figure A.3: Simply supported partially loaded square plate

$\frac{a_1}{a}$	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
$\frac{b_1}{a}$	*Coeff* β *in* $(M_u)_{max} = \beta P$									
0.1	0.284	0.232	0.197	0.170	0.150	0.134	0.120	0.108	0.098	0.088
0.2	0.254	0.214	0.184	0.161	0.142	0.127	0.114	0.103	0.093	0.084
0.3	0.225	0.195	0.168	0.151	0.134	0.120	0.108	0.098	0.088	0.080
0.4	0.203	0.179	0.158	0.141	0.126	0.113	0.102	0.092	0.084	0.076
0.5	0.185	0.164	0.146	0.131	0.116	0.106	0.096	0.087	0.079	0.071
0.6	0.168	0.150	0.135	0.121	0.109	0.099	0.090	0.081	0.074	0.067
0.7	0.153	0.137	0.124	0.112	0.101	0.091	0.083	0.076	0.069	0.062
0.8	0.140	0.126	0.114	0.103	0.094	0.085	0.077	0.070	0.063	0.057
0.9	0.127	0.115	0.104	0.094	0.086	0.078	0.070	0.064	0.058	0.053
1.0	0.115	0.105	0.095	0.086	0.078	0.071	0.064	0.058	0.053	0.048

Table A.3: Numerical results from [85], simply supported square plate, partial load

ordu = 3, ordv = 3, segu = 2, segv = 2										
$\frac{a_1}{a}$	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
$\frac{b_1}{a}$	*Coeff* β *in* $(M_u)_{max} = \beta P$									
0.1	0.083	0.082	0.080	0.078	0.076	0.073	0.069	0.065	0.060	0.055
0.2	0.082	0.081	0.080	0.078	0.075	0.072	0.069	0.064	0.060	0.055
0.3	0.080	0.080	0.078	0.076	0.074	0.071	0.067	0.063	0.059	0.054
0.4	0.078	0.078	0.076	0.074	0.072	0.069	0.066	0.062	0.057	0.052
0.5	0.076	0.075	0.074	0.072	0.070	0.067	0.064	0.060	0.056	0.051
0.6	0.073	0.072	0.071	0.069	0.067	0.064	0.061	0.058	0.053	0.049
0.7	0.069	0.069	0.067	0.066	0.064	0.061	0.058	0.055	0.051	0.046
0.8	0.065	0.064	0.063	0.062	0.060	0.058	0.055	0.051	0.048	0.044
0.9	0.060	0.060	0.059	0.057	0.056	0.053	0.051	0.048	0.044	0.040
1.0	0.055	0.055	0.054	0.052	0.051	0.049	0.046	0.044	0.040	0.037

ordu = 3, ordv = 3, segu = 2, segv = 2										
$\frac{a_1}{a}$	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
$\frac{b_1}{a}$	*Coeff* β *in* $(M_u)_{max} = \beta P$									
0.1	0.083	0.082	0.080	0.078	0.076	0.073	0.069	0.065	0.060	0.055
0.2	0.082	0.081	0.080	0.078	0.075	0.072	0.069	0.064	0.060	0.055
0.3	0.080	0.080	0.078	0.076	0.074	0.071	0.067	0.063	0.059	0.054
0.4	0.078	0.078	0.076	0.074	0.072	0.069	0.066	0.062	0.057	0.052
0.5	0.076	0.075	0.074	0.072	0.070	0.067	0.064	0.060	0.056	0.051
0.6	0.073	0.072	0.071	0.069	0.067	0.064	0.061	0.058	0.053	0.049
0.7	0.069	0.069	0.067	0.066	0.064	0.061	0.058	0.055	0.051	0.046
0.8	0.065	0.064	0.063	0.062	0.060	0.058	0.055	0.051	0.048	0.044
0.9	0.060	0.060	0.059	0.057	0.056	0.053	0.051	0.048	0.044	0.040
1.0	0.055	0.055	0.054	0.052	0.051	0.049	0.046	0.044	0.040	0.037

ordu = 3, ordv = 3, segu = 4, segv = 4										
$\frac{a_1}{a}$	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
$\frac{b_1}{a}$	*Coeff* β *in* $(M_u)_{max} = \beta P$									
0.1	0.136	0.134	0.130	0.125	0.118	0.110	0.101	0.092	0.084	0.076
0.2	0.134	0.132	0.128	0.123	0.116	0.108	0.099	0.091	0.083	0.075
0.3	0.131	0.128	0.125	0.120	0.113	0.105	0.097	0.089	0.081	0.073
0.4	0.126	0.124	0.120	0.116	0.109	0.102	0.094	0.086	0.078	0.071
0.5	0.120	0.118	0.115	0.110	0.104	0.097	0.090	0.082	0.075	0.068
0.6	0.113	0.111	0.108	0.104	0.098	0.092	0.085	0.078	0.071	0.064
0.7	0.105	0.104	0.101	0.097	0.092	0.086	0.079	0.073	0.066	0.060
0.8	0.097	0.096	0.093	0.090	0.085	0.079	0.074	0.067	0.062	0.056
0.9	0.089	0.088	0.085	0.082	0.078	0.073	0.068	0.062	0.057	0.051
1.0	0.081	0.080	0.078	0.075	0.071	0.066	0.061	0.056	0.052	0.047

ordu = 3, ordv = 3, segu = 6, segv = 6										
$\frac{a_1}{a}$	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
$\frac{b_1}{a}$	*Coeff* β *in* $(M_u)_{max} = \beta P$									
0.1	0.173	0.167	0.159	0.146	0.133	0.120	0.109	0.098	0.089	0.081
0.2	0.169	0.164	0.155	0.143	0.130	0.118	0.107	0.097	0.087	0.079
0.3	0.162	0.157	0.149	0.138	0.126	0.114	0.103	0.094	0.085	0.077
0.4	0.153	0.149	0.141	0.131	0.120	0.109	0.099	0.090	0.081	0.074
0.5	0.143	0.139	0.132	0.123	0.113	0.103	0.093	0.085	0.077	0.070
0.6	0.132	0.128	0.123	0.114	0.105	0.096	0.088	0.080	0.073	0.066
0.7	0.122	0.118	0.113	0.106	0.098	0.089	0.082	0.074	0.068	0.061
0.8	0.112	0.109	0.104	0.097	0.090	0.083	0.076	0.069	0.063	0.057
0.9	0.102	0.099	0.095	0.089	0.083	0.076	0.069	0.063	0.058	0.052
1.0	0.093	0.090	0.086	0.081	0.075	0.069	0.063	0.058	0.052	0.047

ordu = 3, ordv = 3, segu = 8, segv = 8										
$\frac{a}{a}$	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
$\frac{b}{a}$	*Coeff* β *in* $(M_u)_{max} = \beta P$									
0.1	0.199	0.189	0.174	0.156	0.139	0.125	0.112	0.101	0.092	0.083
0.2	0.192	0.183	0.168	0.151	0.135	0.122	0.110	0.099	0.090	0.081
0.3	0.181	0.173	0.160	0.144	0.130	0.117	0.106	0.095	0.086	0.078
0.4	0.168	0.161	0.149	0.136	0.123	0.111	0.100	0.091	0.082	0.075
0.5	0.155	0.149	0.139	0.127	0.115	0.104	0.095	0.086	0.078	0.071
0.6	0.143	0.137	0.128	0.118	0.107	0.098	0.089	0.080	0.073	0.066
0.7	0.131	0.126	0.118	0.109	0.099	0.091	0.082	0.075	0.068	0.062
0.8	0.120	0.116	0.109	0.100	0.092	0.084	0.076	0.069	0.063	0.057
0.9	0.110	0.106	0.099	0.092	0.084	0.077	0.070	0.064	0.058	0.052
1.0	0.099	0.096	0.090	0.083	0.076	0.070	0.064	0.058	0.053	0.048

ordu = 3, ordv = 3, segu = 10, segv = 10										
$\frac{a}{a}$	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
$\frac{b}{a}$	*Coeff* β *in* $(M_u)_{max} = \beta P$									
0.1	0.218	0.203	0.181	0.160	0.142	0.127	0.114	0.103	0.093	0.084
0.2	0.208	0.195	0.174	0.155	0.138	0.124	0.111	0.100	0.091	0.082
0.3	0.193	0.181	0.164	0.147	0.131	0.118	0.106	0.096	0.087	0.079
0.4	0.177	0.167	0.153	0.138	0.124	0.112	0.101	0.091	0.083	0.075
0.5	0.163	0.154	0.142	0.128	0.116	0.105	0.095	0.086	0.078	0.071
0.6	0.149	0.142	0.131	0.119	0.108	0.098	0.089	0.081	0.073	0.066
0.7	0.137	0.130	0.121	0.110	0.100	0.091	0.083	0.075	0.068	0.062
0.8	0.125	0.120	0.111	0.101	0.092	0.084	0.076	0.070	0.063	0.057
0.9	0.114	0.109	0.101	0.093	0.085	0.077	0.070	0.064	0.058	0.052
1.0	0.104	0.099	0.092	0.084	0.077	0.070	0.064	0.058	0.053	0.048

ordu = 3, ordv = 3, segu = 12, segv = 12										
$\frac{a}{a}$	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
$\frac{b}{a}$	*Coeff* β *in* $(M_u)_{max} = \beta P$									
0.1	0.233	0.212	0.185	0.163	0.144	0.129	0.116	0.104	0.094	0.085
0.2	0.219	0.201	0.177	0.157	0.139	0.125	0.112	0.101	0.092	0.083
0.3	0.201	0.186	0.166	0.148	0.132	0.119	0.107	0.097	0.088	0.079
0.4	0.183	0.171	0.154	0.139	0.125	0.112	0.101	0.092	0.083	0.075
0.5	0.168	0.157	0.143	0.129	0.117	0.105	0.095	0.086	0.078	0.071
0.6	0.154	0.145	0.132	0.120	0.109	0.098	0.089	0.081	0.073	0.066
0.7	0.141	0.133	0.122	0.111	0.101	0.091	0.083	0.075	0.068	0.062
0.8	0.129	0.122	0.112	0.102	0.093	0.084	0.077	0.070	0.063	0.057
0.9	0.118	0.111	0.102	0.093	0.085	0.077	0.070	0.064	0.058	0.053
1.0	0.107	0.101	0.093	0.085	0.077	0.070	0.064	0.058	0.053	0.048

ordu = 4, ordv = 4, segu = 2, segv = 2										
$\frac{a}{a}$	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
$\frac{b}{a}$	*Coeff* β *in* $(M_u)_{max} = \beta P$									
0.1	0.195	0.189	0.181	0.170	0.158	0.145	0.131	0.118	0.106	0.096
0.2	0.190	0.185	0.177	0.167	0.155	0.142	0.129	0.116	0.104	0.094
0.3	0.184	0.179	0.171	0.161	0.150	0.138	0.125	0.113	0.101	0.091
0.4	0.176	0.171	0.164	0.154	0.144	0.132	0.120	0.109	0.098	0.088
0.5	0.166	0.161	0.154	0.146	0.136	0.125	0.114	0.103	0.093	0.084
0.6	0.155	0.151	0.144	0.137	0.128	0.118	0.107	0.097	0.088	0.079
0.7	0.143	0.139	0.134	0.127	0.118	0.109	0.100	0.091	0.082	0.074
0.8	0.131	0.128	0.123	0.116	0.109	0.101	0.092	0.084	0.076	0.069
0.9	0.119	0.116	0.112	0.106	0.100	0.092	0.085	0.077	0.070	0.063
1.0	0.108	0.105	0.101	0.096	0.090	0.084	0.077	0.070	0.063	0.057

ordu = 4, ordv = 4, segu = 4, segv = 4										
$\frac{a_1}{a}$	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
$\frac{b_1}{a}$	*Coeff* β *in* $(M_u)_{max} = \beta P$									
0.1	0.249	0.234	0.213	0.189	0.164	0.142	0.124	0.109	0.098	0.089
0.2	0.239	0.225	0.205	0.182	0.159	0.138	0.120	0.106	0.095	0.086
0.3	0.224	0.212	0.194	0.172	0.151	0.131	0.115	0.102	0.091	0.083
0.4	0.207	0.196	0.180	0.161	0.141	0.124	0.109	0.096	0.087	0.078
0.5	0.189	0.179	0.165	0.148	0.131	0.115	0.102	0.090	0.081	0.073
0.6	0.172	0.163	0.151	0.136	0.121	0.107	0.094	0.084	0.076	0.068
0.7	0.156	0.148	0.137	0.124	0.111	0.098	0.087	0.078	0.070	0.063
0.8	0.142	0.135	0.125	0.114	0.101	0.090	0.080	0.072	0.065	0.058
0.9	0.129	0.123	0.114	0.104	0.093	0.082	0.073	0.066	0.059	0.053
1.0	0.117	0.112	0.104	0.094	0.084	0.075	0.067	0.060	0.054	0.049

ordu = 4, ordv = 4, segu = 6, segv = 6										
$\frac{a_1}{a}$	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
$\frac{b_1}{a}$	*Coeff* β *in* $(M_u)_{max} = \beta P$									
0.1	0.281	0.253	0.217	0.182	0.155	0.136	0.122	0.110	0.100	0.090
0.2	0.263	0.238	0.205	0.174	0.149	0.131	0.117	0.106	0.096	0.087
0.3	0.239	0.217	0.190	0.162	0.140	0.124	0.111	0.100	0.091	0.082
0.4	0.214	0.196	0.173	0.150	0.131	0.116	0.104	0.094	0.085	0.077
0.5	0.193	0.178	0.158	0.138	0.121	0.108	0.097	0.088	0.080	0.072
0.6	0.175	0.162	0.145	0.128	0.113	0.100	0.091	0.082	0.074	0.067
0.7	0.160	0.148	0.133	0.118	0.104	0.093	0.084	0.076	0.069	0.063
0.8	0.146	0.136	0.122	0.108	0.096	0.086	0.078	0.070	0.064	0.058
0.9	0.133	0.124	0.112	0.099	0.088	0.079	0.071	0.065	0.059	0.053
1.0	0.121	0.112	0.101	0.090	0.080	0.072	0.065	0.059	0.053	0.048

ordu = 4, ordv = 4, segu = 8, segv = 8										
$\frac{a_1}{a}$	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
$\frac{b_1}{a}$	*Coeff* β *in* $(M_u)_{max} = \beta P$									
0.1	0.298	0.256	0.210	0.175	0.153	0.137	0.123	0.110	0.100	0.090
0.2	0.271	0.235	0.196	0.166	0.145	0.130	0.117	0.106	0.095	0.086
0.3	0.240	0.212	0.180	0.155	0.136	0.122	0.110	0.099	0.090	0.081
0.4	0.214	0.191	0.165	0.144	0.127	0.114	0.103	0.093	0.084	0.076
0.5	0.194	0.174	0.152	0.134	0.119	0.107	0.097	0.088	0.079	0.072
0.6	0.177	0.160	0.141	0.124	0.111	0.100	0.090	0.082	0.074	0.067
0.7	0.161	0.146	0.129	0.114	0.103	0.093	0.084	0.076	0.069	0.062
0.8	0.147	0.134	0.119	0.105	0.095	0.086	0.078	0.070	0.064	0.058
0.9	0.133	0.122	0.108	0.096	0.087	0.078	0.071	0.064	0.058	0.053
1.0	0.121	0.110	0.098	0.087	0.079	0.071	0.065	0.059	0.053	0.048

ordu = 4, ordv = 4, segu = 10, segv = 10										
$\frac{a_1}{a}$	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
$\frac{b_1}{a}$	*Coeff* β *in* $(M_u)_{max} = \beta P$									
0.1	0.307	0.251	0.203	0.174	0.154	0.137	0.122	0.110	0.099	0.090
0.2	0.272	0.229	0.190	0.164	0.145	0.130	0.116	0.105	0.095	0.086
0.3	0.239	0.206	0.175	0.152	0.136	0.122	0.109	0.099	0.089	0.081
0.4	0.214	0.187	0.162	0.142	0.127	0.114	0.103	0.093	0.084	0.076
0.5	0.194	0.172	0.149	0.132	0.119	0.107	0.096	0.087	0.079	0.072
0.6	0.177	0.157	0.138	0.123	0.111	0.100	0.090	0.082	0.074	0.067
0.7	0.161	0.144	0.127	0.113	0.102	0.093	0.084	0.076	0.069	0.062
0.8	0.146	0.131	0.116	0.104	0.094	0.085	0.077	0.070	0.064	0.058
0.9	0.133	0.120	0.106	0.095	0.086	0.078	0.071	0.064	0.058	0.053
1.0	0.121	0.109	0.097	0.087	0.078	0.071	0.064	0.058	0.053	0.048

ordu = 4, ordv = 4, segu = 12, segv = 12										
$\frac{a}{L}$	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
$\frac{b}{L}$	*Coeff* β *in* $(M_u)_{max} = \beta P$									
0.1	0.310	0.246	0.200	0.174	0.154	0.136	0.122	0.110	0.099	0.090
0.2	0.270	0.223	0.186	0.163	0.144	0.129	0.115	0.104	0.094	0.085
0.3	0.237	0.202	0.173	0.152	0.135	0.121	0.109	0.098	0.089	0.080
0.4	0.214	0.184	0.160	0.142	0.127	0.114	0.103	0.093	0.084	0.076
0.5	0.193	0.169	0.148	0.132	0.119	0.107	0.096	0.087	0.079	0.072
0.6	0.176	0.155	0.137	0.123	0.110	0.099	0.090	0.082	0.074	0.067
0.7	0.160	0.142	0.126	0.113	0.102	0.092	0.084	0.076	0.069	0.062
0.8	0.146	0.130	0.115	0.104	0.094	0.085	0.077	0.070	0.064	0.058
0.9	0.132	0.118	0.105	0.095	0.086	0.078	0.071	0.064	0.058	0.053
1.0	0.120	0.107	0.096	0.086	0.078	0.071	0.064	0.058	0.053	0.048

ordu = 5, ordv = 5, segu = 2, segv = 2										
$\frac{a}{L}$	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
$\frac{b}{L}$	*Coeff* β *in* $(M_u)_{max} = \beta P$									
0.1	0.143	0.140	0.136	0.130	0.123	0.116	0.107	0.098	0.089	0.081
0.2	0.140	0.138	0.134	0.128	0.122	0.114	0.105	0.097	0.088	0.079
0.3	0.137	0.134	0.130	0.125	0.119	0.111	0.103	0.094	0.086	0.078
0.4	0.132	0.130	0.126	0.121	0.115	0.107	0.100	0.091	0.083	0.075
0.5	0.126	0.124	0.120	0.115	0.110	0.103	0.095	0.087	0.080	0.072
0.6	0.119	0.117	0.114	0.109	0.104	0.097	0.090	0.083	0.076	0.068
0.7	0.111	0.109	0.106	0.102	0.097	0.091	0.085	0.078	0.071	0.064
0.8	0.103	0.101	0.098	0.094	0.090	0.084	0.079	0.072	0.066	0.060
0.9	0.094	0.092	0.090	0.087	0.082	0.078	0.072	0.066	0.061	0.055
1.0	0.085	0.084	0.082	0.079	0.075	0.070	0.066	0.060	0.055	0.050

ordu = 5, ordv = 5, segu = 4, segv = 4										
$\frac{a}{L}$	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
$\frac{b}{L}$	*Coeff* β *in* $(M_u)_{max} = \beta P$									
0.1	0.179	0.173	0.165	0.153	0.140	0.126	0.113	0.101	0.091	0.082
0.2	0.175	0.169	0.161	0.150	0.137	0.124	0.111	0.099	0.089	0.081
0.3	0.168	0.163	0.155	0.145	0.133	0.120	0.108	0.096	0.087	0.078
0.4	0.160	0.155	0.148	0.138	0.127	0.115	0.103	0.092	0.083	0.075
0.5	0.150	0.145	0.138	0.130	0.119	0.108	0.097	0.088	0.079	0.071
0.6	0.138	0.135	0.128	0.120	0.111	0.101	0.091	0.082	0.074	0.067
0.7	0.127	0.124	0.118	0.111	0.103	0.094	0.085	0.076	0.069	0.062
0.8	0.116	0.113	0.108	0.102	0.094	0.086	0.078	0.071	0.064	0.058
0.9	0.106	0.103	0.099	0.093	0.086	0.079	0.071	0.065	0.058	0.053
1.0	0.096	0.094	0.090	0.084	0.078	0.072	0.065	0.059	0.053	0.048

ordu = 5, ordv = 5, segu = 6, segv = 6										
$\frac{a}{L}$	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
$\frac{b}{L}$	*Coeff* β *in* $(M_u)_{max} = \beta P$									
0.1	0.210	0.200	0.184	0.165	0.146	0.129	0.115	0.103	0.094	0.085
0.2	0.203	0.193	0.178	0.160	0.142	0.125	0.112	0.101	0.092	0.083
0.3	0.192	0.183	0.169	0.153	0.136	0.120	0.108	0.097	0.088	0.080
0.4	0.179	0.171	0.158	0.144	0.128	0.114	0.102	0.092	0.084	0.076
0.5	0.164	0.157	0.146	0.133	0.120	0.107	0.096	0.087	0.079	0.071
0.6	0.150	0.144	0.135	0.123	0.111	0.100	0.090	0.081	0.074	0.067
0.7	0.137	0.132	0.124	0.113	0.102	0.092	0.083	0.075	0.069	0.062
0.8	0.126	0.121	0.113	0.104	0.094	0.085	0.077	0.070	0.063	0.057
0.9	0.115	0.110	0.104	0.095	0.086	0.078	0.070	0.064	0.058	0.053
1.0	0.104	0.100	0.094	0.087	0.079	0.071	0.064	0.058	0.053	0.048

ordu = 5, ordv = 5, segu = 8, segv = 8										
$\frac{a_1}{a}$	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
$\frac{b_1}{a}$	*Coeff* β *in* $(M_u)_{max} = \beta P$									
0.1	0.234	0.217	0.193	0.168	0.146	0.130	0.117	0.106	0.095	0.086
0.2	0.223	0.207	0.185	0.162	0.141	0.126	0.113	0.103	0.093	0.084
0.3	0.206	0.193	0.173	0.153	0.134	0.120	0.108	0.098	0.089	0.080
0.4	0.188	0.177	0.160	0.142	0.126	0.113	0.102	0.093	0.084	0.076
0.5	0.172	0.162	0.148	0.132	0.118	0.106	0.096	0.087	0.079	0.071
0.6	0.157	0.148	0.136	0.122	0.110	0.099	0.090	0.081	0.074	0.067
0.7	0.144	0.136	0.125	0.113	0.102	0.092	0.083	0.076	0.069	0.062
0.8	0.132	0.125	0.115	0.104	0.094	0.085	0.077	0.070	0.063	0.057
0.9	0.120	0.114	0.105	0.095	0.086	0.078	0.070	0.064	0.058	0.053
1.0	0.109	0.103	0.095	0.086	0.078	0.070	0.064	0.058	0.053	0.048

ordu = 5, ordv = 5, segu = 10, segv = 10										
$\frac{a_1}{a}$	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
$\frac{b_1}{a}$	*Coeff* β *in* $(M_u)_{max} = \beta P$									
0.1	0.252	0.227	0.195	0.167	0.147	0.132	0.118	0.106	0.096	0.087
0.2	0.235	0.214	0.186	0.161	0.142	0.127	0.114	0.103	0.093	0.084
0.3	0.214	0.196	0.173	0.151	0.134	0.120	0.109	0.098	0.089	0.080
0.4	0.194	0.179	0.159	0.141	0.126	0.113	0.102	0.092	0.084	0.076
0.5	0.177	0.164	0.147	0.131	0.117	0.106	0.096	0.087	0.079	0.071
0.6	0.162	0.151	0.136	0.122	0.109	0.099	0.090	0.081	0.074	0.067
0.7	0.148	0.138	0.125	0.112	0.101	0.092	0.083	0.076	0.069	0.062
0.8	0.135	0.126	0.115	0.103	0.093	0.085	0.077	0.070	0.063	0.057
0.9	0.123	0.115	0.105	0.094	0.086	0.078	0.071	0.064	0.058	0.053
1.0	0.112	0.105	0.095	0.086	0.078	0.071	0.064	0.058	0.053	0.048

ordu = 5, ordv = 5, segu = 12, segv = 12										
$\frac{a_1}{a}$	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
$\frac{b_1}{a}$	*Coeff* β *in* $(M_u)_{max} = \beta P$									
0.1	0.264	0.231	0.195	0.168	0.149	0.133	0.119	0.107	0.097	0.088
0.2	0.243	0.215	0.184	0.160	0.143	0.127	0.114	0.103	0.093	0.084
0.3	0.218	0.196	0.171	0.150	0.134	0.120	0.108	0.098	0.088	0.080
0.4	0.197	0.179	0.158	0.140	0.126	0.113	0.102	0.092	0.084	0.076
0.5	0.180	0.165	0.147	0.131	0.118	0.106	0.096	0.087	0.079	0.071
0.6	0.164	0.151	0.135	0.121	0.110	0.099	0.090	0.081	0.074	0.067
0.7	0.150	0.139	0.125	0.112	0.102	0.092	0.083	0.076	0.069	0.062
0.8	0.137	0.127	0.114	0.103	0.094	0.085	0.077	0.070	0.063	0.057
0.9	0.125	0.116	0.104	0.094	0.086	0.078	0.071	0.064	0.058	0.053
1.0	0.113	0.105	0.095	0.086	0.078	0.071	0.064	0.058	0.053	0.048

ordu = 6, ordv = 6, segu = 2, segv = 2										
$\frac{a_1}{a}$	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
$\frac{b_1}{a}$	*Coeff* β *in* $(M_u)_{max} = \beta P$									
0.1	0.195	0.188	0.176	0.162	0.146	0.130	0.115	0.102	0.092	0.083
0.2	0.190	0.182	0.171	0.157	0.142	0.127	0.113	0.100	0.090	0.081
0.3	0.181	0.175	0.164	0.151	0.137	0.122	0.109	0.097	0.087	0.079
0.4	0.171	0.165	0.155	0.143	0.130	0.116	0.104	0.093	0.083	0.075
0.5	0.159	0.153	0.145	0.134	0.122	0.109	0.098	0.087	0.079	0.071
0.6	0.146	0.141	0.133	0.124	0.113	0.102	0.091	0.082	0.074	0.067
0.7	0.134	0.129	0.122	0.114	0.104	0.094	0.085	0.076	0.068	0.062
0.8	0.122	0.118	0.112	0.104	0.095	0.086	0.078	0.070	0.063	0.057
0.9	0.111	0.107	0.102	0.095	0.087	0.079	0.071	0.064	0.058	0.052
1.0	0.101	0.097	0.092	0.086	0.079	0.072	0.065	0.058	0.053	0.048

ordu = 6, ordv = 6, segu = 4, segv = 4										
$\frac{a}{L}$	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
$\frac{b}{L}$	*Coeff* β *in* $(M_u)_{max} = \beta P$									
0.1	0.225	0.211	0.192	0.169	0.148	0.130	0.115	0.104	0.095	0.086
0.2	0.216	0.203	0.185	0.164	0.144	0.126	0.112	0.102	0.092	0.084
0.3	0.202	0.191	0.174	0.155	0.137	0.121	0.108	0.097	0.089	0.080
0.4	0.187	0.177	0.162	0.145	0.129	0.114	0.102	0.092	0.084	0.076
0.5	0.170	0.162	0.149	0.134	0.120	0.107	0.096	0.087	0.079	0.071
0.6	0.155	0.148	0.137	0.124	0.111	0.099	0.089	0.081	0.074	0.067
0.7	0.142	0.135	0.125	0.114	0.102	0.092	0.083	0.075	0.068	0.062
0.8	0.130	0.124	0.115	0.105	0.094	0.085	0.076	0.069	0.063	0.057
0.9	0.119	0.113	0.105	0.096	0.086	0.078	0.070	0.064	0.058	0.053
1.0	0.108	0.103	0.096	0.087	0.079	0.071	0.064	0.058	0.053	0.048

ordu = 6, ordv = 6, segu = 6, segv = 6										
$\frac{a}{L}$	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
$\frac{b}{L}$	*Coeff* β *in* $(M_u)_{max} = \beta P$									
0.1	0.250	0.227	0.198	0.170	0.147	0.130	0.118	0.107	0.096	0.087
0.2	0.235	0.215	0.189	0.163	0.142	0.126	0.114	0.103	0.093	0.084
0.3	0.215	0.198	0.176	0.153	0.134	0.120	0.108	0.098	0.089	0.080
0.4	0.195	0.181	0.162	0.142	0.126	0.113	0.102	0.093	0.084	0.076
0.5	0.177	0.165	0.148	0.132	0.117	0.105	0.096	0.087	0.079	0.071
0.6	0.162	0.151	0.137	0.122	0.109	0.098	0.089	0.081	0.074	0.067
0.7	0.148	0.139	0.126	0.113	0.101	0.091	0.083	0.076	0.069	0.062
0.8	0.136	0.127	0.116	0.104	0.093	0.084	0.077	0.070	0.063	0.057
0.9	0.123	0.116	0.106	0.095	0.085	0.077	0.070	0.064	0.058	0.053
1.0	0.112	0.105	0.096	0.086	0.078	0.070	0.064	0.058	0.053	0.048

ordu = 6, ordv = 6, segu = 8, segv = 8										
$\frac{a}{L}$	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
$\frac{b}{L}$	*Coeff* β *in* $(M_u)_{max} = \beta P$									
0.1	0.268	0.235	0.197	0.168	0.148	0.133	0.120	0.107	0.097	0.088
0.2	0.247	0.218	0.186	0.160	0.142	0.128	0.115	0.103	0.093	0.085
0.3	0.221	0.199	0.172	0.150	0.134	0.120	0.109	0.098	0.089	0.080
0.4	0.199	0.181	0.159	0.140	0.125	0.113	0.102	0.092	0.084	0.076
0.5	0.181	0.166	0.147	0.130	0.117	0.106	0.096	0.087	0.079	0.071
0.6	0.166	0.152	0.136	0.121	0.109	0.099	0.090	0.081	0.074	0.067
0.7	0.152	0.140	0.125	0.112	0.101	0.092	0.083	0.076	0.069	0.062
0.8	0.138	0.128	0.115	0.103	0.093	0.085	0.077	0.070	0.063	0.057
0.9	0.126	0.116	0.105	0.094	0.085	0.078	0.071	0.064	0.058	0.053
1.0	0.114	0.106	0.095	0.086	0.078	0.071	0.064	0.058	0.053	0.048

ordu = 6, ordv = 6, segu = 10, segv = 10										
$\frac{a}{L}$	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
$\frac{b}{L}$	*Coeff* β *in* $(M_u)_{max} = \beta P$									
0.1	0.279	0.236	0.195	0.169	0.150	0.134	0.120	0.108	0.097	0.088
0.2	0.252	0.217	0.184	0.160	0.143	0.128	0.114	0.103	0.093	0.084
0.3	0.224	0.197	0.170	0.150	0.134	0.120	0.108	0.098	0.088	0.080
0.4	0.202	0.180	0.158	0.140	0.126	0.113	0.102	0.092	0.084	0.076
0.5	0.184	0.166	0.146	0.131	0.118	0.106	0.096	0.087	0.079	0.071
0.6	0.168	0.152	0.135	0.121	0.110	0.099	0.090	0.081	0.074	0.067
0.7	0.153	0.139	0.124	0.112	0.102	0.092	0.083	0.076	0.069	0.062
0.8	0.140	0.127	0.114	0.103	0.094	0.085	0.077	0.070	0.063	0.057
0.9	0.127	0.116	0.104	0.094	0.086	0.078	0.071	0.064	0.058	0.053
1.0	0.115	0.105	0.095	0.086	0.078	0.071	0.064	0.058	0.053	0.048

ordu = 7, ordv = 7, segu = 2, segv = 2										
$\frac{a_1}{a}$	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
$\frac{b_1}{a}$	*Coeff* β *in* $(M_u)_{max} = \beta P$									
0.1	0.184	0.177	0.168	0.156	0.142	0.128	0.114	0.102	0.091	0.082
0.2	0.179	0.173	0.164	0.152	0.139	0.125	0.112	0.100	0.089	0.081
0.3	0.172	0.167	0.158	0.147	0.134	0.121	0.108	0.097	0.087	0.078
0.4	0.163	0.158	0.150	0.139	0.128	0.115	0.103	0.092	0.083	0.075
0.5	0.152	0.148	0.140	0.131	0.120	0.109	0.098	0.088	0.079	0.071
0.6	0.141	0.137	0.130	0.121	0.112	0.101	0.091	0.082	0.074	0.067
0.7	0.129	0.125	0.119	0.112	0.103	0.094	0.085	0.076	0.069	0.062
0.8	0.118	0.115	0.109	0.102	0.095	0.086	0.078	0.070	0.064	0.057
0.9	0.107	0.104	0.100	0.093	0.086	0.079	0.071	0.064	0.058	0.053
1.0	0.097	0.095	0.090	0.085	0.078	0.072	0.065	0.059	0.053	0.048

ordu = 7, ordv = 7, segu = 4, segv = 4										
$\frac{a_1}{a}$	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
$\frac{b_1}{a}$	*Coeff* β *in* $(M_u)_{max} = \beta P$									
0.1	0.206	0.197	0.182	0.164	0.146	0.129	0.114	0.103	0.093	0.085
0.2	0.200	0.190	0.176	0.160	0.142	0.126	0.112	0.101	0.091	0.083
0.3	0.189	0.181	0.168	0.153	0.136	0.121	0.108	0.097	0.088	0.080
0.4	0.177	0.169	0.157	0.143	0.129	0.115	0.102	0.092	0.084	0.076
0.5	0.163	0.156	0.146	0.133	0.120	0.107	0.096	0.087	0.079	0.072
0.6	0.149	0.143	0.134	0.123	0.111	0.100	0.090	0.081	0.074	0.067
0.7	0.136	0.131	0.123	0.113	0.103	0.093	0.083	0.075	0.068	0.062
0.8	0.125	0.120	0.113	0.104	0.095	0.085	0.077	0.070	0.063	0.057
0.9	0.114	0.110	0.103	0.095	0.087	0.078	0.071	0.064	0.058	0.053
1.0	0.103	0.100	0.094	0.087	0.079	0.071	0.064	0.058	0.053	0.048

ordu = 7, ordv = 7, segu = 6, segv = 6										
$\frac{a_1}{a}$	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
$\frac{b_1}{a}$	*Coeff* β *in* $(M_u)_{max} = \beta P$									
0.1	0.227	0.212	0.191	0.168	0.146	0.129	0.116	0.105	0.095	0.086
0.2	0.217	0.204	0.184	0.162	0.142	0.125	0.113	0.102	0.092	0.084
0.3	0.203	0.191	0.173	0.154	0.135	0.120	0.108	0.098	0.089	0.080
0.4	0.186	0.176	0.161	0.143	0.127	0.113	0.102	0.093	0.084	0.076
0.5	0.170	0.161	0.148	0.133	0.119	0.106	0.096	0.087	0.079	0.071
0.6	0.155	0.147	0.136	0.123	0.110	0.099	0.089	0.081	0.074	0.067
0.7	0.142	0.135	0.125	0.113	0.102	0.092	0.083	0.075	0.069	0.062
0.8	0.130	0.124	0.115	0.104	0.094	0.085	0.077	0.070	0.063	0.057
0.9	0.119	0.113	0.105	0.095	0.086	0.078	0.070	0.064	0.058	0.053
1.0	0.108	0.103	0.095	0.087	0.078	0.071	0.064	0.058	0.053	0.048

ordu = 7, ordv = 7, segu = 8, segv = 8										
$\frac{a_1}{a}$	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
$\frac{b_1}{a}$	*Coeff* β *in* $(M_u)_{max} = \beta P$									
0.1	0.245	0.224	0.195	0.168	0.146	0.131	0.118	0.106	0.096	0.087
0.2	0.231	0.212	0.186	0.161	0.141	0.126	0.114	0.103	0.093	0.084
0.3	0.212	0.196	0.174	0.152	0.134	0.120	0.109	0.098	0.089	0.080
0.4	0.192	0.179	0.160	0.142	0.126	0.113	0.102	0.093	0.084	0.076
0.5	0.175	0.163	0.148	0.131	0.117	0.106	0.096	0.087	0.079	0.071
0.6	0.160	0.150	0.136	0.122	0.109	0.099	0.090	0.081	0.074	0.067
0.7	0.147	0.138	0.126	0.113	0.101	0.092	0.083	0.076	0.069	0.062
0.8	0.134	0.126	0.115	0.104	0.093	0.085	0.077	0.070	0.063	0.057
0.9	0.122	0.115	0.105	0.095	0.085	0.078	0.071	0.064	0.058	0.053
1.0	0.111	0.104	0.095	0.086	0.078	0.070	0.064	0.058	0.053	0.048

ordu = 7, ordv = 7, segu = 10, segv = 10										
$\frac{a}{L}$	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
$\frac{b}{L}$	*Coeff* β *in* $(M_u)_{max} = \beta P$									
0.1	0.259	0.230	0.196	0.168	0.148	0.133	0.119	0.107	0.097	0.087
0.2	0.241	0.216	0.186	0.160	0.142	0.128	0.114	0.103	0.093	0.084
0.3	0.217	0.197	0.172	0.151	0.134	0.121	0.108	0.098	0.089	0.080
0.4	0.196	0.180	0.159	0.140	0.126	0.113	0.102	0.092	0.084	0.076
0.5	0.179	0.165	0.147	0.131	0.117	0.106	0.096	0.087	0.079	0.071
0.6	0.164	0.151	0.136	0.121	0.109	0.099	0.090	0.081	0.074	0.067
0.7	0.150	0.139	0.125	0.112	0.101	0.092	0.083	0.076	0.069	0.062
0.8	0.136	0.127	0.115	0.103	0.093	0.085	0.077	0.070	0.063	0.057
0.9	0.124	0.116	0.105	0.094	0.085	0.078	0.071	0.064	0.058	0.053
1.0	0.113	0.105	0.095	0.086	0.078	0.071	0.064	0.058	0.053	0.048

ordu = 7, ordv = 7, segu = 12, segv = 12										
$\frac{a}{L}$	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
$\frac{b}{L}$	*Coeff* β *in* $(M_u)_{max} = \beta P$									
0.1	0.270	0.233	0.195	0.168	0.150	0.133	0.119	0.107	0.097	0.088
0.2	0.246	0.216	0.184	0.160	0.143	0.127	0.114	0.103	0.093	0.084
0.3	0.220	0.197	0.171	0.150	0.134	0.120	0.108	0.098	0.088	0.080
0.4	0.199	0.180	0.158	0.140	0.126	0.113	0.102	0.092	0.084	0.076
0.5	0.182	0.165	0.147	0.131	0.118	0.106	0.096	0.087	0.079	0.071
0.6	0.166	0.152	0.135	0.121	0.110	0.099	0.090	0.081	0.074	0.067
0.7	0.151	0.139	0.125	0.112	0.102	0.092	0.083	0.076	0.069	0.062
0.8	0.138	0.127	0.114	0.103	0.094	0.085	0.077	0.070	0.063	0.057
0.9	0.126	0.116	0.104	0.094	0.086	0.078	0.070	0.064	0.058	0.053
1.0	0.114	0.105	0.095	0.086	0.078	0.071	0.064	0.058	0.053	0.048

ordu = 8, ordv = 8, segu = 2, segv = 2										
$\frac{a}{L}$	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
$\frac{b}{L}$	*Coeff* β *in* $(M_u)_{max} = \beta P$									
0.1	0.217	0.205	0.187	0.167	0.147	0.129	0.115	0.104	0.094	0.085
0.2	0.209	0.197	0.181	0.162	0.143	0.126	0.112	0.101	0.092	0.083
0.3	0.197	0.186	0.171	0.154	0.136	0.121	0.108	0.097	0.088	0.080
0.4	0.182	0.173	0.160	0.144	0.129	0.114	0.102	0.092	0.084	0.076
0.5	0.167	0.159	0.148	0.134	0.120	0.107	0.096	0.087	0.079	0.072
0.6	0.152	0.146	0.135	0.123	0.111	0.099	0.089	0.081	0.074	0.067
0.7	0.139	0.133	0.124	0.114	0.103	0.092	0.083	0.075	0.068	0.062
0.8	0.127	0.122	0.114	0.104	0.094	0.085	0.077	0.070	0.063	0.057
0.9	0.117	0.112	0.104	0.096	0.086	0.078	0.070	0.064	0.058	0.053
1.0	0.106	0.101	0.095	0.087	0.079	0.071	0.064	0.058	0.053	0.048

ordu = 8, ordv = 8, segu = 4, segv = 4										
$\frac{a}{L}$	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
$\frac{b}{L}$	*Coeff* β *in* $(M_u)_{max} = \beta P$									
0.1	0.235	0.218	0.194	0.169	0.147	0.129	0.117	0.106	0.096	0.086
0.2	0.224	0.208	0.186	0.162	0.142	0.126	0.113	0.103	0.093	0.084
0.3	0.208	0.194	0.174	0.153	0.135	0.120	0.108	0.098	0.089	0.080
0.4	0.190	0.178	0.161	0.143	0.127	0.113	0.102	0.093	0.084	0.076
0.5	0.172	0.162	0.148	0.133	0.118	0.106	0.096	0.087	0.079	0.071
0.6	0.157	0.149	0.136	0.123	0.110	0.099	0.089	0.081	0.074	0.067
0.7	0.144	0.137	0.125	0.113	0.102	0.091	0.083	0.076	0.069	0.062
0.8	0.132	0.125	0.115	0.104	0.094	0.084	0.077	0.070	0.063	0.057
0.9	0.121	0.114	0.105	0.095	0.086	0.077	0.070	0.064	0.058	0.053
1.0	0.109	0.104	0.096	0.087	0.078	0.070	0.064	0.058	0.053	0.048

ordu = 8, ordv = 8, segu = 6, segv = 6										
$\frac{a_1}{a}$	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
$\frac{b_1}{a}$	*Coeff* β *in* $(M_u)_{max} = \beta P$									
0.1	0.252	0.228	0.197	0.168	0.147	0.131	0.119	0.107	0.096	0.087
0.2	0.236	0.215	0.187	0.161	0.141	0.127	0.115	0.103	0.093	0.084
0.3	0.215	0.197	0.174	0.152	0.134	0.120	0.109	0.098	0.089	0.080
0.4	0.195	0.180	0.160	0.141	0.125	0.113	0.102	0.092	0.084	0.076
0.5	0.177	0.164	0.148	0.131	0.117	0.106	0.096	0.087	0.079	0.071
0.6	0.162	0.151	0.136	0.122	0.109	0.099	0.090	0.081	0.074	0.067
0.7	0.149	0.139	0.126	0.113	0.101	0.092	0.083	0.076	0.069	0.062
0.8	0.136	0.127	0.115	0.103	0.093	0.085	0.077	0.070	0.063	0.057
0.9	0.123	0.116	0.105	0.095	0.085	0.078	0.071	0.064	0.058	0.053
1.0	0.112	0.105	0.095	0.086	0.078	0.070	0.064	0.058	0.053	0.048

ordu = 8, ordv = 8, segu = 8, segv = 8										
$\frac{a_1}{a}$	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
$\frac{b_1}{a}$	*Coeff* β *in* $(M_u)_{max} = \beta P$									
0.1	0.266	0.233	0.196	0.168	0.149	0.133	0.119	0.107	0.097	0.088
0.2	0.245	0.217	0.185	0.160	0.142	0.128	0.115	0.103	0.093	0.084
0.3	0.220	0.198	0.172	0.150	0.134	0.121	0.108	0.098	0.089	0.080
0.4	0.198	0.180	0.159	0.140	0.126	0.113	0.102	0.092	0.084	0.076
0.5	0.181	0.165	0.147	0.131	0.117	0.106	0.096	0.087	0.079	0.071
0.6	0.165	0.152	0.136	0.121	0.109	0.099	0.090	0.081	0.074	0.067
0.7	0.151	0.139	0.125	0.112	0.101	0.092	0.083	0.076	0.069	0.062
0.8	0.138	0.127	0.115	0.103	0.093	0.085	0.077	0.070	0.063	0.057
0.9	0.125	0.116	0.105	0.094	0.085	0.078	0.071	0.064	0.058	0.053
1.0	0.114	0.105	0.095	0.086	0.078	0.071	0.064	0.058	0.053	0.048

ordu = 8, ordv = 8, segu = 10, segv = 10										
$\frac{a_1}{a}$	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
$\frac{b_1}{a}$	*Coeff* β *in* $(M_u)_{max} = \beta P$									
0.1	0.276	0.234	0.195	0.169	0.150	0.134	0.119	0.108	0.097	0.088
0.2	0.250	0.216	0.183	0.160	0.143	0.128	0.114	0.103	0.093	0.084
0.3	0.222	0.197	0.170	0.150	0.134	0.120	0.108	0.098	0.088	0.080
0.4	0.201	0.180	0.158	0.140	0.126	0.113	0.102	0.092	0.084	0.076
0.5	0.183	0.165	0.146	0.131	0.118	0.106	0.096	0.087	0.079	0.071
0.6	0.167	0.152	0.135	0.121	0.110	0.099	0.090	0.081	0.074	0.067
0.7	0.152	0.139	0.124	0.112	0.102	0.092	0.083	0.076	0.069	0.062
0.8	0.139	0.127	0.114	0.103	0.094	0.085	0.077	0.070	0.063	0.057
0.9	0.127	0.116	0.104	0.094	0.086	0.078	0.070	0.064	0.058	0.053
1.0	0.115	0.105	0.095	0.086	0.078	0.071	0.064	0.058	0.053	0.048

ordu = 8, ordv = 8, segu = 12, segv = 12										
$\frac{a_1}{a}$	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
$\frac{b_1}{a}$	*Coeff* β *in* $(M_u)_{max} = \beta P$									
0.1	0.283	0.233	0.195	0.171	0.151	0.133	0.120	0.108	0.097	0.088
0.2	0.252	0.214	0.183	0.161	0.143	0.127	0.114	0.103	0.093	0.084
0.3	0.224	0.195	0.170	0.151	0.134	0.120	0.108	0.098	0.088	0.080
0.4	0.203	0.179	0.158	0.141	0.126	0.113	0.102	0.092	0.084	0.076
0.5	0.185	0.165	0.146	0.131	0.118	0.106	0.096	0.087	0.079	0.071
0.6	0.168	0.151	0.135	0.122	0.110	0.099	0.090	0.081	0.074	0.067
0.7	0.153	0.139	0.124	0.112	0.102	0.092	0.083	0.076	0.069	0.062
0.8	0.140	0.127	0.114	0.103	0.094	0.085	0.077	0.070	0.063	0.057
0.9	0.127	0.116	0.104	0.094	0.086	0.078	0.071	0.064	0.058	0.053
1.0	0.115	0.105	0.095	0.086	0.078	0.071	0.064	0.058	0.053	0.048

A.4 Uniformly loaded rectangular plate with two edges simply supported and two edges clamped

Coefficients $\alpha, \beta, \beta_1, \beta_2$ for uniformly loaded (load F_1) rectangular plate with two edges simply supported ($u = 0, u = a$) and two edges clamped ($v = 0, v = b$)

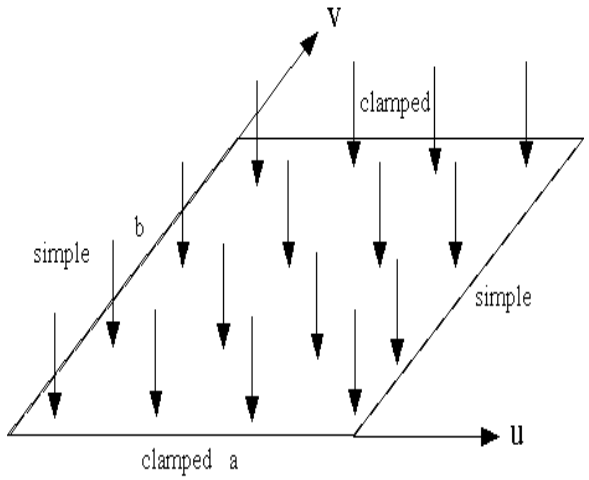


Figure A.4: Uniformly loaded rectangular plate with two edges simply supported and two edges clamped

$\frac{b}{a}$	$u = \frac{a}{2}, v = 0$ $x_{max} = \alpha \frac{F_1 a^4}{Eh^3}$	$u = \frac{a}{2}, v = \frac{b}{2}$ $M_v = \beta F_1 a^2$	$u = \frac{a}{2}, v = 0$ $M_u = \beta_1 F_1 a^2$	$u = \frac{a}{2}, v = 0$ $M_v = \beta_2 F_1 a^2$
1.0000	0.0209	-0.070	0.024	0.033
1.2000	0.0340	-0.087	0.038	0.040
1.4000	0.0502	-0.100	0.052	0.045
1.6000	0.0658	-0.109	0.065	0.047
1.8000	0.0799	-0.115	0.077	0.048
2.0000	0.0987	-0.119	0.087	0.047
3.0000	0.1276	-0.125	0.114	0.042

Table A.4: Numerical results from [85], rectangular plate, two simple edges, two clamped, uniform load

ordu = 4, ordv = 4, segu = 2, segv = 2				
1.0000	0.0212	-0.0466	0.0328	0.0523
1.2000	0.0353	-0.0647	0.0431	0.0424
1.4000	0.0505	-0.0809	0.0596	0.0474
1.6000	0.0660	-0.0917	0.0749	0.0495
1.8000	0.0802	-0.0991	0.0890	0.0516
2.0000	0.0916	-0.0821	0.0987	0.0466
3.0000	0.1276	-0.0859	0.1331	0.0483

ordu = 4, ordv = 4, segu = 4, segv = 4				
1.0000	0.0209	-0.0611	0.0246	0.0355
1.2000	0.0348	-0.0774	0.0375	0.0393
1.4000	0.0504	-0.0900	0.0527	0.0456
1.6000	0.0657	-0.0990	0.0664	0.0472
1.8000	0.0799	-0.1050	0.0790	0.0484
2.0000	0.0922	-0.1065	0.0896	0.0481
3.0000	0.1276	-0.1116	0.1188	0.0433

ordu = 4, ordv = 4, segu = 6, segv = 6				
1.0000	0.0209	-0.0657	0.0246	0.0344
1.2000	0.0349	-0.0818	0.0377	0.0397
1.4000	0.0504	-0.0941	0.0522	0.0451
1.6000	0.0658	-0.1027	0.0657	0.0470
1.8000	0.0799	-0.1085	0.0779	0.0480
2.0000	0.0922	-0.1131	0.0881	0.0477
3.0000	0.1276	-0.1184	0.1163	0.0426

ordu = 4, ordv = 4, segu = 8, segv = 8				
1.0000	0.0209	-0.0675	0.0245	0.0339
1.2000	0.0349	-0.0837	0.0377	0.0399
1.4000	0.0504	-0.0960	0.0520	0.0449
1.6000	0.0658	-0.1047	0.0654	0.0469
1.8000	0.0799	-0.1104	0.0774	0.0478
2.0000	0.0922	-0.1156	0.0876	0.0476
3.0000	0.1276	-0.1210	0.1155	0.0424

ordu = 4, ordv = 4, segu = 10, segv = 10				
1.0000	0.0209	-0.0683	0.0245	0.0337
1.2000	0.0349	-0.0847	0.0377	0.0399
1.4000	0.0504	-0.0972	0.0519	0.0448
1.6000	0.0658	-0.1059	0.0652	0.0469
1.8000	0.0799	-0.1116	0.0772	0.0478
2.0000	0.0922	-0.1168	0.0873	0.0475
3.0000	0.1276	-0.1223	0.1151	0.0423

ordu = 4, ordv = 4, segu = 12, segv = 12				
1.0000	0.0209	-0.0688	0.0245	0.0335
1.2000	0.0349	-0.0853	0.0377	0.0400
1.4000	0.0504	-0.0978	0.0518	0.0447
1.6000	0.0658	-0.1066	0.0652	0.0469
1.8000	0.0799	-0.1124	0.0771	0.0477
2.0000	0.0922	-0.1175	0.0872	0.0474
3.0000	0.1276	-0.1230	0.1149	0.0423

ordu = 5, ordv = 5, segu = 2, segv = 2				
1.0000	0.0215	-0.0629	0.0271	0.0367
1.2000	0.0350	-0.0834	0.0391	0.0397
1.4000	0.0506	-0.0976	0.0534	0.0454
1.6000	0.0659	-0.1074	0.0662	0.0470
1.8000	0.0801	-0.1139	0.0779	0.0480
2.0000	0.0921	-0.1094	0.0877	0.0475
3.0000	0.1277	-0.1148	0.1147	0.0424

ordu = 5, ordv = 5, segu = 4, segv = 4				
1.0000	0.0209	-0.0691	0.0246	0.0336
1.2000	0.0349	-0.0859	0.0378	0.0400
1.4000	0.0504	-0.0989	0.0518	0.0446
1.6000	0.0658	-0.1081	0.0651	0.0469
1.8000	0.0799	-0.1142	0.0770	0.0477
2.0000	0.0922	-0.1178	0.0870	0.0474
3.0000	0.1276	-0.1232	0.1144	0.0421

ordu = 5, ordv = 5, segu = 6, segv = 6				
1.0000	0.0209	-0.0697	0.0244	0.0333
1.2000	0.0349	-0.0865	0.0377	0.0401
1.4000	0.0504	-0.0995	0.0517	0.0446
1.6000	0.0658	-0.1087	0.0650	0.0469
1.8000	0.0799	-0.1148	0.0768	0.0476
2.0000	0.0922	-0.1188	0.0869	0.0474
3.0000	0.1276	-0.1243	0.1144	0.0421

ordu = 5, ordv = 5, segu = 8, segv = 8				
1.0000	0.0209	-0.0698	0.0244	0.0333
1.2000	0.0349	-0.0867	0.0377	0.0401
1.4000	0.0504	-0.0996	0.0517	0.0446
1.6000	0.0658	-0.1088	0.0650	0.0469
1.8000	0.0799	-0.1149	0.0768	0.0476
2.0000	0.0922	-0.1189	0.0869	0.0474
3.0000	0.1276	-0.1245	0.1144	0.0421

ordu = 5, ordv = 5, segu = 10, segv = 10				
1.0000	0.0209	-0.0698	0.0244	0.0333
1.2000	0.0349	-0.0867	0.0377	0.0401
1.4000	0.0504	-0.0997	0.0517	0.0445
1.6000	0.0658	-0.1089	0.0650	0.0469
1.8000	0.0799	-0.1150	0.0768	0.0476
2.0000	0.0922	-0.1190	0.0869	0.0474
3.0000	0.1276	-0.1245	0.1144	0.0421

ordu = 5, ordv = 5, segu = 12, segv = 12				
1.0000	0.0209	-0.0698	0.0244	0.0332
1.2000	0.0349	-0.0867	0.0377	0.0401
1.4000	0.0504	-0.0997	0.0517	0.0445
1.6000	0.0658	-0.1090	0.0650	0.0469
1.8000	0.0799	-0.1151	0.0768	0.0476
2.0000	0.0922	-0.1190	0.0869	0.0474
3.0000	0.1276	-0.1246	0.1144	0.0421

ordu = 6, ordv = 6, segu = 2, segv = 2				
1.0000	0.0209	-0.0689	0.0240	0.0327
1.2000	0.0349	-0.0866	0.0376	0.0404
1.4000	0.0504	-0.0995	0.0513	0.0443
1.6000	0.0657	-0.1089	0.0648	0.0468
1.8000	0.0799	-0.1150	0.0767	0.0476
2.0000	0.0922	-0.1176	0.0866	0.0472
3.0000	0.1276	-0.1232	0.1144	0.0422

ordu = 6, ordv = 6, segu = 4, segv = 4				
1.0000	0.0209	-0.0699	0.0243	0.0331
1.2000	0.0349	-0.0869	0.0377	0.0401
1.4000	0.0504	-0.0999	0.0516	0.0445
1.6000	0.0658	-0.1091	0.0650	0.0469
1.8000	0.0799	-0.1152	0.0768	0.0476
2.0000	0.0922	-0.1191	0.0868	0.0473
3.0000	0.1276	-0.1247	0.1143	0.0421

ordu = 6, ordv = 6, segu = 6, segv = 6				
1.0000	0.0209	-0.0698	0.0244	0.0332
1.2000	0.0349	-0.0868	0.0377	0.0401
1.4000	0.0504	-0.0998	0.0517	0.0445
1.6000	0.0658	-0.1090	0.0650	0.0469
1.8000	0.0799	-0.1152	0.0768	0.0476
2.0000	0.0922	-0.1191	0.0869	0.0474
3.0000	0.1276	-0.1246	0.1144	0.0421

ordu = 6, ordv = 6, segu = 8, segv = 8				
1.0000	0.0209	-0.0698	0.0244	0.0332
1.2000	0.0349	-0.0868	0.0377	0.0401
1.4000	0.0504	-0.0998	0.0517	0.0445
1.6000	0.0658	-0.1090	0.0650	0.0469
1.8000	0.0799	-0.1152	0.0768	0.0476
2.0000	0.0922	-0.1191	0.0869	0.0474
3.0000	0.1276	-0.1246	0.1144	0.0421

ordu = 6, ordv = 6, segu = 10, segv = 10				
1.0000	0.0209	-0.0698	0.0244	0.0332
1.2000	0.0349	-0.0868	0.0377	0.0401
1.4000	0.0504	-0.0998	0.0517	0.0445
1.6000	0.0658	-0.1090	0.0650	0.0469
1.8000	0.0799	-0.1152	0.0768	0.0476
2.0000	0.0922	-0.1191	0.0869	0.0474
3.0000	0.1276	-0.1246	0.1144	0.0421

ordu = 6, ordv = 6, segu = 12, segv = 12				
1.0000	0.0209	-0.0698	0.0244	0.0332
1.2000	0.0349	-0.0868	0.0377	0.0401
1.4000	0.0504	-0.0998	0.0517	0.0445
1.6000	0.0658	-0.1090	0.0650	0.0469
1.8000	0.0799	-0.1152	0.0768	0.0476
2.0000	0.0922	-0.1191	0.0869	0.0474
3.0000	0.1276	-0.1246	0.1144	0.0421

ordu = 7, ordv = 7, segu = 2, segv = 2				
1.0000	0.0209	-0.0699	0.0244	0.0333
1.2000	0.0349	-0.0867	0.0377	0.0401
1.4000	0.0504	-0.0998	0.0517	0.0445
1.6000	0.0658	-0.1089	0.0650	0.0469
1.8000	0.0799	-0.1150	0.0768	0.0476
2.0000	0.0922	-0.1194	0.0868	0.0472
3.0000	0.1276	-0.1249	0.1144	0.0422

ordu = 7, ordv = 7, segu = 4, segv = 4				
1.0000	0.0209	-0.0699	0.0244	0.0332
1.2000	0.0349	-0.0868	0.0377	0.0401
1.4000	0.0504	-0.0999	0.0517	0.0445
1.6000	0.0658	-0.1091	0.0650	0.0469
1.8000	0.0799	-0.1152	0.0768	0.0476
2.0000	0.0922	-0.1191	0.0869	0.0474
3.0000	0.1276	-0.1247	0.1144	0.0421

ordu = 7, ordv = 7, segu = 6, segv = 6				
1.0000	0.0209	-0.0698	0.0244	0.0332
1.2000	0.0349	-0.0868	0.0377	0.0401
1.4000	0.0504	-0.0998	0.0517	0.0445
1.6000	0.0658	-0.1090	0.0650	0.0469
1.8000	0.0799	-0.1152	0.0768	0.0476
2.0000	0.0922	-0.1191	0.0869	0.0474
3.0000	0.1276	-0.1246	0.1144	0.0421

ordu = 7, ordv = 7, segu = 8, segv = 8				
1.0000	0.0209	-0.0698	0.0244	0.0332
1.2000	0.0349	-0.0868	0.0377	0.0401
1.4000	0.0504	-0.0998	0.0517	0.0445
1.6000	0.0658	-0.1090	0.0650	0.0469
1.8000	0.0799	-0.1152	0.0768	0.0476
2.0000	0.0922	-0.1191	0.0869	0.0474
3.0000	0.1276	-0.1246	0.1144	0.0421

ordu = 7, ordv = 7, segu = 10, segv = 10				
1.0000	0.0209	-0.0698	0.0244	0.0332
1.2000	0.0349	-0.0868	0.0377	0.0401
1.4000	0.0504	-0.0998	0.0517	0.0445
1.6000	0.0658	-0.1090	0.0650	0.0469
1.8000	0.0799	-0.1152	0.0768	0.0476
2.0000	0.0922	-0.1191	0.0869	0.0474
3.0000	0.1276	-0.1246	0.1144	0.0421

ordu = 7, ordv = 7, segu = 12, segv = 12				
1.0000	0.0209	-0.0698	0.0244	0.0332
1.2000	0.0349	-0.0868	0.0377	0.0401
1.4000	0.0504	-0.0998	0.0517	0.0445
1.6000	0.0658	-0.1090	0.0650	0.0469
1.8000	0.0799	-0.1152	0.0768	0.0476
2.0000	0.0922	-0.1191	0.0869	0.0474
3.0000	0.1276	-0.1246	0.1144	0.0421

ordu = 8, ordv = 8, segu = 2, segv = 2				
1.0000	0.0209	-0.0699	0.0244	0.0332
1.2000	0.0349	-0.0869	0.0377	0.0401
1.4000	0.0504	-0.0999	0.0517	0.0445
1.6000	0.0658	-0.1091	0.0650	0.0469
1.8000	0.0799	-0.1152	0.0768	0.0476
2.0000	0.0922	-0.1193	0.0869	0.0474
3.0000	0.1276	-0.1249	0.1143	0.0421

ordu = 8, ordv = 8, segu = 4, segv = 4				
1.0000	0.0209	-0.0698	0.0244	0.0332
1.2000	0.0349	-0.0868	0.0377	0.0401
1.4000	0.0504	-0.0998	0.0517	0.0445
1.6000	0.0658	-0.1090	0.0650	0.0469
1.8000	0.0799	-0.1152	0.0768	0.0476
2.0000	0.0922	-0.1191	0.0869	0.0474
3.0000	0.1276	-0.1246	0.1144	0.0421

ordu = 8, ordv = 8, segu = 6, segv = 6				
1.0000	0.0209	-0.0698	0.0244	0.0332
1.2000	0.0349	-0.0868	0.0377	0.0401
1.4000	0.0504	-0.0998	0.0517	0.0445
1.6000	0.0658	-0.1090	0.0650	0.0469
1.8000	0.0799	-0.1152	0.0768	0.0476
2.0000	0.0922	-0.1191	0.0869	0.0474
3.0000	0.1276	-0.1246	0.1144	0.0421

ordu = 8, ordv = 8, segu = 8, segv = 8				
1.0000	0.0209	-0.0698	0.0244	0.0332
1.2000	0.0349	-0.0868	0.0377	0.0401
1.4000	0.0504	-0.0998	0.0517	0.0445
1.6000	0.0658	-0.1090	0.0650	0.0469
1.8000	0.0799	-0.1152	0.0768	0.0476
2.0000	0.0922	-0.1191	0.0869	0.0474
3.0000	0.1276	-0.1246	0.1144	0.0421

A.5 Uniformly loaded rectangular plate with two opposite edges simply supported, the third edge free and the fourth clamped

Bending moments along the sides and maximum deflection for a uniformly loaded rectangular plate with two opposite edges simply supported, one free and one clamped.

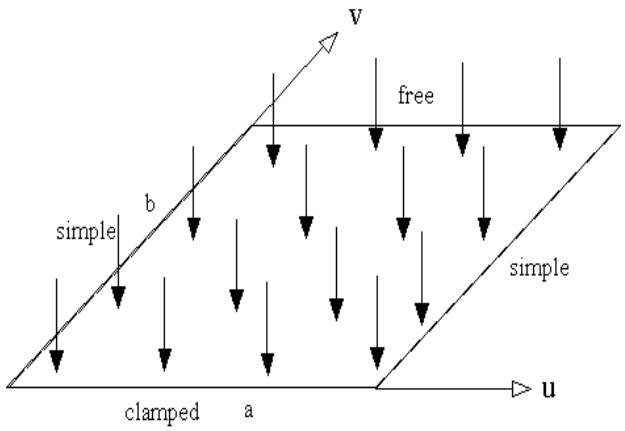


Figure A.5: Uniformly loaded rectangular plate with two edges simply supported ($u = 0, u = a$), the third edge free ($v = b$) and the fourth clamped ($v = 0$)

$\frac{b}{a}$	$(x)_{max}$	$(M_u)_{u=a/2,v=b}$	$(M_v)_{u=a/2,v=0}$
0.3333	1.03	0.0078	-0.428
0.5000	0.635	0.0293	-0.319
0.6667	0.366	0.0558	-0.227
1.0000	0.123	0.0972	-0.119
1.5000	0.154	0.123	-0.124
2.0000	0.164	0.131	-0.125
3.0000	0.166	0.133	-0.125

Table A.5: Numerical results from [85], rectangular plate, two simple edges, one free, one clamped, uniform load

ordu = 3, ordv = 3, segu = 2, segv = 2			
0.3333	0.9566	0.0080	-0.2268
0.5000	0.5692	0.0258	-0.1379
0.6667	0.3180	0.0467	-0.0794
1.0000	0.1022	0.0774	-0.0277
1.5000	0.1257	0.0873	-0.0482
2.0000	0.1299	0.0915	-0.0304
3.0000	0.1312	0.0920	-0.0305

ordu = 3, ordv = 3, segu = 4, segv = 4			
0.3333	1.0153	0.0091	-0.3136
0.5000	0.6224	0.0309	-0.2123
0.6667	0.3554	0.0570	-0.1361
1.0000	0.1176	0.0963	-0.0576
1.5000	0.1473	0.1179	-0.0665
2.0000	0.1549	0.1240	-0.0613
3.0000	0.1572	0.1255	-0.0615

ordu = 3, ordv = 3, segu = 6, segv = 6			
0.3333	1.0209	0.0085	-0.3475
0.5000	0.6300	0.0306	-0.2432
0.6667	0.3615	0.0575	-0.1613
1.0000	0.1204	0.0985	-0.0729
1.5000	0.1513	0.1227	-0.0765
2.0000	0.1596	0.1290	-0.0772
3.0000	0.1621	0.1309	-0.0774

ordu = 3, ordv = 3, segu = 8, segv = 8			
0.3333	1.0232	0.0083	-0.3660
0.5000	0.6326	0.0303	-0.2602
0.6667	0.3636	0.0574	-0.1756
1.0000	0.1214	0.0989	-0.0821
1.5000	0.1527	0.1241	-0.0834
2.0000	0.1612	0.1305	-0.0868
3.0000	0.1638	0.1325	-0.0870

ordu = 3, ordv = 3, segu = 10, segv = 10			
0.3333	1.0242	0.0082	-0.3776
0.5000	0.6337	0.0302	-0.2710
0.6667	0.3646	0.0572	-0.1848
1.0000	0.1219	0.0989	-0.0882
1.5000	0.1533	0.1245	-0.0886
2.0000	0.1619	0.1310	-0.0932
3.0000	0.1646	0.1331	-0.0934

ordu = 3, ordv = 3, segu = 12, segv = 12			
0.3333	1.0249	0.0081	-0.3855
0.5000	0.6344	0.0300	-0.2785
0.6667	0.3651	0.0571	-0.1912
1.0000	0.1221	0.0988	-0.0926
1.5000	0.1537	0.1247	-0.0925
2.0000	0.1623	0.1312	-0.0977
3.0000	0.1650	0.1334	-0.0980

ordu = 4, ordv = 4, segu = 2, segv = 2			
0.3333	1.0381	0.0103	-0.4018
0.5000	0.6401	0.0367	-0.2832
0.6667	0.3673	0.0684	-0.1875
1.0000	0.1224	0.1165	-0.0822
1.5000	0.1547	0.1473	-0.1066
2.0000	0.1633	0.1571	-0.0860
3.0000	0.1661	0.1599	-0.0861

ordu = 4, ordv = 4, segu = 4, segv = 4			
0.3333	1.0254	0.0071	-0.4191
0.5000	0.6359	0.0297	-0.3082
0.6667	0.3663	0.0580	-0.2148
1.0000	0.1227	0.1020	-0.1058
1.5000	0.1545	0.1294	-0.1139
2.0000	0.1632	0.1372	-0.1116
3.0000	0.1660	0.1396	-0.1119

ordu = 4, ordv = 4, segu = 6, segv = 6			
0.3333	1.0262	0.0077	-0.4246
0.5000	0.6359	0.0296	-0.3143
0.6667	0.3663	0.0569	-0.2214
1.0000	0.1227	0.0994	-0.1125
1.5000	0.1545	0.1262	-0.1176
2.0000	0.1632	0.1335	-0.1185
3.0000	0.1660	0.1359	-0.1188

ordu = 4, ordv = 4, segu = 8, segv = 8			
0.3333	1.0262	0.0077	-0.4259
0.5000	0.6358	0.0294	-0.3163
0.6667	0.3663	0.0564	-0.2238
1.0000	0.1227	0.0984	-0.1150
1.5000	0.1545	0.1250	-0.1195
2.0000	0.1632	0.1322	-0.1211
3.0000	0.1660	0.1345	-0.1214

ordu = 4, ordv = 4, segu = 10, segv = 10			
0.3333	1.0263	0.0078	-0.4267
0.5000	0.6358	0.0294	-0.3173
0.6667	0.3663	0.0562	-0.2249
1.0000	0.1227	0.0980	-0.1162
1.5000	0.1545	0.1244	-0.1207
2.0000	0.1632	0.1316	-0.1223
3.0000	0.1660	0.1339	-0.1226

ordu = 4, ordv = 4, segu = 12, segv = 12			
0.3333	1.0263	0.0078	-0.4271
0.5000	0.6358	0.0293	-0.3178
0.6667	0.3663	0.0561	-0.2255
1.0000	0.1227	0.0977	-0.1168
1.5000	0.1545	0.1241	-0.1214
2.0000	0.1632	0.1313	-0.1230
3.0000	0.1660	0.1336	-0.1233

ordu = 5, ordv = 5, segu = 2, segv = 2			
0.3333	1.0414	0.0092	-0.4346
0.5000	0.6398	0.0311	-0.3193
0.6667	0.3674	0.0575	-0.2227
1.0000	0.1228	0.0989	-0.1092
1.5000	0.1545	0.1236	-0.1223
2.0000	0.1632	0.1310	-0.1149
3.0000	0.1660	0.1332	-0.1152

ordu = 5, ordv = 5, segu = 4, segv = 4			
0.3333	1.0260	0.0079	-0.4267
0.5000	0.6359	0.0294	-0.3181
0.6667	0.3663	0.0560	-0.2260
1.0000	0.1227	0.0974	-0.1171
1.5000	0.1545	0.1234	-0.1228
2.0000	0.1632	0.1306	-0.1233
3.0000	0.1660	0.1329	-0.1236

ordu = 5, ordv = 5, segu = 6, segv = 6			
0.3333	1.0262	0.0078	-0.4282
0.5000	0.6358	0.0293	-0.3190
0.6667	0.3663	0.0559	-0.2268
1.0000	0.1227	0.0972	-0.1181
1.5000	0.1545	0.1234	-0.1234
2.0000	0.1632	0.1305	-0.1243
3.0000	0.1660	0.1328	-0.1246

ordu = 5, ordv = 5, segu = 8, segv = 8			
0.3333	1.0263	0.0078	-0.4278
0.5000	0.6358	0.0293	-0.3189
0.6667	0.3663	0.0559	-0.2268
1.0000	0.1227	0.0972	-0.1183
1.5000	0.1545	0.1233	-0.1235
2.0000	0.1632	0.1305	-0.1245
3.0000	0.1660	0.1328	-0.1248

ordu = 5, ordv = 5, segu = 10, segv = 10			
0.3333	1.0263	0.0078	-0.4280
0.5000	0.6358	0.0293	-0.3190
0.6667	0.3663	0.0559	-0.2269
1.0000	0.1227	0.0972	-0.1183
1.5000	0.1545	0.1233	-0.1236
2.0000	0.1632	0.1305	-0.1246
3.0000	0.1660	0.1328	-0.1249

ordu = 5, ordv = 5, segu = 12, segv = 12			
0.3333	1.0263	0.0078	-0.4279
0.5000	0.6358	0.0293	-0.3190
0.6667	0.3663	0.0559	-0.2269
1.0000	0.1227	0.0972	-0.1184
1.5000	0.1545	0.1233	-0.1237
2.0000	0.1632	0.1305	-0.1246
3.0000	0.1660	0.1328	-0.1249

ordu = 6, ordv = 6, segu = 2, segv = 2			
0.3333	1.0245	0.0074	-0.4261
0.5000	0.6356	0.0290	-0.3181
0.6667	0.3662	0.0557	-0.2261
1.0000	0.1227	0.0972	-0.1169
1.5000	0.1545	0.1233	-0.1235
2.0000	0.1632	0.1306	-0.1233
3.0000	0.1660	0.1329	-0.1236

ordu = 6, ordv = 6, segu = 4, segv = 4			
0.3333	1.0262	0.0078	-0.4284
0.5000	0.6358	0.0292	-0.3192
0.6667	0.3663	0.0558	-0.2271
1.0000	0.1227	0.0972	-0.1185
1.5000	0.1545	0.1233	-0.1238
2.0000	0.1632	0.1305	-0.1247
3.0000	0.1660	0.1328	-0.1250

ordu = 6, ordv = 6, segu = 6, segv = 6			
0.3333	1.0263	0.0078	-0.4278
0.5000	0.6358	0.0293	-0.3189
0.6667	0.3663	0.0558	-0.2269
1.0000	0.1227	0.0972	-0.1184
1.5000	0.1545	0.1233	-0.1237
2.0000	0.1632	0.1305	-0.1247
3.0000	0.1660	0.1328	-0.1250

ordu = 6, ordv = 6, segu = 8, segv = 8			
0.3333	1.0263	0.0078	-0.4280
0.5000	0.6358	0.0293	-0.3190
0.6667	0.3663	0.0559	-0.2269
1.0000	0.1227	0.0972	-0.1184
1.5000	0.1545	0.1233	-0.1237
2.0000	0.1632	0.1305	-0.1247
3.0000	0.1660	0.1328	-0.1250

ordu = 6, ordv = 6, segu = 10, segv = 10			
0.3333	1.0263	0.0078	-0.4279
0.5000	0.6358	0.0293	-0.3190
0.6667	0.3663	0.0559	-0.2269
1.0000	0.1227	0.0972	-0.1184
1.5000	0.1545	0.1233	-0.1237
2.0000	0.1632	0.1305	-0.1247
3.0000	0.1660	0.1328	-0.1250

ordu = 6, ordv = 6, segu = 12, segv = 12			
0.3333	1.0263	0.0078	-0.4280
0.5000	0.6358	0.0293	-0.3190
0.6667	0.3663	0.0559	-0.2269
1.0000	0.1227	0.0972	-0.1184
1.5000	0.1545	0.1233	-0.1237
2.0000	0.1632	0.1305	-0.1247
3.0000	0.1660	0.1328	-0.1250

ordu = 7, ordv = 7, segu = 2, segv = 2			
0.3333	1.0255	0.0077	-0.4266
0.5000	0.6358	0.0293	-0.3184
0.6667	0.3663	0.0559	-0.2266
1.0000	0.1227	0.0972	-0.1183
1.5000	0.1545	0.1233	-0.1236
2.0000	0.1632	0.1305	-0.1249
3.0000	0.1660	0.1328	-0.1252

ordu = 7, ordv = 7, segu = 4, segv = 4			
0.3333	1.0262	0.0078	-0.4282
0.5000	0.6358	0.0293	-0.3191
0.6667	0.3663	0.0559	-0.2270
1.0000	0.1227	0.0972	-0.1185
1.5000	0.1545	0.1233	-0.1238
2.0000	0.1632	0.1305	-0.1247
3.0000	0.1660	0.1328	-0.1250

ordu = 7, ordv = 7, segu = 6, segv = 6			
0.3333	1.0263	0.0078	-0.4279
0.5000	0.6358	0.0293	-0.3189
0.6667	0.3663	0.0559	-0.2269
1.0000	0.1227	0.0972	-0.1184
1.5000	0.1545	0.1233	-0.1237
2.0000	0.1632	0.1305	-0.1247
3.0000	0.1660	0.1328	-0.1250

ordu = 7, ordv = 7, segu = 8, segv = 8			
0.3333	1.0263	0.0078	-0.4280
0.5000	0.6358	0.0293	-0.3190
0.6667	0.3663	0.0559	-0.2269
1.0000	0.1227	0.0972	-0.1184
1.5000	0.1545	0.1233	-0.1237
2.0000	0.1632	0.1305	-0.1247
3.0000	0.1660	0.1328	-0.1250

ordu = 7, ordv = 7, segu = 10, segv = 10			
0.3333	1.0263	0.0078	-0.4279
0.5000	0.6358	0.0293	-0.3190
0.6667	0.3663	0.0559	-0.2269
1.0000	0.1227	0.0972	-0.1184
1.5000	0.1545	0.1233	-0.1237
2.0000	0.1632	0.1305	-0.1247
3.0000	0.1660	0.1328	-0.1250

ordu = 7, ordv = 7, segu = 12, segv = 12			
0.3333	1.0263	0.0078	-0.4279
0.5000	0.6358	0.0293	-0.3190
0.6667	0.3663	0.0559	-0.2269
1.0000	0.1227	0.0972	-0.1184
1.5000	0.1545	0.1233	-0.1237
2.0000	0.1632	0.1305	-0.1247
3.0000	0.1660	0.1328	-0.1250

ordu = 8, ordv = 8, segu = 2, segv = 2			
0.3333	1.0263	0.0078	-0.4284
0.5000	0.6358	0.0293	-0.3192
0.6667	0.3663	0.0558	-0.2271
1.0000	0.1227	0.0972	-0.1185
1.5000	0.1545	0.1233	-0.1238
2.0000	0.1632	0.1305	-0.1249
3.0000	0.1660	0.1328	-0.1252

ordu = 8, ordv = 8, segu = 4, segv = 4			
0.3333	1.0263	0.0078	-0.4278
0.5000	0.6358	0.0293	-0.3189
0.6667	0.3663	0.0559	-0.2269
1.0000	0.1227	0.0972	-0.1184
1.5000	0.1545	0.1233	-0.1237
2.0000	0.1632	0.1305	-0.1247
3.0000	0.1660	0.1328	-0.1250

ordu = 8, ordv = 8, segu = 6, segv = 6			
0.3333	1.0263	0.0078	-0.4280
0.5000	0.6358	0.0293	-0.3190
0.6667	0.3663	0.0559	-0.2269
1.0000	0.1227	0.0972	-0.1184
1.5000	0.1545	0.1233	-0.1237
2.0000	0.1632	0.1305	-0.1247
3.0000	0.1660	0.1328	-0.1250

ordu = 8, ordv = 8, segu = 8, segv = 8			
0.3333	1.0263	0.0078	-0.4279
0.5000	0.6358	0.0293	-0.3190
0.6667	0.3663	0.0559	-0.2269
1.0000	0.1227	0.0972	-0.1184
1.5000	0.1545	0.1233	-0.1237
2.0000	0.1632	0.1305	-0.1247
3.0000	0.1660	0.1328	-0.1250

ordu = 8, ordv = 8, segu = 10, segv = 10			
0.3333	1.0263	0.0078	-0.4280
0.5000	0.6358	0.0293	-0.3190
0.6667	0.3663	0.0559	-0.2269
1.0000	0.1227	0.0972	-0.1184
1.5000	0.1545	0.1233	-0.1237
2.0000	0.1632	0.1305	-0.1247
3.0000	0.1660	0.1328	-0.1250

ordu = 8, ordv = 8, segu = 12, segv = 12			
0.3333	1.0263	0.0078	-0.4279
0.5000	0.6358	0.0293	-0.3190
0.6667	0.3663	0.0559	-0.2269
1.0000	0.1227	0.0972	-0.1184
1.5000	0.1545	0.1233	-0.1237
2.0000	0.1632	0.1305	-0.1247
3.0000	0.1660	0.1328	-0.1250

A.6 Uniformly loaded rectangular plate with three edges simply supported and the fourth edge free

Bending moments and maximum deflection for uniformly loaded rectangular plate with three edges simply supported and one free.

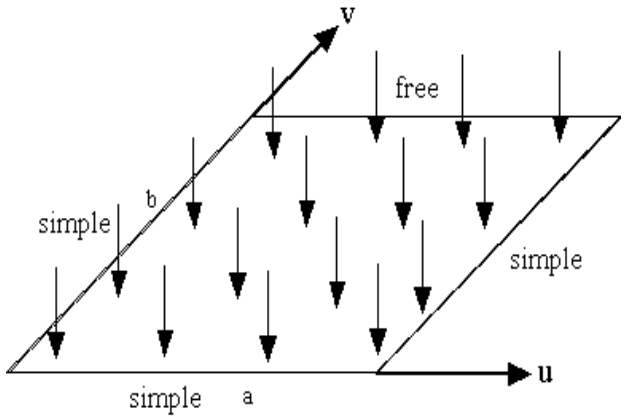


Figure A.6: Uniformly loaded rectangular plate with three edges simply supported and the fourth edge free

	$(u = a/2, v = b/2)$	$(u = a/2, v = b)$	$u = a/2, v = b/2$	
$\frac{b}{a}$	$(x_{max})_{a/2,b/2}$	$(M_u)_{max}$	M_u	M_v
0.5000	0.0775	0.060	0.039	0.022
0.7143	0.1117	0.088	0.059	0.032
0.8333	0.1265	0.100	0.069	0.036
1.0000	0.1404	0.112	0.080	0.039
1.2000	0.1511	0.121	0.090	0.041
1.4000	0.1575	0.126	0.098	0.042
2.0000	0.1646	0.132	0.113	0.041

Table A.6: Numerical results from [85], rectangular plate, three simple edges, one free, uniform load

ordu = 3, ordv = 3, segu = 2, segv = 2				
	$u = a/2, v = b/2$		$u = a/2, v = b/2$	
0.5000	0.0698	0.0513	0.0272	0.0086
0.7143	0.0972	0.0713	0.0398	0.0123
0.8333	0.1073	0.0786	0.0455	0.0137
1.0000	0.1167	0.0854	0.0521	0.0152
1.2000	0.1239	0.0863	0.0629	0.0259
1.4000	0.1277	0.0874	0.0684	0.0280
2.0000	0.1306	0.0918	0.0770	0.0261

ordu = 3, ordv = 3, segu = 4, segv = 4				
	$u = a/2, v = b/2$		$u = a/2, v = b/2$	
0.5000	0.0760	0.0613	0.0361	0.0179
0.7143	0.1091	0.0884	0.0548	0.0254
0.8333	0.1220	0.0990	0.0632	0.0282
1.0000	0.1349	0.1095	0.0730	0.0308
1.2000	0.1445	0.1156	0.0832	0.0349
1.4000	0.1502	0.1193	0.0906	0.0366
2.0000	0.1561	0.1248	0.1036	0.0362

ordu = 3, ordv = 3, segu = 6, segv = 6				
	$u = a/2, v = b/2$		$u = a/2, v = b/2$	
0.5000	0.0769	0.0616	0.0373	0.0198
0.7143	0.1109	0.0900	0.0571	0.0281
0.8333	0.1244	0.1012	0.0661	0.0312
1.0000	0.1379	0.1125	0.0765	0.0341
1.2000	0.1482	0.1200	0.0867	0.0371
1.4000	0.1542	0.1245	0.0944	0.0385
2.0000	0.1608	0.1300	0.1084	0.0385

ordu = 3, ordv = 3, segu = 8, segv = 8				
	$u = a/2, v = b/2$		$u = a/2, v = b/2$	
0.5000	0.0771	0.0615	0.0377	0.0205
0.7143	0.1116	0.0902	0.0579	0.0293
0.8333	0.1253	0.1016	0.0671	0.0325
1.0000	0.1390	0.1131	0.0778	0.0355
1.2000	0.1494	0.1212	0.0880	0.0381
1.4000	0.1556	0.1260	0.0958	0.0393
2.0000	0.1624	0.1315	0.1101	0.0395

ordu = 3, ordv = 3, segu = 10, segv = 10				
	$u = a/2, v = b/2$		$u = a/2, v = b/2$	
0.5000	0.0773	0.0614	0.0379	0.0210
0.7143	0.1118	0.0901	0.0583	0.0299
0.8333	0.1256	0.1016	0.0688	0.0373
1.0000	0.1395	0.1132	0.0784	0.0363
1.2000	0.1500	0.1216	0.0886	0.0387
1.4000	0.1563	0.1266	0.0964	0.0398
2.0000	0.1632	0.1320	0.1109	0.0399

ordu = 3, ordv = 3, segu = 12, segv = 12				
	$u = a/2, v = b/2$		$u = a/2, v = b/2$	
0.5000	0.0773	0.0612	0.0380	0.0212
0.7143	0.1120	0.0900	0.0586	0.0303
0.8333	0.1258	0.1015	0.0679	0.0337
1.0000	0.1398	0.1132	0.0787	0.0368
1.2000	0.1503	0.1217	0.0890	0.0391
1.4000	0.1566	0.1268	0.0968	0.0402
2.0000	0.1636	0.1322	0.1113	0.0402

ordu = 4, ordv = 4, segu = 2, segv = 2				
	$u = a/2, v = b/2$		$u = a/2, v = b/2$	
0.5000	0.0779	0.0729	0.0459	0.0282
0.7143	0.1128	0.1072	0.0707	0.0406
0.8333	0.1267	0.1209	0.0820	0.0452
1.0000	0.1407	0.1348	0.0951	0.0496
1.2000	0.1513	0.1441	0.1049	0.0459
1.4000	0.1576	0.1497	0.1141	0.0470
2.0000	0.1646	0.1585	0.1304	0.0453

ordu = 4, ordv = 4, segu = 4, segv = 4				
	$u = a/2, v = b/2$		$u = a/2, v = b/2$	
0.5000	0.0775	0.0623	0.0394	0.0233
0.7143	0.1124	0.0927	0.0612	0.0335
0.8333	0.1263	0.1049	0.0712	0.0372
1.0000	0.1404	0.1173	0.0827	0.0405
1.2000	0.1511	0.1264	0.0933	0.0424
1.4000	0.1574	0.1318	0.1016	0.0432
2.0000	0.1646	0.1383	0.1169	0.0426

ordu = 4, ordv = 4, segu = 6, segv = 6				
	$u = a/2, v = b/2$		$u = a/2, v = b/2$	
0.5000	0.0775	0.0612	0.0390	0.0228
0.7143	0.1123	0.0905	0.0603	0.0327
0.8333	0.1263	0.1023	0.0700	0.0364
1.0000	0.1404	0.1143	0.0812	0.0397
1.2000	0.1511	0.1232	0.0916	0.0417
1.4000	0.1574	0.1285	0.0996	0.0425
2.0000	0.1646	0.1346	0.1144	0.0420

ordu = 4, ordv = 4, segu = 8, segv = 8				
	$u = a/2, v = b/2$		$u = a/2, v = b/2$	
0.5000	0.0775	0.0607	0.0388	0.0226
0.7143	0.1123	0.0897	0.0599	0.0324
0.8333	0.1263	0.1014	0.0695	0.0360
1.0000	0.1403	0.1131	0.0806	0.0394
1.2000	0.1511	0.1221	0.0910	0.0415
1.4000	0.1574	0.1273	0.0989	0.0423
2.0000	0.1646	0.1333	0.1136	0.0417

ordu = 4, ordv = 4, segu = 10, segv = 10				
	$u = a/2, v = b/2$		$u = a/2, v = b/2$	
0.5000	0.0775	0.0605	0.0387	0.0225
0.7143	0.1123	0.0894	0.0597	0.0323
0.8333	0.1263	0.1009	0.0693	0.0359
1.0000	0.1403	0.1126	0.0803	0.0393
1.2000	0.1511	0.1215	0.0907	0.0414
1.4000	0.1574	0.1268	0.0986	0.0422
2.0000	0.1646	0.1327	0.1132	0.0416

ordu = 4, ordv = 4, segu = 12, segv = 12				
	$u = a/2, v = b/2$		$u = a/2, v = b/2$	
0.5000	0.0775	0.0604	0.0386	0.0224
0.7143	0.1123	0.0892	0.0596	0.0322
0.8333	0.1263	0.1007	0.0691	0.0358
1.0000	0.1403	0.1123	0.0802	0.0392
1.2000	0.1511	0.1212	0.0905	0.0413
1.4000	0.1574	0.1265	0.0984	0.0421
2.0000	0.1646	0.1324	0.1130	0.0416

ordu = 5, ordv = 5, segu = 2, segv = 2				
	$u = a/2, v = b/2$		$u = a/2, v = b/2$	
0.5000	0.0777	0.0616	0.0396	0.0229
0.7143	0.1125	0.0899	0.0605	0.0329
0.8333	0.1264	0.1013	0.0701	0.0368
1.0000	0.1404	0.1129	0.0812	0.0405
1.2000	0.1511	0.1208	0.0909	0.0415
1.4000	0.1575	0.1259	0.0986	0.0422
2.0000	0.1646	0.1321	0.1127	0.0414

ordu = 5, ordv = 5, segu = 4, segv = 4				
	$u = a/2, v = b/2$		$u = a/2, v = b/2$	
0.5000	0.0775	0.0603	0.0385	0.0223
0.7143	0.1123	0.0888	0.0594	0.0321
0.8333	0.1263	0.1003	0.0690	0.0357
1.0000	0.1404	0.1118	0.0800	0.0391
1.2000	0.1511	0.1206	0.0902	0.0412
1.4000	0.1574	0.1258	0.0981	0.0420
2.0000	0.1646	0.1317	0.1125	0.0414

ordu = 5, ordv = 5, segu = 6, segv = 6				
	$u = a/2, v = b/2$		$u = a/2, v = b/2$	
0.5000	0.0775	0.0602	0.0385	0.0223
0.7143	0.1123	0.0887	0.0593	0.0320
0.8333	0.1263	0.1002	0.0689	0.0356
1.0000	0.1403	0.1117	0.0799	0.0390
1.2000	0.1511	0.1205	0.0901	0.0411
1.4000	0.1574	0.1258	0.0980	0.0420
2.0000	0.1646	0.1316	0.1125	0.0414

ordu = 5, ordv = 5, segu = 8, segv = 8				
	$u = a/2, v = b/2$		$u = a/2, v = b/2$	
0.5000	0.0775	0.0602	0.0385	0.0223
0.7143	0.1123	0.0887	0.0593	0.0320
0.8333	0.1263	0.1002	0.0689	0.0356
1.0000	0.1403	0.1117	0.0799	0.0390
1.2000	0.1511	0.1205	0.0901	0.0411
1.4000	0.1574	0.1258	0.0980	0.0420
2.0000	0.1646	0.1316	0.1125	0.0414

ordu = 5, ordv = 5, segu = 10, segv = 10				
	$u = a/2, v = b/2$		$u = a/2, v = b/2$	
0.5000	0.0775	0.0602	0.0385	0.0223
0.7143	0.1123	0.0887	0.0593	0.0320
0.8333	0.1263	0.1001	0.0689	0.0356
1.0000	0.1403	0.1117	0.0799	0.0390
1.2000	0.1511	0.1205	0.0901	0.0411
1.4000	0.1574	0.1258	0.0980	0.0420
2.0000	0.1646	0.1316	0.1125	0.0414

ordu = 5, ordv = 5, segu = 12, segv = 12				
	$u = a/2, v = b/2$		$u = a/2, v = b/2$	
0.5000	0.0775	0.0602	0.0385	0.0223
0.7143	0.1123	0.0887	0.0593	0.0320
0.8333	0.1263	0.1001	0.0689	0.0356
1.0000	0.1403	0.1117	0.0799	0.0390
1.2000	0.1511	0.1205	0.0901	0.0411
1.4000	0.1574	0.1258	0.0980	0.0420
2.0000	0.1646	0.1316	0.1125	0.0414

ordu = 6, ordv = 6, segu = 2, segv = 2				
	$u = a/2, v = b/2$		$u = a/2, v = b/2$	
0.5000	0.0775	0.0599	0.0382	0.0222
0.7143	0.1123	0.0886	0.0591	0.0319
0.8333	0.1263	0.1001	0.0686	0.0355
1.0000	0.1403	0.1117	0.0797	0.0387
1.2000	0.1511	0.1205	0.0900	0.0411
1.4000	0.1574	0.1257	0.0979	0.0420
2.0000	0.1646	0.1317	0.1124	0.0414

ordu = 6, ordv = 6, segu = 4, segv = 4				
	$u = a/2, v = b/2$		$u = a/2, v = b/2$	
0.5000	0.0775	0.0601	0.0385	0.0223
0.7143	0.1123	0.0887	0.0593	0.0320
0.8333	0.1263	0.1001	0.0688	0.0356
1.0000	0.1403	0.1117	0.0798	0.0389
1.2000	0.1511	0.1205	0.0901	0.0411
1.4000	0.1574	0.1257	0.0980	0.0420
2.0000	0.1646	0.1316	0.1125	0.0414

ordu = 6, ordv = 6, segu = 6, segv = 6				
	$u = a/2, v = b/2$		$u = a/2, v = b/2$	
0.5000	0.0775	0.0602	0.0385	0.0223
0.7143	0.1123	0.0887	0.0593	0.0320
0.8333	0.1263	0.1001	0.0689	0.0356
1.0000	0.1403	0.1117	0.0798	0.0390
1.2000	0.1511	0.1205	0.0901	0.0411
1.4000	0.1574	0.1258	0.0980	0.0420
2.0000	0.1646	0.1316	0.1125	0.0414

ordu = 6, ordv = 6, segu = 8, segv = 8				
	$u = a/2, v = b/2$		$u = a/2, v = b/2$	
0.5000	0.0775	0.0602	0.0385	0.0223
0.7143	0.1123	0.0887	0.0593	0.0320
0.8333	0.1263	0.1001	0.0689	0.0356
1.0000	0.1403	0.1117	0.0799	0.0390
1.2000	0.1511	0.1205	0.0901	0.0411
1.4000	0.1574	0.1258	0.0980	0.0420
2.0000	0.1646	0.1316	0.1125	0.0414

ordu = 6, ordv = 6, segu = 10, segv = 10				
	$u = a/2, v = b/2$		$u = a/2, v = b/2$	
0.5000	0.0775	0.0602	0.0385	0.0223
0.7143	0.1123	0.0887	0.0593	0.0320
0.8333	0.1263	0.1001	0.0689	0.0356
1.0000	0.1403	0.1117	0.0799	0.0390
1.2000	0.1511	0.1205	0.0901	0.0411
1.4000	0.1574	0.1258	0.0980	0.0420
2.0000	0.1646	0.1316	0.1125	0.0414

ordu = 6, ordv = 6, segu = 12, segv = 12				
	$u = a/2, v = b/2$		$u = a/2, v = b/2$	
0.5000	0.0775	0.0602	0.0385	0.0223
0.7143	0.1123	0.0887	0.0593	0.0320
0.8333	0.1263	0.1001	0.0689	0.0356
1.0000	0.1403	0.1117	0.0799	0.0390
1.2000	0.1511	0.1205	0.0901	0.0411
1.4000	0.1574	0.1258	0.0980	0.0420
2.0000	0.1646	0.1316	0.1125	0.0414

ordu = 7, ordv = 7, segu = 2, segv = 2				
	$u = a/2, v = b/2$		$u = a/2, v = b/2$	
0.5000	0.0775	0.0602	0.0384	0.0223
0.7143	0.1123	0.0887	0.0593	0.0320
0.8333	0.1263	0.1002	0.0688	0.0356
1.0000	0.1403	0.1117	0.0799	0.0390
1.2000	0.1511	0.1205	0.0901	0.0411
1.4000	0.1574	0.1258	0.0980	0.0420
2.0000	0.1646	0.1316	0.1125	0.0414

ordu = 7, ordv = 7, segu = 4, segv = 4				
	$u = a/2, v = b/2$		$u = a/2, v = b/2$	
0.5000	0.0775	0.0602	0.0385	0.0223
0.7143	0.1123	0.0887	0.0593	0.0320
0.8333	0.1263	0.1001	0.0689	0.0356
1.0000	0.1403	0.1117	0.0799	0.0390
1.2000	0.1511	0.1205	0.0901	0.0411
1.4000	0.1574	0.1257	0.0980	0.0420
2.0000	0.1646	0.1316	0.1125	0.0414

ordu = 7, ordv = 7, segu = 6, segv = 6				
	$u = a/2, v = b/2$		$u = a/2, v = b/2$	
0.5000	0.0775	0.0602	0.0385	0.0223
0.7143	0.1123	0.0887	0.0593	0.0320
0.8333	0.1263	0.1001	0.0689	0.0356
1.0000	0.1403	0.1117	0.0799	0.0390
1.2000	0.1511	0.1205	0.0901	0.0411
1.4000	0.1574	0.1258	0.0980	0.0420
2.0000	0.1646	0.1316	0.1125	0.0414

ordu = 7, ordv = 7, segu = 8, segv = 8				
	$u = a/2, v = b/2$		$u = a/2, v = b/2$	
0.5000	0.0775	0.0602	0.0385	0.0223
0.7143	0.1123	0.0887	0.0593	0.0320
0.8333	0.1263	0.1001	0.0689	0.0356
1.0000	0.1403	0.1117	0.0799	0.0390
1.2000	0.1511	0.1205	0.0901	0.0411
1.4000	0.1574	0.1258	0.0980	0.0420
2.0000	0.1646	0.1316	0.1125	0.0414

ordu = 7, ordv = 7, segu = 10, segv = 10				
	$u = a/2, v = b/2$		$u = a/2, v = b/2$	
0.5000	0.0775	0.0602	0.0385	0.0223
0.7143	0.1123	0.0887	0.0593	0.0320
0.8333	0.1263	0.1001	0.0689	0.0356
1.0000	0.1403	0.1117	0.0799	0.0390
1.2000	0.1511	0.1205	0.0901	0.0411
1.4000	0.1574	0.1258	0.0980	0.0420
2.0000	0.1646	0.1316	0.1125	0.0414

ordu = 7, ordv = 7, segu = 12, segv = 12				
	$u = a/2, v = b/2$		$u = a/2, v = b/2$	
0.5000	0.0775	0.0602	0.0385	0.0223
0.7143	0.1123	0.0887	0.0593	0.0320
0.8333	0.1263	0.1001	0.0689	0.0356
1.0000	0.1403	0.1117	0.0799	0.0390
1.2000	0.1511	0.1205	0.0901	0.0411
1.4000	0.1574	0.1258	0.0980	0.0420
2.0000	0.1646	0.1316	0.1125	0.0414

ordu = 8, ordv = 8, segu = 2, segv = 2				
	$u = a/2, v = b/2$		$u = a/2, v = b/2$	
0.5000	0.0775	0.0602	0.0385	0.0223
0.7143	0.1123	0.0887	0.0593	0.0320
0.8333	0.1263	0.1001	0.0689	0.0356
1.0000	0.1403	0.1117	0.0799	0.0390
1.2000	0.1511	0.1205	0.0901	0.0411
1.4000	0.1574	0.1257	0.0980	0.0420
2.0000	0.1646	0.1316	0.1125	0.0414

ordu = 8, ordv = 8, segu = 4, segv = 4				
	$u = a/2, v = b/2$		$u = a/2, v = b/2$	
0.5000	0.0775	0.0602	0.0385	0.0223
0.7143	0.1123	0.0887	0.0593	0.0320
0.8333	0.1263	0.1001	0.0689	0.0356
1.0000	0.1403	0.1117	0.0799	0.0390
1.2000	0.1511	0.1205	0.0901	0.0411
1.4000	0.1574	0.1258	0.0980	0.0420
2.0000	0.1646	0.1316	0.1125	0.0414

ordu = 8, ordv = 8, segu = 6, segv = 6				
	$u = a/2, v = b/2$		$u = a/2, v = b/2$	
0.5000	0.0775	0.0602	0.0385	0.0223
0.7143	0.1123	0.0887	0.0593	0.0320
0.8333	0.1263	0.1001	0.0689	0.0356
1.0000	0.1403	0.1117	0.0799	0.0390
1.2000	0.1511	0.1205	0.0901	0.0411
1.4000	0.1574	0.1258	0.0980	0.0420
2.0000	0.1646	0.1316	0.1125	0.0414

ordu = 8, ordv = 8, segu = 8, segv = 8				
	$u = a/2, v = b/2$		$u = a/2, v = b/2$	
0.5000	0.0775	0.0602	0.0385	0.0223
0.7143	0.1123	0.0887	0.0593	0.0320
0.8333	0.1263	0.1001	0.0689	0.0356
1.0000	0.1403	0.1117	0.0799	0.0390
1.2000	0.1511	0.1205	0.0901	0.0411
1.4000	0.1574	0.1258	0.0980	0.0420
2.0000	0.1646	0.1316	0.1125	0.0414

ordu = 8, ordv = 8, segu = 10, segv = 10				
	$u = a/2, v = b/2$		$u = a/2, v = b/2$	
0.5000	0.0775	0.0602	0.0385	0.0223
0.7143	0.1123	0.0887	0.0593	0.0320
0.8333	0.1263	0.1001	0.0689	0.0356
1.0000	0.1403	0.1117	0.0799	0.0390
1.2000	0.1511	0.1205	0.0901	0.0411
1.4000	0.1574	0.1258	0.0980	0.0420
2.0000	0.1646	0.1316	0.1125	0.0414

ordu = 8, ordv = 8, segu = 12, segv = 12				
	$u = a/2, v = b/2$		$u = a/2, v = b/2$	
$\frac{b}{a}$	$(x)_{max}$	$(M_u)_{max}$	M_u	M_v
0.5000	0.0775	0.0602	0.0385	0.0223
0.7143	0.1123	0.0887	0.0593	0.0320
0.8333	0.1263	0.1001	0.0689	0.0356
1.0000	0.1403	0.1117	0.0799	0.0390
1.2000	0.1511	0.1205	0.0901	0.0411
1.4000	0.1574	0.1258	0.0980	0.0420
2.0000	0.1646	0.1316	0.1125	0.0414

A.7 Uniformly loaded rectangular plate with all edges clamped

Deflection and bending moments for uniformly loaded rectangular plate will all edges clamped.

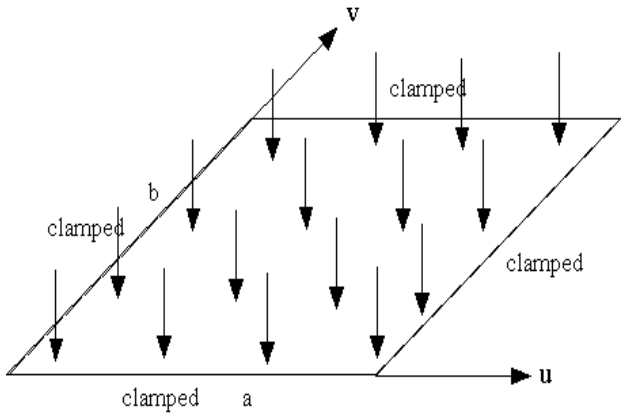


Figure A.7: Uniformly loaded rectangular plate with all edges clamped

$\frac{b}{a}$	$(x)_{u=a/2,v=b/2}$	$(M_u)_{u=a,v=b/2}$	$(M_v)_{u=a/2,v=b}$	$(M_u)_{u=a/2,v=b/2}$	$(M_v)_{u=a/2,v=b/2}$
1.0000	0.0138	-0.0513	-0.0513	-	-
1.2000	0.0188	-0.0639	-0.0554	0.0299	0.0228
1.4000	0.0226	-0.0726	-0.0568	0.0349	0.0212
1.6000	0.0251	-0.0780	-0.0571	0.0381	0.0193
1.8000	0.0267	-0.0812	-0.0571	0.0401	0.0174
2.0000	0.0277	-0.0829	-0.0571	-	-

Table A.7: Numerical results from [85], rectangular plate, all edges clamped, uniform load

ordu = 4, ordv = 4, segu = 2, segv = 2					
1.0000	0.0145	-0.0318	-0.0318	0.0413	0.0413
1.2000	0.0195	-0.0429	-0.0358	0.0483	0.0308
1.4000	0.0226	-0.0496	-0.0413	0.0524	0.0242
1.6000	0.0251	-0.0552	-0.0435	0.0576	0.0249
1.8000	0.0267	-0.0587	-0.0449	0.0604	0.0231
2.0000	0.0271	-0.0596	-0.0315	0.0591	0.0161

ordu = 4, ordv = 4, segu = 4, segv = 4					
1.0000	0.0138	-0.0423	-0.0423	0.0237	0.0237
1.2000	0.0188	-0.0553	-0.0464	0.0318	0.0232
1.4000	0.0226	-0.0648	-0.0480	0.0380	0.0221
1.6000	0.0251	-0.0710	-0.0486	0.0421	0.0206
1.8000	0.0267	-0.0747	-0.0487	0.0444	0.0184
2.0000	0.0277	-0.0768	-0.0471	0.0458	0.0169

ordu = 4, ordv = 4, segu = 6, segv = 6					
1.0000	0.0138	-0.0469	-0.0469	0.0236	0.0236
1.2000	0.0188	-0.0598	-0.0504	0.0310	0.0231
1.4000	0.0226	-0.0690	-0.0515	0.0364	0.0216
1.6000	0.0251	-0.0748	-0.0516	0.0400	0.0199
1.8000	0.0267	-0.0783	-0.0515	0.0420	0.0178
2.0000	0.0277	-0.0802	-0.0521	0.0432	0.0163

ordu = 4, ordv = 4, segu = 8, segv = 8					
1.0000	0.0138	-0.0487	-0.0487	0.0233	0.0233
1.2000	0.0188	-0.0615	-0.0523	0.0305	0.0230
1.4000	0.0226	-0.0705	-0.0533	0.0358	0.0215
1.6000	0.0251	-0.0762	-0.0533	0.0392	0.0196
1.8000	0.0267	-0.0795	-0.0530	0.0412	0.0176
2.0000	0.0277	-0.0813	-0.0541	0.0423	0.0161

ordu = 4, ordv = 4, segu = 10, segv = 10					
1.0000	0.0138	-0.0496	-0.0496	0.0232	0.0232
1.2000	0.0188	-0.0624	-0.0532	0.0303	0.0229
1.4000	0.0226	-0.0712	-0.0543	0.0355	0.0214
1.6000	0.0251	-0.0768	-0.0543	0.0388	0.0195
1.8000	0.0267	-0.0801	-0.0540	0.0408	0.0175
2.0000	0.0277	-0.0819	-0.0551	0.0419	0.0160

ordu = 4, ordv = 4, segu = 12, segv = 12					
1.0000	0.0138	-0.0501	-0.0501	0.0231	0.0231
1.2000	0.0188	-0.0628	-0.0538	0.0302	0.0229
1.4000	0.0226	-0.0716	-0.0549	0.0353	0.0214
1.6000	0.0251	-0.0772	-0.0549	0.0386	0.0194
1.8000	0.0267	-0.0804	-0.0547	0.0406	0.0174
2.0000	0.0277	-0.0822	-0.0556	0.0417	0.0159

ordu = 4, ordv = 4, segu = 14, segv = 14					
1.0000	0.0138	-0.0504	-0.0504	0.0230	0.0230
1.2000	0.0188	-0.0631	-0.0542	0.0302	0.0229
1.4000	0.0226	-0.0719	-0.0553	0.0352	0.0213
1.6000	0.0251	-0.0774	-0.0554	0.0385	0.0194
1.8000	0.0267	-0.0806	-0.0552	0.0404	0.0174
2.0000	0.0277	-0.0824	-0.0560	0.0415	0.0159

ordu = 5, ordv = 5, segu = 2, segv = 2					
1.0000	0.0145	-0.0425	-0.0425	0.0276	0.0276
1.2000	0.0191	-0.0558	-0.0516	0.0322	0.0226
1.4000	0.0228	-0.0669	-0.0543	0.0371	0.0221
1.6000	0.0253	-0.0742	-0.0555	0.0397	0.0196
1.8000	0.0269	-0.0788	-0.0559	0.0411	0.0176
2.0000	0.0276	-0.0808	-0.0480	0.0413	0.0153

ordu = 5, ordv = 5, segu = 4, segv = 4					
1.0000	0.0138	-0.0501	-0.0501	0.0232	0.0232
1.2000	0.0188	-0.0630	-0.0542	0.0302	0.0228
1.4000	0.0226	-0.0720	-0.0557	0.0352	0.0214
1.6000	0.0251	-0.0777	-0.0560	0.0383	0.0193
1.8000	0.0267	-0.0810	-0.0560	0.0402	0.0174
2.0000	0.0277	-0.0828	-0.0556	0.0412	0.0158

ordu = 5, ordv = 5, segu = 6, segv = 6					
1.0000	0.0138	-0.0509	-0.0509	0.0229	0.0229
1.2000	0.0188	-0.0637	-0.0549	0.0300	0.0228
1.4000	0.0226	-0.0725	-0.0563	0.0350	0.0213
1.6000	0.0251	-0.0779	-0.0565	0.0382	0.0192
1.8000	0.0267	-0.0811	-0.0565	0.0401	0.0174
2.0000	0.0277	-0.0829	-0.0565	0.0412	0.0158

ordu = 5, ordv = 5, segu = 8, segv = 8					
1.0000	0.0138	-0.0512	-0.0512	0.0229	0.0229
1.2000	0.0188	-0.0638	-0.0552	0.0300	0.0228
1.4000	0.0226	-0.0725	-0.0566	0.0350	0.0213
1.6000	0.0251	-0.0780	-0.0568	0.0382	0.0192
1.8000	0.0267	-0.0812	-0.0567	0.0401	0.0174
2.0000	0.0277	-0.0829	-0.0568	0.0412	0.0158

ordu = 5, ordv = 5, segu = 14, segv = 14					
1.0000	0.0138	-0.0513	-0.0513	0.0229	0.0229
1.2000	0.0188	-0.0639	-0.0554	0.0300	0.0228
1.4000	0.0226	-0.0726	-0.0567	0.0350	0.0213
1.6000	0.0251	-0.0780	-0.0570	0.0382	0.0193
1.8000	0.0267	-0.0812	-0.0570	0.0401	0.0174
2.0000	0.0277	-0.0829	-0.0570	0.0412	0.0158

ordu = 6, ordv = 6, segu = 2, segv = 2					
1.0000	0.0138	-0.0498	-0.0498	0.0221	0.0221
1.2000	0.0188	-0.0630	-0.0546	0.0296	0.0230
1.4000	0.0226	-0.0720	-0.0562	0.0346	0.0211
1.6000	0.0251	-0.0777	-0.0567	0.0380	0.0192
1.8000	0.0267	-0.0811	-0.0568	0.0400	0.0173
2.0000	0.0277	-0.0829	-0.0548	0.0411	0.0159

ordu = 6, ordv = 6, segu = 4, segv = 4					
1.0000	0.0138	-0.0512	-0.0512	0.0228	0.0228
1.2000	0.0188	-0.0638	-0.0553	0.0299	0.0228
1.4000	0.0226	-0.0727	-0.0567	0.0349	0.0212
1.6000	0.0251	-0.0781	-0.0570	0.0381	0.0193
1.8000	0.0267	-0.0812	-0.0569	0.0400	0.0173
2.0000	0.0277	-0.0829	-0.0568	0.0411	0.0158

ordu = 6, ordv = 6, segu = 6, segv = 6					
1.0000	0.0138	-0.0514	-0.0514	0.0229	0.0229
1.2000	0.0188	-0.0639	-0.0554	0.0300	0.0228
1.4000	0.0226	-0.0726	-0.0568	0.0350	0.0213
1.6000	0.0251	-0.0780	-0.0571	0.0382	0.0192
1.8000	0.0267	-0.0812	-0.0571	0.0401	0.0174
2.0000	0.0277	-0.0829	-0.0570	0.0412	0.0158

ordu = 6, ordv = 6, segu = 8, segv = 8					
1.0000	0.0138	-0.0513	-0.0513	0.0229	0.0229
1.2000	0.0188	-0.0639	-0.0554	0.0300	0.0228
1.4000	0.0226	-0.0726	-0.0568	0.0350	0.0213
1.6000	0.0251	-0.0781	-0.0571	0.0382	0.0193
1.8000	0.0267	-0.0812	-0.0571	0.0401	0.0174
2.0000	0.0277	-0.0829	-0.0570	0.0412	0.0158

ordu = 6, ordv = 6, segu = 10, segv = 10					
1.0000	0.0138	-0.0513	-0.0513	0.0229	0.0229
1.2000	0.0188	-0.0639	-0.0554	0.0300	0.0228
1.4000	0.0226	-0.0726	-0.0568	0.0350	0.0213
1.6000	0.0251	-0.0780	-0.0571	0.0382	0.0193
1.8000	0.0267	-0.0812	-0.0571	0.0401	0.0174
2.0000	0.0277	-0.0829	-0.0570	0.0412	0.0158

ordu = 6, ordv = 6, segu = 12, segv = 12					
1.0000	0.0138	-0.0513	-0.0513	0.0229	0.0229
1.2000	0.0188	-0.0639	-0.0554	0.0300	0.0228
1.4000	0.0226	-0.0726	-0.0568	0.0350	0.0213
1.6000	0.0251	-0.0780	-0.0571	0.0382	0.0193
1.8000	0.0267	-0.0812	-0.0571	0.0401	0.0174
2.0000	0.0277	-0.0829	-0.0570	0.0412	0.0158

ordu = 6, ordv = 6, segu = 14, segv = 14					
1.0000	0.0138	-0.0513	-0.0513	0.0229	0.0229
1.2000	0.0188	-0.0639	-0.0554	0.0300	0.0228
1.4000	0.0226	-0.0726	-0.0568	0.0350	0.0213
1.6000	0.0251	-0.0780	-0.0571	0.0382	0.0193
1.8000	0.0267	-0.0812	-0.0571	0.0401	0.0174
2.0000	0.0277	-0.0829	-0.0570	0.0412	0.0158

ordu = 7, ordv = 7, segu = 2, segv = 2					
1.0000	0.0138	-0.0512	-0.0512	0.0228	0.0228
1.2000	0.0188	-0.0639	-0.0554	0.0300	0.0229
1.4000	0.0226	-0.0727	-0.0569	0.0350	0.0212
1.6000	0.0251	-0.0782	-0.0572	0.0382	0.0193
1.8000	0.0267	-0.0814	-0.0572	0.0401	0.0174
2.0000	0.0276	-0.0829	-0.0569	0.0411	0.0157

ordu = 7, ordv = 7, segu = 4, segv = 4					
1.0000	0.0138	-0.0513	-0.0513	0.0229	0.0229
1.2000	0.0188	-0.0640	-0.0553	0.0300	0.0229
1.4000	0.0226	-0.0726	-0.0567	0.0350	0.0213
1.6000	0.0251	-0.0780	-0.0570	0.0382	0.0193
1.8000	0.0267	-0.0812	-0.0570	0.0401	0.0174
2.0000	0.0277	-0.0828	-0.0569	0.0412	0.0158

ordu = 7, ordv = 7, segu = 6, segv = 6					
1.0000	0.0138	-0.0514	-0.0514	0.0229	0.0229
1.2000	0.0188	-0.0639	-0.0554	0.0300	0.0228
1.4000	0.0226	-0.0726	-0.0568	0.0350	0.0213
1.6000	0.0251	-0.0781	-0.0571	0.0382	0.0193
1.8000	0.0267	-0.0812	-0.0571	0.0401	0.0174
2.0000	0.0277	-0.0829	-0.0570	0.0412	0.0158

ordu = 7, ordv = 7, segu = 8, segv = 8					
1.0000	0.0138	-0.0513	-0.0513	0.0229	0.0229
1.2000	0.0188	-0.0639	-0.0554	0.0300	0.0228
1.4000	0.0226	-0.0726	-0.0568	0.0350	0.0213
1.6000	0.0251	-0.0780	-0.0571	0.0382	0.0193
1.8000	0.0267	-0.0812	-0.0571	0.0401	0.0174
2.0000	0.0277	-0.0829	-0.0570	0.0412	0.0158

ordu = 7, ordv = 7, segu = 10, segv = 10					
1.0000	0.0138	-0.0513	-0.0513	0.0229	0.0229
1.2000	0.0188	-0.0639	-0.0554	0.0300	0.0228
1.4000	0.0226	-0.0726	-0.0568	0.0350	0.0213
1.6000	0.0251	-0.0780	-0.0571	0.0382	0.0193
1.8000	0.0267	-0.0812	-0.0571	0.0401	0.0174
2.0000	0.0277	-0.0829	-0.0570	0.0412	0.0158

ordu = 7, ordv = 7, segu = 12, segv = 12					
1.0000	0.0138	-0.0513	-0.0513	0.0229	0.0229
1.2000	0.0188	-0.0639	-0.0554	0.0300	0.0228
1.4000	0.0226	-0.0726	-0.0568	0.0350	0.0213
1.6000	0.0251	-0.0780	-0.0571	0.0382	0.0193
1.8000	0.0267	-0.0812	-0.0571	0.0401	0.0174
2.0000	0.0277	-0.0829	-0.0570	0.0412	0.0158

ordu = 7, ordv = 7, segu = 14, segv = 14					
1.0000	0.0138	-0.0513	-0.0513	0.0229	0.0229
1.2000	0.0188	-0.0639	-0.0554	0.0300	0.0228
1.4000	0.0226	-0.0726	-0.0568	0.0350	0.0213
1.6000	0.0251	-0.0780	-0.0571	0.0382	0.0193
1.8000	0.0267	-0.0812	-0.0571	0.0401	0.0174
2.0000	0.0277	-0.0829	-0.0570	0.0412	0.0158

ordu = 8, ordv = 8, segu = 2, segv = 2					
1.0000	0.0138	-0.0513	-0.0513	0.0229	0.0229
1.2000	0.0188	-0.0638	-0.0554	0.0299	0.0228
1.4000	0.0226	-0.0727	-0.0567	0.0350	0.0213
1.6000	0.0251	-0.0781	-0.0570	0.0382	0.0193
1.8000	0.0267	-0.0812	-0.0570	0.0401	0.0174
2.0000	0.0277	-0.0830	-0.0570	0.0412	0.0158

ordu = 8, ordv = 8, segu = 4, segv = 4					
1.0000	0.0138	-0.0514	-0.0514	0.0229	0.0229
1.2000	0.0188	-0.0640	-0.0555	0.0300	0.0228
1.4000	0.0226	-0.0726	-0.0569	0.0350	0.0213
1.6000	0.0251	-0.0780	-0.0572	0.0382	0.0193
1.8000	0.0267	-0.0812	-0.0571	0.0401	0.0174
2.0000	0.0277	-0.0829	-0.0570	0.0412	0.0158

ordu = 8, ordv = 8, segu = 6, segv = 6					
1.0000	0.0138	-0.0513	-0.0513	0.0229	0.0229
1.2000	0.0188	-0.0639	-0.0554	0.0300	0.0228
1.4000	0.0226	-0.0726	-0.0568	0.0350	0.0213
1.6000	0.0251	-0.0781	-0.0571	0.0382	0.0193
1.8000	0.0267	-0.0812	-0.0571	0.0401	0.0174
2.0000	0.0277	-0.0829	-0.0570	0.0412	0.0158

ordu = 8, ordv = 8, segu = 8, segv = 8					
1.0000	0.0138	-0.0513	-0.0513	0.0229	0.0229
1.2000	0.0188	-0.0639	-0.0554	0.0300	0.0228
1.4000	0.0226	-0.0726	-0.0568	0.0350	0.0213
1.6000	0.0251	-0.0780	-0.0571	0.0382	0.0193
1.8000	0.0267	-0.0812	-0.0571	0.0401	0.0174
2.0000	0.0277	-0.0829	-0.0570	0.0412	0.0158

ordu = 8, ordv = 8, segu = 10, segv = 10					
1.0000	0.0138	-0.0513	-0.0513	0.0229	0.0229
1.2000	0.0188	-0.0639	-0.0554	0.0300	0.0228
1.4000	0.0226	-0.0726	-0.0568	0.0350	0.0213
1.6000	0.0251	-0.0780	-0.0571	0.0382	0.0193
1.8000	0.0267	-0.0812	-0.0571	0.0401	0.0174
2.0000	0.0277	-0.0829	-0.0570	0.0412	0.0158

ordu = 8, ordv = 8, segu = 12, segv = 12					
1.0000	0.0138	-0.0513	-0.0513	0.0229	0.0229
1.2000	0.0188	-0.0639	-0.0554	0.0300	0.0228
1.4000	0.0226	-0.0726	-0.0568	0.0350	0.0213
1.6000	0.0251	-0.0780	-0.0571	0.0382	0.0193
1.8000	0.0267	-0.0812	-0.0571	0.0401	0.0174
2.0000	0.0277	-0.0829	-0.0570	0.0412	0.0158

Appendix B

Class Definitions

The class definitions for the principle classes making up the software package developed as part of this thesis and which wraps the algorithms documented in the report are presented here.

class: Vector

Vector class for storage. Uses the standard library class as a base access and manipulation functions plus read/write functions

```
template<class T>
class Vector :  public std::vector<T>, public ReadWriteObject
{
  int min; minimum (base) index
  int Math::max; maximum index
  int num; number of elements
public:
  constructors
  Vector(int);
  ...
  read and write functions
  virtual void read(std::istream&);
  virtual void write(std::ostream&);
  ...
};
```

class: Matrix

Matrix class for storage. Uses the Vector class as a base access and manipulation functions plus read/write functions

```
template<class T>
class Matrix :  public Vector<Vector<T> >
{
  int r1,r2; start, end row
  int c1,c2; start, end col
public:
  constructors
  Matrix(int, int);
  Matrix(const Vector<T>&);
  ...
  u varies in direction of the columns
  v varies in direction of the rows
  Get a column of the matrix
  Vector<T> GetU(int) const;
  Get a row of the matrix
  Vector<T> GetV(int) const;
  ...
  insert a row into the matrix
  Matrix<T>& InsertRow(const Vector<T>&, int);
  insert a column into the matrix
  Matrix<T>& InsertCol(const Vector<T>&, int);
  ...
  read and write functions
  virtual void read(std::istream&);
  virtual void write(std::ostream&);
  ...
};
```

class: Matrix3D

Volume matrix class for storage. Uses the Vector class as a base access and manipulation functions plus read/write functions

```
template<class T>
class Matrix3D : public Vector<Matrix<T> >
{
  int r1,r2; start, end row
  int c1,c2; start, end col
  int h1,h2; start, end layer
public:
  constructors
  Matrix3D(int, int, int);
  Matrix3D(const Vector<Matrix<T> >&, int);
  ...
  get the UV layer matrix at w=index
  Matrix<T> GetUV(int index);
  get the UW layer matrix at v=index
  Matrix<T> GetUW(int index);
  get the VW layer matrix at u=index
  Matrix<T> GetVW(int index);
  get the U line vector at indv, indw
  Vector<T> GetU(int indv, int indw);
  get the V line vector at indu, indw
  Vector<T> GetV(int indu, int indw);
  get the W line vector at indu, indv
  Vector<T> GetW(int indu, int indv);
  ...
  read and write functions
  virtual void read(std::istream&);
  virtual void write(std::ostream&);
  ...
};
```

class: ReadWriteObject

abstract base class for reading and writing, keyboard, screen and file
4 read/write pure virtual functions to be overridden

```
class ReadWriteObject
{
  overloaded keyboard input
  friend std::istream& operator>>(std::istream& is, ReadWriteObject& r);
    { r.read(is); return is; }
  overloaded screen output
  friend std::ostream& operator<<(std::ostream& os, ReadWriteObject& r);
    { r.write(os); return os; }
  overloaded file input
  friend std::ifstream& operator>>(std::ifstream& ifs, ReadWriteObject& r);
    { r.readfile(ifs); return ifs; }
  overloaded file output
  friend std::ofstream& operator<<(std::ofstream& ofs, ReadWriteObject& r);
    { r.writefile(ofs); return ofs; }
public:
  pure virtual functions to be overriden in derived classes
  virtual void read(std::istream& is) = 0;
  virtual void write(std::ostream& os) = 0;
  virtual void readfile(std::istream& ifs) = 0;
  virtual void writefile(std::ostream& ofs) = 0;
};
```

class: Curve

abstract base curve class
pure virtual point/derivative, limit functions and read/write functions

```
template<class T>
class Curve :  public ReadWriteObject
{
public:
  pure virtual functions to be overridden in derived classes
  virtual T operator()(double) const = 0;
  virtual T Deriv(int, double) const = 0;
  virtual double GetLeftLimit() const = 0;
  virtual double GetRightLimit() const = 0;
  virtual void read(std::istream&) = 0;
  virtual void readfile(std::ifstream&) = 0;
  virtual void write(std::ostream&) = 0;
  virtual void writefile(std::ofstream&) = 0;
};
```

class: Surf

abstract base surface class
pure virtual point/derivative, limit functions and read/write functions

```
template<class T>
class Surf :  public ReadWriteObject
{
public:
  pure virtual functions to be overridden in derived classes
  virtual T operator()(double, double) const = 0;
  virtual T Deriv(int, int, double, double) const = 0;
  virtual double GetLeftLimitU() const = 0;
  virtual double GetRightLimitU() const = 0;
  virtual double GetLeftLimitV() const = 0;
  virtual double GetRightLimitV() const = 0;
  virtual void read(std::istream&) = 0;
  virtual void readfile(std::ifstream&) = 0;
  virtual void write(std::ostream&) = 0;
  virtual void writefile(std::ofstream&) = 0;
};
```

class: Vol

abstract base volume class
pure virtual point/derivative, limit and read/write functions

```
template<class T>
class Vol :  public ReadWriteObject
{
public:
  pure virtual functions to be overridden
  virtual T operator()(double, double, double) const = 0;
  virtual T Deriv(int, int, double, double, double) const = 0;
  virtual double GetLeftLimitU() const = 0;
  virtual double GetRightLimitU() const = 0;
  virtual double GetLeftLimitV() const = 0;
  virtual double GetRightLimitV() const = 0;
  virtual double GetLeftLimitW() const = 0;
  virtual double GetRightLimitW() const = 0;
  virtual void read(std::istream&) = 0;
  virtual void readfile(std::ifstream&) = 0;
  virtual void write(std::ostream&) = 0;
  virtual void writefile(std::ofstream&) = 0;
};
```

class: KnotSet

encapsulates a B-spline knot set

```
class KnotSet :  public ReadWriteObject
{
  int nk; number of knots
  int ord; order of curve
  int nd; number of distinct knots
  Vector<double> kts; vector of knots
  Vector<double> dts; vector of distinct knots
  Vector<double> mult; vector of multiplicities
  mathematical set representation
  std::multiset<double> mset;
  ...
public:
  constructors
  KnotSet(const Vector<double>&, int, int);
  ...
  find the index, ind, such that t ∈ [t_ind, t_ind + 1)
  int FindIndex(double x);
  ...
  create the matrix D_0^1
  Matrix<double> CreateMatrixDeriv() const;
  create the matrix D_0^lev
  Matrix<double> CreateMatrixDeriv(int) const;
  create interpolation vector
  Vector<double> CreateVectorInterp(double) const;
  create derivative interpolation vector
  Vector<double> CreateVectorInterpDeriv(int, double) const;
  ...
  create the product knot set
  CreateKnotSetProduct(const KnotSet&) const;
  create the derivative knot set
  CreateKnotSetDeriv(int) const;
  create the integral knot set
  CreateKnotSetIntegrate(int) const;
  ...
  input/output
  virtual void read(std::istream&);
  ...
};
```

class: BspCurvBasisFunc

encapsulates single B-spline basis function
algorithms for integration, evaluation

```
class BspCurvBasisFunc :  public Curve<double> {
  order of basis function
  int ord;
  vector of knots
  Vector<double> kts;
  BspCurv representation of basis function
  BspCurv<double> b;
  creates the BspCurv representation
  BspCurv<double> CreateBspCurv();
  ...
public:
  constructors
  BspCurvBasisFunc(const Vector<double>&, int);
  ...
  integrate the basis function
  double Integrate(double, double);
  ...
  point and derivative evaluation
  virtual double operator()(double) const;
  virtual double Derive(int, double) const;
  ...
  limits
  virtual double GetLeftLimit() const;
  virtual double GetRightLimit() const;
  ...
  input/output
  virtual void read(std::istream&);
  ...
};
```

class: BspCurvBasisFuncSet

encapsulates a complete set of basis functions for a given KnotSet
algorithms for integration by parts, minimisation

```
class BspCurvBasisFuncSet :  public ReadWriteObject {
  order of basis functions
  int ord;
  knots making up the set
  Vector<double> kts;
  vector of basis functions
  Vector<double> BspCurvBasisFunc bs;
  ...
public:
  constructors
  BspCurvBasisFuncSet(const Vector<double>&, int, int);
  ...
  integrate the basis function set
  Vector<double> Integrate(double x1, double x2);
  integrate product with B-spline
  Vector<double> IntegrateProduct1(const BspCurv<double>&, double, double);
  integrate product with general curve
  Vector<double> IntegrateProduct2(const Curve<double>&, double, double);
  Create the matrix of the integrals of the products of derivative
  basis functions
  Matrix<double> CreateMatrixIntegral(int, double, double);
  Create the matrix of the integrals of products of derivatives
  (lev1, lev2) of basis functions
  Matrix<double> CreateMatrixIntegral(int, int, double, double);
  Create the minimisation matrix, symmetrical form
  Matrix<double> CreateMatrixMinimisation(int, double, double);
  Create the minimisation matrix, unsymmetrical form
  Matrix<double> CreateMatrixMinimisation(int, int, double, double);
  ...
  input/output
  virtual void read(std::istream&);
  ...
};
```

class: BspCurv

encapsulates a B-spline curve
algorithms for differentiation, integration, product...

```
template<class T>
class BspCurv :  public Curve<T>
{
  int ord; order of curve
  int num; number of control points
  Vector<T> cpts; vector control points
  Vector<double> kts; vector of knots
  KnotSet kset; KnotSet object
  ...
public:
  constructors
  BspCurv(...);
  ...
  differentiate the curve
  BspCurv<T> Derive(int);
  integrate the curve
  BspCurv<T> Integrate(int);
  form the product B-spline curve
  BspCurv<T> Product(const BspCurv<T>&);
  ...
  point and derivative evaluation
  virtual T operator()(double) const;
  virtual T Derive(int, double) const;
  ...
  get limits
  virtual double GetLeftLimit() const;
  virtual double GetRightLimit() const;
  ...
  read and write
  virtual void read(std::istream&);
  ...
};
```

class: FBspCurv

encapsulates a functional B-spline curve
algorithms for finite element analysis and smoothing

```
class FBspCurv :  public BspCurv<Point1D>
{
public:
  constructors
  FBspCurv();
  build an FBspCurv from a BspCurv
  FBspCurv(const BspCurv<Point1D>&);
  ...
  functions for linear finite element beam problems
  FBspCurv ComputeBeamBending(...)  const;
  ...
  functions to smooth curves
  FBspCurv ComputeSmoothCurve(...)  const;
  ...
};
```

class: PBspCurv3D

encapsulates a parametric B-spline curve
algorithms for finite element analysis and smoothing

```
class PBspCurv3D : public BspCurv<Point3D>
{
public:
  constructors
  PBspCurv3D();
  build an PBspCurv3D from a BspCurv
  PBspCurv3D(const BspCurv<Point3D>& v);
  ...
  functions for curvature analysis
  Point3D ComputeCurvatureVec(double) const;
  double ComputeCurvature(double) const;
  Vector<Point2D> ComputeCurvatureArray(int m) const;
  functions for approximation
  PBspCurv3D ComputeLeastSquares(...);
  ...
  functions to smooth curves
  PBspCurv3D ComputeSmoothCurve(...)  const;
  ...
};
```

class: BezCurv

encapsulates a single segment Bezier curve
algorithms for differentiation, integration, product...

```
template<class T>
class BezCurv :  public Curve<T>
{
  int ord; order of curve
  Vector<T> cpts; vector control points
  Vector<double> kts; vector of knots
  KnotSet kset; KnotSet object
  ...
public:
  constructors
  BezCurv(...);
  ...
  differentiate the curve
  BezCurv<T> Derive(int) const;
  integrate the curve
  BezCurv<T> Integrate(int) const;
  form the product Bezier curve
  BezCurv<T> Product(const BezCurv<T>&) const;
  ...
  point and derivative evaluation
  virtual T operator()(double) const;
  virtual T Derive(int, double) const;
  ...
  get limits
  virtual double GetLeftLimit() const;
  virtual double GetRightLimit() const;
  ...
  read and write
  virtual void read(std::istream&);
  ...
};
```

class: CompBezCurv

encapsulates a multi-segment Bezier curve
algorithms for differentiation, integration, product...

```
template<class T>
class CompBezCurv :  public Curve<T>
{
  int ord; order of curve
  int num; number of control points
  Vector<T> cpts; vector control points
  Vector<double> kts; vector of knots
  KnotSet kset; KnotSet object
  ...
public:
  constructors
  CompBezCurv(...);
  ...
  find a segment number given a parameter
  int FindSegment(double) const;
  extract a given segment index
  BezCurv<T> GetSegment(int) const;
  form the product curve
  CompBezCurv<T> Product(const CompBezCurv<T>&) const;
  ...
  point and derivative evaluation
  virtual T operator()(double) const;
  virtual T Derive(int, double) const;
  ...
  get limits
  virtual double GetLeftLimit() const;
  virtual double GetRightLimit() const;
  ...
  read and write
  virtual void read(std::istream&);
  ...
};
```

class: BspSurfBasisFunc

Stores a single surface basis function
algorithms for integration, evaluation

```
class BspSurfBasisFunc :  public Surf<double>
{
  orders of basis function in u and v
  int ordu, ordv;
  vector of knots in u and v
  Vector<double> ktsu, ktsv;
  BspSurf representation of basis function
  BspSurf<double> b;
  creates the BspSurf representation
  BspSurf<double> CreateBspSurf();
  ...
public:
  constructors
  BspSurfBasisFunc(int, Vector<double>&);
  ...
  integrate the basis function
  double Integrate(double, double, double, double) const;
  ...
  point and derivative evaluation
  virtual double operator()(double, double) const;
  virtual double Derive(int, int, double, double) const;
  ...
  limits in u and v
  virtual double GetLeftLimitU() const;
  ...
  input/output
  virtual void read(std::istream&);
  ...
};
```

class: BspSurfBasisFuncSet

encapsulates a complete set of basis functions given knot sets in u and v
algorithms for integration, evaluation

```
class BspSurfBasisFuncSet :  public ReadWriteObject
{
  order of basis functions in u and v
  int ordu, ordv;
 knots making up the set in u and v
  Vector<double> ktsu, ktsv;
  matrix of basis functions
  Matrix<double> BspSurfBasisFunc bs;
  ...
public:
  constructors
  build a set from knot set objects in u and v
  BspSurfBasisFuncSet(const KnotSet&, const KnotSet&);
  ...
  integrate the basis function set
  Matrix<double> Integrate(double, double, double, double);
  integrate product with B-spline surface
  Matrix<double> IntegrateProduct1(const BspSurf<double>&, double, double,
    double, double);
  integrate product with general surface
  Matrix<double> IntegrateProduct2(const Surf<double>&, double, double,
    double, double);
  ...
  input/output
  virtual void read(std::istream&);
  ...
};
```

class: BspSurf

encapsulates a B-spline surface
algorithms for differentiation, integration, product...

```
template<class T>
class BspSurf :  public Surf<T>
{
  int ordu, ordv; order of surface
  int numu, numv; number of control points
  Matrix<T> cpts; matrix of control points
  Vector<double> ktsu, ktsv; vectors of knots in u and v
  KnotSet ksetu; KnotSet object in u
  KnotSet ksetv; KnotSet object in v
  ...
public:
  constructors
  BspSurf(...);
  ...
  differentiate the surface
  BspSurf<T> Deriv(int, int);
  integrate the surface
  BspSurf<T> Integrate(int, int);
  form the product B-spline surface
  BspSurf<T> Product(const BspSurf<T>&);
  ...
  point and derivative evaluation
  virtual T operator()(double, double) const;
  virtual T Derive(int, int, double, double) const;
  ...
  get limits in u and v
  virtual double GetLeftLimitU() const;
  virtual double GetRightLimitU() const;
  ...
  read and write
  virtual void read(std::istream&);
  ...
};
```

class: FBspSurf

encapsulates a functional B-spline surface
algorithms for finite element analysis, data fitting and smoothing

```
class FBspSurf :  public BspSurf<Point1D>
{
public:
  constructors
  FBspSurf();
  build an FBspSurf from a BspSurf
  FBspSurf(const BspSurf<Point1D>&);
  ...
  functions for finite element analysis, plate bending
  FBspSurf ComputeFiniteElement(...);
  FBspSurf ComputePlateBending(...);
  ...
  least squares fitting
  FBspSurf ComputeLeastSquares(...);

  functions for surface smoothing
  FBspSurf ComputeSmoothSurface(...);
  ...
};
```

class: PBspSurf3D

encapsulates a parametric B-spline surface
algorithms for approximation, analysis and smoothing

```
class PBspSurf3D : public BspSurf<Point3D>
{
public:
  constructors
  PBspSurf3D();
  build a PBspSurf3D from a BspSurf
  PBspSurf3D(const BspSurf<Point3D>&);
  ...
  functions to smooth surfaces
  PBspSurf3D ComputeSmoothSurface(...)  const;
  ...
  algorithms for approximation
  PBspSurf3D ComputeLeastSquares(...);
  functions to perform surface analysis
  double ComputeGaussian(double, double) const;
  double ComputeMean(double, double) const;
  ...
};
```

class: BezSurf

encapsulates a single patch B-spline surface, (Bezier surface)
algorithms for differentiation, integration, product...

```
template<class T>
class BezSurf :  public Surf<T>
{
  int ordu, ordv; order of surface
  Matrix<T> cpts; matrix of control points
  Vector<double> ktsu, ktsv; vectors of knots in u and v
  KnotSet ksetu; KnotSet object in u
  KnotSet ksetv; KnotSet object in v
  ...
public:
  constructors
  BezSurf(...);
  ...
  differentiate the surface
  BezSurf<T> Deriv(int, int);
  integrate the surface
  BezSurf<T> Integrate(int, int);
  form the product B-spline surface
  BezSurf<T> Product(const BezSurf<T>&);
  ...
  point and derivative evaluation
  virtual T operator()(double, double) const;
  virtual T Derive(int, int, double, double) const;
  ...
  get limits in u and v
  virtual double GetLeftLimitU() const;
  virtual double GetRightLimitU() const;
  ...
  read and write
  virtual void read(std::istream&);
  ...
};
```

class: CompBezSurf

encapsulates a multi-patch Bezier surface
algorithms for differentiation, integration, product...

```
template<class T>
class CompBezSurf :  public Surf<T>
{
  int ordu, ordv; order of surface
  int numu, numv; number of control points in u and v
  Matrix<T> cpts; vector control points
  Vector<double> ktsu, ktsv; vector of knots in u and v
  KnotSet ksetu, ksetv; KnotSet objects
  ...
public:
  constructors
  CompBezSurf(...);
  ...
  find a segment number given a parameter
  int FindSegmentU(double) const;
  find a segment number given a parameter
  int FindSegmentV(double) const;
  extract a given patch
  BezSurf<T> GetPatch(int, int) const;
  form the product surface
  CompBezSurf<T> Product(const CompBezSurf<T>&) const;
  ...
  point and derivative evaluation
  virtual T operator()(double, double) const;
  virtual T Derive(int, int, double, double) const;
  ...
  get limits in u and v
  virtual double GetLeftLimitU() const;
  virtual double GetRightLimitU() const;
  ...
  read and write
  virtual void read(std::istream&);
  ...
};
```

class: BspVolBasisFunc

Encapsulates a single B-spline volume basis function
functions to integrate, access plus read/write functions

```
class BspVolBasisFunc :  public Vol<double>
{
  orders of basis function in u, v and w
  int ordu, ordv, ordw;
  vector of knots in u, v and w
  Vector<double> ktsu, ktsv, ktsw;
  BspVol representation of basis function
  BspVol b;
  creates the BspVol representation
  BspVol CreateBspVol();
  ...
public:
  constructors
  BspVolBasisFunc(...);
  ...
  integrate the basis function
  double Integrate(double, double, double, double, double, double);
  ...
  point and derivative evaluation
  virtual T operator()(double, double, double) const;
  virtual T Derive(int, int, int, double, double, double) const;
  ...
  get limits in u, v and w
  virtual double GetLeftLimitU() const;
  virtual double GetRightLimitU() const;
  ...
  read and write
  virtual void read(std::istream&);
  ...
};
```

class: BspVolBasisFuncSet

encapsulates a complete set of B-spline volume basis functions
functions to integrate, access
plus read/write functions

```
class BspVolBasisFuncSet :  public ReadWriteObject
{
  order of basis functions in u, v and w
  int ordu, ordv, ordw;
  knots making up the set in u, v and w
  Vector<double> ktsu, ktsv, ktsw;
  matrix3D of basis functions
  Matrix3D<double> BspVolBasisFunc bs;
  ...
public:
  constructors, build a set from u,v,w knot sets
  BspVolBasisFuncSet(const KnotSet&, const KnotSet&, const KnotSet&);
  ...
  integrate the basis function set
  Matrix<double> Integrate(double, double, double, double,
    double, double);
  integrate product with B-spline volume
  Matrix<double> IntegrateProduct1(const BspVol<double>&, double, double,
    double, double, double, double);
  integrate product with general volume
  Matrix<double> IntegrateProduct2(const Vol<double>&, double, double,
    double, double, double, double);
  ...
  read and write
  virtual void read(std::istream&);
};
```

class: BspVol

encapsulates a B-spline volume
algorithms for differentiation, integration, product...

```
template<class T>
class BspVol :  public Vol<T>
{
  int ordu, ordv, ordw; order of surface
  int numu, numv, numw; number of control points
  Matrix3D<T> cpts; matrix3D of control points
  Vector<double> ktsu, ktsv, ktsw; vectors of knots in u and v
  KnotSet ksetu; KnotSet object in u
  KnotSet ksetv; KnotSet object in v
  KnotSet ksetw; KnotSet object in w
  ...
public:
  constructors
  BspVol(...);
  ...
  differentiate the volume
  BspVol<T> Deriv(int, int, int);
  integrate the volume
  BspVol<T> Integrate(int, int, int);
  form the product B-spline volume
  BspVol<T> Product(const BspVol<T>&);
  ...
  point and derivative evaluation
  virtual T operator()(double, double, double) const;
  virtual T Derive(int, int, int, double, double, double) const;
  ...
  get limits in u, v and w
  virtual double GetLeftLimitU() const;
  virtual double GetRightLimitU() const;
  ...
  read and write
  virtual void read(std::istream&);
  ...
};
```

class: FBspVol

encapsulates a functional B-spline volume
algorithms for finite element analysis, fitting and smoothing

```
class FBspVol :  public BspVol<Point1D>
{
public:
  constructors
  build an FBspVol from a BspVol
  FBspVol(const BspVol<Point1D>&);
  ...
  functions for analysis of linear elastic solids
  FBspVol ComputeFiniteElement(...);
  ...
  functions for smoothing functional volumes
  FBspVol ComputeSmoothVolume(...);
  ...
};
```

class: PBspVol3D

encapsulates a parametric B-spline volume
algorithms for analysis, approximation and smoothing

```
class PBspVol3D : public BspVol<Point3D>
{
public:
  constructors
  build an BspVol3D from a BspVol
  PBspVol3D(const BspVol<Point3D>& v);
  ...
  functions for analysis of linear elastic solids
  PBspvol3D ComputeFiniteElement(...);
  ...
  functions for data approximation
  PBspVol3D ComputeLeastSquares(...);
  ...
  functions for smoothing parametric volumes
  PBspVol3D ComputeSmoothVolume(...);
  ...
};
```

class: BezVol

encapsulates a single hyperpatch B-spline volume, (a Bezier volume)
algorithms for differentiation, integration, product...

```
template<class T>
class BezVol :  public Vol<T>
{
  int ordu, ordv, ordw; order of surface
  Matrix3D<T> cpts; matrix3D of control points
  Vector<double> ktsu, ktsv, ktsw; vectors of knots in u and v
  KnotSet ksetu; KnotSet object in u
  KnotSet ksetv; KnotSet object in v
  KnotSet ksetw; KnotSet object in w
  ...
public:
  constructors
  BezVol(...);
  ...
  differentiate the volume
  BezVol<T> Deriv(int, int, int);
  integrate the volume
  BezVol<T> Integrate(int, int, int);
  form the product B-spline volume
  BezVol<T> Product(const BezVol<T>&);
  ...
  point and derivative evaluation
  virtual T operator()(double, double, double) const;
  virtual T Derive(int, int, int, double, double, double) const;
  ...
  get limits in u, v and w
  virtual double GetLeftLimitU() const;
  virtual double GetRightLimitU() const;
  ...
  read and write
  virtual void read(std::istream&);
  ...
};
```

class: Math

functions for solving linear systems
functions for multiplying and manipulating Vector/Matrix/Matrix3D objects

```
class Math
{
  combinatorial coefficient and factorial
  static double Comb(int, int);
  static Fact(int);
  ...
  Guassian quadrature
  static double Integral(Curve<T>&);
  static double Integral(Surf<T>&);
  static double Integral(Vol<T>&);
  ...
  kronecker products and associated functions
  static Matrix<T> MatrixFromVector(const Vector<T>&, int, int);
  static Matrix3D<T> Matrix3DFromVector(const Vector<T>&, int, int, int);
  static Vector<T> CreateKroneckerVector(const Matrix<T>&);
  static Vector<T> CreateKroneckerVector(const Matrix3D<T>&);
  static Matrix<double> KroneckerProduct(const Matrix<double>&, const Matrix<double>&);
  ...
  Vector/Matrix/Matrix3D multiplication operations
  static Matrix<T> VM3DI_1(const Vector<double>&, const Matrix3D<T>&);
  static Matrix<T> VM3DI_2(const Vector<double>&, const Matrix3D<T>&);
  ...
  static Matrix3D<T> MM3DI_1(const Matrix<double>&, const Matrix3D<T>&);
  static Matrix3D<T> MM3DI_2(const Matrix<double>&, const Matrix3D<T>&);
  ...
  elimination algorithms for reducing constrained systems
  static Matrix<double> EliminateMat(...);
  static Matrix<double> EliminateVec(...);
  static Matrix<double> EliminateMVariables(...);
  static Vector<T> EliminateVVariables(...);
  ...
  solve linear system
  static Vector<T> Solve(const Matrix<double>&, const Vector<T>&);
  ...
};
```

Appendix C

Inheritance Structure Diagrams

The principle inheritance relationships amongst the classes detailed in appendix B are presented here.

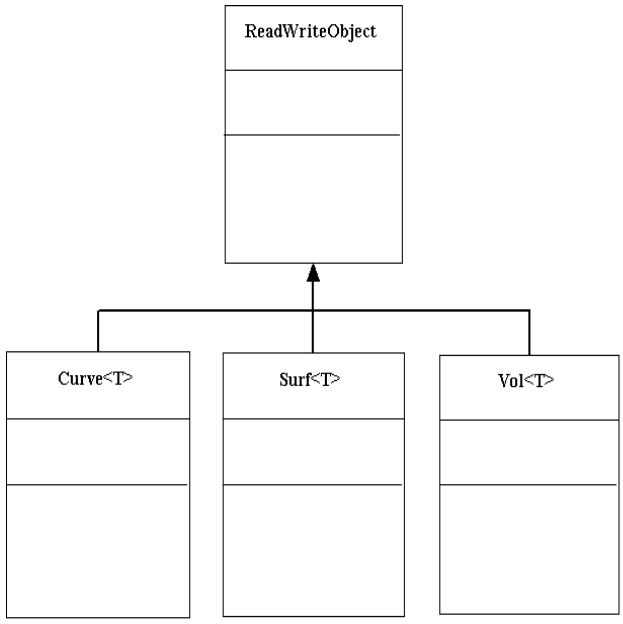


Figure C.1: Inheritance structure for Curve/Surf/Vol classes

Figure C.2: Inheritance structure for curve entities

Figure C.3: Inheritance structure for surface entities

Figure C.4: Inheritance structure for volume entities

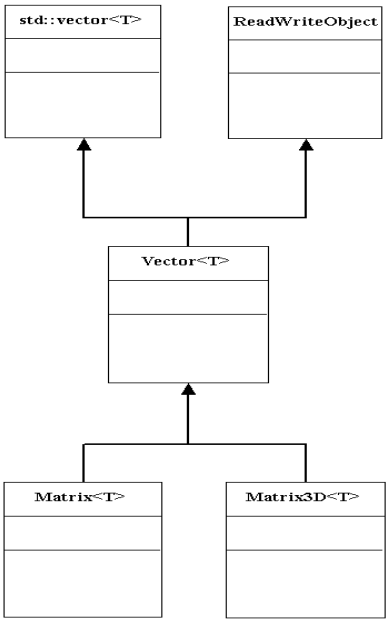


Figure C.5: Inheritance structure for vector/matrix/matrix3D

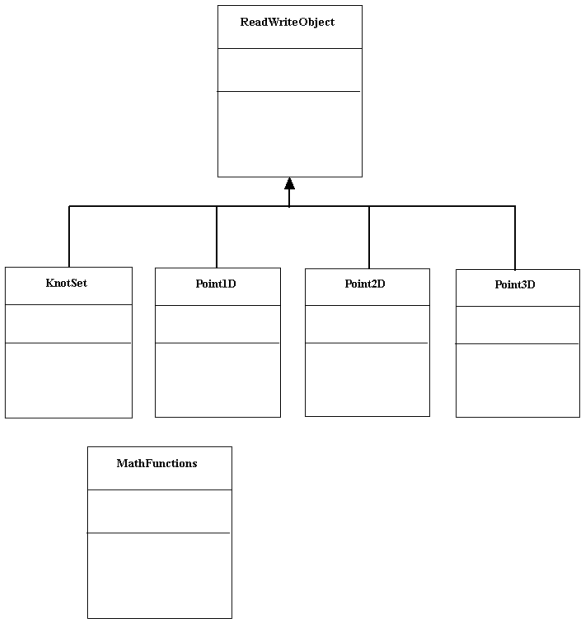


Figure C.6: Ancillary classes

Appendix D

Kronecker Product and Related Functions

The code for the functions used to compute the Kronecker product of two matrices and the associated vector conversion routines covered in Chapter 2 of the report is presented here.

class: Math

Function: MatrixFromVector
Algorithm: to construct a Matrix from a long vector
Parameters: Vector, number of rows cols of the matrix

```
Matrix<T> Math::MatrixFromVector(const Vector<T>& vec, int numu, int numv)
{
  Matrix<T> mat(numu, numv);

  int count=0;

  for (int j=0; j<numv; j++)
    for (int i=0; i<numu; i++) {
      mat[i][j] = vec[count];
      count++;
    }
  return mat;
}
```

class: Math

Function: Matrix3DFromVector
Algorithm: to construct a Matrix3D from a long vector
Parameters: Vector, number of rows, cols and layers of the matrix

```
Matrix3D<T> Math::Matrix3DFromVector(const Vector<T>& vec, int numu, int numv, int numw)
{
  Matrix<T> mat(numu, numv, numw);

  int count=0;

  for (int k=0; k<numw; k++)
    for (int j=0; j<numv; j++)
      for (int i=0; i<numu; i++) {
        mat[k][i][j] = vec[count];
        count++;
      }
  return mat;
}
```

class: Math

Function: CreateKroneckerVector
Algorithm: to construct a Vector from a Matrix3D
Parameters: Matrix3D

```
Vector<T> Math::CreateKroneckerVector(const Matrix3D<T>& mat)
{
  int num = mat.GetNumRows()*GetNumCols()*GetNumLayers();

  Vector<T> vec(numu);

  int count=0;
  for (int k=0; k<mat.GetNumLayers(); k++)
    for (int j=0; j<mat.GetNumCols(); j++)
      for (int i=0; i<mat.GetNumRows(); i++) {
        vec[count] = mat[k][i][j];
        count++;
      }
  return vec;
}
```

class: Math

Function: KroneckerProduct
Algorithm: to construct the Kronecker product
Parameters: Two matrices, mat1, mat2

```
Matrix<double> Math::KroneckerProduct(const Matrix<double>& mat1, const Matrix<double>& mat2)
{
  Matrix<double> mat(mat.GetNumRows()*mat2.GetNumRows(),mat1.GetNumCols()*mat2.GetNumCols());
  int c1=0;
  int c2=0;
  int rowStart, colStart;

  for (int i=0; i<mat1.GetNumRows(); i++) {
    rowStart = i * mat2.GetNumRows();
    c1 = rowStart;
    for (int j=0; j<mat1.GetNumCols(); j++) {
      colStart = j*mat2.GetNumCols();
      c2 = colStart;
      for (int k=0; k<mat2.GetNumRows(); k++) {
        for (int l=0; l<mat2.GetNumCols(); l++) {
          mat[c1][c2] = mat1[i][j]*mat2[k][l];
          c2++;
        }
        c2 = colStart;
        c1++;
      }
      c1 = rowStart;
    }
  }
  return mat;
}
```

class: Math

Function: VM3DI_1
Algorithm: to compute the contraction of a Matrix3D with a Vector object
Parameters: Matrix3D and a Vector

```
Matrix<double> Math::VM3DI_1(Matrix3D<double>& D, Vector<double>& P)
{
  double sum=0.0;
  int m = D.GetNumRows();
  int n = D.GetNumCols();
  int p = D.GetNumLays();

  Matrix<double> T(n,p);

  for (int k=0; k<p; k++)
    for (int j=0; j<n; j++) {
      for (int i=0; i<m; i++)
        sum+=D[i][j][k]*P[i];
      T[j][k]=sum;
      sum=0.0
    }

  return T;
}
```

class: Math

Function: VM3DI_2
Algorithm: to compute the contraction of a Matrix3D with a Vector object
Parameters: Matrix3D and a Vector

```
Matrix<double> Math::VM3DI_2(Matrix3D<double>& D, Vector<double>& P)
{
  double sum=0.0;
  int m = D.GetNumRows();
  int n = D.GetNumCols();
  int p = D.GetNumLays();
  Matrix<double> T(n,p);

  for (int j=0; j<n; j++)
    for (int k=0; k<p; k++) {
      for (int i=0; i<m; i++)
        sum+=D[i][j][k]*P[i];
      T[j][k]=sum;
      sum=0.0;
    }

  return T;
}
```

class: Math

Function: MM3DI_1
Algorithm: to compute the contraction of a Matrix3D with a Matrix object
Parameters: Matrix3D and a Matrix

```
Matrix<double> Math::MM3DI_1(Matrix3D<double>& D, Matrix<double>& A)
{
  double sum=0.0;
  int m = D.GetNumRows();
  int n = D.GetNumCols();
  int p = D.GetNumLays();
  int c = A.GetNumCols();
  Matrix3D<double> T(c,n,p);

  for (int k=0; k<p; k++)
    for (int l=0; l<c; l++)
      for (int j=0; j<n; j++) {
        for (int i=0; i<m; i++)
          sum+=D[i][j][k]*A[i][l];
        T[l][j][k]=sum;
        sum=0.0;
      }

  return T;
}
```

class: Math

Function: MM3DI_2
Algorithm: to compute the contraction of a Matrix3D and a Matrix object
Parameters: Matrix3D and a Matrix

```
Matrix<double> Math::MM3DI_2(Matrix3D<double>& D, Matrix<double>& A)
{
  double sum=0.0;
  int m = D.GetNumRows();
  int n = D.GetNumCols();
  int p = D.GetNumLays();
  int c = A.GetNumCols();
  Matrix3D<double> T(c,n,p);

  for (int j=0; j<n; j++)
    for (int l=0; l<c; l++)
      for (int k=0; k<p; k++) {
        for (int i=0; i<m; i++)
          sum+=D[i][j][k]*A[i][l];
        T[l][j][k]=sum;
        sum=0.0;
      }

  return T;
}
```

Appendix E

B-spline Derivative and Knot Insertion Formulae

The various B-spline formulae used in Chapter 2 are given brief derivations in this appendix. Further details can be found in [10] and [40].

E.1 Derivative Formula

For a B-spline curve given by

$$f(t) = \sum_{i=1}^{n} d_i N_{i,k}(t)$$

we have the result

$$f'(t) = \sum_{i} d_i N'_{i,k}(t) = (k-1) \sum_{i} d_i^{(1)} N_{i,k-1}(t),$$

where

$$d_i^{(1)} = (d_i - d_{i-1})/(t_{i+k-1} - t_i).$$

To show this we recall the definition of the normalised B-spline basis functions in terms of the divided difference of truncated power functions. The kth divided difference of a function g on the points $t_i, \ldots, t_{i+k}$ is given by

$$g[t_i, t_{i+1}, \ldots, t_{i+k}] = \frac{g[t_{i+1}, \ldots, t_{i+k}] - g[t_i, \ldots, t_{i+k-1}]}{t_{i+k} - t_i} = \sum_{j=i}^{i+k} \frac{g(t_j)}{w'(t_j)}, \tag{E.1}$$

where

$$w'(t_j) = \prod_{l=i, l\neq j}^{i+k} (t_j - t_l).$$

If

$$\phi_k(s;t) = (s-t)_+^{k-1} = \begin{cases} (s-t)^{k-1} & s \geq t, \\ 0 & s < t \end{cases}$$

then the *standardized* B-spline basis function $M_{i,k}(t)$ is defined to be the kth divided difference of $\phi_k(s;t)$ in s on $t_i, \ldots, t_{i+k}$ for fixed t:

$$M_{i,k}(t) = \phi_k[t_i, \ldots, t_{i+k}; t] = \frac{\phi_k[t_{i+1}, \ldots, t_{i+k}; t] - \phi_k[t_i, \ldots, t_{i+k-1}; t]}{t_{i+k} - t_i}. \tag{E.2}$$

The *normalized* B-spline basis function $N_{i,k}(t)$ is defined as

$$N_{i,k}(t) = (t_{i+k} - t_i) M_{i,k}(t). \tag{E.3}$$

To differentiate $f(t)$ we note that from the divided difference recursion formula E.1 we have,

$$\begin{aligned} N_{i,k}^{(1)}(t) &= (d/dt)\Big((t_{i+k} - t_i)\phi_k[t_i, \ldots, t_{i+k}; t]\Big) \\ &= (d/dt)\Big(\phi_k[t_{i+1}, \ldots, t_{i+k}; t] - \phi_k[t_i, \ldots, t_{i+k-1}; t]\Big) \\ &= -(k-1)\Big(M_{i+1,k-1}(t) - M_{i,k-1}(t)\Big). \end{aligned}$$

Hence

$$\begin{aligned} f^{(1)}(t) &= (k-1)\sum_i d_i \Big(M_{i,k-1}(t) - M_{i+1,k-1}(t)\Big) \\ &= (k-1)\sum_i d_i^{(1)} N_{i,k-1}(t), \end{aligned}$$

where

$$d_i^{(1)} = (d_i - d_{i-1})/(t_{i+k-1} - t_i).$$

E.2 Basis Function Recursion

To show the B-spline basis function recursive formula

$$\begin{aligned} N_{i,1}(t) &= \begin{cases} 1 & t_i \leq t < t_{i+1} \\ 0 & \text{otherwise} \end{cases} \\ N_{i,k}(t) &= \frac{t - t_i}{t_{i+k-1} - t_i} N_{i,k-1}(t) + \frac{t_{i+k} - t}{t_{i+k} - t_{i+1}} N_{i+1,k-1}(t), \quad k \geq 2. \end{aligned} \tag{E.4}$$

we first recall Leibniz's formula for divided differences: *If* $f(x) = g(x)h(x)$ *for all* x *i.e* f *is the product of two functions, then*

$$f[t_i, \ldots, t_{i+k}] = \sum_{r=i}^{i+k} g[t_i, \ldots, t_r]h[t_r, \ldots, t_{i+k}].$$

If we apply this formula to the function

$$h(s) = \phi_k(s;t) = \phi_{k-1}(s;t)(s-t)$$

we get

$$\phi_k[t_i, \ldots, t_{i+k}; t] = \phi_{k-1}[t_i, \ldots, t_{i+k-1}; t].1 + \phi_{k-1}[t_i, \ldots, t_{i+k}; t].(t_{i+k} - t),$$

where all divided differences of $(s-t)$ of order 2 and higher vanish. Hence, using E.2 and E.1 we have

$$\begin{aligned} M_{i,k}(t) &= M_{i,k-1}(t) + \frac{t_{i+k} - t}{t_{i+k} - t_i}(M_{i+1,k-1}(t) - M_{i,k-1}(t)), \\ &= \frac{t - t_i}{t_{i+k} - t_i} M_{i,k-1}(t) + \frac{t_{i+k} - t}{t_{i+k} - t_i} M_{i+1,k-1}(t). \end{aligned}$$

This identity states that for $t_i < t < t_{i+k}$, $M_{i,k}(t)$ is strictly a *convex combination* of the numbers $M_{i,k-1}(t)$ and $M_{i+1,k-1}(t)$ (similarly for the $N_{i,k}(t)$). Since $M_{i,1}(t)$ is positive for $t_i \leq t < t_{i+1}$ and zero otherwise, it follows by induction that, for $k > 1$, $M_{i,k}(t)$ is positive for $t_i < t < t_{i+k}$ and zero otherwise.

E.3 Knot Insertion Formula

We wish to derive the relationship given by 2.36 in chapter 2 between the two sets of B-spline basis functions $\{N_{i,k}(t)\}$ and $\{\hat{N}_{i,k}(t)\}$. The second set differs from the first in that it has had one extra knot inserted at $\hat{t} = t_{p+1}$. Using E.2 and E.3 we have the following result,

$$(\hat{t} - t_i)\phi_k[t_i, \ldots, t_{i+k-1}, \hat{t}; t] + (t_{i+k} - \hat{t})\phi_k[\hat{t}, t_{i+1}, \ldots, t_{i+k}; t]$$

$$-(t_{i+k} - t_i)\phi_k[t_i, \ldots, t_{i+k}; t] = 0 \quad \text{for } i = p-k+1, \ldots, l.$$

Hence

$$(t_{i+k} - t_i)M_{i,k}(t) = (\hat{t} - t_i)M^1_{i,k}(t) + (t_{i+k} - \hat{t})M^1_{i+1,k}(t),$$

or,

$$N_{i,k}(t) = \frac{\hat{t} - t^1_i}{t^1_{i+k} - t^1_i} N^1_{i,k}(t) + \frac{t^1_{i+k+1} - \hat{t}}{t^1_{i+k+1} - t^1_{i+1}} N^1_{i+1,k}(t)\,, \quad i = p-k+1, \ldots, l$$

as was to be shown. Note that if t_{p+1} occurs with multiplicity s then $\hat{t} = t^1_{p+1} = \ldots = t^1_{p+s+1}$ and so $N^1_{i,k}(t) = N^1_{i,k}(t)$ for $i \leq p-k+s$ and $N^1_{i,k}(t) = N_{i+1,k}(t)$ for $i \geq p+1$.

E.4 Schoenberg-Whitney Conditions

We assume we have a set of B-spline basis functions $(N_{i,k}(t))_{i=1}^n$ the n over the knot vector $(t_i)_{i=1}^{n+k}$ and that $(\tau_i)_{i=1}^m$ a set of parameter values. Then the matrix $\mathbf{B}$ for curve fitting given by

$$\mathbf{B} = \begin{pmatrix} N_{1,k}(\tau_1) & \ldots & N_{n,k}(\tau_1) \\ \vdots & \ddots & \vdots \\ N_{1,k}(\tau_m) & \ldots & N_{n,k}(\tau_m) \end{pmatrix}$$

is of full column rank if and only if there exists $(\eta)_{i=1}^n$ contained in $(\tau_i)_{i=1}^m$ such that

$$N_{i,k}(\eta_i) \neq 0, \quad i = 1, \ldots, n,$$

or, equivalently,

$$t_i < \eta_i < t_{i+k}, \quad i = 1, \ldots, n.$$

In the case of k multiple knots over the definition domain of the $N_{i,k}(t)$ the ith condition for η_i can be relaxed to

$$t_i \leq \eta_i < t_{i+k}, \quad t_i = \ldots = t_{i+k-1} < t_{i+k}, \quad t_i < \eta_i \leq t_{i+k}, \quad t_i < t_{i+1} = \ldots = t_{i+k}$$

E.4.1 Surface case

If $(N_{i,k}(u))_{i=1}^p, (N_{j,l}(v))_{j=1}^q$ are the respective u, v basis function sets defined over knot vectors $(u_i)_{i=1}^{p+k}, (v_j)_{j=1}^{q+l}$, and parameter values $(\tau_i)_{i=1}^m, (\mu_j)_{j=1}^n$ for u and v are given, then there must exist $(\gamma_i)_{i=1}^p$ contained in $(\tau_i)_{i=1}^m$ and $(\zeta_i)_{i=1}^q$ contained in $(\mu_j)_{j=1}^n$ such that

$$N_{i,k}(\gamma_i) N_{j,l}(\zeta_j) \neq 0, \quad i = 1, 2, \ldots, p; \quad j = 1, \ldots, q;$$

or equivalently,

$$u_i < \gamma_i < u_{i+k}, \quad v_j < \zeta_j < v_{j+l}, \quad i = 1, \ldots, p; \quad j = 1, \ldots, q.$$

Again the conditions can be relaxed in the same way as for curves if there are k or l fold knots in the respective basis functions. The volume case generalises this result in the natural way.

Appendix F

Curvature Formulae

The various curvature formulae for curves, surfaces and volumes used in Chapter 4 are given in this appendix (without formal proofs). Further details can be found in standard differential geometry texts.

F.1 Curve Formulae

F.1.1 Functional curves

The curvature is given by the second derivative with respect to arc length (s). For a functional curve $x(t)$ we have

$$\kappa(s) = \frac{d^2x}{ds^2},$$

and in terms of the actual parameter t:

$$ds = \sqrt{(1 + [x'(t)]^2)}dt, \quad \kappa(t) = \frac{x''(t)}{(1 + [x'(t)]^2)^{\frac{3}{2}}}.$$

F.1.2 Parametric curves

For a parametric curve $\mathbf{x}(t) = (x(t), y(t))$ we have

$$\kappa(t) = \frac{|\ \mathbf{x}'(t) \times \mathbf{x}''(t)\ |}{|\ \mathbf{x}'(t)\ |^3}.$$

For the 2D planar case this gives

$$\kappa(t) = \frac{x'(t)y''(t) - y'(t)x''(t)}{([x'(t)]^2 + [y'(t)]^2)^{\frac{3}{2}}}.$$

F.2 Surface Formulae

F.2.1 Parametric surfaces

The curvature of a parametric surface is defined in terms of the so-called first and second fundamental forms. They can be defined as follows.

For a parametric surface $\mathbf{x}(u,v) = (x(u,v), y(u,v), z(u,v))$ the unit normal is given by

$$\mathbf{N} = \frac{\mathbf{x}_u \times \mathbf{x}_v}{\mid \mathbf{x}_u \times \mathbf{x}_v \mid}.$$

The moving frame $\{\mathbf{x}_u, \mathbf{x}_v, \mathbf{N}\}$ is called the *Guass frame*. The tangent plane at a point (u,v) is given by

$$T_u\mathbf{x} = \{\mathbf{x}(u,v,w) + \lambda\mathbf{x}_u(u,v,w) + \mu\mathbf{x}_v(u,v,w) \mid (\lambda,\mu) \in R^2\}.$$

The bilinear form on $T_u\mathbf{x}$ given by the inner product of R^3 is called the first fundamental form of the surface. It has the following matrix form with respect to the basis $(\mathbf{x}_u, \mathbf{x}_v)$:

$$\mathbf{G} = (g_{ij}) = \begin{pmatrix} g_{11} & g_{12} \\ g_{21} & g_{22} \end{pmatrix} = \begin{pmatrix} \mathbf{x}_u \cdot \mathbf{x}_u & \mathbf{x}_u \cdot \mathbf{x}_v \\ \mathbf{x}_v \cdot \mathbf{x}_u & \mathbf{x}_v \cdot \mathbf{x}_v \end{pmatrix}.$$

The first fundamental form is symmetric and geometrically invariant.

The second fundamental form of the surface $\mathbf{x}$ is given by a matrix $\mathbf{H}$ where

$$\mathbf{H} = (h_{ij}) = \begin{pmatrix} h_{11} & h_{12} \\ h_{21} & h_{22} \end{pmatrix} = \begin{pmatrix} \mathbf{N} \cdot \mathbf{x}_{uu} & \mathbf{N} \cdot \mathbf{x}_{uv} \\ \mathbf{N} \cdot \mathbf{x}_{vu} & \mathbf{N} \cdot \mathbf{x}_{vv} \end{pmatrix}.$$

The matrix $\mathbf{H}\mathbf{G}^{-1}$ is symmetric and has two real eigenvalues κ_1, κ_2, with corresponding eigenvectors. κ_1 and κ_2 are called the *principal curvatures* of the surface $\mathbf{x}$. Geometrically these curvatures are the magnitudes of the maximum and minimum of all possible normal curvatures[1]. The product of the principle curvatures $K = \kappa_1\kappa_2 = det(\mathbf{H})/det(\mathbf{G})$ is called the *Gaussian curvature*, and the average $H = (1/2)(\kappa_1 + \kappa_2)$ is called the *mean curvature*.

An alternative way but related of looking at this is as follows. For each point on the surface we have a map to the normal vector at that point:

$$\mathbf{p} = \mathbf{x}(u,v) \longrightarrow \mathbf{N}_{(u,v)}.$$

[1]The normal curvature at a point on a surface in a direction specified by a tangent vector is determined from the intersection curve of the surface with the plane spanned by the surface normal and the tangent vector

This can be viewed as a map from the surface $\mathbf{x}$ to the unit sphere S^2 and in this form it is called the *Guass map*. The derivative of the Guass map, $d\mathbf{N_p}$ at a point $\mathbf{p}$ on the surface measures the variation of the normal near $\mathbf{p}$, that is how the surface 'curves' near $\mathbf{p}$. The Jacobian matrix of $-d\mathbf{N_p}$ with respect to the basis $(\mathbf{x}_u, \mathbf{x}_v)$ is equal to the matrix $\mathbf{HG}^{-1}$. This matrix is often called the *shape matrix* of the surface and generalises to higher dimensions.

F.2.2 Functional surfaces

We can write a functional surface $x = x(u, v)$ parametrically as $\mathbf{x}(u, v) = (u, v, x(u, v))$. Using this form we obtain the following formulae for the coefficients of the first and second fundamental forms:

$$\mathbf{x}_u \cdot \mathbf{x}_v = 1 + x_u^2, \quad \mathbf{x}_u \cdot \mathbf{x}_v = x_u x_v, \quad \mathbf{x}_v \cdot \mathbf{x}_v = 1 + x_v^2$$

$$\mathbf{N} = \frac{\mathbf{x}_u \times \mathbf{x}_v}{\mid \mathbf{x}_u \times \mathbf{x}_v \mid} = \frac{(-x_u, -x_v, 1)}{\sqrt{1 + x_u^2 + x_v^2}}.$$

and

$$\mathbf{N} \cdot \mathbf{x}_{uu} = \frac{x_{uu}}{\sqrt{1 + x_u^2 + x_v^2}}, \quad \mathbf{N} \cdot \mathbf{x}_{uv} = \frac{x_{uv}}{\sqrt{1 + x_u^2 + x_v^2}}, \quad \mathbf{N} \cdot \mathbf{x}_{vv} = \frac{x_{vv}}{\sqrt{1 + x_u^2 + x_v^2}}.$$

From these one obtains the mean and Gaussian curvature:

$$H = \frac{(1 + x_u^2)x_{vv} - 2x_u x_v x_{uv} + (1 + x_v^2)x_{uu}}{(1 + x_u^2 + x_v^2)^{3/2}}, \quad K = \frac{x_{uu}x_{vv} - x_{uv}^2}{(1 + x_u^2 + x_v^2)^2}.$$

F.3 Volume Formulae

F.3.1 Functional volumes

We can write a functional volume $x(u, v, w)$ in the parametric 4D form $\mathbf{x}(u, v, w) = (u, v, w, x(u, v, w))$. This form can be used to calculate curvatures and other geometrical attributes. A basis for the tangent plane is given by the vectors

$$\mathbf{x}_u = (1, 0, 0, x_u), \quad \mathbf{x}_v = (0, 1, 0, x_v), \quad \mathbf{x}_w = (0, 0, 1, x_w).$$

The Gauss map is then given by

$$\mathbf{N_p} = \frac{(-x_u, -x_v, -x_w, 1)}{\sqrt{1 + x_u^2 + x_v^2 + x_w^2}}.$$

The derivative of the Gauss map with respect to the tangent plane basis vectors above is such that

$$\begin{aligned} \mathbf{N}_u &= a_{11}x_u + a_{12}x_v + a_{13}x_w \\ \mathbf{N}_v &= a_{21}x_u + a_{22}x_v + a_{23}x_w \\ \mathbf{N}_w &= a_{31}x_u + a_{32}x_v + a_{33}x_w \end{aligned} \tag{F.1}$$

where $\mathbf{N}_u = \mathbf{N}'_{\mathbf{p}}(\mathbf{x}_u)$. Since $\mathbf{N} \cdot \mathbf{x}_u = 0$ we have, taking derivatives,

$$\begin{aligned} \mathbf{N}_u \cdot \mathbf{x}_u &= -\mathbf{N} \cdot \mathbf{x}_{uu} \\ \mathbf{N}_v \cdot \mathbf{x}_v &= -\mathbf{N} \cdot \mathbf{x}_{vv} \\ \mathbf{N}_z \cdot \mathbf{x}_w &= -\mathbf{N} \cdot \mathbf{x}_{zz} \end{aligned}$$

This together with the expressions for $\mathbf{x}_u, \mathbf{x}_v, \mathbf{x}_w$ and $\mathbf{N}$ gives us the equation

$$\text{Hess}(x) = \mathbf{AM},$$

where $\mathbf{A} = (a_{ij})$ is the shape matrix, $\text{Hess}(x)$ is the Hessian matrix:

$$\text{Hess}(x) = \begin{pmatrix} x_{uu} & x_{uv} & x_{uw} \\ x_{uv} & x_{vv} & x_{vw} \\ x_{uw} & x_{vw} & x_{ww} \end{pmatrix},$$

and $\mathbf{M}$ is the matrix

$$\mathbf{M} = -\frac{1}{\sqrt{1 + x_u^2 + x_v^2 + x_w^2}} \begin{pmatrix} 1 + x_u^2 & x_u x_v & x_u x_w \\ x_u x_v & 1 + x_v^2 & x_v x_w \\ x_u x_w & x_v x_w & 1 + x_w^2 \end{pmatrix}.$$

Hence in the volumetric case the shape matrix is given by $\text{Hess}(x)\mathbf{M}^{-1}$. The principal curvatures, $\kappa_1, \kappa_2, \kappa_3$ are the eigenvalues of this matrix and the Guassian and mean curvatures are given by

$$K = \kappa_1\kappa_2\kappa_3, \quad H = (\kappa_1 + \kappa_2 + \kappa_3)/3.$$

In addition there is one further curvature measure obtained by taking the products in pairs of the principal curvatures (sometimes called the scalar curvature)

$$M = \kappa_1\kappa_2 + \kappa_1\kappa_3 + \kappa_2\kappa_3.$$

In explicit terms of the partial derivatives these curvatures are given by the following expressions:

$$K = \frac{x_{uu}x_{vv}x_{ww} + 2x_{uv}x_{uw}x_{vw} - x_{uu}x_{vw}^2 - x_{ww}x_{uv}^2 - x_{vv}x_{uw}^2}{(1 + x_u^2 + x_v^2 + x_w^2)^{\frac{5}{2}}}$$

$$3H = \frac{(1 + x_u^2 + x_w^2)x_{vv} + (1 + x_v^2 + x_w^2)x_{uu} + (1 + x_u^2 + x_v^2)x_{ww}}{(1 + x_u^2 + x_v^2 + x_w^2)^{\frac{3}{2}}}$$
$$- \frac{2x_u x_v x_{uv} + 2x_u x_w x_{uw} + 2x_v x_w x_{vw}}{(1 + x_u^2 + x_v^2 + x_w^2)^{\frac{3}{2}}}$$

$$M = \frac{(1 + x_u^2)(x_{vv}x_{ww} - x_{vw}^2) + (1 + x_w^2)(x_{uu}x_{vv} - x_{uv}^2) + (1 + x_v^2)(x_{uu}x_{ww} - x_{uw}^2)}{(1 + x_u^2 + x_v^2 + x_w^2)^2} +$$
$$\frac{2x_u x_v(x_{uw}x_{vw} - x_{uv}x_{ww}) + 2x_u x_w(x_{uv}x_{vw} - x_{uw}x_{vv}) + 2x_v x_w(x_{uv}x_{uw} - x_{vw}x_{uu})}{(1 + x_u^2 + x_v^2 + x_w^2)^2}$$

F.3.2 Volume level surface curvature

An alternative approach for dealing with curvature is to consider level surfaces (also called isosurfaces) of the functional volume $x(u, v, w)$. A level surface is given by

$$S = \{(u, v, w) \mid x(u, v, w) = k\},$$

where k is arbitrary. The surface normal of an isosurface is given by the normalised gradient vector:

$$\mathbf{N}_{(u,v,w)} = \frac{(x_u, x_v, x_w)}{\sqrt{x_u^2 + x_v^2 + x_w^2}}$$

The description of the curvature of the isosurface is given by the shape matrix of the function x. This is given by the projection of the Gauss map onto the tangent plane of the isosurface. For the derivative of the Gauss map given by

$$d\mathbf{N}_\mathbf{p} = (\mathbf{N}_u, \mathbf{N}_v, \mathbf{N}_w),$$

the normal projection operator $\mathbf{P}$ is defined as

$$\mathbf{P} = \frac{1}{(x_u^2 + x_v^2 + x_w^2)} \begin{pmatrix} x_u^2 & x_u x_v & x_u x_w \\ x_u x_v & x_v^2 & x_v x_w \\ x_u x_w & x_v x_w & x_w^2 \end{pmatrix}.$$

The tangential projection operator is $\mathbf{T} = \mathbf{I} - \mathbf{P}$ and the shape matrix is given by

$$-d\mathbf{N_p}\mathbf{T} = \mathbf{T}\text{Hess}(x)\mathbf{T}.$$

This matrix has three real eigenvalues, $\kappa_1, \kappa_2, \kappa_3$ where $\kappa_3 = 0$. The corresponding eigenvectors are the principle directions in the tangent plane and normal respectively. Concentrating on the principal curvatures for the isosurface, the mean and Gaussian curvatures are given by

$$H = \frac{\kappa_1 + \kappa_2}{2}, \quad K = \kappa_1\kappa_2$$

In terms of partial derivatives the expressions are

$$K = \Big(x_w^2(x_{uu}x_{vv} - x_{uv}^2) + x_v^2(x_{uu}x_{ww} - x_{uw}^2) + x_u^2(x_{vv}x_{ww} - x_{vw}^2) + 2\big[x_u x_v(x_{uw}x_{vw} - x_{uv}x_{ww})$$

$$+ x_u x_w(x_{uv}x_{vw} - x_{uw}x_{vv}) + x_v x_w(x_{uv}x_{uw} - x_{vw}x_{uu})\big] \Big) / (x_u^2 + x_v^2 + x_w^2)^2$$

$$H = \frac{\Big(x_u^2(x_{vv} + x_{ww}) + x_v^2(x_{uu} + x_{ww}) + x_w^2(x_{uu} + x_{vv}) - 2x_u x_v x_{uv} - 2x_u x_w x_{uw} - 2x_v x_w x_{vw} \Big)}{2(x_u^2 + x_v^2 + x_w^2)^{\frac{3}{2}}}.$$

Bibliography

[1] M Alhanaty and M Bercovier. Curve and Surface Fitting and Design by Optimal Control Methods. *CAGD*, (33):167–182, 2001.

[2] H Antes. Bicubic Fundamental Splines in Plate Bending. *International Journal for Numerical Methods in Engineering*, 8:503–511, 1974.

[3] M Bro-Neilson. Modelling Elasticity in Solids using Active Cubes - Applications to Simulated Operations. In *Proceedings Computer Vision, Virtual Reality and Robotics in Medicine*, pages 1–13, 1995.

[4] A Barr. Global and Local Deformations of Solid Primitives. *Computer Graphics*, 18(3):21–30, 1984.

[5] J Beck, R Farouki, and J K Hinds. Surface Analysis Methods. *IEEE CG&A*, pages 18–35, 1986.

[6] M Bloor, M J Wilson, and H Hagen. The Smoothing Properties of Variational Schemes for Surface Design. *CAGD*, 12:381–394, 1995.

[7] M Bloor and M Wilson. Using Partial Differential Equations to Generate Free-form Surfaces. *CAD*, 22:202–212, 1990.

[8] M Bloor, M Wilson, R Schneider, and L Kobbett. Mesh Fairing Based on an Intrinsic PDE Approach. *CAD*, 33:767–777, 2001.

[9] M Bloor and M Wilson. Generating Blend Surfaces Using PDE's. *CAD*, 21:165–171, 1989.

[10] C de Boor. *A Practical Guide to Splines*. Springer Verlag, 2001.

[11] W Boehm, G Farin, and J Kahmann. A Survey of Curve and Surface Methods in CaGD. *CAGD*, 1:1–60, 1984.

[12] J M Brown, M Bloor, M S Bloor, and M Wilson. The Use of Multiple Knots for B-spline Finite Element Approximation to PDE Surfaces. *Computing Supplement*, 10:87–99, 1995.

[13] M Casale and E Stanton. An Overview of Analytic Solid Modelling. *IEEE CG&A*, pages 45–56, Feb 1985.

[14] G Celniker and D Gossard. Deformable Curve and Surface Finite Elements for Free-form Shape Design. *Computer Graphics*, 25(4), 1991.

[15] S Cheng and C Dade. Dynamic Analysis of Stiffened Plates and Shells Using Spline Gauss Collocation Method. *Computers and Structures*, 36(4):623–629, 1990.

[16] L D Cohen and I Cohen. Finite-Element Methods for active Contour Models and Balloons for 2D and 3D Images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 15(11):1131–1147, 1993.

[17] M Eck and J Hadenfeld. Local Energy Fairing of B-spline Curves. *Computing*, 10:129–147, 1995.

[18] L Elsgolts. *Differential Equations and the Calculus of Variations*. Mir Publishers, Moscow, 1977.

[19] L Fang and D Gossard. Multidimensional Curve Fitting to Unorganised Data Points by Non-linear Optimisation. *CAD*, 27(1):48–58, 1995.

[20] G Farin, G Rein, N Sapadis, and A Worsey. Fairing Cubic B-spline Curves. *CAGD*, (4):91–103, 1987.

[21] G Farin and N Sapadis. Curvature and the Fairness of Curves and Surfaces. *CGA*, pages 52–57, 1989.

[22] A Fischer, P Kagan, P Z Bar-Yoseph, and M Shpitalni. *Product Modelling for Computer Integrated Design and Manufacture*, chapter A B-spline Finite Element Approach for Designing and Analysing Sculptured Objects, pages 129–138. Chapman and Hall, 1996.

[23] J Gardiner, A Laub, J Amato, and C Moler. Solution of the Sylvester Equation $AXB^T + CXD^T = E$. *ACM Transactions on Mathematical Software*, 18(2):223–231, 1992.

[24] G H Golub and C Van Loan. *Matrix Computations*. John Hopkins University Press, 1996.

[25] A Graham. *Kronecker Products and Matrix Calculus with Applications*. Ellis Horwood Ltd, 1981.

[26] G Greiner. *Wavelets, Images and Surface Fitting*, chapter Surface Construction Based on Variational Principles, pages 277–286. 1-56881-040-7, 1994.

[27] G Greiner and J Loos. Data Dependent Thin Plate Energy and its use in Interactive Surface Modelling. *Computer Graphics Forum*, 15(3):175–185, 1996.

[28] G Greiner. Variational Design and Fairing of Spline Surfaces. *Computer Graphics Forum*, 13(3):143–154, 1994.

[29] G Greiner. *Creating Fair and Shape-Preserving Curves and Surfaces*, chapter Modelling of Curves and Surfaces based on Optimisation Techniques, pages 11–27. B G Teubner, 1998.

[30] J Greissmair and W Purgathofer. Deformation of Solids with Trivariate B-splines. *Eurographics 89*, pages 137–148, 1989.

[31] H Hagen and G P Bonneau. Variational Surface Design and Surface Interrogation. *Eurographics 93*, 12(3):447–459, 1993.

[32] H Hagen and P Santarelli. *Topics in Surface Modelling*, chapter Variational Design of Smooth B-spline Surfaces, pages 85–94. SIAM, 1992.

[33] H Hagen and H Schulze. *Geometric Modelling*, chapter Variational Principles in Curve and Surface Design, pages 161–184. Springer, 1990.

[34] B Hahmann and T A Foley. A Quartic Spline Based on a Variational Approach. *Computing Supplement*, 10:199–210, 1995.

[35] H Henderson, F Pukelsheim, and S Searle. On the History of the Kronecker Product. *Linear and Multilinear Algebra*, (14):113–120, 1983.

[36] H V Henderson and S R Searle. The Vec Operator and Kronecker Products: A Review. *Linear and Multiliner Algebra*, (9):271–288, 1983.

[37] K Hollig. *Finite Element Methods with B-splines*. SIAM, 2003.

[38] J Hoschek. Intrinsic Parameterisation for Approximation. *CAGD*, (5):27–31, 1988.

[39] J Hoschek. Smoothing of Curves and Surfaces. *CAGD*, (2):97–105, 1985.

[40] J Hoschek and D Lasser. *Fundamentals of Computer Aided Geometric Design*. Wellesley, 1993.

[41] K I Joy. Utilising Parametric Hyperpatch Methods for Modelling and Display of Free-form Solids. pages 455–472. Proceedings of the Symposium on Solid Modelling Foundatations and CAD/CAM Applications, 1991.

[42] K I Joy. Mechanical Deformation of Hyperpatch Solids. pages 567–582. Proceedings of the Computer Graphics International, 1992.

[43] M Kallay, H Hagen, S Hahmann, and T Schreiber. Constrained Optimisation in Surface Design. *IEEE CG&A*, pages 53–59, Sept 1992.

[44] M Kallay and B Ravani. Optimal Twist Vectors as a Tool for Interpolating a Network of Curves with a Minimum Energy Surface. *CAGD*, 7:465–473, 1990.

[45] M Kass, A Witkin, and D Terzopoulos. Snakes, Active Contour Models. *International Journal of Computer Vision*, 2:321–331, 1988.

[46] J Kjellander. Smoothing of Bicubic Parametric Surfaces. *CAD*, 15(5):288–293, 1983.

[47] J Kjellander. Smoothing of Cubic Parametric Splines. *CAD*, 15(3):175–179, 1983.

[48] G J Klein. *Deformable Models for Volume Feature Tracking*. PhD thesis, University of California, Berkeley, 1999.

[49] U Langbecker and H Nowacki. A Knowledge Based System for Geometric Design. *Computing Supplement*, 10:211–226, 1995.

[50] D Lasser. Rational Tensor Product Bézier Volumes. *Computer and Mathematics with Applications*, 28:49–62, 1994.

[51] D Lasser. Bernstein-Bézier Representation of Volumes. *CAGD*, 2:145–149, 1985.

[52] A Y T Leung and F T K Au. Spline Finite Elements for Beam and Plate. *Computers and Structures*, 37(5):717–729, 1986.

[53] N J Lott and D I Pullin. Method for Fairing B-spline Surfaces. *CAD*, 20(10):597–604, 1988.

[54] K Maccallum and J Zhang. Curve Smoothing Techniques using B-splines. *The Computer Journal*, 29(6):564–569, 1986.

[55] A Marsan and D Dutta. On the Application of Tensor Product Solids in Heterogeneous Solid Modelling. ASME Design Engineering Technical Conferences, Proceedings of DETC98, 1998.

[56] H Meier. *Creating Fair and Shape-Preserving Curves and Surfaces*, chapter FAIR, An Interpolation/Approximation Method and Tool, pages 73–87. B G Teubner, 1998.

[57] H Meier and H Nowacki. Interpolating Curves with Gradual Changes in Curvature. *CAGD*, (4):297–305, 1987.

[58] D Metaxas and D Terzopoulos. Dynamic Deformation of Solid Primitives with Constraints. In *Proceedings of Siggraph 92*, pages 309–312, 1992.

[59] H P Moreton and C H Sequin. Functional Optimisation in Fair Surface Design. *Computer Graphics*, 26(2):167–176, 1992.

[60] K M Morken. Some Identities for Products and Degree Raising of Splines. *Constructive Approximation*, 7:195–208, 1991.

[61] H Nowacki and X Lu. Fairing Composite Polynomial Curves with Constraints. *CAGD*, (11):1–15, 1994.

[62] H Nowacki, G Westgaard, and J Heimann. *Creating Fair and Shape-Preserving Curves and Surfaces*, chapter Creation of Fair Surfaces Based on Higher Order Fairness Measures with Interpolation Constraints, pages 141–161. B G Teubner, 1998.

[63] G Nielson. Scattered Data Modelling. *IEEE CG&A*, pages 60–70, Jan 1993.

[64] G Nielson. *Volume Graphics*, chapter Volume Modelling, pages 29–48. Springer, 2000.

[65] R Parnes. *Solid Mechanics in Engineering*. Wiley, 2001.

[66] L Piegl and W Tiller. Symbolic Operators for NURBS. *CAD*, 29(5):361–368, 1997.

[67] J Poliakoff. An Improved Algorithm for Automatic Fairing of Non-Uniform Parametric Cubic Splines. *CAD*, 28(1):59–66, 1996.

[68] H Pottmann. Smooth Curves Under Tension. *CAD*, 22(4), 1990.

[69] H Qin and D Terzopoulos. D-NURBS, a Physics Based Design Framework for Geometric Design. *IEEE Transactions on Visualization and Computer Graphics*, 2(1):85–96, 1996.

[70] S S Rao. *The Finite Element Method in Engineering*. Butterworth-Heinemann, 1999.

[71] P Radeva, A Amini, and J Huang. Deformable B-solids and Implicit Snakes for 3D Localization and Tracking of MRI-SPaMM Data. *International journal on Computer Vision and Image Understanding*, 66(2):163–178, 1997.

[72] P Radeva, A Amini, J Huang, and E Marti. Deformable B-solids: Applications for Localisation and Tracking of MRi-SPaMM Data. Technical report, UPIIA, UAB, Barcelona, Spain, 1996.

[73] A Rappoport, A Sheffer, and M Bercovier. Volume-Preserving Freeform Solids. *IEEE Transactions on Visualisation and Computer Graphics*, 2(1):19–27, 1996.

[74] S Roth, M Gross, S Turello, and F Carls. A Bernstein-Bézier Approach to Soft Tissue Simulation. Technical Report 282, Institute of Scientific Computing, Ecole Polytechnique Federal de Zurich, 1998.

[75] M Sabin. Spline Finite Elements. *http://www.damtp.cam.ac.uk/user/na/people/Malcolm*, 2000.

[76] N Sapadis and G Farin. Automatic Fairing Algorithm for B-spline Curves. *CAD*, 22(2):121–129, 1990.

[77] B Schmitt, M Kazakov, A Paska, and V Savchenko. Volume Sculpting with 4D Volume Splines. In *International Conference on Imaging Science, Systems and Technology*, pages 475–483, 2000.

[78] T Sederberg and S Parry. Free-form Deformation of Solid Geometric Models. *Computer Graphics*, pages 151–160, 1986.

[79] E L Stanton and L M Crain. A Parametric Cubic Modelling System for General Solids of Composite Material. *International Journal for Numerical Methods in Engineering*, 11:653–670, 1997.

[80] G Strang. *Introduction to Applied Mathematics*. Wellesley-Cambridge Press, 1986.

[81] D Terzopoulos, J Platt, A Barr, and K Fleischer. Elastically Deformable Models. *Computer Graphics*, 21(4):205–215, 1987.

[82] D Terzopoulos and H Qin. Dynamic NURBS with Geometric Constraints for Interactive Sculpting. *ACM Transactions on Graphics*, 13(2):103–136, 1994.

[83] S Tikhonov and V Aresnin. *Solutions of Ill-Posed Problems*. Winston, 1977.

[84] W Tiller. Knot-Removal algorithms for NURB Curves and Surfaces. *CAD*, 24(8):445–453, 1992.

[85] S Timoshenko. *Theory of Plates and Shells*. McGraw-Hill, 1940.

[86] A H Vermeulen, R H Bartels, and G R Heppler. Integrating Products of B-splines. *SIAM Journal of Scientific and Statistical Computation*, 13(4):1025–1038, 1992.

[87] X Wang, F Cheng, and B Barsky. Energy and B-spline Approximation. *CAD*, 29(7):485–496, 1997.

[88] S Wang. A Unified Timoshenko Beam B-spline Rayleigh-Ritz Method for Vibration and Buckling Analysis of Thick and Thin beams and Plates. *International Journal for Numerical Methods in Engineering*, 40:473–491, 1997.

[89] J Wang and P C Shen. Solution Governing Differential Equations of Vibrating Cylindrical Shells Using B-spline Functions. *Numerical Methods for Partial Differential Equations*, 2:173–185, 1986.

[90] W Weiss, L Andor, G Rener, and T Varady. Advanced Surface Fitting Techniques. *CAGD*, 19:19–42, 2002.

[91] W Welch and A Witkin. Variational Surface Modelling. *Computer Graphics*, 26(2):157–166, 1992.

[92] W Wesselink and R C Veltkamp. Interactive Design of Constrained Variational Curves. *CAGD*, 12:533–546, 1995.

[93] G Westgaard and H Nowacki. A Process for Surface Fairing in Irregular Meshes. *CAGD*, 18:619–638, 2001.

[94] C Zhang, P Zhang, and F Cheng. Fairing Spline Curves and Surfaces by Minimising Energy. *CAD*, 33:913–923, 2001.