STATICS AND DYNAMICS OF PULP FIBRES

by

Yuri Andrew Lawryshyn

A thesis submitted in conformity with the requirements for the degree of Doctor of Philosophy
Graduate Department of Chemical Engineering and Applied Chemistry
University of Toronto

© Copyright by Yuri Andrew Lawryshyn 1997
The author has granted a non-exclusive licence allowing the National Library of Canada to reproduce, loan, distribute or sell copies of this thesis in microform, paper or electronic formats.

The author retains ownership of the copyright in this thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without the author’s permission.

L’auteur a accordé une licence non exclusive permettant à la Bibliothèque nationale du Canada de reproduire, prêter, distribuer ou vendre des copies de cette thèse sous la forme de microfiche/film, de reproduction sur papier ou sur format électronique.

L’auteur conserve la propriété du droit d’auteur qui protège cette thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

0-612-27989-8
To my parents and my fiancée Jody -
through the clouds,
the mountain looms above
Flexible pulp fibres produce paper of higher quality than their stiff counterparts. The pulp and paper industry has a keen interest to measure fibre flexibility and to fractionate fibres based on flexibility. The objective of this thesis is to theoretically study the static and dynamic behaviour of pulp fibres with direct applications to fibre flexibility measurement devices and fibre screening. Governing equations, which represent the deflection and motion of pulp fibres, are developed and numerical methods are utilized to solve the mathematical formulations. Static large deflection beam theory is applied to the geometry of four existing fibre flexibility measurement devices to determine the advantages and shortcomings of each method. It is shown that the small deflection analysis predicts flexibility with an error of less than 10% when compared to the large deflection analysis for fibre deflections of less than 20% of the span length. Furthermore, it is concluded that a better estimate of the hydrodynamic forces acting on the pulp fibres is required. To study the behaviour of flexible fibres in pulp screening applications, non-linear equations representing the motion of a flexible fibre are developed. Two methods to represent the dynamic interaction of a fibre with the flow domain walls are proposed. Together with a Computational Fluid Dynamics (CFD) analysis, the motion of fibres in a channel flow with a slot are studied and the effect of fibre flexibility on the ability of the fibres to pass through the slot is examined. For the first time, a theoretical model has been used to show that screening based on fibre flexibility does occur. However, it is shown that the predominant property which governs the fractionation of fibres is the fibre length. To propose a direct method to model the flow of a flexible fibre for future applications and a method to predict the hydrodynamic forces acting on a fibre, an automatic three-dimensional finite element mesh generating algorithm, based on the Delaunay triangulation, is developed for use with CFD software. A unique method of mesh refinement is defined and it is shown that the method is extremely efficient for typical fibre geometries.
Acknowledgements

I would like to thank Professor D.C.S. Kuhn for his guidance and supervision. His dedication to all of his students is unique and greatly appreciated. He has always looked out for my interests first, and his own second - a special characteristic not common to man. Because of him, my experience as a Ph.D. student was a wonderful period of my life which I will always cherish in my thoughts.

I would like to thank my thesis committee members for their support and guidance. Thanks to all students, staff, faculty and alumni of the University of Toronto Pulp and Paper Centre - my learning experience was greatly accentuated because of the Centre.

I would like to acknowledge my colleagues, Lanre Oshinowo, Ted Mao, Fred Gocool, Xintong Lu, Ivan Gohn for their help in all manners of my work. I would like to thank James Olsen and Robert Gooding for their input and stimulating conversations.

Finally, I would like to thank my parents and Jody for their love, encouragement, support and help throughout the program.
## Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Abstract</strong></td>
<td>i</td>
</tr>
<tr>
<td><strong>Acknowledgements</strong></td>
<td>ii</td>
</tr>
<tr>
<td><strong>Nomenclature</strong></td>
<td>vii</td>
</tr>
<tr>
<td><strong>1 Introduction</strong></td>
<td>1</td>
</tr>
<tr>
<td><strong>2 Literature Review</strong></td>
<td>6</td>
</tr>
<tr>
<td>2.1 Flow Past Cylinders and Fibre Motion Models</td>
<td>7</td>
</tr>
<tr>
<td>2.1.1 Analytical and Semi-Emperical Estimates of the</td>
<td>7</td>
</tr>
<tr>
<td>Forces acting on Cylindrical Bodies</td>
<td>7</td>
</tr>
<tr>
<td>2.1.2 Analytical and Experimental Investigations of Fibre Motion</td>
<td>9</td>
</tr>
<tr>
<td>2.2 Application of CFD and Mesh Generation to Fibre Flow</td>
<td>11</td>
</tr>
<tr>
<td>2.3 Methods to Measure Fibre Flexibility</td>
<td>12</td>
</tr>
<tr>
<td>2.4 Dynamics of Flexible Beams</td>
<td>14</td>
</tr>
<tr>
<td>2.5 Methods of Discretizing Beams in Vibration</td>
<td>15</td>
</tr>
<tr>
<td>2.6 Fibre Screening</td>
<td>18</td>
</tr>
<tr>
<td>2.7 Summary</td>
<td>20</td>
</tr>
</tbody>
</table>
3 Fibre Statics

3.1 Static Deflection of Fibres

3.2 Numerical Solution of the Large Deflection Equations

3.3 Fibre Stiffness Measurement Methods
   3.3.1 Cantilevered Fibre in a T-Junction
   3.3.2 Clamped Cantilevered Fibre in a Channel Flow
   3.3.3 Simply Supported Fibre in a Capillary
   3.3.4 Conforming Fibre on a Wire

3.4 Small and Large Deflection Analyses
   3.4.1 Cantilevered Fibre Methods
   3.4.2 Simply Supported Fibre in a Capillary
   3.4.3 Conforming Fibre on a Wire

3.5 Conclusions

4 Fibre Dynamics: Equations of Motion

4.1 Simple Dynamic Motion of Two Masses

4.2 Flexible Fibre in Free Vibration

4.3 Equations of Motion of a Flexible Fibre

4.4 Discretization of the Equations of Motion

4.5 Inertialess Equations of Motion

4.6 Modelling the Contact Between the Fibre and the Channel and Slot Walls
   4.6.1 Compliant Wall Contact Model
   4.6.2 Constraint Wall Contact Model
   4.6.3 Advantages and Shortcomings of the Wall Contact Models

4.7 Numerical Solution of the Equations of Motion of a Flexible Fibre in a Shear Flow

4.8 Comparison with Static Analysis

4.9 Conclusions
5 Fibre Dynamics: Motion Simulations
5.1 Channel and Slot Geometry
5.2 CFD Simulation of the Channel and Slot
5.3 General Aspects of the Fibre Motion Model
  5.3.1 Integration of the Flow Domain
  5.3.2 Practical Considerations of the Fibre Motion Model
  5.3.3 Effect of Inertia on the Fibre Motion Simulations
5.4 Fibre Motion Simulations: Qualitative Analysis
5.5 Fibre Motion Simulations: Quantitative Analysis
5.6 Discussion
5.7 Summary

6 Automatic Three-Dimensional Mesh Generation
6.1 The Three-Dimensional Automatic Mesh Generating Algorithm
6.2 The Delaunay Triangulation
6.3 The Rectangular Three-Dimensional Domain
6.4 Generation of the Fibre Nodes
6.5 Generation of the Internal Flow Domain Nodes
6.6 Mesh Refinement
6.7 The Final Steps
6.8 Flow Simulation
7 Concluding Remarks 137
  7.1 Summary and Discussion 137
    7.1.1 Fibre Statics 138
    7.1.2 Fibre Dynamics 139
    7.1.3 Automatic Three-Dimensional Mesh Generating Algorithm 139
  7.2 Recommendations for Future Work 139
    7.2.1 Application of Large Deflection Equations 139
    7.2.2 Flexible Fibre Motion Model 140
    7.2.3 Application of CFD to Model the Hydrodynamic Forces Acting on a Fibre 140
    7.2.4 Pulp Screening 140

8 References 141

A Object Oriented Programming 148

B Vector and Matrix Template Classes 160
Nomenclature

\( a \) channel half-width for Clamped Cantilevered Fibre
\( a \) span length for Conforming Fibre on a Wire
\( A \) cross-sectional area of a fibre
\( A_i \) constant of integration for a vibrating fibre
\( \mathbf{a}_i \) position vector used in the meshing algorithm
\( \mathbf{A}_i \) rotation matrix used in the meshing algorithm
\( b \) channel half-height for Clamped Cantilevered Fibre
\( b(x) \) distributed structural damping coefficient of the fibre per unit length
\( b_q \) structural damping coefficients of the fibre
\( B_i \) constant of integration for a vibrating fibre
\( \mathbf{B}_i \) screw theory rotation matrix used in the meshing algorithm
\( \mathbf{b}_n \) unit vector used in the meshing algorithm
\( \mathbf{B} \) structural damping matrix of the fibre
\( c_{\text{wall}} \) wall damping coefficient of Compliant Wall Contact Model
\( C(\cdot) \) fibre concentration function
\( C_{\text{avg}} \) average fibre concentration in the channel
\( C_{E\text{Lavg}} \) average fibre concentration in the exit layer
\( C_y \) constant of integration for a vibrating fibre
\( C_D \) drag coefficient for a circular cylinder
\( C_{\text{avg}} \) average fibre concentration in the slot
\( C_{u\text{avg}} \) average fibre concentration in the channel upstream from the slot
\( C(\cdot) \) vector of Coriolis and centripetal forces in the dynamic analysis
\( d \)  diameter of wire for Conforming Fibre on a Wire
\( d_1, d_2 \)  lengths of different geometries of the T-Junction
\( D \)  diameter of the cylinder
\( E \)  Young's Modulus
\( f_n \)  force per unit length acting normal to the cylinder
\( f_t \)  force per unit length acting tangential to the cylinder
\( f_w \)  weight applied, per unit length for Conforming Fibre on a Wire
\( f(x,t) \)  applied force per unit length for a fibre in vibration
\( F_c \)  wall and fibre contact force for the Constraint Wall Contact Model
\( F_{\text{max}} \)  maximum applied force for the static and dynamic comparison
\( F_{n_j} \)  \( j \)-th point force applied on the fibre in the normal direction
\( F_{v_j} \)  \( j \)-th point force applied on the fibre in the tangential direction
\( F_y \)  force applied on the fibre in the \( Y \) direction
\( F_x \)  force applied on the fibre in the \( X \) direction
\( F_\theta \)  moment applied on the fibre
\( f \)  hydrodynamic force vector
\( F \)  applied force vector in the dynamic analysis
\( F_v \)  applied force vector in the inertialess model
\( g_n(\cdot) \)  normal hydrodynamic force function
\( g_t(\cdot) \)  tangential hydrodynamic force function
\( G_y(\cdot) \)  function used for smoothing tetrahedra in the meshing algorithm
\( G \)  vector of functions used in the Runge-Kutta method
\( H \)  height above the lower channel wall
\( H_c \)  channel height
\( H_{\text{el}} \)  exit layer height
\( H_f \)  height above the channel wall that the fibre is initially positioned in the fibre motion simulations
\( H_{\text{fem}} \)  height above the channel wall that the fibre is initially positioned in the fibre motion simulations, where it enters the slot and any placement above this value would result in the fibre not passing through the slot.
\[ i \quad \text{integral counter} \]
\[ I \quad \text{cross-sectional moment of inertia of a fibre} \]
\[ I_{int} \quad \text{turbulence intensity} \]
\[ I \quad \text{identity matrix} \]
\[ I_{m\times m} \quad \text{identity matrix of size } m \times m \]
\[ j \quad \text{integral counter} \]
\[ k \quad \text{constant for calculating drag for Clamped Cantilevered Fibre} \]
\[ k \quad \text{spring stiffness for Conforming Fibre on a Wire} \]
\[ k \quad \text{constant of integration for a vibrating fibre} \]
\[ k \quad \text{turbulent kinetic energy parameter used in the standard } k-\varepsilon \text{ turbulence model} \]
\[ k_n \quad \text{normal hydrodynamic force coefficient} \]
\[ k_t \quad \text{tangential hydrodynamic force coefficient} \]
\[ k_{wall} \quad \text{wall spring stiffness coefficient of the Compliant Wall Contact Model} \]
\[ K_q \quad \text{terms of the stiffness matrix of the dynamic analysis} \]
\[ k \quad \text{finite element stiffness matrix} \]
\[ K \quad \text{diagonal stiffness matrix of the dynamic analysis} \]
\[ K_1, K_2, \ldots, K_s \quad \text{Runge-Kutta parameters} \]
\[ h \quad \text{step size in the Runge-Kutta method} \]
\[ l \quad \text{length of a finite beam element} \]
\[ L \quad \text{length of the fibre} \]
\[ L_c \quad \text{length of the channel} \]
\[ L_{ca} \quad \text{length from the channel inlet to the slot} \]
\[ L_s \quad \text{length of the slot} \]
\[ L_s \quad \text{characteristic length scale for turbulence modelling} \]
\[ m \quad \text{number of modes of vibration used to discretize the dynamic equations} \]
\[ M \quad \text{applied moment in the static analysis} \]
\[ M \quad \text{mass of fibre in the dynamic analysis} \]
\[ M, M(\circ) \quad \text{mass matrix} \]
\[ M_n(\circ) \quad \text{inertialess "mass" matrix} \]
\[ n \quad \text{an integer used for the turbulent flow power law equation} \]
\( n_c \)  
number of centre points defining the fibre in the meshing algorithm

\( n_{arc} \)  
number of nodes at one centreline point used to define the fibre shape in the meshing algorithm

\( N(x) \)  
finite element shape function

\( N_s \)  
rate of fibre passage through the slot per unit depth

\( \mathbf{n}_a \)  
unit vector normal to an ellipse of intersection in the meshing algorithm

\( p(\bullet) \)  
probability of fibre passage

\( P \)  
passage ratio

\( P_i \)  
component of the passage ratio due to the "turning effect"

\( P_i \)  
i-th point or node in the Delaunay triangulation

\( P_{CCG} \)  
passage ratio for a constant concentration gradient

\( P_{OCGM} \)  
maximum passage ratio when the Olsen's (1996) concentration gradient is used

\( P_w \)  
component of the passage ratio due to the "wall effect"

\( Q \)  
volume flow rate for Clamped Cantilevered Fibre

\( Q_{fem} \)  
volume flow rate through between the channel wall and \( H_{fem} \) per unit depth

\( Q_s \)  
volume flow rate through the slot per unit depth, also equal to the exit layer volume flow rate per unit depth

\( Q(x) \)  
shape function - natural mode of vibration

\( Q_i(x) \)  
i-th shape function - natural mode of vibration

\( \mathbf{q} \)  
vector of unknown time-dependent co-ordinates to be determined in the dynamic analysis

\( \mathbf{Q}(x) \)  
vector of shape functions

\( Re \)  
Reynolds number

\( R \)  
hydraulic radius of the channel

\( R_{fibre} \)  
radius of fibre used in the meshing algorithm

\( \mathbf{r}_i \)  
position vector used in the meshing algorithm

\( \mathbf{R} \)  
position vector of a point on the fibre

\( \mathbf{R}_a \)  
position vector of the i-th centreline point in the meshing algorithm

\( s \)  
distance measurement along the length of a fibre

\( S_i \)  
length of the i-th side of a tetrahedron
<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$S_{\text{rms}}$</td>
<td>root-mean-square of the six sides of a tetrahedron</td>
</tr>
<tr>
<td>$t$</td>
<td>time</td>
</tr>
<tr>
<td>$T$</td>
<td>tensile force in the large deflection analysis</td>
</tr>
<tr>
<td>$T$</td>
<td>potential energy in the dynamic equations</td>
</tr>
<tr>
<td>$\mathbf{t}_c$</td>
<td>unit vector used in the meshing algorithm</td>
</tr>
<tr>
<td>$u, u(x,t)$</td>
<td>dynamic displacement of a fibre or beam from the undeflected position</td>
</tr>
<tr>
<td>$u(y)$</td>
<td>flow rate</td>
</tr>
<tr>
<td>$U$</td>
<td>magnitude of the fluid velocity</td>
</tr>
<tr>
<td>$U_m$</td>
<td>maximum flow speed for the T-Junction</td>
</tr>
<tr>
<td>$U_{\text{max}}$</td>
<td>maximum flow speed for the static and dynamic comparison</td>
</tr>
<tr>
<td>$U_n$</td>
<td>normal (with respect to a fibre) fluid velocity component</td>
</tr>
<tr>
<td>$U_s$</td>
<td>average slot velocity</td>
</tr>
<tr>
<td>$U_t$</td>
<td>tangential (with respect to a fibre) fluid velocity component</td>
</tr>
<tr>
<td>$U_u$</td>
<td>average channel velocity upstream of the slot</td>
</tr>
<tr>
<td>$U_x$</td>
<td>$X$ or $x$ component of the fluid velocity</td>
</tr>
<tr>
<td>$U_y$</td>
<td>$Y$ or $y$ component of the fluid velocity</td>
</tr>
<tr>
<td>$U$</td>
<td>fluid velocity vector</td>
</tr>
<tr>
<td>$v_{nj}$</td>
<td>components of $v_n$ ($j = 1,2,3$)</td>
</tr>
<tr>
<td>$V$</td>
<td>shear force in the large deflection analysis</td>
</tr>
<tr>
<td>$V$</td>
<td>potential energy in the dynamic equations</td>
</tr>
<tr>
<td>$V$</td>
<td>volume of a tetrahedron</td>
</tr>
<tr>
<td>$V_t$</td>
<td>Voronoi region</td>
</tr>
<tr>
<td>$V_x$</td>
<td>$X$ or $x$ component of the fibre velocity</td>
</tr>
<tr>
<td>$V_y$</td>
<td>$Y$ or $y$ component of the fibre velocity</td>
</tr>
<tr>
<td>$\mathbf{v}_n$</td>
<td>screw vector used in the meshing algorithm</td>
</tr>
<tr>
<td>$\mathbf{V}$</td>
<td>fibre velocity vector</td>
</tr>
<tr>
<td>$w(\bullet)$</td>
<td>finite element node displacement</td>
</tr>
<tr>
<td>$W$</td>
<td>weight per unit length of a fibre in the large deflection analysis</td>
</tr>
<tr>
<td>$W_s$</td>
<td>slot width</td>
</tr>
<tr>
<td>$\delta W$</td>
<td>virtual work in the dynamic analysis</td>
</tr>
</tbody>
</table>
\( \mathbf{w} \)  vector of finite element node displacements
\( x \)  Cartesian co-ordinate
\( X \)  global Cartesian co-ordinate
\( X_o \)  global \( X \) position of point \( O \) of the fibre
\( \mathbf{x}_tn \)  position of a point on the fibre in tangential and normal co-ordinates
\( \mathbf{x}_{xy} \)  position of a point on the fibre in Cartesian co-ordinates
\( y \)  Cartesian co-ordinate
\( Y \)  global Cartesian co-ordinate
\( Y_o \)  global \( Y \) position of point \( O \) of the fibre

**Greek Symbols**

\( \alpha \)  initial angle of a fibre in the T-Junction
\( \varepsilon \)  turbulence dissipation
\( \theta \)  fibre orientation angle
\( \delta \)  deflection for the static analysis
\( \delta \)  distance of penetration into the wall for the Compliant Wall Contact Model
\( \phi \)  angle used for creating nodes on the fibre in the meshing algorithm
\( \varphi \)  independent variable in the Runge-Kutta method
\( \gamma \)  tetrahedron aspect ratio
\( \gamma_{max1} \)  maximum tetrahedron aspect ratio for centroid addition
\( \gamma_{max2} \)  maximum tetrahedron aspect ratio for running the smoothing algorithm
\( \mu \)  fluid viscosity
\( \eta \)  rotated \( y \) co-ordinate for the T-Junction
\( \xi \)  rotated \( x \) co-ordinate for the T-Junction
\( r_i(t) \)  \( i \)-th mode displacement coefficient for the dynamic analysis
\( \mathbf{r}(t) \)  vector of the displacement coefficients for the dynamic analysis
\( \rho \)  density of the fluid
\( \rho_c \) density of fibre
\( \omega \) natural frequency
\( \psi \) vector of dependent variables to be solved for in the Runge-Kutta method
\( \psi_{\text{inera}} \) vector of dependent variables to be solved for in the Runge-Kutta method for the equations of motion with inertia
\( \psi_{\text{no-in}} \) vector of dependent variables to be solved for in the Runge-Kutta method for the equations of motion with no inertia
\( \chi \) variable of integration
\( \sigma_i \) constant of integration for a vibrating fibre

**Relation Symbols and Operators**

\( \in \) the expression on the left is an element of the set on the right
\( \equiv \) expression of the left is defined to be the expression on the right
\( \forall \) for all
\( \cdot \) derivative with respect to time
\( - \) denotes dimensionless variable
\( / \) partial derivative with respect to \( x \) in the dynamic analysis
\( \partial \) partial derivative with respect to \( t \) in the dynamic analysis
\( ^\wedge \) used to denote kinetic or potential energy per unit length of the fibre in the dynamic analysis
Chapter 1

Introduction

The product of the pulping process, or stock, requires some form of screening operation to remove undesirable, oversized and troublesome materials from good papermaking fibres. Of particular interest in the pulp and paper industry, although presenting great difficulty, is the desire to fractionate fibres based on length and stiffness. Shives, which are bundles of fibres that have not been separated during pulping and require further processing, are characterized by being much stiffer than individual fibres. Stiff fibres and shives reduce the strength, surface and optical properties of paper and can cause problems on printing presses and in coating operations. Flexible fibres produce stronger paper because of their ability to consolidate better than stiff ones; this leads to a greater degree of hydrogen bonding.

Since fibre flexibility plays an important role in the paper quality, the pulp and paper industry has a keen interest in the ability to measure wet fibre flexibility easily and quickly. A number of wet fibre flexibility measurement methods have been proposed, to date, but no measurement device exists which can accomplish the task efficiently and accurately. Generally, the methods rely on the hydrodynamic forces of a fluid, in which a fibre is suspended, to deflect the fibre. A stiffness value for the fibre is estimated by comparing the flow rates to the fibre deflection and utilizing static small deflection beam bending theory.

Pressurized screens are the primary unit for shive removal. They utilize a cylindrical perforated plate (see Figure 1.1). A typical screen plate contour geometry is shown in Figure 1.2.
The accepts flow through the perforations and are fed forward in the process for papermaking, whereas the rejects are typically processed further with the stiff fibre component refined and recycled. The perforations are holes or slots whose smallest dimension is greater than the diameter of a fibre or shive but much less than the length of either. This means that the screen plate does not act as a physical barrier to shives, but rather, must rely on other mechanisms to separate shives and stiff fibres from desirable flexible ones. Fractionation based on fibre flexibility, the inverse of stiffness, must rely on mechanisms which take advantage of the ability of flexible fibres to deform in a shear flow to separate them from stiff ones.

Although technical information is available on the working performance of existing pressure screens, little is known or published on the fundamental mechanisms of fibre fractionation. Most of the work published to date focuses on experimental studies of screening and fibre fractionation. No mathematical or numerical models have been proposed to study the flow of flexible fibres through small apertures.

The objective of this thesis is to theoretically study the static and dynamic behaviour of pulp fibres with direct applications to fibre flexibility measurement devices and fibre screening. Governing equations, which represent the deflection and motion of pulp fibres, are developed and numerical methods are utilized to solve the mathematical formulations. In particular, static large-deflection beam theory is applied to the geometry of four existing fibre flexibility measurement devices.

![Diagram](image.png)

**Figure 1.1**: Screen plate geometry.
devices to determine the advantages and shortcomings of each method. To study the behaviour of flexible fibres in pulp screening applications, non-linear equations representing the motion of a flexible fibre are developed. Together with a Computational Fluid Dynamics (CFD) analysis, the motion of fibres in a channel flow with a slot are studied and the effect of fibre flexibility on the ability of the fibres to pass through the slot is examined. An automatic three-dimensional finite element mesh generating algorithm is developed for use with CFD software to model the fluid flow around a fibre of arbitrary shape. The motivation behind the automatic mesh generating algorithm is to introduce a direct method to model the flow of a flexible fibre for future applications, where it is expected that the computer speeds will be capable of solving such a problem in a practical duration of time. The algorithm can also be used to predict the hydrodynamic forces acting on the fibre in static situations, such as in fibre flexibility measurement devices.

The numerical models developed in this thesis are tools which can be more broadly utilized in the development of pulp fibre property measurement devices and processing equipment. As mentioned above, the work focuses on the static deflection of fibres, the motion of flexible fibres in a shear flow and the development of an automatic three-dimensional mesh generating algorithm for modelling the flow of a fluid around fibres. Three numerical "tools" are developed:

1. a large deflection analysis applied to static fibre deflection to study fibre flexibility measurement devices;
2. a flexible fibre motion model in a shear flow to model the flow of a flexible fibre in screening geometries;
3. an automatic three-dimensional mesh generating algorithm for creating a mesh around a fibre of any shape in a rectangular flow domain.

As presented in the literature, all of the fibre flexibility measurement devices, which predict a flexibility value, analytically relate a fibre's deflection to its flexibility using small deflection analysis. In many cases, during the measurement, the fibre's deflection is greater than 10% of its span length, violating the assumptions of small deflection beam theory. A numerical model which predicts the static deflection of a fibre using large deflection analysis was developed. Utilizing the model, an investigation of the error associated with the small deflection analysis is performed and an assessment of the error due to physical effects that may not be accounted for by small deflection analysis is presented.

Although numerical models for the flow of rigid or perfectly flexible cylinders and fibres have been proposed in literature, no models have been proposed to study the flow of a fibre with nominal stiffness. The task of numerically modelling the motion of a flexible fibre can be grouped into two sub-tasks: 1) the prediction of the forces acting on the fibre by the fluid flow and 2) the calculation of the dynamic motion of the flexible cylinder. Although CFD, using automatic mesh generation, could be utilized to tackle sub-task 1, the computation times would be prohibitive with today's computer speeds. For this reason, analytical approximations are used to estimate the fluid forces acting on the fibre. A method to solve sub-task 2 is introduced. The dynamic motion of a flexible fibre in a channel with a slot is investigated to show the applicability of the model for screen plate design and analysis.

Of key importance in the development of a fibre flexibility device is the ability to model the fluid flow around the fibre and to determine the forces of the fluid acting on the fibre. This task can be accomplished by using computational fluid dynamics (CFD). A difficulty associated with using CFD is the generation of a mesh for flow domains with awkward geometries. Furthermore, the development of a direct method to model the motion of a flexible fibre is complicated by the fact that the pulp fibres will move and deform in any shear flow, requiring mesh regeneration at every time step. Generally, mesh generation is labour intensive. To overcome this problem an automatic three-dimensional mesh generating algorithm, which creates a mesh around a cylindrical body representing a fibre, was developed. A unique method for mesh optimization is proposed and applied. The meshing algorithm can be integrated with a CFD
package to study the flow of a fluid around the fibre. It is expected that this algorithm will prove to be extremely useful in the design of a fibre flexibility measurement device.

Relevant literature pertaining to the above topics will be presented in Chapter 2. Chapter 3 will introduce the equations for large static deflection theory and apply the equations to four fibre flexibility measurement methods proposed in literature. The dynamic equations describing the motion of a flexible fibre will be developed in Chapter 4. Chapter 5 will utilize the numerical models developed in Chapter 4 to study the flow of flexible fibres in a channel with a slot. The effect of fibre flexibility on the ability of fibres to pass through the slot will be investigated. Qualitative and quantitative comparisons of the results from the numerical simulations will be made with experimental work published in the literature. The automatic three-dimensional mesh generating algorithm will be discussed in Chapter 6. Chapter 7 will give general conclusions and provide recommendations for future work. The numerical models presented in this thesis are developed using the C++ object oriented programming language. The advantages of using the C++ language are discussed in Appendix A, as is the design of data structures used throughout the programs of the thesis.
Chapter 2

Literature Review

Three main topics are associated with the study of the statics and dynamics of pulp fibres in a fluid suspension, namely, the fluid forces acting on the fibres, the static deflection of fibres and the dynamic motion of fibres. The first section of the literature review will focus on the fluid mechanics of flows around cylindrical bodies, analytical and empirical methods to estimate the hydrodynamic forces acting on cylinders and long slender bodies immersed in a fluid, and fibre flow models. Relevant literature regarding the application of computational fluid mechanics (CFD) to model the flow of a fluid around a cylindrical body, as well as mesh generation, will be discussed in section two. The third section will focus on static deflection of fibres with a brief review of literature pertaining to existing methods for measuring fibre flexibility. The dynamics of flexible body motion will be reviewed in the fourth section. The equations of motion of a flexible body are generally of integro-differential form and must be discretized to attain a numerical solution. For the case of a flexible fibre, the equations can be discretized by using the modes of free vibration of a beam. The development of these equations will be given in Chapter 4, but methods to discretize them will be discussed in the fifth section of this chapter. These equations will be utilized to study the ability of flexible fibres to pass through small apertures, which has a direct application to pulp screening. The sixth section will discuss relevant literature pertaining to pulp screening, and the last section will summarize the discussion of relevant literature.
2.1 Flow Past Cylinders and Fibre Motion Models

2.1.1 Analytical and Semi-Emperical Estimates of the Forces acting on Cylindrical Bodies

The crossflow of a stream past a circular cylinder is a very common fluids engineering problem. For a Reynolds number, \( Re \), based on the diameter of the cylinder and the speed of the free stream flow of less than one, the flow around the cylinder is laminar and no separation or resulting wake takes place (Fox and McDonald (1985)). As the Reynolds number increases, flow separation takes place resulting in a laminar wake. According to White (1991), at \( Re > 35 \) a vortex street occurs, called the Karman vortex street. Schlichting (1979) presented the work by Blenk et al. (1935) where it was shown that a regular Karman street is observed for \( 60 < Re < 5000 \). Above \( Re = 5000 \) complete turbulent mixing was observed.

The drag coefficient for a circular cylinder in a cross flow is defined as

\[
C_D = \frac{2f_n}{\rho U^2 D}
\]

where \( f_n \) is the force per unit length acting normal to the cylinder axis, \( \rho \) is the density of the fluid, \( U \) is the speed of the free stream and \( D \) is the diameter of the cylinder. Numerous experimental studies have been carried out to determine \( C_D \) as a function of \( Re \). White (1991) presented a curve-fit formula developed by Sucher and Brauer (1975)

\[
C_D = 1.18 + \frac{6.8}{Re^{0.89}} + \frac{1.96}{Re^{0.5}} - \frac{0.0004Re}{1 + 3.64Re - 7Re^2}
\]

which is valid for the range \( 10^4 < Re < 2 \times 10^5 \). Unfortunately, experimental data is not available for normal and tangential forces acting on cylinders at different angles to the free stream velocity.

Stokes (1851) (presented by Schlichting (1979)) solved the equations for the creeping motion flow past a three-dimensional sphere to determine the drag coefficient of the sphere for low Reynolds number flows. As was stated by Stokes, an analogous solution to the two-dimensional flow past a cylinder cannot be determined since the no-slip boundary condition on the surface of the cylinder and the freestream boundary conditions cannot be simultaneously satisfied by applying the creeping flow equations in two dimensions. This is known as the Stokes paradox. White (1991) presented the work developed by Oseen (1910) where an \textit{ad hoc} linearized convective acceleration term is added to the momentum equation, removing the Stokes paradox for two-dimensional creeping flows. The Oseen solution for determining the flow field
around an infinite cylinder was discussed in a text by Lamb (1932). Making use of Oseen's equations, the drag on a cylinder in a crossflow was developed by Tomotika and Aoi (1951). White (1991) compared the drag coefficient of Tomotika and Aoi with experimental data of Tritton (1959) and showed that the analytical prediction for the drag coefficient compares well with experimental data for a Reynolds number up to one, but then diverges.

Cox (1970) developed analytical equations for the hydrodynamic forces acting on a long slender body immersed in a general undisturbed flow field. By assuming the fluid inertia to be negligible, the force per unit length on the body is obtained by an asymptotic expansion in terms of the ratio of the diameter to the length of the body. Similar work has been done by Burgers (1938), Batchelor (1970), and De Mestre and Russel (1975). Cox's first-order approximation is given by

\[
f = \frac{2\pi \mu}{\ln(L/D)} \left( 2I - \frac{\partial R}{\partial s} \frac{\partial R}{\partial s} \right) \cdot (U - V)
\]  

(2.3)

where \( f \) is the force vector per unit length, \( \mu \) is the viscosity of the fluid, \( L \) is the length of the body, \( D \) is the diameter of the body, \( I \) is the unit matrix, \( R \) is the position vector of a point on the fibre, \( s \) is measured along the fibre length, \( U \) is the undisturbed fluid velocity and \( V \) is the velocity of the fibre (see Figure 2.1). Equation (2.3) is valid for \( Re << 1 \) and \( D/L << 1 \).

Referring to Figure 2.1 for the definition of \( \theta \), the partial derivative of \( R \) with respect to \( s \) can be written as

\[
\frac{\partial R}{\partial s} = \begin{bmatrix} \cos(\theta) \\ \sin(\theta) \end{bmatrix}
\]  

(2.4)

![Figure 2.1: Long slender body.](image)
and equation (2.3) becomes

\[
f = \frac{2\pi \mu}{\ln(L/D)} \left[ (2 - \cos^2(\theta))(u_x - v_x) + \cos(\theta)\sin(\theta)(U_y - V_y) \\
+ (2 - \sin^2(\theta))(u_y - v_y) + \cos(\theta)\sin(\theta)(U_x - V_x) \right].
\]

(2.5)

Here \(U_x\) and \(U_y\) are the x and y components of the vector \(U\), and \(V_x\) and \(V_y\) are the x and y components of the vector \(V\). The vectors \(f\), \(U\) and \(V\) are transformed from the \(x\)-\(y\) co-ordinate frame to the \(t\)-\(n\) co-ordinate frame by

\[
x_{xy} = \begin{bmatrix}
cos(\theta) & -sin(\theta) \\
sin(\theta) & cos(\theta)
\end{bmatrix} x_{tn}
\]

(2.6)

where \(x_{xy}\) is any vector with components in \(x\)-\(y\) co-ordinates and \(x_{tn}\) is the same vector with its components in \(t\)-\(n\) co-ordinates. The resulting normal and tangential force per unit length acting on the body is given by

\[
\begin{align*}
f_t &= 2\pi \mu (U_t - V_t) \left( \ln \left( \frac{L}{D} \right) \right)^{-1} \\
f_n &= 4\pi \mu (U_n - V_n) \left( \ln \left( \frac{L}{D} \right) \right)^{-1}
\end{align*}
\]

(2.7)

The subscripts \(t\) and \(n\) designate the normal and tangential components of the respective vectors. The directions of \(t\) and \(n\) are defined in Figure 2.1.

More recently, Khayat and Cox (1989) developed the equations for the hydrodynamic forces acting on a long slender body with the consideration of inertia effects. As expected, the first order approximations to these equations are equivalent to equation (2.7) for \(Re\) approaching zero. It was shown that for the theory to be valid, \(Re\) must satisfy

\[Re < (D/L)^{1.5}\text{ as } D/L \rightarrow 0\]

where \(\delta\) is a fixed positive constant much smaller than unity. Modifications were made to the equations for the hydrodynamic forces for bodies of infinite length. Although equation (2.7) is a first order approximation, it is easily applied and has been used in fibre motion models presented by Gradon and Podgorski (1990), Podgorski et al. (1995) and Olsen (1996). It will be utilized in the fibre motion models developed in the thesis.

2.1.2 Analytical and Experimental Investigations of Fibre Motion

The equations of motion of a non-sedimenting ellipsoid suspended in a flow field was theoretically studied by Jeffery (1922). In the analysis, inertia effects were neglected. It was found that the
forces acting on the ellipsoid tend to make the ellipsoid adopt the same rotation as the surrounding fluid and to set the ellipsoid with its axes parallel to the principal axes of distortion of the surrounding fluid. Thus, if there is a velocity gradient in the fluid, the ellipsoid will be subjected to a torque arising from the uneven viscous stresses of the fluid. It was shown that the torque caused the ellipsoid to rotate end over end in the flow field and an equation for the frequency of rotation was developed. Experimental and analytical studies of rigid fibre motion and rotation based on Jeffery's model have recently been carried out by Szeri et al. (1992) and Stover et al. (1992). Mason (1950) applied Jeffery's equations to derive equations which relate the collision frequency of fibres in a shear flow to the concentration of fibres in the fluid, fibre size and shear rate. The "critical concentration", which designates the maximum concentration of fibres at which no fibre interaction takes place, was calculated by assuming that there is only one fibre within each volume swept by a fibre.

Forgacs and Mason (1959) examined the flow of flexible fibres in a simple shear flow. Two types of fibre deformation motions were reported: "springy orbits" and "snake orbits". For stiffer fibres, small deflections, or "leaf spring" bending, were observed (Figure 2.2 A). As the flexibility was increased, the fibre deformations became larger and were observed to make "snake turns" (Figure 2.2 B).

Gradon et al. (1987) analytically determined the motion and deposition of rigid fibres around and onto an infinite cylinder. The analytic solution to the velocity flow field around an infinite cylinder, which was solved by Lamb (1932) by applying the Oseen equation, was used.
together with equations for the force and torque acting on an immersed particle in a low Reynolds number flow presented in a text by Happel and Brenner (1965). Gradon and Podgorski (1990) and Podgorski et al. (1995) considered the motion and deposition of perfectly flexible, or thread-like, fibres around and onto an infinite cylinder. Again, an analytic velocity profile was used for the flow around an infinite cylinder. In these analyses, Cox’s (1970) leading order approximation for the forces acting on the fibre was used. It was found that the perfectly flexible fibres tended to align with the streamlines near the cylinder and were much less likely to make contact with it than rigid fibres.

All of the studies presented in this section calculate the hydrodynamic forces acting on a particle or fibre suspended in a fluid using analytical equations, where the solution to the flow field is determined with no particle present in the flow. The next section will discuss the use of CFD to determine the motion of particles in a fluid. Since the use of CFD requires the flow domain to be discretized into a mesh of a finite number of elements, automatic meshing strategies will also be discussed. Experimental work of the flow of fibres through apertures in pulp screening applications will be discussed in Section 2.6.

2.2 Application of CFD and Mesh Generation to Fibre Flow

Hu et al. (1992) developed a package that simulates an unsteady two-dimensional two-phase flow of solids and liquids. By using the finite element method the Navier-Stokes equations were solved to determine the flow velocities and the surface stresses acting on the particles. The surface stresses were then used to calculate the motion of the particles, using Newton’s equations of motion. An automatic two-dimensional meshing algorithm was used to generate a mesh of the domain composed of 6 node triangular elements. The numerical simulations presented in the work showed the effect of vortex shedding on the motion of infinite cylinders and reproduced the kind of drafting, kissing and tumbling motion which is observed in the sedimentation of spheres. Feng et al. (1994) used the package developed by Hu et al. (1992) to model various two-dimensional motions of particles in a fluid.

The development of an automatic three-dimensional mesh generating algorithm is an essential step in the development of a three-dimensional two-phase fibre-fluid flow package.
similar to the one presented by Hu et al. (1992) in two dimensions. Although the development of such a package is beyond the scope of this thesis, an automatic three-dimensional mesh generating algorithm has been written with the intention that it will be utilized in the future in the development of a dynamic fibre bending simulation to be applied in the design of an accurate fibre flexibility measurement device.

A number of automatic mesh generating algorithms for two dimensions have been developed. Blacker and Stephenson (1991) presented a mesh generating technique called "paving" which meshes arbitrary two-dimensional geometries with quadrilateral elements. Most automatic mesh generating algorithms apply the Delaunay (1934) triangulation which provides a method to triangulate nodes in a convex domain. Bowyer (1981) and Watson (1981), in the same issue of the Computer Journal, presented algorithms for the construction of the Delaunay triangulation. A description of the triangulation will be given in Chapter 6. Taniguchi et al. (1992) proposed a two-dimensional mesh generating method based on the Delaunay triangulation in which provisions were made to allow for complex non-convex geometries. Weatherill and Hassan (1994) developed a method which constructs three-dimensional unstructured tetrahedral meshes using the Delaunay triangulation and is based on the algorithm of Bowyer (1981). The efficiency of the algorithm reduces the computer time for mesh generation of realistic flow domains to the order of minutes on today's workstations. The automatic three-dimensional mesh generating algorithm developed in this thesis for three-dimensional pulp fibres is based on the work presented by Weatherill et al. and will be discussed in detail in Chapter 6.

2.3 Methods to Measure Fibre Flexibility

As mentioned in the Introduction, pulp fibre flexibility plays an important role in paper quality and this fact has contributed to industries' keen interest in the ability to measure wet fibre flexibility. Although a number of fibre flexibility measurement methods have been proposed, no on-line fibre flexibility measurement device is available at this time. The reason for this is that no method proposed to date is fast and accurate enough to be used in an industrial process.

Seborg and Simmonds (1941) developed an apparatus to directly measure the stiffness of wet or dry fibres. A single fibre is supported as a cantilever and a bending stress is applied at the
free end by a quartz spring. Although the apparatus was able to accurately determine the stiffness of straight fibres, the procedure is extremely tedious and labour intensive.

Robertson et al. (1961) proposed a method to classify fibres according to a "flexibility index". The method is based on the observation that the motion of fibres in a laminar shear flow describe rotational orbits which reflect their individual flexibilities. Rigid fibres were seen to rotate with no flexing or bending and were classified as "type A orbits". "Type B orbits" designated fibres that bend as a whole along their entire lengths where the motion of the fibre ends were not independent of each other. Very flexible fibres were classified as "type C orbits" where the bending was the same as the "snake orbits" observed by Forgacs and Mason (1959). A different type of classifying approach was proposed by Shallhorn and Karrin (1981) and Kuhn et al. (1990) where fibres of different stiffnesses are classified by their screening characteristics.

Frolander and Hartler (1970) used a semi-empirical relation to estimate fibre stiffness from the shear modulus of a fibre network. As with attempts to use mat compressibility to determine fibre stiffness (Jones (1963) and Elias (1967)), no direct relationship between mat shear modulus or mat compressibility with fibre flexibility was established.

Samuelsson (1963) developed a method to estimate the stiffness values of fibres by clamping individual fibres as a cantilever to the wall of a channel through which water flows in the laminar regime. By estimating the hydrodynamic forces acting on the fibre and measuring the deflection, a value for fibre stiffness can be determined. The method could not be used to measure the stiffness of fibres whose length was less than approximately 1.5 mm.

Mohlin (1975) and Steadman and Luner (1983) both developed conformability methods to determine fibre stiffness. In both methods a wet standard hand sheet is placed and compressed against a glass slide which has wires attached to it. The flexibility of a fibre lying across a wire is estimated by relating the compressive force and the distance from the wire to the contact point of the fibre and slide.

Tam Doo and Kerekes (1981) and (1982) developed a fibre flexibility measurement method in which a single fibre is placed in a notched capillary and is deflected by a fluid flowing through the capillary. Fibre deflection is controlled by the flow rate of the fluid and is measured by a microscope. The stiffness value is determined by relating the estimate of the hydrodynamic forces acting on the fibre with the measured deflection.
Kuhn et al. (1995) developed an apparatus to measure fibre flexibility based on the deflection of fibres as they enter normal to a main channel from a capillary flow. Again, an estimate of the hydrodynamic forces and the measured deflection is used to determine a stiffness value. All of the fibre flexibility measurement methods, which relate the applied forces and deflections to determine stiffness values, published to date rely on small deflection analysis. A large deflection analysis has not been performed on the fibre stiffness measurement methods to quantify the limitations associated with the small deflection analysis. Large deflection analysis also provides the ability to investigate important physical effects that cannot be accounted for by small deflection analysis, such as contact friction.

2.4 Dynamics of Flexible Beams

To develop a mathematical model describing the motion of a flexible pulp fibre, a dynamic analysis must be performed. The development of the dynamic equations of motion of a flexible fibre will be presented in Chapter 4. It will be shown that the equations of motion are of integro-differential form and need to be discretized to be solved numerically. Since the geometry of a pulp fibre can be approximated by a long slender cylinder, the dynamics of fibres can be represented by the equations of motion for a beam in vibration. The vibration of stationary beams\(^1\) is treated in many textbooks (e.g. Rao (1986)); however the problem of a translating, rotating and vibrating beam is more complicated. A method to model a vibrating body in general motion is to superimpose the natural or free vibrations of the body onto the equations for rigid body motion. The natural vibrations of any body are determined by observing the vibrations of the body that arise when the body is allowed to vibrate freely from an initial deflected state. Although no dynamic model for a flexible, but not perfectly flexible, pulp fibre has been proposed in literature, the tools developed in the field of research of the motion of robots and space vehicles with flexible links can be utilized. It will be assumed that all deflections are small so that Euler-Bernoulli beam theory can be applied.

Hughes (1974) examined the dynamics of flexible appendages of a space vehicle. It was stated that the general motion (i.e. with external influences such as forces present) of a flexible

\(^1\) Here stationary refers to the fact that the undeflected axis of the beam is stationary, i.e. the beam is vibrating in a stationary location.
body can be viewed as a superposition of natural motions where the natural motions vary in time, and was referred to as a modal expansion. Two families of natural vibration, "constrained" and "unconstrained", were discussed. Constrained natural motions of vibration of the flexible appendages referred to those that were fixed to the main body in which no attitude motion of the main body was allowed. Unconstrained motions referred to the case where the main body, as a whole, was allowed to oscillate. It was shown that both constrained and unconstrained modal expansions had their own advantages. Furthermore, it was shown that the two were mathematically equivalent in the infinite limit. For the case of a flexible pulp fibre moving freely in a flow field, the unconstrained natural motions are most appropriate for a modal expansion. This implies that using the modes of free vibration of a beam in lateral vibration with both ends unconstrained or free can be applied to solve the general motion of a flexible fibre.

Book (1984) developed non-linear equations of motion for a robot manipulator with flexible links and rotary joints. The kinematics of both the rotary-joint motion and the link deformation were described by $4 \times 4$ transformation matrices. Since the link deflections were assumed to be small, the link transformation was shown to be composed of summations of assumed link shapes. These assumed link shapes are determined from modal expansions. Wang and Vidyasagar (1992) used a similar approach to model a robot with a flexible last link. Hastings and Book (1987) developed a linear state-space model for a single link flexible manipulator using assumed link shapes as in Book (1984), and compared the simulation results to a direct-drive arm. It was shown that the nodal eigenvalues agreed with experimentally determined frequency values. The relevance of the work performed in the study of robots with flexible links is that the modal expansion method provides a viable method to mathematically model the general motion of a flexible beam or fibre.

2.5 Methods of Discretizing Beams in Vibration

By applying Hamilton's principle (see Goldstein (1980)) the equations of motion of a flexible pulp fibre will be developed in Chapter 4. These equations are coupled and integro-differential in form. To solve them, approximate numerical methods must be used. In particular, the function representing the deflection of the fibre from its undeflected axis, $u(x,t)$, must be linearized by
discretization. The independent variables \( x \) and \( t \) represent the position along the axis of the undeflected beam or fibre and time, respectively. Some methods discretize \( u(x,t) \) by modelling the beam by a set of elements which interact with each other at their respective nodes. Other methods use a number of assumed mode shape functions to discretize \( u(x,t) \).

The finite element method is a numerical procedure in which a complex structure, i.e. the vibrating beam, is considered as an assemblage of a number of smaller elements which are interconnected at joints or nodes. These elements are considered to be continuous structural members. A representation of an element is depicted in Figure 2.3. Here, \( f \) represents forces and moments applied on the element, \( w \) represents displacements and rotations and \( l \) represents the length of the beam. Using Euler-Bernoulli theory for beams, it is possible to develop equations for the beam element. Letting \( w(x,t) \) represent the displacement/rotation in the direction of vibration within the element, it can be expressed in terms of the unknown node displacements \( w_i(t) \) in the form

\[
w(x, t) = \sum_{i=1}^{j} N_i(x) w_i(t) \tag{2.8}
\]

where \( N_i(x) \) is the shape function corresponding to the node displacement \( w_i(t) \) and \( j \) is the number of unknown node displacements. In the case depicted in Figure 2.3, \( j = 4 \). It should be noted that \( w_i(t) \) is treated as an unknown. Applying the necessary boundary equations, in this case

\[
w(0, t) = w_1(t), \quad \frac{\partial w(0, t)}{\partial x} = w_2(t), \quad w(l, t) = w_3(t), \quad \frac{\partial w(l, t)}{\partial x} = w_4(t) \tag{2.9}
\]

it is possible to find \( j \) shape functions to satisfy equation (2.9), provided that appropriate functions are chosen. The mass, \( M \), and stiffness, \( k \), matrices for the element can be determined and the equations of motion for the element become

\[
M \ddot{w} + kw = f \tag{2.10}
\]

where \( w \) is a vector of node displacements, \( f \) is a vector of node forces which can be determined by considering virtual work, and the dot represents the derivative with respect to time. The solution to the system can then be calculated by integrating all the elements and solving the resulting system of ordinary differential equations.

Rao (1986) presented the mass and stiffness matrices for the beam element shown in Figure 2.3 by assuming the shape function to be a third order polynomial of the form

\[
N_i(x) = a_i + b_i x + c_i x^2 + d_i x^3, \quad i = 1, 2, 3, 4. \tag{2.11}
\]
Paz (1980) discretized the motion of a beam in vibration by letting

\[ w(x, t) = Q(x) \sin \omega t \]

and introducing it into the equation of motion for a beam in free vibration. Here \( Q(x) \) represents the amplitude of a mode of free vibration and at frequency \( \omega \). Applying the required boundary conditions, the equations of motion can be obtained.

Meirovitch (1990) discussed the Rayleigh-Ritz method, a form of the finite element method. While the general finite element method, proposed above, discretizes the structure by an assemblage of smaller elements, the Rayleigh-Ritz method considers the entire structure as one element and discretizes the equations by a number of functions. The method consists of selecting \( m \) functions \( Q_1, Q_2, \ldots, Q_m \) from a complete set of admissible functions (see Meirovitch (1990) for a definition of admissible functions) and discretizing \( u(x, t) \) as follows:

\[
\begin{align*}
  u(x, t) &= \sum_{n=1}^{m} Q_n(x) \tau_n(t) \\
  & \text{(2.12)}
\end{align*}
\]

where \( \tau_n(t) \) are the displacement coefficients to be determined. While the Rayleigh-Ritz method is a form of the finite element method, the difference between it, in this context, and the general finite element method is that the Rayleigh-Ritz method discretizes the function \( u(x, t) \) but not the beam, as the general finite element method does. Meirovitch (1990) showed that convergence of the Rayleigh-Ritz method is very rapid for the solution to the eigenvalue problem of a beam in free vibration. Comparing results from the general finite element method (where a polynomial
function was used to represent the shape of the element) and the Rayleigh-Ritz method for the eigenvalue problem, Meirovitch (1990) showed that the Rayleigh-Ritz method yields considerably better accuracy than the general finite element method. In fact, the first two natural frequencies computed by the Rayleigh-Ritz method with \( m = 3 \) were more accurate than those computed by the general finite element method with \( m = 10 \).

The advantages of the general finite element method become apparent when the beam is not uniform. The finite element method permits the use of different beam elements which can model a nonuniform beam. The Rayleigh-Ritz method is applicable only to regularly shaped objects. Weaver et al. (1990) stated that for a beam with a regular geometry the Rayleigh-Ritz method will always be superior in accuracy and computational efficiency to the general finite element method.

As mentioned previously, Hughes (1974) defined natural motion of a dynamic system in vibration as the motion when no external influences, meaning forces and torques, are present. It was stated that the general motion may be viewed as a superposition of the natural motions, where contributions of the natural motions vary in time. This leads to the assumed modes method, as presented by Book (1984). The assumed modes method is a form of the Rayleigh-Ritz method where the natural motion functions of Hughes (1974) are used as the admissible functions, \( Q_i(x) \), in equation (2.12).

### 2.6 Fibre Screening

Pulp fibre screening is a unique process since it attempts to fractionate fibres based on stiffness. As mentioned in the Introduction, pressurized screens are the primary unit for shive removal. In pulp screening, a flow parallel to the screen plate is drawn into a series of apertures. Separation of pulp fibres is determined by the ability of fibres with different properties to pass through the apertures.

Thomas and Cornelius (1982) investigated a boundary layer flow over a plate with a slot. A recirculating zone adjacent to the upstream side of the slot was observed. Thomas and Cornelius also noted that the flow passing through the slot originated from an "exit layer" in the main flow, adjacent to the plate of the main flow.
Riese et al. (1969) used a dilute suspension of rigid nylon fibres to observe the passage of the fibres through a perforated plate normal to the flow field. It was observed that the fluid forces aligned the fibres perpendicular to the plate and the alignment effect was increased with increased flow rate. Fluid flow patterns were found to be the most important variable in determining the probability of fibre passage.

Gooding and Kerekes (1992) defined a quantity called the "passage ratio" as the pulp consistency (defined as (the mass of the fibres)/(the mass of the suspension)) in the flow through the screen aperture divided by the consistency in the flow immediately upstream of the aperture. Gooding (1986) and Kumar (1991) studied the motion of fibres in a very dilute suspension in a laboratory flow loop consisting of a channel with a single slot. Most importantly, both experimental studies showed that fibre flexibility plays an important role in fractionation. Gooding (1986) also filmed the motion of individual fibres at a channel and slot junction and classified five different types of motion. It was shown that fibre interaction with the downstream slot wall plays an important role in determining the passage of fibres through the slot.

Gooding (1986) and Gooding and Kerekes (1989) suggested that two mechanisms affect screening: the "wall effect" and the "entry effect" or "turning effect". Since only fibres in the exit layer are candidates for entering a screen slot, the concentration of fibres in the exit layer compared to the overall upstream channel concentration will affect the passage ratio. Since the exit layer thickness is less than the length of a fibre, it is reasonable to expect that the concentration of fibres in the exit layer will be less than the main channel fibre concentration. Gooding and Kerekes observed this concentration difference and attributed it to the wall effect. The effect of fibre flexibility on the wall effect is unknown. The entry effect was described to be influenced by the ability of fibres to turn and enter the slot. Since the thickness of the exit layer is much less than a fibre length, any fibre that is eligible for passage through the slot must be aligned almost parallel to the channel wall, or screen plate. The ability of the fibre to exit through the slot is therefore determined by the ability of the fibre to turn or bend to enter the slot. Flexible fibres have an advantage over stiff ones in that they have a greater ability to bend along the streamlines and will have a better opportunity to flow through the slot.

Very recently, Olsen (1996) developed a numerical model for the motion of rigid fibres in a channel with a slot. To model the contact between the fibre and the flow domain walls, Olsen
used constraint equations to solve for the force of contact. A limitation of this model was the inability to predict the trajectory of a fibre after it contacted the channel and slot corner and slot wall simultaneously.

Typical fibre stiffnesses are on the order of $10^{-11}$ to $10^{-14}$ Nm$^2$ (Samuelsson (1963), Tam Doo and Kerekes (1981) and (1982), Steadman and Luner (1983)). Gooding (1986) estimated the stiffness of mechanical pulp shives to be on the order of $10^{-4}$ to $10^{-8}$ Nm$^2$. As was presented by the work of Gradon and Podgorski (1990) and Podgorski et al. (1995), a perfectly flexible fibre in the exit layer would be expected to enter the slot. For this reason, a numerical model of nominally flexible fibres is required in order to study the effect of flexibility on the ability of fibres to pass through a slot in screening applications. To this date no such model has been proposed.

2.7 Summary

An estimate of the hydrodynamic forces acting on fibres suspended in a shear flow is essential in the study of the static and dynamic behaviour of the fibres in pulp processing and measurement equipment. Two general methods can be applied to determine these forces. One of them involves analytical and empirical equations to estimate the forces while the other attempts to determine the forces directly, from a rigorous computational analysis using CFD. Section 2.1 discussed topics in literature which pertain to analytical and empirical methods for force estimation, while Section 2.2 discussed the application of CFD to determine the forces. While it is clear that a properly developed CFD model will be able to more accurately determine these forces, its application to all but the simplest problems is limited by computer resources. For this reason the analytical method for estimating forces (Cox’s (1970) equations) was utilized in this thesis for the fibre motion modelling. All of the fibre motion models presented to date have assumed the fibres to be rigid or perfectly flexible. Since a perfectly flexible fibre tends to follow the streamlines (Gradon and Podgorski (1990) and Podgorski et al. (1995)), its application is of limited use.

The application of static beam theory is required in the development of fibre flexibility measurement devices. Section 2.3 presented different methods which have been proposed to estimate the flexibility of pulp fibres. All of the methods used small deflection analysis. Large deflection analysis will be applied to four fibre flexibility measurement method geometries to
study the errors associated with the application of the small deflection analysis. The implications of the errors associated with the use of small deflection analysis are important because of the fact that the flexible fibre motion models rely on small deflection, or Euler-Bernoulli, beam theory.

Section 2.4 presented literature relevant to modelling the motion of a flexible body. The discussion centred around the fact that the motion of a flexible body can be modelled by the equations of rigid body motion and the superposition of the modes of free vibration. In Section 2.5 it was shown that the Rayleigh-Ritz method utilizes the assumed modes of free vibration to discretize the dynamic equations of motion of a vibrating body.

Section 2.6 discussed experimental investigations of the flow of fibres for the application of fibre screening and fractionation. The experimental results clearly show that the flexibility of a fibre is an important property that can affect the fractionation of pulp fibres. To date, no numerical models which predict the trajectories of flexible fibres have been proposed. By utilizing the analytical equations for hydrodynamic force estimation reviewed in Section 2.1 and the methods to model flexible body motion of Sections 2.4 and 2.5, a numerical model will be developed in Chapter 4 which will model the flow of flexible pulp fibres in geometries similar to those reviewed in Section 2.6. Chapter 5 will provide qualitative and quantitative comparisons between the numerical flexible fibre motion model and the experimental work presented in the literature.
Chapter 3

Fibre Statics

Wet fibre flexibility determines the ability of fibres to deform and entangle during the consolidation stage of papermaking, and is recognized as an important fundamental fibre property influencing the strength, and surface and optical properties of the resulting paper. A number of experimental methods have been proposed to measure fibre flexibility. All the work published to date has used small deflection beam theory to analyse the static deflections of the pulp fibres. The purpose of this chapter is to determine the errors associated with using small deflection theory by applying large deflection analysis to four fibre flexibility measurement methods and to assess errors due to physical effects that may not be accounted for by small deflection analysis. The equations for the large deflection of beams will be introduced in the first section. These equations consist of coupled ordinary differential equations which lead to a boundary value problem. The shooting method is used to solve the equations and will be described in the second section. The third section will detail four fibre flexibility measurement devices, proposed in literature, that attempt to give a numerical value for fibre stiffness. The fourth section will investigate the error associated with the small deflection analysis for each of the four fibre flexibility measurement methods detailed in the third section, and assess the error due to physical effects that may not be accounted for by small deflection analysis. Conclusions will be presented in the fifth section.
3.1 Static Deflection of Fibres

The stiffness of an object is determined by the application of a known force and a comparison of the resulting deflection to a theoretical deflection. This section will discuss the theory of the static deflection of fibres or beams.

The equation governing the deflection of a beam is given by

\[ \frac{M}{EI} = \frac{d\theta}{ds} \]  

(3.1)

where \( M \) is the bending moment, \( E \) is the modulus of elasticity, \( I \) is the cross-sectional moment of inertia (\( EI \) is defined as the beam stiffness), \( \theta \) is the angle between a tangent to the element and a vertical reference line, and \( s \) is a measure of length along the beam. For any function in the \( x-y \) Cartesian plane (Frish-Fay (1962)),

\[ \frac{d\theta}{ds} = \frac{d^2y/dx^2}{(1 + (dy/dx)^2)^{3/2}}. \]  

(3.2)

In the small deflection analysis, it is assumed that the deflection of a beam positioned along the length of the \( x \)-axis is small, and \((dy/dx)^2\) is small compared to 1, giving the differential equation for beam deformation, from equation (3.2), as

\[ \frac{d^2y}{dx^2} = \frac{M}{EI}. \]  

(3.3)

For simple geometries, equation (3.3) is easy to evaluate analytically. Since the deflections are assumed to be small in the small deflection analysis, a normal force acting on a beam remains normal to the undeflected axis and the end co-ordinate of the deflected beam is the same as that of the undeflected beam. This has the consequence of theoretically making the deflected beam increase in length. Furthermore, the small deflection analysis, as applied above, only allows the application of forces normal to the undeflected beam shape. Tensile forces cannot be applied directly using equation (3.3).

In large deflection theory, \((dy/dx)^2\) is not negligible (Frish-Fay (1962)). A moment balance for a beam element, as shown in Figure 3.1, gives

\[ V = \frac{dM}{ds} = EI \frac{d^2\theta}{ds^2} \]  

(3.4)

where \( V \) is the shear force. A force balance in the normal \( n \)-direction gives

\[ \frac{dV}{ds} = T \frac{d\theta}{ds} + f_n - W \sin(\theta) \]  

(3.5)
where $T$ is the tensile force within the beam, $W$ is the distributed weight per unit length of the beam and $f_n$ is the normal component of the distributed force acting on the beam. Balancing the forces in the tangential $t$-direction leads to

$$\frac{dT}{ds} = V\frac{d\theta}{ds} + f_t + W\cos(\theta) \tag{3.6}$$

where $f_t$ is the tangential component of the distributed force acting on the beam. Equations (3.4), (3.5) and (3.6) are combined to give the following system of non-linear ordinary differential equations,

$$\begin{align*}
E I \frac{d^3 \theta}{ds^3} &= T \frac{d\theta}{ds} + f_n - W\sin(\theta) \\
-\frac{dT}{ds} &= EI \frac{d^2 \theta}{ds^2} \frac{d\theta}{ds} + f_t + W\cos(\theta).
\end{align*} \tag{3.7}$$

Equations (3.7) are non-dimensionalized with $\bar{s} = \frac{s}{L}, \bar{f}_n = \frac{f_n L^3}{EI}, \bar{f}_t = \frac{f_t L^3}{EI}, \bar{T} = \frac{T L^2}{EI}, \bar{W} = \frac{W L^3}{EI}$,
where \( L \) is the length of the fibre, to give

\[
\frac{d^3 \theta}{ds^3} = \frac{f}{d} + f_n - \overline{W} \sin(\theta) \\
\frac{d \tilde{T}}{ds} = \frac{d^2 \theta}{ds^2} \frac{d \theta}{ds} + f_r + \overline{W} \cos(\theta)
\]

Equations (3.7) and (3.8) are all in terms of the position variables \( s \) and \( \theta \). In cases where the locations of both ends of the fibre or beam are required, such as in a simply supported geometry, the \( x \) and \( y \) co-ordinates of each point along the fibre are required. This can be achieved by including the following equations in the analysis,

\[
\frac{dx}{ds} = \frac{d \bar{x}}{ds} = \sin(\theta), \quad \frac{dy}{ds} = \frac{d \bar{y}}{ds} = -\cos(\theta)
\]

where \( \bar{x} = x/L \) and \( \bar{y} = y/L \).

The force balance equations, together with the boundary conditions, lead to a boundary value problem. Simple Runge-Kutta algorithms cannot be used directly, since all of the initial boundary conditions are not available. However, the equations can be solved using relaxation methods, e.g. the finite difference method, or by solving the equations directly. The latter approach was applied in this work by employing the shooting method (Wang and Kitipornchai (1992)). The following section will discuss the numerical approach taken to solve the equations.

### 3.2 Numerical Solution of the Large Deflection Equations

Equations (3.8) and (3.9) represent a system of coupled ordinary differential equations. As mentioned above, since boundary conditions are generally applied at both ends of the beam, the problem is a boundary value problem. Runge-Kutta algorithms which solve ordinary differential equations can only solve them if all the initial boundary conditions are known. This fact makes the algorithms popular for solving time marching problems, where all the initial conditions are known. To solve the boundary value problem, on the other hand, the shooting method was employed. The method applies a non-linear root finding algorithm, in this case the Newton-Raphson method, to a fifth-order Runge-Kutta algorithm, using guesses for the initial unknown boundary conditions to match the final, known boundary conditions (Press et al. (1992)).
The Runge-Kutta method is a subject that is described in many texts, but a brief description will be given here to show its application to the problem at hand. Consider a set of first order differential equations which can be written as
\[
\frac{d\psi_j}{d\varphi} = G_j(\varphi, \psi_1, \psi_2, \ldots, \psi_n)
\]
or in vector form
\[
\psi' = G(\varphi, \psi)
\]  
(3.10)
where the prime ('') represents the derivative with respect to an independent variable, \( \varphi \), \( \psi \) is a vector of the variables for which a solution is sought and \( G \) is a vector of functions. The fifth-order Runge-Kutta method employs a recurrence formula to solve equation (3.10) numerically as follows (see Press et al. (1992))
\[
\psi_{i+1} = \psi_i + \frac{37}{378} K_1 + \frac{250}{621} K_3 + \frac{125}{594} K_4 + \frac{512}{1771} K_6
\]  
(3.11)
where
\[
\begin{align*}
K_1 &= hG(\varphi_i, \psi_i) \\
K_2 &= hG(\varphi_i + \frac{1}{5} h, \psi_i + \frac{1}{5} K_1) \\
K_3 &= hG(\varphi_i + \frac{3}{10} h, \psi_i + \frac{3}{10} K_1 + \frac{9}{40} K_2) \\
K_4 &= hG(\varphi_i + \frac{3}{5} h, \psi_i + \frac{3}{5} K_1 - \frac{9}{10} K_2 + \frac{6}{5} K_3) \\
K_5 &= hG(\varphi_i + h, \psi_i - \frac{11}{25} K_1 + \frac{7}{25} K_2 - \frac{19}{25} K_3 + \frac{573}{500} K_4) \\
K_6 &= hG(\varphi_i + \frac{7}{8} h, \psi_i + \frac{1631}{55296} K_1 + \frac{175}{512} K_2 - \frac{575}{13824} K_3 + \frac{44275}{105024} K_4 + \frac{253}{4096} K_5)
\end{align*}
\]  
(3.12)
and \( h \) is the step size of the independent variable. The advantage of the adaptive step size routine is that the step size, \( h \), is varied according to derivatives of the functions, \( G_j \). Thus, when the functions are changing slowly, large step sizes are taken to speed up the solution, whereas when the functions are changing rapidly, small step sizes are used to ensure the accuracy of the solution.

The steps involved in the Runge-Kutta method are summarized below:

1. Apply the initial conditions for \( \varphi_0 = 0 \) to determine \( \psi_0 \).
2. Find \( \psi_{i+1} \) by applying equations (3.11) and (3.12).
3. Update \( \varphi_{i+1} = \varphi_i + h. \)
4. If \( \varphi_{i+1} \) is less than the final value, then go to step 2.
5. End of algorithm.

To apply the Runge-Kutta method to equations (3.8) and (3.9), the vector, \( \psi \), is defined
as follows

\[
\psi = \begin{bmatrix}
\theta \\
\frac{d\theta}{d\bar{s}} \\
\frac{d^2\theta}{d\bar{s}^2} \\
\bar{r} \\
x \\
y
\end{bmatrix}
\]  

(3.13)

and the vector of functions, G, as

\[
G(\phi, \psi) = \begin{bmatrix}
\frac{d\theta}{d\bar{s}} \\
\frac{d^2\theta}{d\bar{s}^2} \\
\frac{d\theta}{d\bar{s}} + \bar{f}_n - \bar{W}\sin(\theta) \\
\frac{d^2\theta}{d\bar{s}^2} - \bar{f}_r - \bar{W}\cos(\theta) \\
\sin(\theta) \\
-cos(\theta)
\end{bmatrix}
\]  

(3.14)

The independent variable, \( \phi \), of equation (3.10) is replaced by \( \bar{s} \).

The shooting method procedure is described below.

1. Determine \( \psi_0 \) by applying the known initial conditions and using guesses for the unknown initial conditions.

2. Apply the Runge-Kutta algorithm to find \( \psi_f \), the final value of the dependent variables.

3. If the known final conditions match those of \( \psi_f \) to within a defined tolerance, then end the algorithm.

4. Use a Newton-Raphson root finding technique (see, for example, Press et al. (1992)) by comparing the known final conditions to those of \( \psi_f \) to find better guesses for the unknown initial conditions.

5. Go to step 1.
Usually more than one mathematical solution exist for each analysis. However, all of the solutions except one are obviously not physically realistic. To arrive at suitable solutions, it was important to seed the analysis with a good initial guess. Convergence criteria for the Newton-Raphson method and the Runge-Kutta algorithm were on the order of 0.1 %. The boundary conditions are dictated by the geometry and will be discussed in detail for each fibre stiffness measurement method.

3.3 Fibre Stiffness Measurement Methods

As mentioned above, wet fibre flexibility determines the ability of fibres to deform and entangle during the consolidation stage of papermaking, and is recognized as an important fundamental fibre property influencing the strength, and surface and optical properties of the resulting paper. A number of methods for measuring the flexibility of fibres have been proposed. Many of the techniques require the fibres to be handled individually, a slow and laborious task. Other techniques classify fibre flexibility by the fibres' orbital deformation in shear flows (Robertson et al. (1961)) or by their screening characteristics (Shallhorn and Karnis (1981) and Kuhn et al. (1990)), but do not provide a numerical measure of flexibility in terms of the modulus of elasticity and moment of inertia. Four methods that have been proposed to predict fibre stiffness, the inverse of flexibility, will be analysed using small and large deflection analyses:

- Cantilevered Fibre in a T-Junction (Kuhn, et al. (1995));
- Clamped Cantilevered Fibre in a Channel Flow (Samuelsson (1963));
- Simply Supported Fibre in a Capillary (Tam Doo and Kerekes (1982));
- Conforming Fibre on a Wire (Steadman and Luner (1983)).

All of the above methods assumed the fibres to be straight with a circular cross-section and a constant stiffness along the length, and used small deflection analysis. To allow an assessment of the methods as defined in literature, the geometric and boundary assumptions of all methods were those defined in the literature. The assumptions that may not be considered physically reasonable are retained in the analysis to allow an assessment of the methods as defined in the literature. The error due to small deflection analysis will remain regardless of other geometric and boundary conditions assumptions.
The geometries and equations for the analyses of the different fibre stiffness methods will be discussed in this section. The theory for small deflection of beams was used in all of the experimental set-ups published to date. Although the fluid dynamic forces loading the fibres in the different methods are not known accurately, the presented work is not compromised, since any errors associated with the forces affect both the large and small deflection analyses equally.

3.3.1 Cantilevered Fibre in a T-Junction

Kuhn et al. (1995) developed an apparatus to measure fibre flexibility based on the deflection of fibres as they enter normal to a main channel from a capillary flow (T-Junction), see Figure 3.2. As a fibre enters the main channel from the capillary, the portion of the fibre in the channel is subjected to a normal force due to the main channel flow, and moves downstream. The fibre hits the downstream edge of the capillary, pivots until the end of the fibre in the capillary hits the upstream side of the capillary, and bends. The fibre is momentarily held in this position in a state of quasi-steady equilibrium before passing into the main channel flow. The analysis considers the portion of the fibre in the main channel flow to be a cantilever. A high speed CCD camera records the fibre's deflected shape at the T-Junction and the fibre's stiffness is then estimated from its deflection. Initial work was done using small deflection analysis.

The flow in the channel is assumed to be laminar, two-dimensional and fully developed, and the effects of the capillary and capillary flow at the T-Junction are ignored, resulting in the following equation for the velocity profile in the channel:

\[ u(y) = \frac{-4U_m}{(d_1 + d_2)^2}(y^2 + (d_2 - d_1)y - d_1d_2) \]  

(3.15)

where \( U_m \) is the maximum channel flow speed, \( d_1 \) is the vertical distance the capillary projects into the main channel and \( d_2 \) is the vertical distance from the end of the capillary (defined as the origin) to the lower channel wall.

29
Figure 3.2: T-Junction.

As mentioned previously, Cox (1970) developed equations for the forces acting on a long, slender body placed in a Stokes flow assuming the inertia of the fluid to be negligible. To estimate the forces acting on a fibre element suspended in a channel flow (Figure 3.1), first order approximations of Cox's equations are used in the T-Junction method. The normal and tangential components of the force per unit length acting on the fibre are given as

\[ f_n = -4\pi \mu U_n \cdot \left( \ln \left( \frac{D}{L} \right) \right)^{-1}, \quad f_t = -2\pi \mu U_t \cdot \left( \ln \left( \frac{D}{L} \right) \right)^{-1} \tag{3.16} \]

respectively. Here \( \mu \) is the viscosity of the fluid, \( D \) is the diameter of the fibre, \( L \) is the length of the fibre, and \( U_n \) and \( U_t \) are the normal and tangential components of the fluid velocity respectively. In the small deflection analysis, tangential effects are not considered and therefore only the normal component of the force is used. Substituting \( U_n = u(y)\cos(\alpha) \) and \( y = \xi \cos(\alpha) \) into the normal force component of equation (3.16) gives

\[ f_n = -4\pi \mu \cdot \left( \ln \left( \frac{D}{L} \right) \right)^{-1} \cos(\alpha) u(-\xi \cos(\alpha)) \tag{3.17} \]

where \( \alpha \) is defined in Figure 3.2 and \( \xi \) is measured along the undeflected fibre length from the downstream capillary contact point.

The moment, as a function of \( \xi \), is given by

\[ M(\xi) = \int_{-\xi}^{L-\xi} f_n(s + \chi) \chi d\chi \tag{3.18} \]

where \( \chi \) is a variable used for the integration. The equation for deflection using small deflection
theory (see equation (3.3)) is given by
\[
\frac{d^2 \eta}{d\xi^2} = \frac{M(\xi)}{EI}
\] (3.19)
where \( \eta \) and \( \xi \) are defined in Figure 3.2 and \( y(x) \) of equation (3.3) is replaced by \( \eta(\xi) \). Substituting equation (3.18) into equation (3.19) and solving with the boundary conditions:
\[
\eta(0) = 0 \text{ and } \frac{d\eta(0)}{d\xi} = 0
\]
gives
\[
\eta(\xi) = \frac{2\pi \mu U_m \cos(\alpha) \xi^2}{45EI \cdot (d_1 + d_2)^2 \ln\left(\frac{D_z}{L}\right)} \left( \cos^2(\alpha)\xi^4 - 3 \cos(\alpha)(d_2 - d_1)\xi^3 - 15d_1d_2\xi^2 
+ 10(L^2\cos(\alpha)(3(d_2 - d_1) - 2L\cos(\alpha)) + 6Ld_1d_2)\xi 
- 60L^3\cos(\alpha)(d_2 - d_1) + 45L^4\cos^2(\alpha) - 90L^2d_1d_2 \right) \tag{3.20}
\]
The end deflection, \( \delta \), can be determined by setting \( \xi = L \) to give
\[
\eta(L) = \delta = \frac{2\pi \mu U_m L^4 \cos(\alpha)}{45EL \cdot (d_1 + d_2)^2 \ln\left(\frac{D_z}{L}\right)} (L \cos(\alpha)(33(d_2 - d_1) - 26L \cos(\alpha)) + 45d_1d_2). 
\tag{3.21}
\]
Equation (3.21) can be used directly to determine the stiffness, \( EI \).

For the large deflection analysis, the boundary equations are determined as follows:
at \( s = 0 \), i.e. \( x = y = 0 \):
- initial angle; \( \theta = \alpha \)
and at \( s = L \):
- zero tension; \( T = 0 \)
- zero shear; \( V = 0 \) or \( \frac{d^2 \theta}{ds^2} = 0 \)
- zero moment; \( M = 0 \) or \( \frac{d\theta}{ds} = 0 \).

### 3.3.2 Clamped Cantilevered Fibre in a Channel Flow
Samuelsson (1963) proposed a method to measure fibre stiffness by fixing a fibre as a cantilever to the wall of a channel through which water flows, Figure 3.3. Fibre deflection is observed by a microscope and the deflection can be varied by varying the water flow rate through the channel. The channel dimensions are 6.0 cm wide by 0.4 cm high and the velocity profile is assumed to be laminar and fully developed.

Samuelsson (1963) estimated the normal force per unit length acting on the fibre by the
The general equation is given by:

\[ f_n = \frac{1}{2} C_D \rho D u^2 \tag{3.22} \]

where \( \rho \) is the density of the fluid and the coefficient of drag, \( C_D \), is estimated as \( k/Re \), with \( k = 20 \). The local Reynolds number, \( Re \), is based on the diameter of the fibre, \( D \), and the local flow velocity in the \( x \)-direction, \( u \). Since Samuelsson (1963) used small deflection analysis, no provision for a tangential component of the force can be applied. Following the same approach as in the previous method, Samuelsson (1963) showed that the deflection is given by

\[ \delta = \frac{11}{320} \left( \frac{Q u k L^5}{ab^2 EI \left( 1 - 0.63 \frac{b}{a} \right)} \right) \left( 1 - \frac{13 L}{33 b} \right) \tag{3.23} \]

where \( a \) is the channel half-width, \( b \) is the channel half-height and \( Q \) is the volumetric flow rate. The flow rate is controlled by a valve to achieve deflections, viewed by a microscope, of approximately 15% based on the fibre length. Equation (3.23) is then applied to determine the stiffness, \( EI \).

The boundary conditions for the large deflection analysis are the same as for the T-Junction geometry, as given above, with the stipulation that \( \alpha = 0 \) at \( s = 0 \).
3.3.3 Simply Supported Fibre in a Capillary

Tam Doo and Kerekes (1982) developed a fibre stiffness measurement method in which a single fibre is placed in a notched capillary, and is deflected by a fluid flowing through it. By controlling the flow rate through the capillary it is possible to control the fibre deflection which is viewed through a microscope. The basic geometry is represented in Figure 3.4 and the forces acting on the fibre are shown schematically in Figure 3.5.

The equation for the stiffness was presented by Tam Doo and Kerekes (1982) as

\[ EI = \frac{\mu^2 R^4}{2 \rho \delta D} C \cdot (Re_b)^K \]  \hspace{1cm} (3.24)

where \( \mu \) is the viscosity of water (N·s/m²), \( R \) is the capillary tube inner radius (0.00075 m), \( \rho \) is the density of water (kg/m³), \( \delta \) is the deflection of the fibre (m), \( D \) is the diameter of the fibre as measured through the microscope (m), \( Re_b \) is the Reynolds number based on the fibre diameter and the average flow velocity, and \( C \) and \( K \) are constants. This equation was derived using small deflection theory concepts and the constants were determined experimentally by observing the deflection of a metal wire of known stiffness. For a fair comparison between the small and large deflection analyses, the force distribution acting on the fibre must be known or assumed. The following analysis is used to estimate the forces acting on the fibre, using the approach of Tam Doo and Kerekes (1982) where possible.

The normal force per unit length acting on a cylinder is given by equation (3.22), but the drag coefficient for a cylinder, in contrast to the estimate used by Samuelsson (1963), is
Figure 3.5: Simply supported beam.

approximated by (Sucher and Brauer (1975))

\[ C_D = 1.18 + \frac{6.8}{Re^{0.89}} + \frac{1.96}{Re^{0.5}} - \frac{0.0004Re}{1 + 3.64Re - 7Re^2} \]  \hspace{1cm} (3.25)

where \( Re \) is the Reynolds number based on the diameter of the cylinder and the undisturbed local fluid velocity. Equation (3.25) is a more accurate description of \( C_D \) than that used by Samuelsson (1963) and is therefore used in this analysis. If the velocity distribution at the capillary entrance is assumed to be uniform, then the normal force is constant across the span length, the reaction forces \( R_1 \) and \( R_2 \) are both equal to \( f_nL/2 \) and the moment at \( x \) becomes

\[ M(x) = \frac{1}{2}f_nL \cdot (L - x) - \frac{1}{2}f_n \cdot (L - x)^2. \]  \hspace{1cm} (3.26)

The boundary conditions require that the deflection must be zero at the ends, \( y(0) = y(L) = 0 \). Substituting equation (3.26) into the equation for small deflection theory (equation (3.3)) and solving for the boundary conditions gives the fibre deflection as a function of \( x \):

\[ y(x) = \frac{f_n x \cdot (x^3 - 2Lx^2 + L^3)}{24EI}. \]  \hspace{1cm} (3.27)

The stiffness is determined by letting \( x = L/2 \) and defining \( \delta = -y(L/2) \):

\[ EI = \frac{5f_nL^4}{384\delta}. \]  \hspace{1cm} (3.28)

Again, small deflection analysis only allows for forces acting normal to the undeflected fibre. Tangential forces due to friction exist at the locations where the fibre contacts the capillary walls. In the large deflection analysis, the effect of these forces can be considered. The boundary conditions for the large deflection analysis are as follows:

at \( s = 0, x = y = 0 \):

34
• zero moment; \( M = 0 \) or \( \frac{d\theta}{ds} = 0 \)

and at \( s = L \):

• zero deflection; \( y = 0 \)
• zero moment; \( M = 0 \) or \( \frac{d\theta}{ds} = 0 \)

• the tensile force is determined by the friction caused by the interaction of the fibre with the capillary wall;
  \[ T_1 = T \sin(\theta) - V \cos(\theta) \] or \[ T_1 = T \sin(\theta) - EI \frac{d^2\theta}{ds^2} \cos(\theta) \].

To account for the constant span length, the length of the fibre, \( L \), is adjusted in the Newton-Raphson root finding algorithm, thus achieving the desired geometry.

3.3.4 Conforming Fibre on a Wire

Steadman and Luner (1983) developed a method to measure fibre stiffness by measuring the span length of fibres placed over a wire and pressed with a known constant force. The method consists of forming a thin fibre network on a standard sheet machine wire, couching the network off the wire and pressing it in contact with a glass slide, which has several 25 \( \mu \)m diameter stainless steel wires attached across its surface. The span length is determined by measuring the distance between the centre of the wire and the position where the fibre first contacts the slide.

Figure 3.6 shows the geometry of a fibre placed on a wire and loaded with a distributed force per unit length, \( f_w \). Since the geometry is symmetric about the \( y \)-axis, the analysis will only include the right half portion of the fibre. The forces acting on the fibre are shown in Figure 3.7. In the small deflection analysis, it is assumed that at the point of contact of the fibre with the glass slide at \( x = a \) the fibre has no curvature. This assumption forces the moment to be zero at this point and will be investigated in the large deflection analysis.

In the small deflection analysis, the moment as a function of \( x \), is given by

\[
M(x) = R \cdot (a - x) - \frac{1}{2} f_w \cdot (a - x)^2.
\]  \hspace{1cm} (3.29)

Substitution of equation (3.29) into the equation for small deflection theory and solving for the boundary conditions:

• at \( x = 0 \) the slope is zero: \( \frac{dy}{dx} = 0 \)
• at \( x = a \) the fibre is resting on the slide: \( y = 0 \)

35
Figure 3.6: Conforming Fibre on a Wire.

gives the deflection as

\[ y(x) = \frac{f_w x^4 + 4(R-f_w a)x^3 - 6a \cdot (2R-f_w a)x^2 + a^3 \cdot (8R-3f_w a)}{24EI}. \]  

(3.30)

The reaction force, \( R \), can be determined by setting \( y(0) = d \), where \( d \) is the wire diameter, to get

\[ R = \frac{3(f_w a^4 - 8dEI)}{8a^3}. \]  

(3.31)

Substituting equation (3.31) and \( x = a \) into equation (3.30) and solving for the stiffness gives

\[ EI = \frac{f_w a^4}{72d}. \]  

(3.32)

Two different approaches were taken to determine the deflection of a conforming fibre using the large deflection analysis. The first method parallels the small deflection analysis, where it is assumed that the curvature of the fibre at the point of contact with the glass slide is zero and consequently the moment is zero. The boundary conditions in this case are as follows:

At \( s = 0 \):

- the fibre is horizontal: \( \theta = \pi/2 \)

and at \( s = a \):

- the fibre is horizontal: \( \theta = \pi/2 \)
- the moment is zero: \( \frac{d\theta}{ds} = 0 \)
- the tension is zero: \( T = 0 \).

As in the simply supported method, the geometry is matched by varying the fibre length variable, \( L \), in the Newton-Raphson root finding algorithm to achieve the desired geometry, i.e. at \( s = 0 \), \( y = d \) and at \( s = L \), \( y = 0 \).

The second method models the reaction force of the glass slide by assuming the slide to act as a linear spring. Mathematically, the reaction force is determined as follows:
if \( y > 0 \) then \( R_{\text{slide}} = 0 \), else \( R_{\text{slide}} = -ky \)

where \( R_{\text{slide}} \) is the reaction force per unit length in the positive \( y \)-direction, and \( k \) is a constant. As \( k \) approaches infinity, the slide is modelled to be rigid, which, practically, it is. Numerically, however, the system of equations becomes unstable as \( k \) is increased. Modelling the reaction force this way can also be interpreted as allowing the fibre's cross-section to deflect as it is compressed by the force \( W \) and the slide. For this reason, the method has some merit and is investigated. The boundary conditions are similar to those of the cantilevered methods:

at \( s = 0 \):

- the fibre is horizontal: \( \theta = \pi/2 \)

and at \( s = L \):

- the moment is zero: \( \frac{d\theta}{ds} = 0 \)
- the shear is zero: \( \frac{d^2\theta}{ds^2} = 0 \)
- the tension is zero: \( T = 0 \).

### 3.4 Small and Large Deflection Analyses

In the following comparison of the small and large deflection analyses, it will be assumed that the results obtained from the large deflection analysis represent the true fibre deflections, to which the small deflection analysis results can be compared. As mentioned previously, in the small deflection analysis, only the initial normal forces acting on a beam or fibre can be considered analytically. In the large deflection analysis, on the other hand, normal and tangential forces as well as the weight can be applied in the numerical analysis. Furthermore, these forces can be updated as the geometry of the fibre changes due to bending. In all the analyses a 30 \( \mu \)m fibre
3.4.1 Cantilevered Fibre Methods

Since the Cantilevered Fibre in a T-Junction and the Clamped Fibre in a Channel Flow measurement methods are geometrically similar, i.e. both are cantilevered, the comparison results will be presented together. For the present analysis, the cantilevered portion of the fibres is assumed to be 2 mm in length.

As mentioned previously, the end co-ordinate of a deflected beam, or fibre, remains the same as that of the undeflected fibre in the small deflection analysis. This has the consequence of increasing the theoretical length of the deflected fibre. In the cantilevered geometry, two values for fibre length, \( L \), of equations (3.20-3.21) and (3.23), were investigated. In one case, \( L \) is determined by the distance measured along the undeflected fibre axis to the end point of the deflected fibre as determined by the respective large deflection analysis, and is labelled "parallel length" (PL). In the other case, \( L \) is considered to be the actual fibre length (2 mm in this case) and is labelled "real length" (RL).

A comparison of the predicted deflections of the small and large deflection analyses for the Cantilevered Fibre in a T-Junction is shown in Figure 3.8. In the large deflection analysis equation (3.16) was used to estimate \( f_r \). The curves labelled "a" and "b" represent the predicted deflections for flow rates, \( U_m \), of 0.17 m/s and 0.34 m/s respectively, and the subscripts "0", "1" and "2" distinguish the large deflection analysis, the small deflection analysis assuming a parallel length value for \( L \), and the small deflection analysis assuming a real length value for \( L \), respectively. As shown in the figure, while assuming the parallel length for \( L \) under predicts the deflection, the use of the real length for \( L \) over predicts the deflection and has the effect of elongating the fibre. It follows, therefore, that the average of the two deflections gives a better estimate of the deflection than either of the two small deflection estimates alone.

The effect of the tangential force component on the deflection was analysed by solving the large deflection equations with \( f_t = 0 \) and \( f_t \) estimated from equation (3.16) for the Cantilevered Fibre in a T-Junction geometry. As expected, the effect of the tangential force is to increase the overall deflection of the cantilevered fibre. However, the increased deflection is small. For example, for a cantilevered fibre with \( \alpha = 0^\circ \), i.e. normal to the flow, and a deflection of
approximately 50% (based on fibre length), the increased deflection is less than 3%. Furthermore, the effect of the tangential force decreases as $\alpha$ increases to 90°. Since the increased deflection due to the tangential force component is small, it is justifiable to ignore the effect in the small deflection analysis.

The absolute error of the stiffness of the small deflection analysis versus the deflection is plotted in Figure 3.9. In the large deflection analysis of the Cantilevered Fibre in a T-Junction method the tangential force component given by Cox's equations was used (equation (3.16)). Since no tangential force component was published for the Clamped Fibre in a Channel Flow method (Samuelsson (1963)) it was ignored in the large deflection analysis. For deflections less than approximately 20%, the error using the small deflection analysis is less than 10%. It is interesting to note that the errors determined by $L$ being the real length ($a_1$ and $b_1$) and parallel length methods ($a_2$ and $b_2$) are very similar but opposite in sign for deflections up to approximately 35%. This would suggest that averaging the stiffness determined by the two small deflection
3.4.2 Simply Supported Fibre in a Capillary

A comparison of the large and small deflection results is shown in Figure 3.10 for a fibre stiffness of $1 \times 10^{-12}$ Nm$^2$ and capillary flow rates ranging from 0.2 to 0.8 m/s, which correspond to a Reynolds number range from 6 to 24, based on the fibre diameter. It is shown that as the flow rate is increased, the deflection increases and the small deflection analysis over predicts the deflection. The large deflection results were determined with zero friction at the capillary walls.

In the large deflection analysis, the effect of friction of the contact between the capillary walls and the fibre can be studied for the Simply Supported Fibre in a Capillary. Figure 3.11 plots the results of the large deflection analysis where the capillary flow rate is 0.4 m/s, the fibre stiffness is $1 \times 10^{-12}$ Nm$^2$ and the coefficient of friction, defined as the ratio of the tensile force ($T_1$ and $T_2$ as defined in Figure 3.5) to the reaction force ($R_1$ and $R_2$ as defined in Figure 3.5), is varied from 0 to 1.0. As expected, as the friction increases the deflection is decreased. Even for a
Figure 3.10: Large and small deflection analyses for the Simply Supported Fibre in a Capillary. $EI = 1 \times 10^{-12}$ Nm$^2$.

A relatively small coefficient of friction of 0.1, there is a substantial difference in the deflection. This would suggest that the unknown coefficient of friction can be a large contributor to error when determining the fibre stiffness, especially if the assumption that the fibre does not stretch in the axial direction is valid. As the coefficient of friction is increased, the stiffness prediction would over predict the true stiffness of the fibre if friction is ignored in the prediction analysis. Since Tam Doo and Kerekes (1982) determined their deflection coefficients experimentally by measuring the deflections of a metal wire of known stiffness, a friction factor is built-in to their analysis. However, the coefficient of friction between a pulp fibre and the capillary walls will most likely be higher than the coefficient of friction between a metal wire and the capillary walls. Underestimation of the coefficient of friction suggests that the Simply Supported Fibre in a Capillary method has the potential to over predict the stiffness.

The absolute error of the stiffness of the small deflection analysis results versus the deflection is plotted in Figure 3.9. For the simply supported geometry, the small deflection
Figure 3.11: Effect of friction (tensile force / reaction force) for the Simply Supported Fibre in a Capillary. \( U = 0.4 \text{m/s} \), \( EI = 1 \times 10^{12} \text{Nm}^2 \).

analysis over predicts the stiffness when compared to the large deflection analysis. For deflections less than 20% the error of the stiffness calculation is less than 10%, using the small deflection analysis. Since the friction forces in the Simply Supported Fibre method are unknown, the cantilevered methods are potentially more suitable for determining fibre flexibility.

3.4.3 Conforming Fibre on a Wire

In this analysis the pressure of 0.41 MPa used by Steadman and Luner (1983) was applied for a fibre diameter of 30 \( \mu \text{m} \) resulting in a load per unit length, \( f_e \), of 12.3 N/m. A wire diameter (see Figure 3.6), \( d \), of 25 \( \mu \text{m} \) was used.

The Conforming Fibre on a Wire method is sensitive to the measured location where the fibre first comes into contact with the slide and, therefore, to the span length, \( a \). Since the stiffness is proportional to the 4-th power of \( a \) (see equation (3.32)), a 5% error in the measurement of \( a \), for example, would give a 20% error in the stiffness prediction. Numerically,
Figure 3.12: The effect of spring stiffness, $k$, for the Conforming Fibre on a Wire. $EI = 1 \times 10^{12}$ Nm$^2$.

at the point of contact, the fibre is assumed to have no curvature in the small deflection analysis and the first method of the large deflection analysis, or very little curvature in the second method of the large deflection analysis. The problem of matching the span length from the actual experimental set-up to the numerical equations is further accentuated if the fibre's cross-section deflects under the loading pressure. To study the effect of fibre cross-section deformation in a qualitative way, the second method of the large deflection analysis is used.

Figure 3.12 plots the deflected fibre shape predicted using the second method of the large deflection analysis with spring stiffness values, $k$, of $1 \times 10^7$, $5 \times 10^7$, $1 \times 10^8$ and $2 \times 10^8$ N/m and a fibre stiffness of $10^{-12}$ Nm$^2$. Also plotted in the figure, for reference, is the shape of the fibre predicted using the first method of the large deflection analysis, curve c. As $k$ approaches larger values, the fibre deflection predicted by the second method appears to approach that of the first. Figure 3.13 plots the $k$ values used above versus the numerically determined span length, $a$. The line at $a = 1.07 \times 10^4$ m represents the value for $a$ determined using the first method. It is evident
Figure 3.13. The effect of spring stiffness, $k$, on the span length, $a$, for the Conforming Fibre on a Wire. $a_{af}$ represents the value determined for $a$ using the first method of large deflection analysis for the Conforming Fibre on a Wire. $EI = 1 \times 10^{-12}$ Nm$^2$.

that as $k$ approaches very large values, $a$ predicted by the second method does not converge on the value for $a$ predicted by the first method. This can be attributed to the fact that even for very large $k$ values the curvature at $a$ is not zero for the second method whereas it is for the first method. Since it is assumed that the fibre has no curvature at $x = a$ in the first method, the fibre requires a greater span length to "touch" the slide. Clearly, a shorter experimental measurement of $a$ than that which is determined in theory would lead to an underestimate of the stiffness and suggests that the Conforming Fibre on a Wire method underestimates the stiffness.

Figure 3.14 compares the results obtained from the small deflection analysis (equation (3.30)) and the first method of the large deflection analysis for fibre stiffnesses of $10^{-11}$ Nm$^2$, $10^{-12}$ Nm$^2$ and $10^{-13}$ Nm$^2$. For stiffnesses of $10^{-11}$ Nm$^2$ and $10^{-12}$ Nm$^2$, the small deflection analysis gives results that are very similar to those predicted using the large deflection analysis. For a stiffness of $10^{-13}$ Nm$^2$ the curve generated using the small deflection analysis deviates slightly from
the one using the large deflection analysis. It was determined that fibres of stiffnesses less than $7.83 \times 10^{-14}$ Nm$^2$ have a radius of curvature less than the wire radius (12.5 μm). This would require the consideration of the wire geometry in the analysis. The deflection ($d/\alpha \times 100\%$) for this case is 47%. It should be emphasized that in this case the geometry of the wire plays an important role in the shape of the deflected fibre requiring a much more complicated deflection analysis and will not be discussed here.

The absolute error of the stiffness predicted using the small deflection analysis versus deflection is plotted in Figure 3.9. The small deflection analysis under predicts the stiffness when compared to the large deflection analysis. As in the other stiffness measurement methods, the error in the stiffness calculated using the small deflection analysis increases as the deflection is increased.
3.5 Conclusions

Four fibre stiffness measurement techniques were analysed using small and large deflection analyses. It was shown that the small deflection analysis generally predicts stiffnesses with an error of less than 10% when compared to the large deflection analysis for fibre deflections of less than 20%. Since the dynamic equations of motion for a fibre in a shear flow, to be developed in the following chapter, rely on small deflection theory, this result is encouraging, for it suggests that the use of small deflection theory for the dynamic case is justified and the added complexity of using large deflection theory to model fibre dynamics is not required. The Conforming Fibre on a Wire method has the potential for introducing large errors in determining the stiffness, due to the difficulty in determining the span length. It was shown that if the fibre deflects in the cross-section at the point of contact with the slide, then the measured span length would be less than predicted from the experimental stiffness analysis, with the consequence of under predicting the stiffness. All other methods, on the other hand, introduce error due to the estimate of the hydrodynamic forces acting on the fibre. Unknown friction forces at the contact points in the Simply Supported Fibre in a Capillary method could contribute to an over predicted stiffness result.

Steadman and Lunar (1983) compared the stiffness results using the Simply Supported Fibre in a Capillary method and the Conforming Fibre on a Wire. For the same batch of fibres, the mean stiffness value predicted using the Simply Supported Fibre in a Capillary method was one order of magnitude greater than the mean stiffness value predicted using the Conforming Fibre on a Wire method. In a qualitative way, these results support the statement above: the friction forces at the contact point of the Simply Supported Fibre in a Capillary method could contribute to an over predicted stiffness result and the deflection of fibres in their cross-section at the point of contact with the slide in the Conforming Fibre on a Wire method could contribute to an under predicted stiffness value.

It would be expected that the cantilevered geometries would predict stiffness values in a range between those of the Simply Supported Fibre in a Capillary method and Conforming Fibre on a Wire method. Kuhn et al. (1995) compared the stiffness results using the Cantilevered Fibre in a T-Junction method and the Simply Supported Fibre in a Capillary method. The results of the
Cantilevered Fibre in a T-Junction method gave a mean stiffness value of one order of magnitude greater than the Simply Supported Fibre in a Capillary method. This inconsistency can be attributed to the inaccurate force estimation of the cantilevered geometries. It is expected that, in the future, the Automatic Three-Dimensional Mesh Generation Algorithm of Chapter 6 will be used with CFD to better predict the fluid forces acting on the fibre in the relatively complicated geometry of the T-Junction.

The results presented suggest that with a better prediction of the hydrodynamic forces acting on the fibre, the cantilevered fibre geometry has the most potential for predicting an accurate fibre stiffness value. The errors associated with the span length measurement of the Conforming Fibre on a Wire method and the unknown contact friction associated with the Simply Supported Fibre in a Capillary method do not affect the stiffness prediction of the cantilevered methods.
Chapter 4

Fibre Dynamics: Equations of Motion

To date, numerical models of the flow of cylinders and ellipsoids have assumed the bodies to be completely rigid or perfectly flexible. The task of numerically modelling the flow of a flexible cylinder or fibre, on the other hand, has not been investigated. This numerical task can be grouped into two sub-tasks: 1) the prediction of the forces acting on the fibre by the fluid flow, and 2) the calculation of the dynamic movement of the flexible fibre.

In Chapter 2 a review of literature pertaining to determining the hydrodynamic forces acting on a long slender body such as a fibre was given. Analytical, empirical and CFD methods for determining the forces were discussed. Generally, empirical results provide a method to determine forces only acting normal to the body. Clearly, tangential forces are just as important in a simulation of fibre motion. A method to estimate forces using CFD and automatic mesh generation will be discussed in Chapter 6. While this method could prove quite useful in future applications, today's computer speeds are too slow to make the CFD method viable for studying numerous simulations. As was mentioned previously, Cox (1970) was able to analytically estimate the forces acting on a long slender body in a creeping or Stokes flow, by using perturbation methods.
As presented previously, in the first order approximation of Coq, the normal and
tangential components of the force per unit length acting on a long slender body are given as

\[ \begin{align*}
 f_n &= -4\pi\mu(U_n - V_n)\left(\ln\left(\frac{D}{L}\right)\right)^{-1} \\
 f_t &= -2\pi\mu(U_t - V_t)\left(\ln\left(\frac{D}{L}\right)\right)^{-1}
\end{align*} \] (4.1)

respectively. Here \( \mu \) is the viscosity of the fluid, \( D \) is the diameter of the body, \( L \) is the length of
the body, \( U_n \) and \( U_t \) are the normal and tangential components of the undisturbed fluid velocity,
respectively, and \( V_n \) and \( V_t \) are the normal and tangential components of the local body velocity,
respectively.

The purpose of this chapter is to introduce a method to calculate the dynamic movement
of a flexible fibre in a two-dimensional plane, i.e. the second sub-task as defined above. The
theoretical development follows closely with methods introduced in the modelling of robots with
flexible links (see, for example, Book (1984)). The dynamic theory relies on Euler-Bernoulli
beam bending theory, i.e. small deflection beam theory. In the previous chapter it was shown that
the static small deflection analysis of beams produces errors of less than 10% when compared to
static large deflection analysis for deflections of fibres, based on the span length, of up to 20%. It
will be assumed that the error using small deflection analysis in the dynamic case will be of the
same order, which justifies the use of Euler-Bernoulli beam theory. To determine the effect of the
fibre's inertia, the equations of motion will first be developed by including all of the inertia terms.
It will be shown in the following chapter that the inertia of the fibre is negligible and for this
reason a more economical inertialess model will be developed by simplifying the model including
inertia. The first section will briefly demonstrate that the problem of modelling the movement of a
flexible fibre or cylinder requires a vibrational analysis. The second section will provide a short
review of the problem of a uniform beam in free vibration. The third section will develop the
equations of motion of the flexible fibre by utilizing Hamilton's principle. These equations
represent the exact equations of motion and include the inertia of the fibre. In the fourth section,
the natural modes of vibration for a beam with both ends free, presented in the second section,
will be used to discretize the equations of motion of the fibre by the assumed modes method. In
the fifth section, the dynamic model will be simplified by eliminating the inertia terms to develop
an inertialess dynamic model. In the following chapter the equations of motion will be utilized to
simulate the trajectories of flexible fibres flowing in a channel with a slot. Of key interest in the study is the ability of the fibres to pass through the slot. Two models to represent contact between the fibre and the channel and slot walls will be presented in the sixth section. The seventh section will present the application of the Runge-Kutta method to solve the coupled ordinary differential equations representing the discretized equations of motion. The dynamic models will be utilized to produce a steady-state solution which will be compared to static analytical theory, in the final section.

4.1 Simple Dynamic Motion of Two Masses

A simple example of two masses separated by a linear spring will be used to demonstrate that a vibrational analysis is required to determine the motion of a flexible cylinder. Consider a force, \( F \), acting in the \( x \)-direction on two masses, each of mass \( m \), separated by a linear spring of stiffness \( k \), as shown in Figure 4.1. The problem represents a two degree of freedom system. The overall acceleration of the centre of mass, \( a_c \), is simply given by the equation

\[
a_c = \frac{F}{2m}.
\]  

(4.2)

To determine the motion of the two masses relative to each other or, equivalently, the motion of each mass relative to a fixed frame, however, requires a vibrational analysis and knowledge of the spring stiffness, \( k \).

The motion of the two masses described above is analogous to the problem of the dynamic motion of a flexible fibre. In both cases, the problem is reduced to rigid body motion in the limit as the stiffness is increased to infinity. The flexible spring adds a degree of freedom to the two mass problem. Since a fibre is continuous, an infinite number of degrees of freedom are required to model all the modes of flexible movement of the body. Thus, the motion of a flexible fibre can only be described if a vibrational analysis is performed.
4.2 Flexible Fibre in Free Vibration

The problem of modelling the motion of a flexible fibre in a shear flow is similar to that of modelling a beam in vibration with both ends free. The solution to the differential equation of motion of a simple beam in lateral vibration will be reviewed in this section. This subject is treated extensively in the literature and the reader is referred to references Paz (1980), Rao (1986) and Weaver et al. (1990) for a complete discussion of the topic.

For a beam with a uniform cross-section, the equation for lateral vibration of a beam is given by Rao (1986)

\[
E I \frac{\partial^4 u(x, t)}{\partial x^4} + \rho_c A \frac{\partial^2 u(x, t)}{\partial t^2} = f(x, t)
\]  

(4.3)

where \( x \) is measured along the length of the undeflected beam (\( x = 0 \) at one end and \( x = L \) at the other end, where \( L \) is the length of the beam); \( t \) is the time; \( u(x, t) \) is the deflection of the beam, defined as positive in the \( y \)-axis direction, normal to the \( x \)-axis; \( E \) is Young's modulus; \( I \) is the moment of inertia of the beam cross-section about the central axis that is perpendicular to both the \( x \)-axis and the \( y \)-axis; \( \rho_c \) is the density of the beam; \( A \) is the area of the beam cross-section; and \( f(x, t) \) is the force per unit length acting in the \( y \)-axis direction on the beam (see Figure 4.2). Equation (4.3) assumes that the deflection of the beam is small when compared to the length of the beam and that Euler-Bernoulli or thin beam theory applies to the beam, i.e. a plane cross-section of the beam remains plane during flexure. In the static analysis of Chapter 3 it was shown that the small deflection analysis produced errors of less than 10% for deflections of approximately 20% of the fibre length when compared to the large deflection analysis. If it is assumed that these results can be extrapolated to the dynamic case, then Euler-Bernoulli theory
should be applicable for most fibre flow cases. Neither the axial effects nor the effects of rotary inertia and shear deformation were considered.

For free vibration, $f(x,t) = 0$, and equation (4.3) reduces to

$$EI \frac{\partial^4 u(x,t)}{\partial x^4} + \rho_c A \frac{\partial^2 u(x,t)}{\partial t^2} = 0. \quad (4.4)$$

Using the method of separation of variables, $u(x,t) = Q(x)\tau(t)$, equation (4.4) gives

$$Q''' - k^4 Q = 0 \quad (4.5)$$

$$\ddot{\tau} + \omega^2 \tau = 0 \quad (4.6)$$

where the prime (') designates the derivative with respect to $x$ and the dot (•) represents the derivative with respect to $t$, and

$$\omega = (kL)^2 \sqrt{\frac{EI}{\rho_c A L^4}}. \quad (4.7)$$

The solution to the above equations for the $i$-th mode of vibration can be written as (Rao (1986))

$$Q_i(x) = C_1 \cos k_i x + C_2 \cosh k_i x + C_3 \sin k_i x + C_4 \sinh k_i x \quad (4.8)$$

$$\tau_i(t) = A_1 \cos \omega_i t + B_1 \sin \omega_i t \quad (4.9)$$

where $A_i, B_i$ and $C_i$ are constants of integration.

As mentioned previously, the modal expansion or the assumed modes approach has been used successfully in modelling the motion of robots with flexible links. Book (1984) incorporated the deflections of links for a general flexible link robot by using the modal analysis approach. It was stated that the approach is valid for small link deflections. In the modal analysis or expansion approach, assumed modes for free vibration are used to approximate the force response solution.

![Figure 4.2: Beam or fibre with a distributed normal load.](image)
The assumed modes approach was discussed in Chapter 2. The important issue here is that a solution for a beam or fibre in free vibration with appropriate boundary conditions is required to be used with the assumed modes method. In the case of an unconstrained fibre flowing in a shear flow, unconstrained boundary conditions should be used.

For the case where the beam is unconstrained at both ends, both the moment and the shear must be zero at the ends. This requirement leads to the following boundary conditions:

- zero moment:
  \[ EI \frac{\partial^2 u(x, t)}{\partial x^2} \bigg|_{x=0} = EI \frac{\partial^2 u(x, t)}{\partial x^2} \bigg|_{x=L} = 0 \]  \hspace{1cm} (4.10)

- zero shear:
  \[ \frac{\partial}{\partial x} EI \frac{\partial^2 u(x, t)}{\partial x^2} \bigg|_{x=0} = \frac{\partial}{\partial x} EI \frac{\partial^2 u(x, t)}{\partial x^2} \bigg|_{x=L} = 0. \] \hspace{1cm} (4.11)

Applying the boundary conditions of equations (4.10) and (4.11) to equation (4.8) the solution to the free vibration problem of an unconstrained beam becomes

\[ Q_i(x) = \cos k_i x + \cosh k_i x - \sigma_i (\sin k_i x + \sinh k_i x) \] \hspace{1cm} (4.12)

where

\[ \sigma_i = \frac{\cosh k_i L - \cos k_i L}{\sinh k_i L - \sin k_i L} \] \hspace{1cm} (4.13)

\[ \omega_i = (k_i L)^2 \sqrt{\frac{EI}{\rho c AL^4}}. \] \hspace{1cm} (4.14)

The constants, \( k_i \), are determined by solving the periodic equation

\[ \cos k_i L \cosh k_i L = 1. \] \hspace{1cm} (4.15)

Table 4.1 gives the first 10 roots calculated by solving equation (4.15). The final solution can be written as

\[ u(x, t) = \sum_{i=1}^{\infty} Q_i(x) \tau_i(t). \] \hspace{1cm} (4.16)

It should be emphasized that equation (4.16) represents the deflection of the beam from its undeflected axis. Since the beam is unconstrained, the 0-th mode of vibration \( (i = 0) \) can be used to represent the rigid body motion. Also,

\[ \int_0^L Q_i(x) dx = 0 \] \hspace{1cm} (4.17)

\[ \int_0^L (Q_i(x))^2 dx = L \] \hspace{1cm} (4.18)

\[ \int_0^L Q_i(x) Q_j(x) dx = 0, \ \forall i \neq j. \] \hspace{1cm} (4.19)
Table 4.1: Natural modes of vibration for a beam with both ends free.

<table>
<thead>
<tr>
<th>Mode of Vibration</th>
<th>$kL$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.000</td>
</tr>
<tr>
<td>1</td>
<td>4.730</td>
</tr>
<tr>
<td>2</td>
<td>7.853</td>
</tr>
<tr>
<td>3</td>
<td>10.996</td>
</tr>
<tr>
<td>4</td>
<td>14.137</td>
</tr>
<tr>
<td>5</td>
<td>17.279</td>
</tr>
<tr>
<td>6</td>
<td>20.420</td>
</tr>
<tr>
<td>7</td>
<td>23.562</td>
</tr>
<tr>
<td>8</td>
<td>26.704</td>
</tr>
<tr>
<td>9</td>
<td>29.845</td>
</tr>
<tr>
<td>10</td>
<td>32.987</td>
</tr>
</tbody>
</table>

Equations (4.17), (4.18) and (4.19) will be used to simplify the discretized model.

Equation (4.12) represents the shape of the $i$-th mode of natural, lateral vibration of a beam with both ends free and it will be used to discretize the equations obtained for a flexible fibre in a flow.

4.3 Equations of Motion of a Flexible Fibre

The equations of motion of the flexible fibre will be developed using Hamilton's principle (see Goldstein (1980) and Meirovitch (1990)). The equations developed here represent the exact\(^1\) dynamics of a flexible fibre in planar motion. These equations will be non-linear and of integro-differential form. Numerical methods which solve these equations will be discussed in the next section.

The geometry and co-ordinate system of the flexible fibre are shown in Figure 4.3. The vector $\mathbf{R}_p$ defines the position of a point on the fibre relative to a global $X-Y$ co-ordinate frame. The origin of the local $x-y$ co-ordinate frame is located at one end of the fibre, with the $x$-axis

---

\(^1\) The equations are exact if all the assumptions of Euler-Bernoulli beam theory are valid, the forces acting on the fibre are known exactly, and the fibre has a constant cross-section and stiffness along its length.
Figure 4.3: Flexible fibre geometry.

running along the undeflected fibre cross-section centre line. The function $u(x,t)$ is the deflection of the fibre from the $x$-axis due to lateral vibrations within the body. Thus, $\mathbf{R}_p$ is given by

$$
\mathbf{R}_p = \begin{bmatrix}
X_o + x \cos \theta - u \sin \theta \\
Y_o + x \sin \theta + u \cos \theta
\end{bmatrix}
$$

(4.20)

where $X_o$ and $Y_o$ represent the global $X$ and $Y$ positions of point $O$ (one end) of the fibre.

To determine the equations of motion using Hamilton's principle, the kinetic and potential energy, in addition to the virtual work, of the system must be derived. The kinetic energy is given as (see Book (1984))

$$
T = \frac{1}{2} \int_0^L \left( \mathbf{\dot{R}}_p \right)^2 dm
$$

(4.21)

where the dot ($\dot{}$) is used to signify the derivative with respect to time and $dm$ is the incremental mass of the fibre. Equation (4.21) can be rewritten as

$$
T = \frac{1}{2} \int_0^L \rho A \left( \mathbf{\dot{R}}_p \right)^2 dx = \int_0^L \mathbf{T} \cdot \mathbf{\dot{R}}_p dx.
$$

(4.22)

The potential energy of the fibre is (see Book (1984))

$$
V = \int_0^L \frac{EI}{2} \left( \frac{\partial^2 u}{\partial x^2} \right)^2 dx = \int_0^L \mathbf{\dot{V}} \cdot \mathbf{\dot{R}}_p dx.
$$

(4.23)

Two forms of virtual work arise in the problem. A distributed force per unit length acts on the fibre due to the fluid forces. Also, point forces act on the fibre at times when the fibre comes in contact with a solid boundary. Let $f_n$ and $f_t$ represent the normal (defined positive in the direction of the $y$-axis) and tangential (defined positive in the direction of the $x$-axis) distributed
forces, respectively. Let the j-th point force acting at position \( x_j \) of the fibre be represented by the normal and tangential components \( F_n \) and \( F_t \), respectively. The virtual work can be written as follows:

\[
\delta W = \int_0^L \left( f_n \cos \theta - f_n \sin \theta \right) \delta R_{\mu \nu} \, dx \\
+ \sum_j \left( F_n \cos \theta - F_n \sin \theta \right) \delta R_{\mu_j \nu} \\
+ \int_0^L \left( f_t \sin \theta + f_t \cos \theta \right) \delta R_{\rho \tau} \, dx \\
+ \sum_j \left( F_t \sin \theta + F_t \cos \theta \right) \delta R_{\rho_j \tau_j}
\]

(4.24)

where \( \delta R_{\mu \nu} \) and \( \delta R_{\rho \tau} \) represent the infinitesimal change of the \( X \) and \( Y \) components of the vector \( \mathbf{R}_p \), respectively. The terms \( \delta R_{\mu_j \nu} \) and \( \delta R_{\rho_j \tau_j} \) represent the infinitesimal change of the vector \( \mathbf{R}_p \) as above, but evaluated at the j-th point of contact. It should be noted that buoyancy effects were not considered in equation (4.24) but can easily be introduced.

Hamilton's principle can now be applied (see Goldstein (1980)):

\[
\int_{t_1}^{t_2} (\delta T - \delta V + \delta W) \, dt = 0.
\]

(4.25)

Equation (4.25) can be rewritten as

\[
\int_{t_1}^{t_2} \left( \int_0^L \left( \mathbf{T} \left( \dot{X}_o, \dot{Y}_o, \theta, \dot{\theta}, u, \dot{u} \right) - \dot{\mathbf{V}}(u''') \right) \, dx + \delta W \right) \, dt = 0
\]

(4.26)

where the dot (\( \bullet \)) is used to represent the derivative with respect to time, as before, and

\[
\dot{u} = \frac{\partial u}{\partial t} \quad \text{and} \quad u''' = \frac{\partial^3 u}{\partial x^3}
\]

(4.27)

where the hollow dot (\( \hat{\bullet} \)) is used to represent the partial derivative with respect to time.

Introducing equations (4.22) and (4.23) in equation (4.26), integrating by parts and simplifying gives

\[
\int_{t_1}^{t_2} \left( \int_0^L \left( \frac{\partial \mathbf{T}}{\partial \theta} \delta \theta - \frac{d}{dt} \left( \frac{\partial \mathbf{T}}{\partial \dot{u}} \right) \delta \dot{u} + \frac{\partial \hat{\mathbf{T}}}{\partial \theta} \delta \theta + \frac{d}{dt} \left( \frac{\partial \hat{\mathbf{T}}}{\partial \dot{u}} \right) \delta \dot{u} \\
- \frac{d}{dt} \left( \frac{\partial \mathbf{T}}{\partial \dot{X}_o} \right) \delta \dot{X}_o - \frac{d}{dt} \left( \frac{\partial \mathbf{T}}{\partial \dot{Y}_o} \right) \delta \dot{Y}_o \right) \, dx \\
+ \delta W - \frac{\partial \hat{\mathbf{V}}}{\partial u'''} \delta u'' \bigg|_0^L + \frac{\partial \hat{\mathbf{V}}}{\partial x} \frac{\partial \hat{\mathbf{V}}}{\partial u'''} \delta u \bigg|_0^L \right) \, dt = 0
\]

(4.28)

The last two terms on the left hand side of equation (4.28) represent boundary conditions required
at the ends of the fibre. After much algebraic manipulation of equations (4.22), (4.23), (4.24) and (4.28), terms multiplied by $\delta X_o$, $\delta Y_o$, $\delta \theta$ and $\delta u$ can be collected and each set can be equated to zero. This leads to the following integro-differential equations:

- for $\delta X_o$:
  \[
  \begin{align*}
  & M \ddot{X}_o - \frac{1}{2} ML c_0 \frac{\partial^2}{\partial x^2} \ddot{X}_o - \frac{1}{2} ML s_0 \ddot{\theta} - \rho c A s_0 \int_0^L \ddot{u} \ dx - 2 \rho c A \dot{c}_0 \int_0^L \dot{u} \ dx \\
  & + \rho c A s_0 \int_0^L \ddot{u} \ dx - \rho c A \int_0^L \dot{u} \ dx = F_x 
  \end{align*}
  \] (4.29)

- for $\delta Y_o$:
  \[
  \begin{align*}
  M \ddot{Y}_o - \frac{1}{2} ML s_0 \ddot{\theta} + \frac{1}{2} ML c_0 \ddot{\theta} \\
  + \rho c A c_0 \int_0^L \ddot{u} \ dx - 2 \rho c A s_0 \dot{u} \ dx \\
  - \rho c A \ddot{\theta} \int_0^L \ddot{u} \ dx - \rho c A s_0 \int_0^L \dot{u} \ dx = F_y 
  \end{align*}
  \] (4.30)

- for $\delta \theta$:
  \[
  \begin{align*}
  \frac{1}{2} ML^2 \ddot{\theta} - \frac{1}{2} ML (s_0 \ddot{X}_o - c_0 \ddot{Y}_o) \\
  + 2 \rho c A \ddot{\theta} \int_0^L \ddot{u} \ dx + \rho c A \dot{\theta} \int_0^L \dot{u}^2 \ dx \\
  - \rho c A (c_0 \ddot{X}_o + s_0 \ddot{Y}_o) \int_0^L \ddot{u} \ dx \\
  + \rho c A \int_0^L x \ddot{u} \ dx = F_\theta 
  \end{align*}
  \] (4.31)

- for $\delta u$, $u$ is a function of $x$ and $t$, therefore $\delta u$ must remain inside the integral, giving:
  \[
  \begin{align*}
  & \int_0^L (\rho c A (\ddot{u} - s_0 \ddot{X}_o + c_0 \ddot{Y}_o + x \ddot{\theta} - u \ddot{\theta}) \\
  & - E l \dddot{u} - f_n) \delta u \ dx - \Sigma_s F_n \delta u = 0 
  \end{align*}
  \] (4.32)

where $M$ is the mass of the fibre, $s_0 = \sin \theta$, $c_0 = \cos \theta$, and the force terms are given as

\[
\begin{align*}
F_x &= \int_0^L (f_s c_0 - f_n s_0) \ dx + \Sigma_s (F_n c_0 - F_n s_0) \\
F_y &= \int_0^L (f_s s_0 + f_n c_0) \ dx + \Sigma_s (F_n s_0 + F_n c_0) \\
F_\theta &= \int_0^L (f_n x - f_n u) \ dx + \Sigma_s (F_n x - F_n u) 
\end{align*}
\] (4.33)

Equations (4.29), (4.30), (4.31) and (4.32) represent the exact equations of motion of a flexible fibre. They can be solved using numerical methods which rely on discretizing the flexible fibre displacement function $u(x,t)$. A discussion of the different methods was given in Chapter 2.
The cross-sectional geometry of a flexible fibre modelled in this analysis was assumed to be constant along the length. This permitted its natural modes of vibration to be determined in Section 4.2. In view of the superiority of the Rayleigh-Ritz method over the finite element method for the study of vibrations of beams with regular geometry (see Chapter 2), the Rayleigh-Ritz method was chosen to discretize the equations of motion developed in the previous section. Furthermore, the assumed modes method can be utilized since the modes of natural vibration are known. The next section will use the assumed modes method to spatially discretize the integro-differential equations developed in this section. This will lead to a system of ordinary differential equations. Aspects of the numerical solution of these equations will be discussed in Section 4.6.

4.4 Discretization of the Equations of Motion

To model the dynamics of a flexible fibre, it is necessary to discretize the general equation of motion for a beam in flexural vibration. Using the assumed modes method, the function representing the natural modes of vibration for an unconstrained beam is chosen to discretize the displacement function, \( u(x,t) \). Taking the first \( m \) modes of vibration to approximate the integro-differential equations developed in Section 4.3 leads to \( m + 3 \) coupled ordinary differential equations of second order. These equations can be solved numerically using the Runge-Kutta method, described in Section 3.2.

Define the vector \( Q \in R^m \)

\[
Q(x) = \begin{bmatrix} Q_1(x) \\ Q_2(x) \\ \vdots \\ Q_m(x) \end{bmatrix}
\]  

(4.34)

where \( Q_i \) represents the shape of the \( i \)-th mode of natural vibration given by equation (4.12). The discretized form of \( u(x,t) \) can be written as

\[
u(x,t) = \sum_{i=1}^{m} Q_i(x) \tau_i(t)
\]

(4.35)

or in vector form

\[
u(x,t) = Q^T \tau
\]

(4.36)
where the superscript "T" is used to denote the transpose and $\tau \in \mathbb{R}^m$ is a vector of unknown time-dependent coefficients to be determined, such that

$$\tau(t) = \begin{bmatrix} \tau_1(t) \\ \tau_2(t) \\ \vdots \\ \tau_m(t) \end{bmatrix}.$$  \hspace{1cm} \text{(4.37)}

Integrals involving terms with the function, $u(x,t)$, can now be simplified (c.f. equations (4.17), (4.18), (4.19)). For example, the integral $\int_0^t u \dot{u} dx$ in equation (4.31) can be written as follows

$$\int_0^t u \dot{u} dx = \int_0^t \tau^T Q Q^T \tau dx = \tau^T \int_0^t Q Q^T dx \tau$$  \hspace{1cm} \text{(4.38)}

but $\int_0^t Q Q^T dx = L I_{m \times m}$, where $I_{m \times m} \in \mathbb{R}^{m \times m}$ is the identity matrix, and therefore $\int_0^t u \dot{u} dx = L \tau^T \tau$.

It should also be noted that $\int_0^t Q dx = 0_m$, $\int_0^t \chi Q dx = 0_m$ and $\delta u = Q^T \delta \tau$.

Substituting equation (4.36) in equations (4.29), (4.30), (4.31) and (4.32), the dynamics of the system become

$$M(q) \ddot{q} + C(q, \dot{q}) + \begin{bmatrix} 0_3 \\ K \tau \end{bmatrix} = F$$  \hspace{1cm} \text{(4.39)}

where $q \in \mathbb{R}^{3+m}$ is a vector of unknown time-dependent, co-ordinates to be determined,

$$q^T = \begin{bmatrix} X_o & Y_o & \theta & \tau^T \end{bmatrix}.$$  \hspace{1cm} \text{(4.40)}

$M(q) \in \mathbb{R}^{(3+m) \times (3+m)}$, the inertia (mass) matrix is given by

$$M = \begin{bmatrix} M & 0 & -\frac{1}{2} M L s_0 & 0_m^T \\ 0 & M & \frac{1}{2} M L c_0 & 0_m^T \\ -\frac{1}{2} M L s_0 & \frac{1}{2} M L c_0 & \frac{1}{3} M L^2 + M \tau^T \tau & 0_m^T \\ 0_m & 0_m & 0_m & M I_{m \times m} \end{bmatrix}.$$  \hspace{1cm} \text{(4.41)}

The vector $C(q, \dot{q}) \in \mathbb{R}^{3+m}$ is

$$C(q, \dot{q}) = \begin{bmatrix} -\frac{1}{2} M L c_0 \dot{\theta}^2 \\ -\frac{1}{2} M L s_0 \dot{\theta}^2 \\ 2M \dot{\theta}^T \tau \dot{\tau} \\ -M \dot{\theta}^2 \tau \end{bmatrix}.$$  \hspace{1cm} \text{(4.42)}
The diagonal stiffness matrix, \( K \in \mathbb{R}^{n \times n} \), is determined by

\[
K_{ii} = EI \int_0^L \frac{d^4 Q_i}{dx^4} \, dx
\]

(4.43)

and the force vector, \( F \in \mathbb{R}^{3+n} \), is given by

\[
F = \begin{bmatrix}
\int_0^L (f_i c_8 - f_n s_8) \, dx + \sum_j (F_{ij} c_8 - F_{nj} s_8) \\
\int_0^L (f_i s_8 + f_n c_8) \, dx + \sum_j (F_{ij} s_8 + F_{nj} c_8) \\
\int_0^L f_n x \, dx - \int_0^L f_i Q^T \, dx + \sum_j (F_{ij} x_j - F_{nj} Q^T(x_j)) \tau \\
\int_0^L f_n Q \, dx + \sum_j F_{nj} Q(x_j)
\end{bmatrix}
\]

(4.44)

At this point no structural damping has been included in the model. Paz (1980) suggested the damping coefficient matrix to be given by

\[
b_{ij} = \int_0^L b(x) Q_i(x) Q_j(x) \, dx
\]

(4.45)

where \( b(x) \) represents the distributed damping coefficient per unit length which, for a uniform beam, is constant. For the flexible fibre, equation (4.45) reduces to \( B = b L I_{m \times m} \), where \( B \in \mathbb{R}^{m \times m} \) is the damping coefficient matrix.

The final form of the discretized system is obtained by adding structural damping to equation (4.39):

\[
M(q) \dddot{q} + C(q, \dot{q}) + \begin{bmatrix} 0_3 \\ K \tau \end{bmatrix} + \begin{bmatrix} 0_3 \\ B \tau \end{bmatrix} = F.
\]

(4.46)

It was assumed that the internal structural damping of a fibre was very small when compared to the damping induced by the fluid and for this reason, the structural damping was usually neglected. Equation (4.46) represents a set of \( 2 \times (3+m) \) coupled ordinary differential equations of second order. Since all the initial (time) boundary conditions are known, or can be assumed, the system can be solved numerically using the same adaptive step size Runge-Kutta routine that was used in the shooting method of Chapter 3.

4.5 Inertialess Equations of Motion

Since the density of pulp fibres is on the order of 10% more than that of water, it can be argued that the inertia terms can be ignored. To develop the inertialess equations of motion of a flexible pulp fibre the inertia terms are eliminated by setting \( M = 0 \) and \( \rho_c = 0 \). This has the consequence of reducing the order of the equations by one. The numerical task is reduced to solving for the
velocity of the fibre instead of the acceleration. To solve the equations using numerical methods, the velocity terms of the fibre must be grouped together.

Generally, the equations for the fluid forces acting on a fibre have the following form:

\[
\begin{align*}
    f_n &= g_n(U_n - V_n) \\
    f_t &= g_t(U_t - V_t)
\end{align*}
\]  \tag{4.47}

where \( f_n \) and \( f_t \) are the normal and tangential forces per unit length acting on the fibre, as before, and \( g_n \) and \( g_t \) are some functions of the relative local normal and tangential speeds of the fluid and the fibre. As mentioned above, when the inertia terms are set to zero, the equations of motion are solved with respect to the fibre velocity. If the functions \( g_n \) and \( g_t \) are non-linear, then the task of solving the equations could prove to be difficult. For the present analysis, the fluid forces are estimated using Cox's (1970) first order approximation of equation (4.1). By setting \( M = 0 \) and \( \rho_e = 0 \) in equation (4.39) and using equation (4.1) for the forces, the equations of motion for an inertialess, flexible fibre become

\[
M_v(q) \ddot{q} + \begin{bmatrix} 0 \\ K\tau \end{bmatrix} = F_v
\]  \tag{4.48}

where \( M_v(q) \in \mathbb{R}^{(3+m)\times(3+m)} \) is

\[
M_v = \begin{bmatrix}
    L(k_1c_0^2 + k_ns_0^2) & Ls_0c_0(k_t - k_n) & -\frac{1}{2}k_nL^2s_0 & 0^T_m \\
    Ls_0c_0(k_t - k_n) & L(k_is_0^2 + k_nc_0^2) & \frac{1}{2}k_nL^2c_0 & 0^T_m \\
    -\frac{1}{2}k_nL^2s_0 & \frac{1}{2}k_nL^2c_0 & \frac{1}{2}k_nL^3 + k_tL\tau^T\tau & 0^T_m \\
    0_m & 0_m & 0_m & k_nL1_{m\times m}
\end{bmatrix}
\]  \tag{4.49}

the matrix \( K \) is the same as defined in equation (4.43), and \( F_v \in \mathbb{R}^{3+m} \) is

\[
F_v = \begin{bmatrix}
    k_1c_0[\frac{L}{0} U_tdx - k_ns_0][\frac{L}{0} U_ndx + \Sigma_f(F_yc_0 - F_\eta s_0)] \\
    k_1s_0[\frac{L}{0} U_tdx + k_nc_0][\frac{L}{0} U_ndx + \Sigma_f(F_\eta s_0 + F_y c_0)] \\
    k_n[\frac{L}{0} U_nxdx - k_1][\frac{L}{0} U_tQ^Tdx + \Sigma_f(F_\eta x_t - F_y Q^T(x_t))\tau] \\
    k_n[\frac{L}{0} U_nQdx + \Sigma_f F_\eta Q(x_t)]
\end{bmatrix}
\]  \tag{4.50}

The constant coefficients, \( k_n \) and \( k_t \), arise from Cox's (1970) first order force estimate (see equation (4.1)) and are given by

\[
\begin{align*}
    k_n &= -4\pi\mu \left( \ln \left( \frac{D}{L} \right) \right)^{-1} \\
    k_t &= -2\pi\mu \left( \ln \left( \frac{D}{L} \right) \right)^{-1}
\end{align*}
\]  \tag{4.51}
Similar to the dynamic model with inertia, the inertialess dynamic model is solved using the adaptive step size Runge-Kutta algorithm. The application of the Runge-Kutta method to the equations of motion will be discussed in Section 4.7.

4.6 Modelling the Contact Between the Fibre and the Channel and Slot Walls

The procedure to model the fibre trajectories is a three step process: first, a two-dimensional flow domain is solved using a CFD (Computational Fluid Dynamics) program; second, the node data of flow velocities from the CFD simulation is integrated with the software written to solve the equations of motion of the fibre; and third, the equation of motion software is run to simulate the flow of a flexible pulp fibre by seeding the fibre in the flow domain. Simulations of fibre motion will be presented in the next chapter. In all cases the geometry of the flow domain consists of a channel with a slot, as shown in Figure 4.4. In this section, two mathematical models will be presented to represent the contact between the fibre and the channel and slot walls. Advantages and shortcomings of each contact model will be discussed.

Figure 4.4: Channel and slot geometry.
4.6.1 Compliant Wall Contact Model

The first method developed to deal with fibre and wall interaction models the flow domain walls as spring and damper systems. Although the walls are modelled to be compliant, which is not very realistic, the analysis would essentially be the same as in the realistic situation where the flow domain walls are modelled to be rigid and the fibre walls are modelled as compliant. It was assumed that the forces acting on the fibre due to wall contact are in the direction normal to the wall face. The procedure for modelling the fibre/wall contact is described below.

The walls of the flow domain are designated by five different names, as depicted in Figure 4.4, namely, Top Channel Wall (TCW), Lower Left Channel Wall (LLCW), Lower Right Channel Wall (LRCW), Left Slot Wall (LSW) and Right Slot Wall (RSW). In the simulation procedure, a fibre is divided into $N$ nodes, typically $N = 1000$. During each step of the iterative numerical analysis, the global $X$ and $Y$ co-ordinates of each of the $N$ nodes is computed. If any of the nodes are located "behind" or "inside" a wall, then a point force is applied at that node according to Table 4.2.

<table>
<thead>
<tr>
<th>Node Location</th>
<th>Applied Point Force</th>
</tr>
</thead>
<tbody>
<tr>
<td>TCW</td>
<td>$F_r = -k_{wall}\delta - c_{wall}V_r$ (4.52)</td>
</tr>
<tr>
<td>LLCW</td>
<td>$F_r = k_{wall}\delta - c_{wall}V_r$ (4.53)</td>
</tr>
<tr>
<td>LRCW</td>
<td>$F_r = k_{wall}\delta - c_{wall}V_r$ (4.54)</td>
</tr>
<tr>
<td>LSW</td>
<td>$F_x = k_{wall}\delta - c_{wall}V_x$ (4.55)</td>
</tr>
<tr>
<td>RSW</td>
<td>$F_x = -k_{wall}\delta - c_{wall}V_x$ (4.56)</td>
</tr>
</tbody>
</table>

Table 4.2: Compliant wall contact forces.

Here $\delta$ represents the perpendicular distance from the wall face to the fibre node, $k_{wall}$ is the contact stiffness coefficient, $c_{wall}$ is the contact damping coefficient and $V_x$ and $V_y$ are $X$ and $Y$ velocity components of the node on the fibre. In the application of the inertialess model, the damping coefficient is set to zero.

Figure 4.5 depicts an example of a fibre in contact with the walls in the channel and slot junction. The fibre is in contact with three wall regions: LSW, RSW and LRCW. For all the nodes of the fibre that are in the LSW region, point forces according to equation (4.55) will be applied at the nodes. Similarly, those nodes located in the RSW region, the point forces will be
calculated by equation (4.56) and those nodes located in the LRCW, point forces will be calculated by equation (4.54).

While the compliant wall model is easy to implement, the simulation run times in which the fibre interacts with the walls were extremely long and often the simulations were found to be unstable. To combat this problem, a second method, utilizing constraint equations, was developed.

4.6.2 Constraint Wall Contact Model
The wall contact model presented here follows a similar approach taken by Olsen (1996). For any portion of the fibre which is in contact with a channel or slot wall, a constraint equation is applied which prevents motion of that portion of the fibre normal to the wall. For each constraint equation, a point contact force is solved to determine the force of contact required to prevent the fibre from moving normal to the wall at the point of contact region. Essentially, three types of contacts, or a combination of the three, can be encountered in the simulations, namely: contact with the channel wall, contact with the slot wall, contact with the channel and slot corner.

The velocity of any point on the fibre is given by

\[
\dot{X}_p = \begin{bmatrix}
X_o - xs_0 \hat{\theta} - QT \hat{s}_0 - QT \hat{c}_0 \hat{\theta} \\
Y_o + xc_0 \hat{\theta} + QT \hat{c}_0 - QT \hat{s}_0 \hat{\theta}
\end{bmatrix},
\]

(4.57)

Figure 4.6 shows a fibre whose end is in contact with the channel wall. The appropriate constraint equation for the inertialess model in this case, to prevent the fibre from moving normal
to the channel wall (i.e. the $Y$ direction), will be

$$\dot{\mathbf{R}}_p \pm\dot{Y}_o + xc_0 \dot{\vartheta} + Q^T \dot{t} c_0 - Q^T t_0 s_\theta \dot{\vartheta} = 0. \quad (4.58)$$

In the case of the inertia model, the acceleration equation would determine the constraint equation by differentiating equation (4.58) with respect to time. The force vector (equation (4.44) for the inertia model) can be written as

$$\mathbf{F} = \left[ \begin{array}{c} \int_0^L f_i c_0 - f_n s_\theta \, dx \\ \int_0^L f_i s_\theta + f_n c_0 \, dx + F_c \\ \int_0^L f_n x \, dx - \int_0^L f_i Q^T \, dx + F_c x_c c_0 - F_c Q^T(x_c) s_\theta \\ \int_0^L f_i Q \, dx + F_c Q(x_c) c_0 \end{array} \right] \quad (4.59)$$

where $F_c$ is the force of contact and $x_c$ is the point along the undeflected fibre length where the contact force acts, which, in this case $x_c$ would be either 0 or $L$. For the inertialess model, the force vector (equation (4.50)) becomes

$$\mathbf{F}_v = \left[ \begin{array}{c} k_c \int_0^L U_i c_0 \, dx - k_n \int_0^L U_i s_\theta \, dx \\ k_i \int_0^L U_i c_0 + k_n c_0 \int_0^L U_i \, dx + F_c \\ k_n \int_0^L U_n x \, dx - k_l \int_0^L U_i Q^T \, dx + F_c x_c c_0 - F_c Q^T(x_c) s_\theta \\ k_l \int_0^L U_i Q \, dx + F_c Q(x_c) c_0 \end{array} \right] \quad (4.60)$$

Thus equation (4.58), or its derivative with respect to time in the inertia model, can be utilized to

Figure 4.6: Constraint channel wall contact.
solve for the contact force, $F_c$.

Figure 4.7 depicts the case where the fibre contacts the right slot wall. In this situation, the appropriate constraint equation must prevent motion of the fibre contact point in the $X$ direction. For the inertialess model, the constraint equation is determined by

$$
(\mathbf{R}_p)_X = \dot{X}_o - x s_9 \dot{\theta} - Q^T \dot{\mathbf{r}} s_9 - Q^T \tau c_9 \dot{\theta} = 0.
$$

(4.61)

and for the inertia model, as before, the constraint equation is equation (4.61) differentiated with respect to time.

The force vector for the inertia model becomes

$$
\mathbf{F} = \begin{bmatrix}
\int_0^L (f_c c_9 - f_n s_9) dx - F_c \\
\int_0^L (f_c s_9 + f_n c_9) dx \\
\int_0^L f_n x dx - \int_0^L f_i Q^T d\tau + F_c x c_9 + F_c Q^T(x_c) \tau c_9 \\
\int_0^L f_n Q dx + F_c Q(x_c) s_9
\end{bmatrix}
$$

(4.62)

and for the inertialess model,

$$
\mathbf{F}_o = \begin{bmatrix}
k_i c_9 \int_0^L U_i dx - k_n s_9 \int_0^L U_n dx - F_c \\
k_i s_9 \int_0^L U_i dx + k_n c_9 \int_0^L U_n dx \\
k_n \int_0^L U_n x dx - k_i \int_0^L U_i Q^T d\tau + F_c x c_9 + F_c Q^T(x_c) \tau c_9 \\
k_n \int_0^L U_n Q dx + F_c Q(x_c) s_9
\end{bmatrix}.
$$

(4.63)

![Figure 4.7: Constraint slot wall contact.](image)
Again, the constraint equation is used to solve for the contact force, $F_c$. In the case where the fibre contacts the left slot wall, $F_c$ is replaced by $(-F_c)$ in equations (4.62) and (4.63).

The case where the fibre contacts the corner of the channel and slot junction is shown in Figure 4.8. Here, motion in the normal direction of the contact point of the fibre is constrained as follows for the inertialess model:

$$\begin{align*}
(\dot{R}_p)_n = - (\dot{R}_p)_x s_\theta + (\dot{R}_p)_y c_\theta = -X_o s_\theta + Y_o c_\theta + Q^T \tau + x_c \dot{\theta} = 0. 
\end{align*}$$

Again, the constraint equation for the inertia model is simply equation (4.64) differentiated with respect to time. Now the force vector for the inertia model becomes

$$F = \begin{bmatrix}
\int_0^L (f_i s_\theta - f_n s_\theta) dx - F_c s_\theta \\
\int_0^L (f_i s_\theta + f_n c_\theta) dx + F_c c_\theta \\
\int_0^L f_n x dx - \int_0^L f_i Q^T dx + F_c x_c \\
\int_0^L f_n Q dx + F_c Q(x_c)
\end{bmatrix}$$

and for the inertialess model,

$$F_v = \begin{bmatrix}
k_i c_\theta \int_0^L U_i dx - k_n s_\theta \int_0^L U_n dx - F_c s_\theta \\
k_i s_\theta \int_0^L U_i dx + k_n c_\theta \int_0^L U_n dx + F_c c_\theta \\
k_n \int_0^L U_n x dx - k_i \int_0^L U_i Q^T dx + F_c x_c \\
k_i \int_0^L U_n Q dx + F_c Q(x_c)
\end{bmatrix}$$

Figure 4.8: Constraint corner of channel and slot contact.
As in the previous two cases, the constraint equation is used to solve for the contact force, $F_c$. In the corner contact case, care must be taken to ensure that the orientation of the fibre is consistent with the defined direction of $F_n$. If the fibre is oriented such that the contact force acts in the negative $n$-direction, then $F_c$ is replaced by $(-F_c)$ in equations (4.65) and (4.66).

In cases where there is more than one point of contact, equations (4.58) to (4.66) are used as many times as there are corresponding contacts. In many simulation runs, the fibre would contact both the slot wall and the channel and slot corner. Applying the constraint equations as explained above would lead to two equations of constraint. Usually, the simulation would "hold" the fibre in this position forever. In this case Olsen (1996) considered the fibre to be vertically stapled and ended the simulation. If the fibre is in contact with the up-stream channel and slot corner and the down-stream slot wall, it is physically unrealistic to assume the fibre to be stapled in this situation. Another difficulty with applying the equations as given above is that once a fibre touches a wall or corner, there is no built-in mechanism to let go of the fibre. The difficulties were overcome by solving the equations for all contact forces, $F_c$, and if any of the forces were found to be negative then the corresponding constraint equation was eliminated and the equations were solved a second time. This elimination process would continue until all of the contact forces had a positive value. This method was found to give satisfactory results.

4.6.3 Advantages and Shortcomings of the Wall Contact Models

The great advantage of the Compliant Wall Contact Model over the Constraint Wall Contact Model is the ease of its applicability. The method is simple to apply to more complicated geometries. The Constraint Wall Contact Model requires that each contact scenario be programmed into the wall contact simulation model. This task becomes extremely difficult even for the simple flow geometry considered here for cases where the fibre is quite flexible. The problem is further complicated by the use of an adaptive step size ordinary differential equation (ODE) solver, such as the Runge-Kutta method introduced previously. Often, the step size, in this case time, $t$, first tried by the solver is too large and the fibre "jumps" to a location that is completely unexpected. While the Compliant Wall Contact Model can deal with this situation in a straightforward method, the Constraint Wall Contact Model must determine that the contact scenario is not possible, based on the previous fibre position, and flag the ODE solver.
appropriately to decrease the time step.

The major shortcoming of the Compliant Wall Contact Model is that it is extremely slow and even unstable in certain circumstances. For example, the flow of a flexible kraft fibre through the slot required a two week simulation time on a 133 MHz Pentium computer using 32-bit compiled code. The same simulation utilizing the Constraint Wall Contact Model required only one half hour on the same computer. This great advantage of the Constraint Wall Contact Model warrants its application in the simulations to be presented in the following chapter. It should be emphasized that the results obtained from the two different contact models gave very similar results.

4.7 Numerical Solution of the Equations of Motion of a Flexible Fibre in a Shear Flow

The equations of motion, equations (4.46) and (4.48), represent systems of coupled ordinary differential equations. In contrast to the large deflection equations developed in Chapter 3, all the initial boundary conditions are known or can be assumed. This fact makes the problems initial value problems and the Runge-Kutta method discussed in Section 4.2 can be applied directly. To apply the Runge-Kutta method equations (4.46) and (4.48) must be written in the form given in equation (3.10) and repeated here for convenience,

\[ \dot{\psi} = G(\varphi, \psi). \]  

(4.67)

In the present analysis, the independent variable, \( \varphi \), is replaced by the independent variable representing time, \( t \). The vector \( \psi \) and the vector of functions \( G \) are defined below.

The dynamic equations for the model including inertia, equation (4.46), can be written as

\[ \ddot{\mathbf{q}} = M(q)^{-1} \left[ F - C(q, \dot{q}) - \begin{bmatrix} 0_3 \\ K_f \end{bmatrix} \right]. \]  

(4.68)

The vector \( \psi \) and the vector of functions \( G \) for the inertia model can now be defined as

\[ \psi_{\text{inertia}} = \begin{bmatrix} \dot{\mathbf{q}} \\ \mathbf{q} \end{bmatrix} \Rightarrow \psi_{\text{inertia}} = \begin{bmatrix} \ddot{\mathbf{q}} \\ \dot{\mathbf{q}} \\ \mathbf{q} \end{bmatrix}, \psi_{\text{inertia}} \in \mathbb{R}^{2(3+m)} \]
where the prime (') was replaced by the dot (\( \cdot \)) to designate differentiation with respect to time, and

\[
\mathbf{G}_{\text{inertia}}(t, \mathbf{\psi}_{\text{inertia}}) = \begin{bmatrix} \mathbf{M}(\mathbf{q})^{-1} \left( \mathbf{F} - \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}}) - \begin{bmatrix} 0_3 \\ \mathbf{K}_T \\ \mathbf{B} \end{bmatrix} \right) \end{bmatrix}, \quad \mathbf{G}_{\text{inertia}}(t, \mathbf{\psi}_{\text{inertia}}) \in \mathbb{R}^{(3+m)}.
\]

The inertialess equations of motion, equation (4.48), can be rewritten as

\[
\dot{\mathbf{q}} = \mathbf{M}_v(\mathbf{q})^{-1} \left( \mathbf{F}_v - \begin{bmatrix} 0_3 \\ \mathbf{K}_T \end{bmatrix} \right).
\]

Thus, for the no inertia model, the vector \( \mathbf{\psi} \) and the vector of functions \( \mathbf{G} \) are defined as

\[
\mathbf{\psi}_{\text{no-in}} = \mathbf{q} \Rightarrow \dot{\mathbf{\psi}}_{\text{no-in}} = \dot{\mathbf{q}}, \quad \mathbf{\psi}_{\text{no-in}} \in \mathbb{R}^{(3+m)}
\]

and

\[
\mathbf{G}_{\text{no-in}}(t, \mathbf{\psi}_{\text{no-in}}) = \begin{bmatrix} \mathbf{M}_v(\mathbf{q})^{-1} \left( \mathbf{F}_v - \begin{bmatrix} 0_3 \\ \mathbf{K}_T \end{bmatrix} \right) \end{bmatrix}, \quad \mathbf{G}_{\text{no-in}}(t, \mathbf{\psi}_{\text{no-in}}) \in \mathbb{R}^{(3+m)}.
\]

It should be emphasized that if the Constraint Wall Contact Model is being used then equations (4.68) and (4.69) will contain the extra constraint equation(s) and the unknown contact force(s), \( \mathbf{F}_c \), which must also be solved.

The diagonal stiffness matrix, \( \mathbf{K} \), is given by

\[
\mathbf{K} = \frac{EI}{L^3} = \begin{bmatrix}
500.56 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 3803.5 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 14616 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 39944 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 89135 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 173881 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 308208 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 508482 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 793403 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1184014
\end{bmatrix}
\]

for the first ten modes of vibration.

Since the normal and tangential fluid forces acting on the fibre must be integrated along the length of the fibre, a numerical integration must be performed. The algorithm used to accomplish this task, described by Press et al. (1992), uses an extended trapezoidal rule in a
sequential manner. One of the difficulties associated with the numerical integrations required in this analysis is that the integrations for the forces often approach zero. For example, in the case where a fibre is flowing parallel to the flow streamlines, the normal forces acting on the fibre should integrate to zero, as should the deflection terms. Numerically, this problem can be difficult to solve for the following reason. The numerical integration routine used in the analysis calls an extended trapezoidal function sequentially until a desired tolerance, defined as

\[
\text{tolerance} = \left| \frac{\text{last calculated value} - \text{second last calculated value}}{\text{last calculated value}} \right|
\]

is reached. When the integration approaches zero, the calculated tolerance can become large. One method to combat this problem is to specify a small value for which calculated values smaller than that value are considered to be zero. A better and more efficient approach was used in the analysis and will be described here.

Consider the following integration

\[
I = \int_a^b g(x) \, dx
\]  
(4.70)

where \( g(x) \) is some function. Equation (4.70) is equivalent to

\[
I = \int_a^b (\beta + g(x)) \, dx - \beta(b-a)
\]  
(4.71)

where \( \beta \) is a constant not equal to zero. If the integral of equation (4.70) approaches zero, then the integral in equation (4.71) will approach a non-zero value of \( \beta(b-a) \). By adding the constant to the integral, the numerical difficulties associated with integrals which approach zero can be avoided. This approach was used to make the integration algorithms of the simulation software more efficient.

**4.8 Comparison with Static Analysis**

To demonstrate the validity of the dynamic numerical analysis a flexible fibre is loaded in such a way that the deflection and motion of the fibre reach a steady-state and the results are compared to static theory results.
To compare the dynamic analysis with a static analysis, a flexible fibre is introduced into a simple shear flow (see Figure 4.9) and the system is allowed to reach a steady-state. The equation for the velocity profile depicted in Figure 4.9 is

\[ U = U_{\text{max}} \left( 1 - \left| \frac{Y}{a} \right| \right) \]  

(4.72)

where \( U_{\text{max}} \) is the maximum velocity in the \( X \)-direction (i.e. \( U(0) = U_{\text{max}} \)), and \( a \) is the point on the \( Y \)-axis where the velocity is zero. The initial fibre position is centred at the origin of the global \( X-Y \) axes, oriented along the \( Y \)-axis. This gives, at \( t = 0 \): \( X_o = 0, Y_o = -L/2, \) and \( \theta = \pi/2 \). Initially the fibre is assumed to have no velocity and no deflection. If \( a = L/4 \), then the forces, acting on the fibre due to the fluid flow, balance and the fibre should only deflect. As \( t \to \infty \), the flexible fibre should reach a constant, steady-state deflected shape.

Using static engineering beam theory (Higdon et al. (1985)), the maximum deflection of a simply supported beam loaded as in Figure 4.9, with \( a = L/4 \) is given by

\[ \Delta = \frac{7}{1920} \frac{F_{\text{max}} L^4}{EI} \]  

(4.73)

where \( F_{\text{max}} \) is the maximum force applied (analogous to \( U_{\text{max}} \) in equation (4.72)).

For the test case, the following parameters were used:

\[ \mu = 1 \times 10^{-3} \text{ Ns/m}^2 \]
\[ U_{\text{max}} = 0.35 \text{ m/s} \]
\[ D = 30 \mu \text{m} \]
\[ L = 3 \text{ mm} \]
\[ EI = 1 \times 10^{-12} \text{ Nm}^2 \]

The maximum applied force, \( F_{\text{max}} \), is determined by substituting the above values in the normal force equation (equation (4.1)) and using the value of \( U_{\text{max}} \) for \( U_a \). This gives \( F_{\text{max}} = 9.6 \times 10^{-5} \text{ N/m} \), which in turn gives \( \Delta = 0.28 \text{ mm} \).

Figure 4.10 plots the maximum deflection of the fibre versus time. The curve labelled "\( M_0 \)" represents the fibre deflection determined by applying the inertialess model, whereas the curves labelled "\( M_1 \)" and "\( M_2 \)" were determined by applying the model including inertia for a fibre mass of \( 9 \times 10^{-10} \text{ kg} \) and \( 4 \times 10^{-9} \text{ kg} \), respectively. The plots were generated by running the simulations with one, two and three modes of vibration (\( m = 1, 2 \) and \( 3 \)) each producing similar results. The horizontal dashed line at \( \Delta \) is the predicted deflection using static analysis. Clearly, the steady-state deflection of the dynamic model matches the static analysis. As expected, the
Inertialess model, "\(M_0\)”, reaches a steady-state deflection much more quickly than the inertia models. The inertia model with less mass, "\(M_1\)”, is overdamped by the hydrodynamic forces and does not overshoot the steady-state value. The more massive model, "\(M_2\)”, on the other hand, overshoots the steady-state value. The shapes of the fibre for the first millisecond in 0.2 ms increments are plotted in Figure 4.11, for \(M = 9 \times 10^{-10}\) kg. The simulation was performed using 3 modes of vibration (i.e. \(m = 3\)).

A similar analysis was performed with the fibre positioned parallel to the \(x\)-axis and the flow running parallel to the \(y\)-axis, and the deflections agreed identically with these described above.
Figure 4.10: Maximum deflection vs. time \((m = 1, 2, 3, M_0 = \text{inertialess}, M_1 = 9 \times 10^{-10} \text{ kg}, M_2 = 4 \times 10^9 \text{ kg})\).

Figure 4.11. Deflection of fibre \((t = 0 \text{ to } 1 \text{ ms}, m = 3, M = 9 \times 10^{-10} \text{ kg})\).
4.9 Conclusions

Using Hamilton's principle, the equations of motion for a flexible fibre in a flow were developed. The equations were discretized by utilizing the assumed modes of vibration for a beam with both ends free. By using a simple shear flow in the simulation, the flexible fibre was allowed to reach a steady-state deflection shape. It was shown that the dynamic simulation compared favourably with analytical static equations for both models, inertia and inertialess. The numerical model developed in this chapter will be used to simulate the flow of flexible fibres in a channel with a slot in Chapter 5, and will be compared to experimental results and observations.
Chapter 5

Fibre Dynamics: Motion Simulations

As mentioned in Chapter 1, stiff fibres and shives reduce paper quality. The fractionation of pulp fibres based on fibre stiffness is an important issue in the pulp and paper industry. In particular, it is desirable to remove shives and other contaminants from flexible fibres. Although a number of experimental studies have been performed to observe the flow of flexible fibres in screening applications, no mathematical or numerical models have been proposed to simulate the motion of flexible fibres in a shear flow. In Chapter 4 the equations of motion for a flexible fibre in a flow were developed. The purpose of this chapter is to use the numerical model to study the motion of fibres of different stiffnesses in a channel flow with a slot to gain a better understanding of the fundamental mechanisms of fibre fractionation based on fibre flexibility. The procedure to model the fibre trajectories is a three step process: first, a two-dimensional flow domain is solved using a CFD (Computational Fluid Dynamics) program; second, the node data of flow velocities from the CFD simulation is integrated with the software written to solve the equations of motion of the fibre; and third, the equation of motion software is run to simulate the flow of a flexible pulp fibre by seeding the fibre in the flow domain. In the analysis, it is assumed that the fibre does not affect the flow of the fluid and that the forces of the fluid acting on the fibre are given by the first order approximation of Cox (1970). The simulation results will be compared to the experimental work performed by Gooding (1986) and Kumar (1991).

The first section will present the geometry of the experimental set-up used by Gooding
and Kumar. The development and validation of the CFD model will be presented in the second section. The third section will discuss general issues with respect to the model, such as the integration of the flow domain with the fibre motion model and the effect of including inertia terms. In the fourth section, a qualitative comparison will be made between observed fibre trajectories reported by Gooding and those predicted by the simulations. The fifth section will quantitatively compare the passage ratios of fibres determined by Kumar with those predicted by the simulations. A discussion of the simulation results will be given in the sixth section.

5.1 Channel and Slot Geometry

Gooding (1986) developed a laboratory scale experimental flow loop to observe the passage of fibres through a plexiglass channel and slot test section. A dilute suspension of fibres in a water medium was used. High speed cine-films were taken of the fibre motion near the slot entry. Kumar (1991) used the set-up to perform numerous experiments. Simulations presented in this chapter will be compared to the results of Gooding and Kumar. Figure 5.1 shows a two-dimensional schematic representation of a channel with a slot. The length of the channel used by Kumar was 350 mm and its cross-sectional dimensions were 20 mm x 20 mm. The distance to the slot was 192 mm. The entry to the channel consisted of an abrupt contraction from a circular

![Figure 5.1: Channel flow with a slot geometry.](image-url)
pipe with a diameter of 38 mm. The test section in the slot area was made to be modular in order that different slot geometries could be tested. All comparisons presented in this thesis will use the experimental results of Kumar and Gooding based on a slot width of 0.5 mm.

Kumar maintained the average upstream channel velocity, $U_r$, at a constant value of approximately 6.5 m/s which corresponds to a Reynolds number of 130 000 based on the height or hydraulic diameter of the channel. Fox and MacDonald (1985) reported that flows between two infinite parallel plates would remain laminar for a Reynolds number, based on the distance between the plates, of less than 1000. Clearly, the flow in the channel is turbulent. Kumar estimated the turbulence intensity in the mid-section of the channel at the channel and slot junction to be on the order of 10%. The average slot velocity, $U_s$, was varied from approximately 1.3 m/s to 7.8 m/s which corresponds to a Reynolds number range, based on the slot width, of 650 to 3900. This Reynolds number range corresponds to a transition from laminar to turbulent flow in the slot. Since the flow in the channel is turbulent, one would expect that in the primary area of interest, at the channel and slot junction, the flow would remain turbulent.

Kumar used four different fibre batches to observe the ability of fibres to pass through the slot based on fibre flexibility:

- **F1. nylon**: $L = 1$ mm, $EI = 329 \times 10^{-12}$ Nm$^2$;
- **F2. rayon**: $L = 1$ mm, $EI = 3.1 \times 10^{-12}$ Nm$^2$;
- **F3. nylon**: $L = 3$ mm, $EI = 329 \times 10^{-12}$ Nm$^2$;
- **F4. kraft**: $L = 3.6$ mm, $EI = 4.4 \times 10^{-12}$ Nm$^2$.

Since the nylon fibres have such a high stiffness value, they will be modelled as rigid fibres in the simulations and a 30 µm fibre diameter will be used.

### 5.2 CFD Simulation of the Channel and Slot

The first step in the simulation procedure is to solve a two-dimensional flow domain by applying CFD. The CFD package used to simulate the flows was the finite element based CFD solver FIDAP version 7.52. As mentioned in Chapter 2, Gooding (1986) and Gooding and Kerekes (1989) suggested that two mechanisms affect screening: the "wall effect" and the "turning effect". In the fibre motion simulations presented in this thesis, only the "turning effect" will be
considered. This means that the most important region of study is at and near the channel and slot junction. The maximum dimension of the flow domain is the channel length, 350 mm, and the minimum dimension is the slot width, 0.5 mm, which represents a maximum to minimum dimension ratio of 700. An accurate two-dimensional CFD modal of the domain depicted in Figure 5.1 with Kumar's dimensions would require on the order of 100,000 nodes. Although this is a manageable problem on today's large and fast computers, the area of interest is in the channel and slot region which implies that the entire domain does not need to be solved provided that appropriate boundary conditions are implemented.

Referring to Figure 5.1, the dimensions of the flow domain for the CFD simulation, were

\[
\begin{align*}
L_c &= 40.5 \text{ mm} \\
H_c &= 20.0 \text{ mm} \\
L_s &= 5.0 \text{ mm} \\
W_s &= 0.5 \text{ mm} \\
L_{ca} &= 20.0 \text{ mm}.
\end{align*}
\]

Nine node quadrilateral elements were used to discretize the velocity, and four node continuous quadrilateral elements were used to discretize the pressure. Figure 5.2 shows a typical finite element mesh used in the simulations for the flow domain and Figure 5.3 shows a close-up view of the mesh in the channel and slot junction. Approximately 10,000 nodes were used to discretize the flow domain. The standard \( k-\varepsilon \) model (see FIDAP (1993)) was used to simulate turbulence. A subroutine was written and linked with the CFD package to apply a fully developed turbulent flow profile at the channel inlet. Schlichting (1979) presented an empirical formula developed

![Figure 5.2: Flow domain finite element mesh.](image)
from experimental work by Nikuradse (1932) for a fully developed turbulent velocity profile in a pipe as

\[ u(y) = U_m \left( \frac{y}{R} \right)^{1/n} \]  

(5.1)

where \( U_m \) is the maximum velocity in the pipe, \( R \) is the radius, \( y \) is measured from the wall towards the centre of the pipe and \( n \) is determined from experiments. For a Reynolds number of 130 000, a value of \( n = 7 \) is appropriate. Although the flow profile in equation (5.1) is for flow in a pipe, it is reasonable to apply it to the two-dimensional problem at hand since the two-dimensional CFD simulation is an approximation of Kumar's channel (i.e. square pipe) flow. Since Kumar used an average speed of 6.5 m/s, equation (5.1) was integrated to determine the corresponding value for \( U_m \) which would give an average flow speed of 6.5 m/s across the channel inlet and this value was found to be 7.4 m/s.

As mentioned earlier, Kumar estimated the turbulence intensity in the mid-section of the channel in the slot area to be approximately 10%. For this reason, a 10% turbulent intensity at the
inlet was specified. Since the kinetic energy per unit mass is

\[ k = (I_m U_a)^2 \]  \hspace{1cm} (5.2)

where \( I_m \) is the turbulence intensity and \( U_a \) is the average velocity at the channel inlet, a value of \( k = 0.4 \) was specified at the channel inlet in the CFD simulations. FIDAP recommends estimating the dissipation according to

\[ \varepsilon = \frac{k^{3/2}}{0.1 L_\delta} \]  \hspace{1cm} (5.3)

where \( L_\delta \) is the characteristic width, defined as half the channel height for channel flow. Therefore a value of \( \varepsilon = 250 \) was specified at the inlet. It should be mentioned that changing the \( k \) and \( \varepsilon \) boundary conditions at the inlet did not have any significant effect on the mean channel velocities.

Since FIDAP uses the finite element method to discretize the Navier-Stokes equations, stress flux boundary conditions are used to model a constant pressure at a boundary. To "pull" or suck the fluid through the slot, a stress flux boundary condition in the \( Y \)-direction was used at the slot outlet. This stress flux was varied to simulate different flow rates through the slot. A zero velocity component in the \( X \)-direction was specified at the slot outlet. Neumann boundary conditions were used for \( k \) and \( \varepsilon \) at the slot outlet and for velocity, \( k \) and \( \varepsilon \) at the channel outlet. The no-slip boundary condition was used at the walls. The walls were also specified as "WALLS" in FIDAP in order that the turbulent "wall function" be implemented in their immediate vicinity (see FIDAP for details).

Seven CFD simulations were performed to be used in the fibre motion simulations for obtaining a comparison with the results of Kumar's experimental work. An eighth simulation was run for comparison with a "large" simulation for numerical validation. Table 5.1 lists the normal stress flux boundary condition specified at the slot outlet, the average slot flow speed and the slot to channel average flow speed ratio for each simulation.
To test the validity of the numerical convergence of the CFD models, a simulation was run considering a larger domain and a finer mesh. Referring to Figure 5.1, the dimensions of the flow domain for the large CFD simulation were

\[
\begin{align*}
L_c &= 100.5 \text{ mm} \\
H_c &= 20.0 \text{ mm} \\
L_s &= 10.0 \text{ mm} \\
W_s &= 0.5 \text{ mm} \\
L_{ca} &= 50.0 \text{ mm}.
\end{align*}
\]

Approximately 30 000 nodes were used to discretize the flow domain. The normal stress flux at the slot outlet was set to a value of -20 000 N/m². For the simulation, \( U_s \) was found to be 3.6 m/s. The corresponding "small" simulation is labelled CFD8 in Table 5.1.

Since the region of interest is near the channel and slot junction, the velocity profiles of five different locations near the channel and slot are plotted. The physical locations of the plots are depicted in Figure 5.4. Figures 5.5a to 5.5e plot the X and Y components of the velocity for each of the respective locations comparing the results obtained from the large simulation with the results from CFD8. In all the plots, the error between the large simulation and CFD8 is less than 10% based on the mean channel flow.

Streamline and velocity vector plots at the channel and slot junction for simulations CFD1 to CFD7 are given in Figures 5.6 to 5.12, respectively. For smaller \( U/U_s \) ratios, a greater recirculation zone is present on the upstream portion of the slot. This is expected, since a \( U/U_s \)

<table>
<thead>
<tr>
<th>Reference</th>
<th>Normal Stress Flux at Slot Outlet</th>
<th>Average Slot Flow Speed, ( U_s )</th>
<th>( U/U_s )</th>
</tr>
</thead>
<tbody>
<tr>
<td>CFD1</td>
<td>-4000 N/m²</td>
<td>1.5 m/s</td>
<td>0.24</td>
</tr>
<tr>
<td>CFD2</td>
<td>-8000 N/m²</td>
<td>2.4 m/s</td>
<td>0.37</td>
</tr>
<tr>
<td>CFD3</td>
<td>-13300 N/m²</td>
<td>3.3 m/s</td>
<td>0.50</td>
</tr>
<tr>
<td>CFD4</td>
<td>-22000 N/m²</td>
<td>4.4 m/s</td>
<td>0.67</td>
</tr>
<tr>
<td>CFD5</td>
<td>-32000 N/m²</td>
<td>5.4 m/s</td>
<td>0.83</td>
</tr>
<tr>
<td>CFD6</td>
<td>-45000 N/m²</td>
<td>6.5 m/s</td>
<td>1.00</td>
</tr>
<tr>
<td>CFD7</td>
<td>-55000 N/m²</td>
<td>7.2 m/s</td>
<td>1.11</td>
</tr>
<tr>
<td>CFD8</td>
<td>-15000 N/m²</td>
<td>3.6 m/s</td>
<td>0.55</td>
</tr>
</tbody>
</table>

Table 5.1: Slot outlet boundary conditions
ratio of zero would represent a cavity driven flow, where only recirculation is present in the slot or cavity. The recirculation zone creates a barrier for fibres trying to pass through the slot. For $U_j/U_w$ ratios equal or greater than one, the streamline plots show an overshoot of the flow at the channel and slot junction.
Figure 5.5a.

Figure 5.5b.

Figure 5.5c.

Figure 5.5d.

Figure 5.5e.

Figure 5.5: Flow velocity comparison between the Large Simulation and CFD8.
Figure 5.6: Streamline and velocity vector plots for CFD1. $U/U_\infty = 0.24$.

Figure 5.7: Streamline and velocity vector plots for CFD2. $U/U_\infty = 0.37$.

Figure 5.8: Streamline and velocity vector plots for CFD3. $U/U_\infty = 0.50$. 
Figure 5.9: Streamline and velocity vector plots for CFD4. $U/U_\infty = 0.67$.

Figure 5.10: Streamline and velocity vector plots for CFD5. $U/U_\infty = 0.83$.

Figure 5.11: Streamline and velocity vector plots for CFD6. $U/U_\infty = 1.00$. 

86
5.3 General Aspects of the Fibre Motion Model

General aspects of the fibre motion model will be discussed in this section. Specifically, the integration of the CFD simulation with the fibre motion will be presented, the simplifying assumptions of the numerical model and their effect on the fibre motion simulation will be discussed, as will the effects of using the inertia and inertialess models developed in the previous chapter.

5.3.1 Integration of the Flow Domain

The equations of motion require the flow velocity to be known at every point of the flow domain. To make the simulation efficient, it is important to evaluate the flow speeds as quickly as possible. As mentioned previously, the velocity of the flow is solved using a commercial CFD package. Since the geometry that was used in these simulations is not complicated, the CFD calculations were performed using a structured mesh; however the simulation software can be integrated with CFD solutions solved with unstructured meshes. This capability is important because it provides the capability to simulate flexible fibre motion in more complicated flow geometries and could be used as a design tool for pulp screen plate manufacturing. The integration of the CFD data with the fibre motion software is described below.

The CFD software outputs to a file a list of the flow velocities and pressures at each node.
of the CFD mesh. The fibre motion software reads the CFD output file and the node/element connectivity file, and determines the size of the smallest element in the flow domain. Based on the smallest element size, the fibre motion software superimposes a square grid over the flow domain. Next, the location of each node of the square grid with respect to the CFD mesh is determined and the x and y velocity components of the CFD mesh are interpolated to the node of the square grid. By making the grid square, the calculation of the velocity of any point in the flow domain becomes very efficient since the task to determine which nodes should be used to interpolate the data is computationally inexpensive.

5.3.2 Practical Considerations of the Fibre Motion Model

As with all numerical work, simplifying assumptions must be made to make the fibre motion model practical and manageable. Assumptions of the model are mentioned here, with justification.

Assumption: The forces acting on the fibre are given by Cox's equations

The first order approximation of Cox's (1970) equations represents the classical linear Stokes drag relation. A linear Stokes force has been used extensively in the literature to model the flow of fibres and particles, both, in laminar flows (see Gradon and Podgorski (1990), Podgorski et al. (1995), for example) and turbulent flows (see Olsen (1996)). Considering a similar geometry, Olsen (1996) theoretically showed that turbulence will affect the mean fibre velocity by only 4.25% and for this reason neglected the effect of turbulence in his rigid fibre motion simulations. Therefore, as an initial development of a flexible fibre motion model, the omission of the effects of fluctuating velocity components due to turbulence and the application of Cox's first order approximation, are justified.

Although a three-dimensional CFD analysis, which will be discussed in Chapter 6, could be applied to model the motion of a fibre in a shear flow, the amount of computational work required using CFD methods is prohibitive with today's computer power. Cox's (1970) equations are a first order approximation of a perturbation analysis of a Stokes flow. This fact, together with the fact that the fibre density is similar to that of water, suggests that the inertialess equations of motion are the most appropriate. In this section both forms of the equations of motion will be
utilized to investigate the effects of including inertia in the model. It should also be mentioned that multiplying the force equations by a factor of 100 did not have any effect on the fibre trajectories of a rigid inertialess fibre. In fact, observation of equation (4.50) shows that while the global $X$ and $Y$ positions of the fibre are affected by both the normal and tangential forces, only the normal force plays a role in the fibre orientation for a rigid fibre. This implies that for a rigid inertialess fibre, the orientation of the fibre is essentially independent of the ratio of the normal to tangential forces acting on the fibre.

**Assumption: Euler-Bernoulli equations for beam bending are applicable**

The equations of motion for a flexible fibre used in this analysis assume the fibre deflections to be small. The large deflection analysis of Chapter 3 showed that in static deflections of fibres, the small deflection analysis was applicable to within 10% error for deflections of up to 20%, based on the fibre length. The assumption here is that the errors for the dynamic case will not be greater and that the equations are appropriate for this analysis. In the large deflection analysis it was shown that the deflections using small deflection analysis were generally greater than those of the large deflection analysis. Extrapolating this fact to the dynamic deflection model, it is expected that the use of small deflection beam theory for modelling the motion of flexible fibres will overestimate the bending of the fibres. The results from the simulations will show that in the critical region of the channel and slot junction, the fibre deflections are small until the fibre makes contact with a channel or slot wall and this initial contact is usually the determining factor for fibre passage through the slot.

**Assumption: Three-dimensional effects or out-of-plane fibre motion is negligible**

The analysis developed for fibre motion is for a two-dimensional problem where the motion of the fibre is constrained to move in the global $X$-$Y$ plane. If a three-dimensional flow is considered, the method of superposition for beams in three-dimensional vibration can be used to model the fibre dynamics. While in the present analysis only the two-dimensional model is considered, out-of-plane effects can be accounted for by shortening the fibre length by a factor representative of the projected fibre length.
Assumption: Initial Fibre Orientation is Horizontal

In a recent study, Olsen (1996) measured the probability density of fibre orientations near a channel wall. As expected, it was found that fibres tended to align parallel to the channel wall due to geometric constraints. Furthermore, it was found that the fibres had a preferential alignment with the downstream end of the fibres pointing away from the wall. In the fibre motion simulations, it was found that the fibres tended to align along the streamlines. Fibres whose initial angle, $\theta$, was positive (i.e. the downstream end of the fibre points away from the wall) were less likely to pass through the slot than horizontally or negatively oriented fibres.

5.3.3 Effect of Inertia on the Fibre Motion Simulations

Using a typical fibre coarseness of 30 mg/100 m (Smook (1992)), the mass of a 3 mm long fibre would be approximately $9 \times 10^{-10}$ kg. The effect of inertia is to resist the change of motion of a body by external forces acting on that body. Although the motions of fibres with mass and the inertialess fibre motions were similar, the fibres with inertia experienced a more "lethargic" motion; i.e. they were more resistant to motion change. For this reason one would expect the trajectories of fibres with inertia to be more resilient to changes in the flow direction, with the result that they would tend to flow through the channel and slot junction and not enter the slot as easily. This is exactly what is observed in the simulations. Fibres of greater mass, initially located 10 mm upstream of the slot with a horizontal orientation, must have their initial location closer to the lower left channel wall than their lighter counterparts in order for them to pass through the slot. Since pulp fibres and the synthetic fibres used by Gooding and Kumar have essentially neutral buoyancy, the inertialess fibre motion model will be used in the simulations of the following sections.

5.4 Fibre Motion Simulations: Qualitative Analysis

The results of the fibre motion simulations will be discussed qualitatively in this section. A general flexible fibre motion simulation will be presented, followed by a discussion of predicted methods for fibre passage through the slot and a qualitative comparison with experimental observations. An important feature to consider in studying the fibre motion simulations is the initial fibre and wall contact. Generally, if the leading edge of the fibre hits the downstream slot
wall, it will enter the slot, whereas if the leading edge hits the downstream channel wall the fibre will either staple the slot or not enter and continue downstream in the main channel flow. Exceptions to this generalization will be discussed later in this section.

In all of the simulations of fibre motion, the fibre is seeded in the channel with its mid-point 10 mm upstream of the upstream corner of the channel and slot and the fibre is oriented horizontally. The distance from the lower channel wall to the initial fibre starting point is defined as $H_f$. The height of the exit layer at this point is defined as $H_{EL}$ (Figure 5.13). As mentioned previously, a fibre diameter of 30 μm will be used in the simulations.

Figure 5.14 shows a typical fibre trajectory from start to finish. The simulation was run

![Figure 5.13: Initial fibre position.](image)

![Figure 5.14: Simulation of a kraft fibre entering the slot.](image)
using the flow model, CFD3, with a kraft fibre, F4, whose properties are given in Section 5.1, with $H_f = 0.2$ mm. Label "a" depicts the starting position. At label "b", the fibre has travelled approximately halfway to the slot. Label "c" depicts the critical region. In this case, the fibre has deflected enough with the streamlines so that its leading edge has made contact with the downstream edge of the slot wall. As mentioned above, generally, if the leading edge of the fibre contacts the downstream slot wall, it will enter the slot, whereas if it contacts the downstream channel wall it will either staple the slot or flow past it. Labels "d" and "e" depict different stages of fibre passage through the slot.

In the experimental study of the flow of fibres in a channel with a slot performed by Gooding (1986), five "Fibre Motion Types" were reported:

1. the fibre moves past the slot without either entering the slot or touching the slot wall;
2. one end of the fibre enters the slot and touches one of the walls, but is then swept out of the slot and back into the mainstream;
3. one end of the fibre enters the slot and the fibre is immobilized - balanced on the downstream edge of the slot (commonly called "stapling");
4. the fibre passes through the slot after contacting one (or both) of the slot walls;
5. the fibre passes through the slot without contacting either slot wall.

All of these motions are observed by the fibre motion simulations except for Fibre Motion Type 3. Often, fibres do become immobilized on the downstream edge of the slot for a period of time, but eventually, they either flow out of the slot or pass through it. In this section, a generalized discussion of the types of fibre motion predicted by the simulations will be presented.

In the simulations, it was observed that fibres which approach the channel and slot junction can be grouped into five general categories according to their motion, as follows:

A. the fibre first contacts the upstream channel and slot corner;
B. the fibre enters the slot without any contact;
C. the fibre first contacts the downstream slot wall;
D. the fibre first contacts the downstream channel wall;
E. the fibre does not contact any walls and simply flows downstream in the channel.

Category A type motion is observed in cases where the fibre's initial starting height, $H_p$, is small. In these cases the fibre flows parallel to the channel wall until its leading portion enters the
channel and slot junction, where the downward flow causes the fibre to contact the upstream channel and slot corner. Often, the fibre eventually hits the downstream slot wall. In most cases the fibre rotates and enters the slot. This type of fibre motion is shown in Figure 5.15, where a rigid fibre is shown to flow into the slot after first contacting the upstream channel and slot corner. For small $U/U_w$ ratios and long rigid fibres, the fibres are swept out of the slot. The larger the recirculation in the slot and the smaller the exit layer height, the easier it is for the fibres to be swept out of the slot. In Figure 5.16 a 3 mm rigid fibre, using the flow of CFD1, approaches the slot, hits the upstream channel and slot corner, and rotates. The large recirculation in the upstream portion of the slot helps to push the leading end of the fibre out of the slot, which is then swept away into the main channel flow. Flexible fibres do not exhibit this behaviour since they tend to bend with the streamlines and are less prone to be affected by the mainstream flow above the exit layer. Figure 5.17 shows a flexible kraft fibre flowing through the slot with the same initial conditions as the rigid fibre of Figure 5.16. The slight overlap between the fibre and the channel and slot walls, shown in the figures, is a consequence of the fibre and wall contact model.

Category B type motion is observed for short fibres. Figure 5.18 shows a 1 mm rigid fibre entering the slot without making wall contact.

Category C type motion is observed for most fibres that enter the slot. Figure 5.19 shows this type of motion. As with Category A type motions, the fibres usually pass into the slot. However, for small $U/U_w$ ratios long rigid fibres do not pass through, but are swept away by the main channel flow, as depicted in Figure 5.16. Again, flexible long fibres do not exhibit this behaviour, but rather, enter into the slot.

Category D type motion is generally observed for fibres that do not enter the slot. Longer fibres usually staple the slot if they first contact the downstream channel wall, as shown in Figure 5.20. Short fibres rotate and either enter the slot at higher $U/U_w$ ratios (Figure 5.21), or partially enter the slot and then get swept into the channel flow for smaller $U/U_w$ ratios (Figure 5.22).

Clearly, Category E type motion is only observed for the trivial case where the fibres do not enter the slot.

In all cases, the effect of fibre flexibility plays an important role in the ability of fibres to
Figure 5.15: Category A type motion for a rigid fibre entering the slot.

Figure 5.16: Category A type motion for a rigid fibre swept away by the mainstream flow.

Figure 5.17: Category A type motion for a flexible fibre entering the slot.
Figure 5.18: Category B type motion for a short fibre entering the slot without wall contact.

Figure 5.19: Category C type motion for a rigid fibre entering the slot.

Figure 5.20: Category D type motion for a long fibre stapling the slot.
Figure 5.21: Category D type motion for a short fibre entering the slot.

Figure 5.22: Category D type motion for a short fibre pivoting into the slot then being swept by the mainstream flow.

pass through the slot. As mentioned previously, flexible fibres are more apt to bend with the streamlines, which makes it easier for them to pass into and through the slot. A quantitative study of fibre passage will be presented in the next section.

5.5 Fibre Motion Simulations: Quantitative Analysis

As mentioned above, in all of the simulations of fibre motion, the fibre is seeded in the channel with its mid-point 10 mm upstream of the upstream corner of the channel and slot and the fibre is oriented horizontally. The distance from the lower channel wall to the initial fibre starting point is defined as $H_f$. The height of the exit layer at this point is defined as $H_{el}$ (Figure 5.13).

The fibre motion simulation results are compared to the experimental work of
Kumar (1991), as described in Section 5.1. Summarizing, Kumar measured the passage ratio, defined below, for varying $U/U_u$ ratios and for four fibre types:

- **F1. nylon**: $L = 1\,\text{mm}, EI = 329 \times 10^{12}\,\text{Nm}^2$;
- **F2. rayon**: $L = 1\,\text{mm}, EI = 3.1 \times 10^{12}\,\text{Nm}^2$;
- **F3. nylon**: $L = 3\,\text{mm}, EI = 329 \times 10^{12}\,\text{Nm}^2$;
- **F4. kraft**: $L = 3.6\,\text{mm}, EI = 4.4 \times 10^{12}\,\text{Nm}^2$.

In all of the simulations, the fibre diameter is assumed to be 30 $\mu\text{m}$. The simulations show that a critical fibre starting height, $H_{fmt}$, determines fibre passage. In all cases, fibres initiated above $H_{fmt}$ do not enter the slot while those initiated at or below $H_{fmt}$ do enter the slot. Table 5.2 gives $H_{fmt}$ for the four different fibres and the seven flows, CFD1 to CFD7, discussed previously. As expected, the results show that shorter and more flexible fibres enter the slot at higher $H_f$ values than stiffer and longer ones. For slot to channel velocity ratios of 0.5 and greater the ratio

<table>
<thead>
<tr>
<th>Flow</th>
<th>$U/U_u$</th>
<th>$H_f$ for F1</th>
<th>$H_{fmt}$ for F2</th>
<th>$H_{fmt}$ for F3</th>
<th>$H_{fmt}$ for F4</th>
</tr>
</thead>
<tbody>
<tr>
<td>CFD1</td>
<td>0.24</td>
<td>0.16</td>
<td>0.16</td>
<td>0.01</td>
<td>0.11</td>
</tr>
<tr>
<td>CFD2</td>
<td>0.37</td>
<td>0.23</td>
<td>0.24</td>
<td>0.11</td>
<td>0.17</td>
</tr>
<tr>
<td>CFD3</td>
<td>0.50</td>
<td>0.30</td>
<td>0.31</td>
<td>0.15</td>
<td>0.22</td>
</tr>
<tr>
<td>CFD4</td>
<td>0.67</td>
<td>0.41</td>
<td>0.43</td>
<td>0.20</td>
<td>0.29</td>
</tr>
<tr>
<td>CFD5</td>
<td>0.83</td>
<td>0.51</td>
<td>0.52</td>
<td>0.24</td>
<td>0.35</td>
</tr>
<tr>
<td>CFD6</td>
<td>1.00</td>
<td>0.61</td>
<td>0.62</td>
<td>0.28</td>
<td>0.41</td>
</tr>
<tr>
<td>CFD7</td>
<td>1.11</td>
<td>0.66</td>
<td>0.68</td>
<td>0.32</td>
<td>0.46</td>
</tr>
</tbody>
</table>

Table 5.2: $H_f$ determined by the fibre motion simulations. $H_{el}$ and $H_{fmt}$ are in mm.
of \( H_{\text{fin}}/H_{\text{EL}} \) is constant for each fibre type.

As suggested by Gooding and Kerekes (1989), only fibres in the exit layer are candidates for passage through the slot which means that \( H_{\text{fin}} \) must be less than or equal to \( H_{\text{EL}} \). This is true in all cases of the simulations in which the initial fibre orientation is horizontal. The passage ratio is defined as

\[
P = \frac{C_{\text{avg}}}{C_{\text{uavg}}}
\]

where \( C_{\text{avg}} \) is the average fibre concentration in the slot and \( C_{\text{uavg}} \) is the average fibre concentration in the mainstream flow upstream of the slot. A passage ratio of 1.0 is similar to a "T" junction where the mass fraction, or concentration, of fibres in one stream is equivalent to the concentration of the other. Passage ratios less than 1.0 define situations where it is more favourable for the fibres to remain in the mainstream channel flow. Passage ratios greater than 1.0 depict situations where it is more favourable for the fibre to enter the slot.

As mentioned previously, Gooding (1986) and Gooding and Kerekes (1989) suggested that two mechanisms affect screening: the "wall effect" and the "entry effect" or "turning effect". It was pointed out that since only fibres in the exit layer are candidates for entering a screen slot, the concentration of fibres in the exit layer compared to the overall upstream channel concentration will affect the passage ratio. If it is assumed that the "wall effect" and "turning effect" are the two mechanisms which fractionate fibres, then the passage ratio can be separated into two components as follows (Gooding (1986))

\[
P = P_w P_t.
\]

Here \( P_w \) represents the "wall effect" passage ratio defined as the average fibre concentration in the exit layer divided by the average fibre concentration in the mainstream flow upstream of the slot,

\[
P_w = \frac{C_{\text{ELavg}}}{C_{\text{uavg}}}
\]

and \( P_t \) represents the "turning effect" passage ratio defined in a similar way to \( P_w \) as

\[
P_t = \frac{C_{\text{avg}}}{C_{\text{ELavg}}}.
\]

It should be emphasized that to conserve fibre mass, \( P_t \) must be less than or equal to 1.0.

For a fair comparison of passage ratio between the simulations and Kumar's work, a fibre concentration profile in the exit layer must be known. The fibre motion simulation software has
no mechanism to determine this concentration gradient. Recently, Olsen (1996) measured the concentration gradient of rigid nylon fibres of 1 mm, 2 mm and 3 mm lengths near the channel wall upstream of a slot in the same flow cell that was used by Kumar and presented the following correlation for fibre concentration as a function of the vertical distance from the channel wall,

$$\frac{C(H)}{C_{avg}} = \begin{cases} k_m H/L & \text{if } H/L < 1/k_m \\ 1 & \text{if } H/L \geq 1/k_m \end{cases} \quad (5.8)$$

where $C_{avg}$ is the average fibre concentration in the channel, $H$ is the vertical distance from the channel wall, and $k_m$ was found to have a value of 3.22. Unfortunately, the effect of varying the slot to channel velocity ratio was not considered.

The following analysis will use Olsen's (1996) fibre concentration observations to predict the theoretical passage ratio. In all of the equations, a unit depth in the global $Z$ direction is assumed. The fibre flow rate in the slot is given by

$$N_z = Q_z C_{avg} \quad (5.9)$$

where $Q_z$ is the slot volume flow rate of the fluid. Since only fibres in the exit layer are candidates for passage through the slot, the number of fibres flowing through the slot can be written as

$$N_z = \int_0^{H_EZ} C(y) u(y) p(y) dy \quad (5.10)$$

Here $u(y)$ is the flow speed in the $X$ direction as given in equation (5.1) and $p(y)$ is the probability of fibre passage. Using the results from the simulations, $p(y)$ can be written as

$$p(y) = \begin{cases} 1 & \text{if } y \leq H_{front} \\ 0 & \text{if } y > H_{front} \end{cases} \quad (5.11)$$

Combining equations (5.8), (5.9), (5.10) and (5.11) gives the passage ratio as

$$P = \frac{1}{Q_z C_{avg}} \int_0^{H_{front}} C(y) u(y) dy. \quad (5.12)$$

Olsen (1996) independently developed a similar expression and suggested that $H_{front} = H_{EZ}$, which implies that $P = 1.0$, i.e. all the fibres in the exit layer will enter into the slot.

Table 5.3 gives the fibre motion simulation results given in Table 5.2, the predicted passage ratios using equation (5.12), $P_{nm}$, the passage ratio as determined by Kumar, $P_k$, and the ratio of $P_{nm}/P_k$. Ideally, this ratio should be 1.0; however, the ratio is much less than 1.0 for all cases in Table 5.3.
Figure 5.23: Passage ratio as determined by Kumar (1991) and equation (5.14).

\( P_{k1} \): Kumar's passage ratio for fibre F1; \( P_{k2} \): Kumar's passage ratio for fibre F2;
\( P_{k3} \): Kumar's passage ratio for fibre F3; \( P_{k4} \): Kumar's passage ratio fibre F4;
\( P_{fl} \): passage ratio by equation (5.14) for fibre F1;
\( P_{f2} \): passage ratio by equation (5.14) for fibre F2;
\( P_{f3} \): passage ratio by equation (5.14) for fibre F3;
\( P_{f4} \): passage ratio by equation (5.14) for fibre F4;

If a constant concentration is assumed across the height of the channel, equation (5.12) reduces to

\[
P = \frac{Q_{fem}}{Q_s} = P_{CCG}
\]  

(5.13)

where \( Q_{fem} \) is the volume flow rate between the channel wall and \( H_{fem} \) and the subscript "CCG" is used for reference and denotes "Constant Concentration Gradient". For the assumed velocity profile of equation (5.1), equation (5.13) becomes

\[
P_{CCG} = \left( \frac{H_{fem}}{H_{EL}} \right)^{8/7}
\]  

(5.14)

In contrast to equation (5.12) with Olsen's (1996) concentration gradient, equation (5.14)
represents the case where $P_o = 1.0$ (instead of $P_t = 1.0$) and, therefore, $P_{ccg} = P_r$. The passage ratio determined by Kumar is plotted with the simulated passage ratio of equation (5.14) in Figure 5.23. As mentioned previously, the $H_{foe}/H_{el}$ ratio is relatively constant for all fibre types for slot to channel velocity ratios greater than or equal to 0.5. Therefore, $P_{ccg}$ would be expected to remain relatively constant for slot to channel velocity ratios greater than or equal to 0.5, as shown in the figure. For slot to channel velocity ratios in the region of 0.5 to 0.6, the experimental results are closest to the simulated ones.

While the same trends affecting the passage ratio with respect to fibre flexibility and fibre length were observed in the simulations as in the experimental results of Kumar (1991), quantitatively, the results did not agree. However, the fact that the simulations show increased fibre passage with increased flexibility, as do experimental studies, verifies the fact that the model can be utilized as a preliminary design tool for developing contoured screen plates. It should be emphasized that the fibre motion model predicts the "turning effect", $P_o$, and that other methods must be used to predict the "wall effect", $P_w$, which is directly related to the concentration gradient. The following section will discuss issues related to the concentration gradient and Kumar's (1991) experimental results.
<table>
<thead>
<tr>
<th></th>
<th>F1</th>
<th>F2</th>
<th>F3</th>
<th>F4</th>
</tr>
</thead>
<tbody>
<tr>
<td>CFD1</td>
<td>0.16</td>
<td>0.18</td>
<td>0.33</td>
<td>0.55</td>
</tr>
<tr>
<td>U/U_u = 0.24</td>
<td>0.23</td>
<td>0.27</td>
<td>0.51</td>
<td>0.53</td>
</tr>
<tr>
<td>H_{SL} = 0.23</td>
<td>0.30</td>
<td>0.36</td>
<td>0.68</td>
<td>0.53</td>
</tr>
<tr>
<td>CFD2</td>
<td>0.30</td>
<td>0.36</td>
<td>0.68</td>
<td>0.53</td>
</tr>
<tr>
<td>U/U_u = 0.37</td>
<td>0.41</td>
<td>0.50</td>
<td>0.83</td>
<td>0.60</td>
</tr>
<tr>
<td>H_{SL} = 0.32</td>
<td>0.51</td>
<td>0.58</td>
<td>0.92</td>
<td>0.63</td>
</tr>
<tr>
<td>CFD3</td>
<td>0.51</td>
<td>0.58</td>
<td>0.92</td>
<td>0.63</td>
</tr>
<tr>
<td>U/U_u = 0.50</td>
<td>0.61</td>
<td>0.64</td>
<td>0.98</td>
<td>0.65</td>
</tr>
<tr>
<td>H_{SL} = 0.41</td>
<td>0.66</td>
<td>0.64</td>
<td>1.00</td>
<td>0.64</td>
</tr>
</tbody>
</table>

Table 5.3: Fibre motion simulation results. $H_{SL}$ and $H_{fcm}$ are in mm and $P_{nm} = P_{OCG}$. 
5.6 Discussion

In the previous section it was shown that the passage ratio determined by using the results from the simulations and Olsen's (1996) concentration gradient was much less than the passage ratio observed by Kumar (1991). Here, it will be shown that if Olsen's concentration gradient is applied, then in certain circumstances, it is not possible to achieve the passage ratios reported by Kumar.

Since Olsen reported that the majority of the fibres tend to align parallel to the channel wall with the leading end pointing slightly away from the wall, it is fair to assume that, on the average, only fibres in the exit layer have a chance of entering the slot. Therefore, the maximum passage ratio that can be achieved with Olsen's concentration gradient is obtained by setting $H_{fem}=H_{EL}$ in equation (5.12). For reference, $P_{OCGM}$ is defined as this quantity, representing the Maximum passage ratio which can be obtained using Olsen's Concentration Gradient. Another interpretation of $P_{OCGM}$ is that all of the fibres are aligned exactly out-of-plane, such that they all enter the slot, but the concentration gradient near the channel wall remains unchanged - as reported by Olsen. Clearly, $P_{OCGM}$ is a function of the fibre length and the exit layer height, but not of fibre flexibility. Olsen plotted his version of $P_{OCGM}$ against the experimental results of Kumar and reported that the two were in good agreement for fibre length to slot width ratios of less than 2.0, but $P_{OCGM}$ was less than the passage ratios observed by Kumar for longer fibres.

Figure 5.24 shows similar trends, where $P_{OCGM}$ is compared to Kumar's data. Clearly, equation (5.12) will always under predict the passage ratio if $H_{fem} < H_{EL}$.

Olsen attributed the discrepancy between $P_{OCGM}$ and Kumar's data to the stapling effect. Gooding and Kerekes (1989) observed the trajectories of 3 mm fibres at a slot to channel velocity ratio of 1.0. It was observed that fibres even beyond the exit layer were found to staple the slot and accumulate and that the fibres entered the slot at the same rate as that of the accumulation. This mechanism of entry could account for passage ratios approaching 1.0 for all fibre lengths. It was also observed that contoured slot profiles, on which the fibres did not staple, did not predict a marked increase in the passage ratio.

The fact that $P_{OCGM}$ compares relatively well with experimental data suggests that the "turning effect" is not a primary factor in fibre passage. It was concluded by Olsen that the
trajectory effects of fibres passing through the slot are secondary compared to the effect of the upstream fibre concentration gradient. While it is possible for fibres, whose centre of mass is outside the exit layer to enter the slot if they are oriented with the leading end pointing towards the wall, Olsen showed that the majority of the fibres are oriented with their leading end pointing away from the wall. This orientation makes it even more difficult for them to pass through the slot. The question then remains: "How is it that essentially all fibres in the exit layer enter the slot?".

One explanation to the question stated above could be that stapling affects the flow in such a way that fibres above $H_{EL}$ are captured. However, this hypothesis does not seem very likely. Gooding and Kerekes (1992), found that the passage ratio decreases as feed concentration is increased and attributed this to stapling and plugging.

Figure 5.24: Passage ratio as determined by Kumar (1991) and equation (5.12) with $H_{font} = H_{EL}$.

* $P_k$: Kumar’s passage ratio for fibre F1; $P_{k2}$: Kumar’s passage ratio for fibre F2;
* $P_{k3}$: Kumar’s passage ratio for fibre F3; $P_{k4}$: Kumar’s passage ratio fibre F4;
* $P_{l1}$: $P_{OCGM}$ for a 1mm fibre; $P_{l2}$: $P_{OCGM}$ for a 3mm fibre; $P_{l3}$: $P_{OCGM}$ for a 3.6mm fibre.
Another factor affecting fibre passage is the out-of-plane orientation of fibres. Gooding and Kerekes (1992) showed that as the aperture velocity was increased the passage ratios of industrial screens increased at a greater rate for slotted screens than screens with holes. Since the greatest dimension of a slot opening of a screen plate is greater than the fibre length, all fibres that are perfectly out-of-plane would behave as point particles and all such oriented fibres would be expected to pass through the slot. Out-of-plane orientation would not have such an effect for a holed screen plate. Olsen measured the projected length (a measure of the degree of "out-of-planeness") of fibres near the channel wall and found that the projected length varied from 0.5 at the wall to a constant value of 0.8 at a distance of $L/3$ away from it. While the out-of-plane effects did increase the simulated passage ratio, the effects were not great enough to account for the question at hand.

Yet another explanation of the aforementioned question could be that the hydrodynamic force estimate is not applicable. Cox's equations were developed for small fluid to fibre relative velocities. This assumption breaks down at the slot and channel junction, where the Reynolds number based on the fibre diameter approaches values on the order of hundreds. Clearly, a Stokes flow is not valid in this situation. In performing the simulations, it was found that changing the ratio of the normal and tangential force component coefficients did not affect the fibre trajectory and fibre passage. However, it is reasonable to assume that the fibre will have an effect on the local flow near the slot and channel junction which could affect the fibre trajectory. This gives further credence to the need for a direct method to model fibre flow by using CFD.

The fibre motion simulation can only predict the trajectory, or "turning effects". However, at slot to channel velocity ratios of 0.5 and greater, the simulated results generally predict passage ratio "enhancement" due to fibre flexibility for a given fibre length as a similar ratio to Kumar's experimental data (i.e. $P_{nn}/P_k$ is constant for a given fibre length and slot to channel velocity ratio). These effects suggest that fibre flexibility enhances the "turning effect" by a small fraction, and that the "turning effect" is the primary effect which fractionates fibres of equal length based on their flexibility. Perhaps the most effective method to fractionate fibres based on flexibility is to first separate them according to length and then according to flexibility.
5.7 Summary

The results of the simulations presented show that fibre flexibility plays a role in pulp fibre fractionation in screening but the effect of fibre length is predominant. As reported in literature, the simulations show that flexible fibres tend to bend along the flow domain streamlines. This bending enables them to pass through an aperture, such as a slot, more easily than rigid fibres. The results of Table 5.2 show that fractionation based on fibre flexibility is easier to achieve for longer fibres than for shorter ones.

Qualitatively, the observed experimental results compared favourably with the results obtained from the simulations. The fibre motion model presented here is the first to theoretically show that fractionation based on fibre flexibility is possible. Furthermore, the constraint wall contact model, which was presented in the previous chapter, was shown to be robust and efficient, solving the flexible fibre motion equations in less than one half hour on a Pentium 133 MHz computer.

Although the results from the simulations could not directly predict the experimentally obtained passage ratios, similar trends were observed between the experimental and simulated data with respect to fractionation based on fibre flexibility and fibre length. It is expected that the fibre motion model will be useful in optimizing contoured screen wall geometries to achieve better fractionation based on fibre flexibility. Due to the industry's keen interest in separating fibres based on their flexibility, the model could be utilized to develop new processing techniques to achieve the desired goals.

The next chapter will propose a method to directly model the flow of a flexible fibre, in which CFD is used to determine the forces acting on the fibre. While a complete fibre motion model requires great computer resources, it is expected that the model will be utilized in specific fibre flow geometries, in which only a few time steps are required to obtain a solution, such as in a fibre flexibility measurement device.
Chapter 6

Automatic Three-Dimensional Mesh Generation

Although analytical and empirical estimates determining the flow of a fluid around cylindrical objects have been proposed and force estimates have been provided, little work has been done to model the flow around a three-dimensional, cylindrical, curved fibre using Computational Fluid Dynamics (CFD). The advantage of using CFD models is that flow patterns and stresses can be estimated for fibres with complex geometries. As well, most of the analytical work published to date is valid for Reynolds numbers, based on the fibre diameter, of less than one, while CFD is not limited by this restriction. The primary difficulty associated with using CFD for irregularly shaped three-dimensional objects is the human time and effort required to discretize the domain into a mesh or grid of regularly shaped three-dimensional cells or elements. Hu et al. (1992) developed a two-dimensional finite element model which simulates the motion of circular particles in a fluid. The extension of this method to three dimensions is straightforward with the limitation that an automatic three-dimensional mesh generating algorithm is required. The purpose of this chapter is to develop such an algorithm, which creates a tetrahedral mesh around a general, curved, three-dimensional fibre enclosed in a regular rectangular box, to overcome the aforementioned limitation. The meshing package developed here can be integrated with the methods proposed by Hu et al. (1992) to develop a direct fibre motion simulation model. While the integration is straightforward, the work effort is extensive and was deemed beyond the scope of this thesis. It is
expected that as computer speed and memory increase, it will become possible for a fully interactive solution using CFD and the equations of motion of a fibre to be used to solve the movement of a fibre in a shear flow.

The development of an automatic three-dimensional mesh generating algorithm will be discussed in this chapter. The mesh generation algorithm follows closely the work presented by Weatherill and Hassan (1994), however the generation of boundary and fibre nodes represents original work as do the mesh refinement procedures.

Of primary importance in the development of the mesh generating algorithm is the ability to automatically mesh the domain from the outside boundary to the fibre walls, with the stipulation that the fibre is a cylindrical three-dimensional body, of non-constant cross-section, curving in three dimensions. Robustness, speed and mesh quality are other important factors which influenced the development. Future applications of the automatic mesh generating algorithm coupled with a CFD package include modelling fibre motion in fibre processing equipment such as screens and the development of fibre measurement devices such as fibre flexibility measurement equipment.

6.1 The Three-Dimensional Automatic Mesh Generating Algorithm

The general procedure for the three-dimensional mesh generating algorithm follows the steps below.

*Three-Dimensional Automatic Mesh Generating Algorithm (3D-AMGA)*

1. Construct the initial rectangular three-dimensional domain consisting of 8 nodes and 5 tetrahedra.
2. Add boundary nodes on the 6 faces of the rectangular domain.\(^1\)
3. Add nodes on the fibre faces.\(^1\)
4. Generate the flow domain nodes which are internal to the outside boundary and external to the fibre and add them to the domain.\(^1\)
5. Check the aspect ratio of all the tetrahedra in the flow domain and add nodes at the

Nodes are added by the application of the Delaunay triangulation algorithm. A discussion of the algorithm is given in Section 6.2.
6. Smooth the mesh.
7. Add nodes centred along all the edges of the tetrahedra to create 10 node quadratic tetrahedral elements.
8. Create node tables for the triangles of each of the 6 boundary faces and the fibre faces and check the integrity of the fibre faces.
9. Save the data structure in the required format for the CFD package.
10. End the program.

The details of some of the more complicated algorithms will be given in the following sections.

6.2 The Delaunay Triangulation

According to Weatherill and Hassan (1994), Dirichlet (1850) first proposed a method to systematically decompose a domain into a set of packed convex polyhedra. For a given set of points, \( \{P_1, P_2, \ldots, P_n\} \), \( n \) regions can be formed where for each region, a point, \( p \), inside the region will be closer to one of the given points, \( P_i \), than to any other point in the given set, \( \{P_1, P_2, \ldots, P_n\} \). Mathematically, each of these regions, \( V_i \), can be expressed as (see Weatherill and Hassan (1994))

\[
V_i = \{P_j : |p - P_i| < |p - P_j|, \forall j \neq i\}. \tag{6.1}
\]

Weatherill and Hassan (1994) stated that the geometric construction is known as the Dirichlet tessellation or Voronoi diagram (Voronoi (1908)). It consists of packed non-overlapping convex polyhedra, called Voronoi regions spanning the entire domain. By joining all the point pairs that share a common Voronoi boundary, a Delaunay triangulation (Delaunay (1934)) is performed. The geometric construction is valid for \( N \)-dimensional Euclidean space.

A two-dimensional example of a Dirichlet tessellation and Delaunay triangulation is shown in Figure 6.1. The bold lines represent the Voronoi boundaries. By definition, these line segments lie half way between the two points on either side of the Voronoi regions. The points defining the outside boundary of the convex domain will have Voronoi regions of infinite size. By joining points that share Voronoi boundaries, the Delaunay triangulation is formed. The fine lines of Figure 6.1 represent the Delaunay triangulation. In \( N \) dimensions, each Delaunay triangle is a
Figure 6.1: Dirichlet tessellation (bold lines) and Delaunay triangulation (fine lines).

A simplex with \( N + 1 \) data points as vertices (Bowyer (1981)). A point that is equidistant from all of the \( N + 1 \) vertices forming a Delaunay triangle is the centre of a hypersphere passing through all of the vertices, called the Voronoi vertex (Weatherill and Hassan (1994)), and is located at the intersection of the \( N + 1 \) Voronoi boundaries. The Voronoi boundary will be a complex polygon lying in the \( N - 1 \) dimensional hyperplane. Of particular importance is the fact that, by definition, no hypersphere can contain a data point. This condition is referred to as the "in-circle criterion" (Weatherill and Hassan (1994)).

In two dimensions, the Voronoi regions will be convex polygons and the Delaunay triangulation will form triangles, whereas in three dimensions, the Voronoi regions will be convex polyhedra and the Delaunay triangulation will form tetrahedra. Since visual representations are easier to do in two dimensions than in three, the Delaunay triangulation algorithm will be
explained for a two-dimensional domain. Extrapolation of the algorithm to three dimensions is straightforward.

The algorithm for two dimensions given here follows closely the work presented by Bowyer (1981) and Watson (1981). Essentially, upon addition of a new point in the domain, Voronoi vertices, which are closer to the new point than their forming points, are deleted and new Voronoi regions are formed. The search for the Voronoi vertices to be deleted is performed by a tree search (see Schildt (1987)). The basic steps of the algorithm are presented here.

**Delaunay Triangulation Algorithm**

1. Build the initial data structure of $n$ points.
2. Add a new point, $P_{n+1}$.
3. Find one Voronoi vertex whose distance to $P_{n+1}$ is less than the distance to its forming points.
4. Create a list of vertices called $V_{\text{delete}}$ and add the vertex found in step three to the list. This list contains the Voronoi vertices which will be deleted.
5. Create another list of vertices, $V_{\text{visited}}$, which represents vertices visited in the tree search but not deleted.
6. Perform a tree search by using the neighbouring Voronoi vertices of the first vertex found. The tree search algorithm continues to search down a branch until a Voronoi vertex is found that should not be deleted. This vertex should be added to the list called $V_{\text{visited}}$. Thus, at the completion of the tree search, all the vertices that are to be deleted will be listed in $V_{\text{delete}}$ and all of the neighbours of those to be deleted will be listed in $V_{\text{visited}}$.
7. Check each neighbour of each Voronoi vertex of $V_{\text{delete}}$ to see if that neighbour is not a vertex that should be deleted. If it is not to be deleted then construct a new vertex formed by the old forming points of the vertex to be deleted, replacing the forming point not associated with the neighbour with the new point, $P_{n+1}$.
8. Update the neighbouring vertex entries for the vertices of $V_{\text{visited}}$ and create new neighbouring entries for the new vertices.
9. Rearrange the data as needed.

Table 6.1 presents the data structure developed for the two-dimensional example depicted
Table 6.1: Data structure for the Delaunay triangulation.

<table>
<thead>
<tr>
<th>Vertex</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>2-3</th>
<th>1-3</th>
<th>1-2</th>
</tr>
</thead>
<tbody>
<tr>
<td>V1</td>
<td>P1</td>
<td>P7</td>
<td>P2</td>
<td>V2</td>
<td>0</td>
<td>V10</td>
</tr>
<tr>
<td>V2</td>
<td>P2</td>
<td>P7</td>
<td>P9</td>
<td>V7</td>
<td>V3</td>
<td>V1</td>
</tr>
<tr>
<td>V3</td>
<td>P2</td>
<td>P9</td>
<td>P3</td>
<td>V4</td>
<td>0</td>
<td>V2</td>
</tr>
<tr>
<td>V4</td>
<td>P9</td>
<td>P8</td>
<td>P3</td>
<td>V5</td>
<td>V3</td>
<td>V7</td>
</tr>
<tr>
<td>V5</td>
<td>P3</td>
<td>P8</td>
<td>P4</td>
<td>V6</td>
<td>0</td>
<td>V4</td>
</tr>
<tr>
<td>V6</td>
<td>P5</td>
<td>P4</td>
<td>P8</td>
<td>V5</td>
<td>V8</td>
<td>0</td>
</tr>
<tr>
<td>V7</td>
<td>P7</td>
<td>P8</td>
<td>P9</td>
<td>V4</td>
<td>V2</td>
<td>V8</td>
</tr>
<tr>
<td>V8</td>
<td>P5</td>
<td>P8</td>
<td>P7</td>
<td>V7</td>
<td>V9</td>
<td>V6</td>
</tr>
<tr>
<td>V9</td>
<td>P6</td>
<td>P5</td>
<td>P7</td>
<td>V8</td>
<td>V10</td>
<td>0</td>
</tr>
<tr>
<td>V10</td>
<td>P6</td>
<td>P7</td>
<td>P1</td>
<td>V1</td>
<td>0</td>
<td>V9</td>
</tr>
</tbody>
</table>

in Figure 6.1 and represents the first step of the two-dimensional algorithm. For each Voronoi vertex, there is a corresponding triangle composed of three forming points and for each side of the triangle there is a neighbouring triangle with its Voronoi vertex. For example, the Voronoi vertex, V7, is formed by the three points, P7, P8 and P9. A convention has been chosen to order the forming points in a counter-clockwise direction. The neighbouring Voronoi vertices are determined as described below. The first neighbour is the one which shares the line segment formed by the second and third points of the vertex. In this example, the second and third points of V7 are P8 and P9. The other vertex which shares the line segment formed by P8 and P9 with V7 is the Voronoi vertex V4. Similarly, the second neighbour is determined by the first and third points, P7 and P9, which corresponds to V2, and the third neighbour is determined by the first and second points, P7 and P8, which corresponds to V8. In cases where a line segment is on the boundary, the neighbour is given the value 0.

The second step of the algorithm is to add a point inside the domain. The addition of a new point, P10, is depicted in Figure 6.2. The next step is to find a Voronoi vertex whose distance to the new point is less than the distance to its forming points. Voronoi vertex, V1, is one such vertex. The lists, $V_{\text{delete}}$ and $V_{\text{visited}}$, are created, with $V_{\text{delete}} = \{V1\}$. Now the tree search of step 6 is performed to find the other vertices that need to be deleted, i.e. those whose distance
to P10 is less than the distance to their forming points. Proceeding with the tree search, the first neighbour of V1, V2, is checked and from Figure 6.2 it can be seen that P10 lies inside the circle formed by the forming point of V2 and therefore V2 is added to $V_{delete}$. The tree search drops down a level and the neighbouring vertices of V2 are searched next. The first neighbour of V2 is V7. From Figure 6.2 it is evident that P10 does not lie inside the circle formed by the forming points of V7, so V7 is added to $V_{marked}$ and the tree search is continued along the same level, i.e. the second neighbour of V2, V3, is checked. Again from Figure 6.2 it can be seen that the circle formed by the forming points of V3 does not contain P10, so V3 is added to $V_{marked}$ and the tree search proceeds to the third neighbour of V2, V1. Since V1 was already marked for deletion the tree search continues one level higher, to the second neighbour of V1. Since the second neighbour of V1 is zero, the tree search continues at the same level and the third neighbour, V10

Figure 6.2: Applying the in-circle criterion for point P10.
is checked. Since $P_{10}$ does lie within the circle formed by the forming points of $V_{10}$, the tree search drops down a level and now checks the neighbours of $V_{10}$. The first neighbour of $V_{10}$ is $V_1$, which, as before, has been marked for deletion and the tree search continues along the level to the second neighbour of $V_{10}$. Since the second neighbour of $V_{10}$ is zero, the search continues to the third neighbour, $V_9$. Again, from Figure 6.2 it is evident that $P_{10}$ does not lie inside the circle formed by the forming points of $V_9$, so $V_9$ is added to $V_{\text{visited}}$ and the tree search continues one level higher. Since all three neighbours of $V_1$ have been checked the tree search is done. The final result of the search is as follows:

$$V_{\text{delete}} = \{V_1, V_2, V_{10}\} \text{ and } V_{\text{visited}} = \{V_7, V_3, V_9\}.$$ 

Now the new Voronoi vertices are constructed using the non-deleted members of the neighbours of $V_{\text{delete}}$. The first neighbour of $V_1$ is $V_2$. Since $V_2$ is to be deleted, no new vertex is formed for this neighbour and the second neighbour is checked. The second neighbour is 0 which means that a new vertex should be formed using the old forming points of $V_1$, but replacing the forming point that does not correspond to this neighbour with the new point, $P_{10}$. The forming points of $V_1$ are $\{P_1, P_7, P_2\}$ and since the first and third points of $V_1$ determine the second neighbour, the second point of $V_1$, $P_7$, is replaced by $P_{10}$. Therefore, the first new Voronoi vertex is formed by the points: $V_{N1} = \{P_1, P_{10}, P_2\}$. The third neighbour of $V_1$ is $V_{10}$ which is to be deleted so the next vertex of $V_{\text{delete}}$, $V_2$ is now studied. The first neighbour of $V_2$ is $V_7$, which is not to be deleted, and thus the second new Voronoi vertex is formed by replacing the first point of $V_2$ by the new point to give: $V_{N2} = \{P_{10}, P_7, P_9\}$. The second neighbour of $V_2$ is $V_3$ which leads to a new vertex, $V_{N3} = \{P_2, P_{10}, P_9\}$ and the third neighbour of $V_2$ is $V_1$ which is to be deleted and no new vertex is formed. Finally, the last member of $V_{\text{delete}}$, $V_{10}$, is analysed. The first neighbour of $V_{10}$ is $V_1$, which is to be deleted and no new vertex is formed. The second neighbour of $V_{10}$ is zero, so it too forms a new Voronoi vertex, $V_{N4} = \{P_6, P_{10}, P_1\}$. The third neighbour of $V_{10}$ is $V_9$, which is not to be deleted, and thus the fifth and final new Voronoi vertex is formed by replacing the third point of $V_{10}$ by the new point to give: $V_{N5} = \{P_6, P_7, P_{10}\}$. The new Dirichlet tessellation and Delaunay triangulation is shown in Figure 6.3 and the new data structure is given in Table 6.2.

The final steps involve creating the new neighbouring vertex entries for the new vertices, updating the neighbouring vertex entries for the visited Voronoi vertices, and rearranging the data.
structure to use computer memory efficiently.

In some cases, the new point being inserted is located exactly on one of the circles of the forming points, which means that the new point is the same distance to the Voronoi index as the forming points. Here, the deletion of the Voronoi index is arbitrary. The addition of a new point on the boundary of the domain results in the formation of triangles with zero area. These triangles can easily be removed without affecting the logic of the data structure, as long as the associated neighbouring vertices are updated accordingly.
Table 6.2: Data structure for the new Delaunay triangulation with P10.

The extrapolation of the Delaunay algorithm to three dimensions is straightforward. Each Voronoi vertex will have four forming points and four neighbours. The forming points of the tetrahedra, or nodes, are ordered as shown in Figure 6.4. The volume of the tetrahedra can be calculated by (see Huebner and Thornton (1982))

$$V = \frac{1}{6} \det \begin{bmatrix} 1 & x_1 & y_1 & z_1 \\ 1 & x_2 & y_2 & z_2 \\ 1 & x_3 & y_3 & z_3 \\ 1 & x_4 & y_4 & z_4 \end{bmatrix}$$ (6.2)

where "det" signifies the determinant of the matrix and $x_i, y_i, z_i$, $i = 1,2,3,4$ are the co-ordinates of the nodes, or forming points, of the tetrahedra. With this convention, the integrity of each tetrahedron can be checked by making sure the volume is greater than zero.

An important feature of the Delaunay triangulation algorithm is that the only floating point calculations required to be performed in the algorithm are those associated with the in-circle criterion distance calculation. This means that any errors associated with the mesh generating algorithm must be attributed to the in-circle check, and this fact makes error tracking easy. In the automatic mesh generating software, a tolerance value was used to check the in-circle criterion.
If the absolute value of the distance from a Voronoi vertex to the new point being added minus the distance from a Voronoi vertex to its forming points is less than the tolerance, then the Voronoi vertex is included in the list for deletion and is marked accordingly. After the generation of the new tetrahedra the integrity of all the new tetrahedra is checked and if any of the newly formed tetrahedra have a volume less than zero, then the Voronoi vertex in question is taken out of the deleted list, added to the visited list, and new tetrahedra are generated afresh. This procedure, suggested by Weatherill and Hassan (1994), was found to be very effective.

6.3 The Rectangular Three-Dimensional Domain

To use the Delaunay triangulation algorithm, an existing convex, triangulated domain is required. The mesh generating algorithm is initiated with a rectangular box consisting of 8 nodes and 5 tetrahedra (see Figure 6.5). The user enters the size of the box by specifying the co-ordinates of node 7. In the example depicted here the co-ordinates of node 7 were chosen to be (1,2,1).

The data structure of the Delaunay triangulation is presented in Table 6.3. The Vector and Matrix class templates, described in Appendix A, are used to store the data within the algorithm. The primary objects used in the algorithm for data storage are listed here:

- `Matrix<long> T_node(num_T,4);`
Figure 6.5: Initial three-dimensional rectangular domain.

- `Matrix<long> T_near(num_T,4);`
- `Matrix<double> N_coord(num_N,3);`
- `Matrix<double> T_coord(num_T,3);`
- `Vector<double> T_rad2(num_T);`

The long type variables `num_T` and `num_N` represent the number of tetrahedra and nodes in the domain, respectively. To avoid constant reallocation of memory, the `Matrix` and `Vector` objects declared here usually have more rows than `num_T` or `num_N`. The `T_node` object stores the forming nodes for the tetrahedra of the domain. `T_near` stores the corresponding neighbouring tetrahedra. `N_coord` stores the x, y, z co-ordinates of the forming nodes. To keep the analysis efficient, the co-ordinates of each Voronoi vertex are stored in the object `T_coord` and the `Vector` object, `T_rad2`, stores the square of the radii of the spheres formed by the forming nodes of the tetrahedra. It should be emphasized that the square of the distance calculation is sufficient for the in-circle criterion check and is therefore used to keep the algorithm as efficient as possible.

The next step in the algorithm is to add the outside boundary nodes on the 6 faces of the rectangular domain by the application of the Delaunay triangulation algorithm. Figure 6.6 shows
the rectangular domain of Figure 6.5 with the addition of outside boundary nodes. In this case, the node spacing was set at 4 nodes per unit length. Figure 6.6 represents 162 nodes and 392 tetrahedra. To keep the grids of the boundary faces regular, the spacing between the nodes on the outside boundary faces is kept constant. Although a regular mesh is relatively simple to create, regular meshes pose problems for the Delaunay triangulation algorithm. The reason for this is that the addition of nodes in a regular pattern creates situations where the nodes are found to be located exactly on the spheres formed for the in-circle criterion. Although this should not pose a problem for the Delaunay triangulation algorithm presented in Section 6.2, it does create a problem in the practical application of the algorithm due to the floating point inaccuracies. The procedure used to combat this problem, described at the end of Section 6.2, alleviated the difficulties.

### 6.4 Generation of the Fibre Nodes

The third step of the automatic mesh generating algorithm is the generation and addition of the nodes which define the fibre. Since it is expected that in the future the mesh generating algorithm will be coupled with fibre motion software, the fibre node generation procedure was designed in such a way as to be fully automatic, but robust enough to handle fibre geometries of any shape. A general fibre shape is depicted in Figure 6.7. The fibre consists of three faces, end face 1 and end face 2 which are located at the fibre ends, and the cylinder face which defines the body of the fibre. The fibre node generation procedure reads a data file which defines the centre points of a fibre with a three-dimensional shape. Ellipses are formed around the centre points to create the

<table>
<thead>
<tr>
<th>T No.</th>
<th>Node 1</th>
<th>Node 2</th>
<th>Node 3</th>
<th>Node 4</th>
<th>2,3,4</th>
<th>1,3,4</th>
<th>1,2,4</th>
<th>1,2,3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>5</td>
<td>2</td>
<td>4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>5</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>0</td>
<td>5</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>7</td>
<td>8</td>
<td>5</td>
<td>0</td>
<td>5</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>3</td>
<td>7</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>5</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>4</td>
<td>7</td>
<td>5</td>
<td>1</td>
<td>3</td>
<td>4</td>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>
Figure 6.6: Rectangular domain with outside boundary nodes.

cylindrical body of the fibre. The ability to vary the radius of the fibre at the different centre points is easy to implement but was not done. The procedure for generating the nodes on the fibre will be discussed in this section and examples will be given, where appropriate. A summary of the procedure is given here.

Fibre Node Generation Procedure (Step 3 of 3D-AMGA)

1. Read the data file containing the centre points of the fibre and store the points in a list.
2. Add the first centre point to the domain.
3. Calculate the equation of the circle of end face 1 and add nodes on the face.
4. Calculate the ellipses located at the centre points internal to the fibre and add nodes on the cylinder face.
5. Calculate the equation of the circle of end face 2 and add nodes on the face.
6. End the procedure.

The first step is straightforward. Consider the list of \( n_c \) fibre centre points to be \( \{ C_1, C_2, \ldots, C_{n_c} \} \). Let \( \mathbf{R}_i \) represent a vector starting at the origin and terminating at the \( i \)-th centre point. The next step of the procedure requires the addition of \( C_i \) to the domain by applying the
End Face 1

Cylinder Face

End Face 2

Figure 6.7: General fibre shape.

Delaunay triangulation algorithm. Now, the points on end face 1 need to be generated, i.e. step 3.

The vector normal to end face 1 is given by

\[ \mathbf{n}_{c1} = \frac{\mathbf{R}_{c1} - \mathbf{R}_{c2}}{|\mathbf{R}_{c1} - \mathbf{R}_{c2}|} \tag{6.3} \]

A vector from the origin to any point on the circle of end face 1 is determined as follows,

\[ \mathbf{R}_1 = \mathbf{R}_{c1} + R_{\text{fibre}} \mathbf{t}_{c1} \tag{6.4} \]

where the tangential vector, \( \mathbf{t}_{c1} \), is determined by solving

\[ \mathbf{n}_{c1} \cdot \mathbf{t}_{c1} = 0 \tag{6.5} \]

and \( R_{\text{fibre}} \) is the radius of the fibre. Since equation (6.4) represents an infinite number of points, a starting point needs to be determined. A simple algorithm to determine a unique value for \( \mathbf{t}_{c1} \) was developed. Define

\[ \mathbf{b}_{c1} = \mathbf{n}_{c1} \times \mathbf{t}_{c1} \tag{6.6} \]

and

\[ \phi = 2\pi/n_{\text{arc}} \tag{6.7} \]

where \( n_{\text{arc}} \) is the number of points located around each centre point of the fibre on the cylinder face. Nodes on the circle end face, \( \mathbf{R}_{1j} \), can now be calculated by

\[ \mathbf{R}_{1j} = \mathbf{R}_{c1} + R_{\text{fibre}} (\cos(\phi \cdot j) \mathbf{t}_{c1} + \sin(\phi \cdot j) \mathbf{b}_{c1}), \quad j = 0, 1, 2, ..., n_{\text{arc}} - 1. \tag{6.8} \]

The nodes are added to the domain by implementing the Delaunay triangulation algorithm. When \( n_{\text{arc}} \) is large, node addition needs to take place on the end face itself, not just on the end face circle, to prevent the generation of very skewed elements. This is done by applying equation (6.8)
with $R_{\text{fibre}}$ replaced by a smaller value and a smaller range for $j$ is used. Figure 6.9 shows the effects of node generation on the end faces. Figure 6.9A was formed using $n_{\text{arc}} = 12$ and $R_{\text{fibre}}=0.1$ units whereas Figure 6.9B was also formed with $n_{\text{arc}} = 12$ and $R_{\text{fibre}} = 0.1$ units, but equation (6.8) was applied three more times with 10, 8 and 6 nodes and radii of 0.066, 0.043 and 0.029 units, respectively. Figure 6.9C shows a typical mesh for a fibre end face that was used in practice. Nodes on end face 2 are generated in a similar fashion in step 5 of the procedure.

The next step in the fibre node generation procedure, step 4, is to add the nodes located on the ellipses of the internal centre points of the fibre, i.e. the centre points $C_2, C_3, ..., C_{nc-1}$. To make the mesh on the cylinder face as regular as possible, the fibre node generation was developed in such a way that the nodes of each centre point line up with all of the other nodes of the other centre points. The normal vector of each ellipse is given by the following,

$$\mathbf{n}_{ci} = \frac{\mathbf{a}_i + \mathbf{a}_{i+1}}{|\mathbf{a}_i + \mathbf{a}_{i+1}|}, \quad i = 2, 3, ..., nc - 1$$  \hspace{1cm} (6.9)
where
\[ a_i = \frac{R_{c(i-1)} - R_{ci}}{|R_{c(i-1)} - R_{ci}|}, \quad i = 2, 3, ..., n_c. \] (6.10)

It should be emphasized that
\[ a_2 = n_c \]
as a consequence of equation (6.10). Vectors \( t_{c2} \) and \( b_{c2} \) are defined as follows
\[ t_{c2} = t_{c1}, \quad b_{c2} = b_{c1}. \]

The equation to generate the nodes on the \( i \)-th ellipse is given by
\[ R_y = R_{ci} + R_{fibre}\left( r_{ij} - \frac{r_{ij} \cdot n_{ci}}{a_i \cdot n_{ci}} a_i \right), \quad i = 2, 3, ..., n_c - 1, \quad j = 0, 1, ..., n_{arc} - 1 \] (6.11)
where the vector, \( r_{ij} \), is defined as
\[ r_{ij} = \cos(\phi \cdot j)t_{ci} + \sin(\phi \cdot j)b_{ci}, \quad i = 2, 3, ..., n_c - 1, \quad j = 0, 1, ..., n_{arc} - 1. \] (6.12)

To keep the nodes lined up along the cylinder face, the vectors \( t_{ci} \) and \( b_{ci} \) are rotated in the plane defined by \( a_{i-1} \) and \( a_i \) through an angle, \( \theta_i \), defined by
\[ \theta_i = a_i \cdot a_{i+1}, \quad i = 2, 3, ..., n_c - 1. \] (6.13)

The rotation proceeds as follows
\[ \begin{align*}
    t_{ci(i+1)} &= A_i t_{ci}, \\
    b_{ci(i+1)} &= A_i b_{ci},
\end{align*} \quad i = 2, 3, ..., n_c - 1 \] (6.14)
where the rotation matrix, \( A_i \in R^{3 \times 3} \), is given by (see McCarthy (1990))
\[ A_i = (I - B_i)^{-1}(I + B_i) \] (6.15)
where \( I \in R^{3 \times 3} \) is the identity matrix and \( B_i \in R^{3 \times 3} \) is determined by the components of the rotation (screw) vector (see McCarthy (1990))
\[ v_n = \frac{a_i \times a_{i+1}}{\cos \theta_i + 1} \] (6.16)
i.e.
\[ B_i = \begin{bmatrix}
0 & -v_{n1} & v_{n2} \\
v_{n1} & 0 & -v_{n3} \\
-v_{n2} & -v_{n3} & 0
\end{bmatrix}. \] (6.17)

Figure 6.10 shows an example plot of the fibre cylinder face mesh plot. The fibre was generated with \( n_c = 96 \) and \( n_{arc} = 12 \). The last step of the fibre generation procedure is to generate the nodes on end face 2. This step is analogous to the generation of the nodes on end face 1. At this point, all of the boundary nodes and fibre nodes have been defined for the flow
domain. Step 4 of the three-dimensional automatic mesh generating algorithm requires the generation of the nodes internal to the flow domain. The procedure for this step will be discussed in the following section.

6.5 Generation of the Internal Flow Domain Nodes

The procedure to generate the internal flow domain nodes, step 4 of the three-dimensional automatic mesh generating algorithm, follows directly from the work presented by Weatherill and Hassan (1994) and will be briefly described in this section. The procedure follows the following steps (see Weatherill and Hassan (1994)).

*Internal Flow Domain Node Generation Procedure (Step 4 of 3D-AMGA)*

1. Compute the node distribution function for each boundary node, \( r_i = (x, y, z) \), i.e. for node 0

\[
dp_0 = \frac{1}{n_{\text{boundary}}} \sum_{i=1}^{n_{\text{boundary}}} |r_i - r_0|
\]  

(6.18)

where it is assumed that node 0 is surrounded by \( n_{\text{boundary}} \) nodes.

2. Initialize the number of interior nodes created, \( n_{\text{int}} = 0 \).
3. For all tetrahedra within the domain, not including those inside the fibre:
   a) Define a prospective node, point \( Q \), to be at the centroid of the tetrahedron;
   b) Derive the point distribution, \( dp_Q \), for the point \( Q \), by interpolating the point
      distribution function from the nodes of the tetrahedron, \( dp_j \), \( j = 1, 2, 3, 4 \);
   c) Compute the distances, \( d_j \), \( j = 1, 2, 3, 4 \), from the prospective point, \( Q \), to each of
      the four nodes of the tetrahedron:
      
      \[
      \text{if } d_j < \alpha dp_Q \text{ for any } j = 1, 2, 3, 4 \text{ then reject the point and return to the beginning of step 3}
      \]
      \[
      \text{if } d_j > \alpha dp_Q \text{ for all } j = 1, 2, 3, 4 \text{ then compute the distance } s_i, \ i = 1, ..., n_{int} \text{ from the }
      \]
      \[
      \text{prospective point } Q, \text{ to other points to be inserted, } P_k, \ k = 1, ..., n_{int} \]
      \[
      \text{if } s_i < \beta dp_j \text{ then reject the point and return to the beginning of step 3}
      \]
      
      \[
      \text{else accept the point } Q \text{ for insertion by the Delaunay triangulation algorithm, increase } n_{int} \text{ by 1 and include } Q \text{ in the list } P_k, \ k = 1, ..., n_{int} ;
      \]
      d) Assign the interpolated value of the point distribution function, \( dp_Q \), to the new
         node;
      e) Go to the next tetrahedra.
   4. If \( n_{int} = 0 \) then go to step 6.
   5. Perform the Delaunay triangulation of the derived points, \( P_k, \ k = 1, ..., n_{int} \) and go to
      step 2.
   6. End the procedure.

The values of \( \alpha \) and \( \beta \) were normally kept at one; however, for complicated fibre
geometries, the generated mesh had better geometric properties for \( \alpha \) in the range of 0.7 to 0.9.
The computer time required to generate 10 000 internal nodes was on order of 10 minutes on a
Pentium PC, 133MHz. Often, the resulting mesh had badly skewed tetrahedra. The following
section will discuss the methods used to refine, or smooth, the mesh.

6.6 Mesh Refinement

After the generation of internal nodes, the flow domain is composed of many tetrahedra. Some of
these tetrahedra are very skewed and could therefore introduce errors in the resulting flow
The amount of skew of a tetrahedra is often referred to as the aspect ratio. A number of methods to measure the aspect ratio of tetrahedra have been proposed in literature, see for example Baker (1989), Cavendish et al. (1985). The method to calculate the aspect ratio, \( \gamma \), in this work is based on the definition proposed by Parthasarathy et al. (1993) and is calculated as follows

\[
\gamma = \frac{\left( \sum_{i=1}^{6} S_i^2 / 6 \right)^{3}}{V} = \frac{S_{rms}^3}{V}
\]  

where \( S_i \) is the length of the \( i \)-th side of the tetrahedron, \( S_{rms} \) is the root-mean-square of the six sides of the tetrahedron and \( V \) is the volume of the tetrahedron. The best value for \( \gamma \) can be calculated by setting all sides equal to one. The resulting tetrahedron will have a volume of 0.11785 cubic units. The aspect ratio calculation is normalized by multiplying the right hand side of equation (6.19) by 0.11785 to give the aspect ratio definition used in this study as

\[
\gamma = 0.11785 \cdot \frac{\left( \sum_{i=1}^{6} S_i^2 / 6 \right)^{3}}{V}.
\]

Generally, tetrahedra with \( \gamma < 5 \) are assumed to be very good, while those with \( 5 < \gamma < 10 \) are assumed to be acceptable. A negative value for \( \gamma \) represents an inverted tetrahedron, whose volume is negative, and is unacceptable. Steps 5 and 6 of the three-dimensional automatic mesh generating algorithm were developed to refine the mesh and will be discussed here. In Step 5, nodes are added at the centroids of tetrahedra that have a large value for \( \gamma \). This technique is unique and has proven to be very successful when applied to the geometry at hand. It is expected that this technique would work well in other situations. Step 6 also represents an original formulation and it too has the capability to significantly improve the mesh quality.

**Procedure for Tetrahedron Centroid Node Addition for Skewed Element (Step 5 of 3D-AMGA)**

1. Find \( \gamma \) for each tetrahedron in the flow domain (i.e. not inside the fibre).
2. If \( \gamma > \gamma_{max} \) then add a node at the centroid of the tetrahedron.
3. End procedure when all tetrahedra have \( \gamma < \gamma_{max} \).

It was found that a value for \( \gamma_{max} \) in the range of 10 to 15 worked well.

The smoothing procedure uses an optimizing function to minimize \( \gamma \). The procedure is
given below.

**Mesh Smoothing Procedure (Step 6 of 3D-AMGA)**

1. Set counter, \( i \), to 1.
2. If \( i > \) the number of tetrahedra in the domain then end the procedure.
3. Calculate the aspect ratio for the \( i \)-th tetrahedron, \( \gamma_i \).
4. If \( \gamma_i < \gamma_{\text{max2}} \) then this tetrahedron does not need smoothing, therefore increment \( i \) by 1 and go to step 2.
5. Determine which of the four nodes of the \( i \)-th tetrahedron are movable, i.e. those nodes that are in the flow domain (not part of the outside boundary or fibre walls), and create a list of these nodes, \( P_k \), \( k = 1, \ldots, n_{\text{move}}, n_{\text{move}} \leq 4 \). If no points of the tetrahedron are movable, \( n_{\text{move}} = 0 \), then increment \( i \) by 1 and go to step 2.
6. By performing a tree search, find all of the other tetrahedra that are formed by at least one of the nodes from the list, \( P_k \), and create a list of these tetrahedra and the \( i \)-th tetrahedron, \( T_r, j = 1, \ldots, n_{\text{move}} \).
7. Find the minimum of the function, \( G_{\gamma}(\gamma_T) \), defined as follows

\[
G_{\gamma}(\gamma_T) = \sum_{j=1}^{n_{\text{move}}} \begin{cases} 
    \gamma_T > 0 & \Rightarrow (\gamma_T)^2 \\
    \gamma_T < 0 & \Rightarrow (100\gamma_T)
\end{cases}
\]  

(6.21)

by moving the nodes, \( P_k \).
8. Increment \( i \) by 1 and go to step 2.

The optimum mesh would have all tetrahedra with \( \gamma < 5 \) and, for this reason, the value for \( \gamma_{\text{max2}} \) was chosen to be 5. The function, \( G_{\gamma}(\gamma_T) \), defined in equation (6.21) was developed in such a way that its minimum would optimize the \( \gamma \) collectively for all the tetrahedra associated with the movable points. Since a negative value for \( \gamma \) is unacceptable, the function multiplies any negative \( \gamma_T \) by a large number, 100 in this case, to make sure that the minimum of \( G_{\gamma}(\gamma_T) \) does not include an inverted tetrahedron. The minimization routine is based on the downhill simplex method in multidimensions presented by Press et al. (1992). It should be emphasized that the **Mesh Smoothing Procedure** destroys the Dirichlet tessellation of the domain which means that no nodes can be added after the smoothing procedure has been run.

An example mesh generated run is considered to show the effects of the mesh refinement.
procedures. Figure 6.11 shows the aspect ratio distribution of the tetrahedra after generating a fibre and running the Internal Flow Domain Node Generation Procedure (Step 4 of 3D-AMGA) with $\alpha = 1.0$. The value of $\alpha$ was then changed to 0.8 and the Internal Flow Domain Node Generation Procedure was run again. The aspect ratio distribution of the tetrahedra is shown in Figure 6.12. The maximum $\gamma$ in this case was 554. Clearly, with 196 tetrahedra with $\gamma$ greater than 10, the mesh is not very good and needs refinement. Figure 6.13 shows the distribution after running the Procedure for Tetrahedron Centroid Node Addition for Skewed Element with $\gamma_{\text{max}} = 10.0$. At this point all of the tetrahedra have acceptable aspect ratios. The results of running the Mesh Smoothing Procedure is shown in Figure 6.14. This mesh is quite good, having only 8 tetrahedra whose aspect ratio is greater than 6 and the majority of the tetrahedra with aspect ratios of less than 2.0. The example given here demonstrates the capabilities of the mesh refinement procedures. After the addition of the quadratic nodes on the sides of the tetrahedra, this example case has over 24 000 nodes. Typical computer run times of the mesh refinement procedures for this size of problem is on order of 10 minutes on a Pentium 133 MHz PC. The bulk of the mesh generation work is completed at this point. The final steps of the three-dimensional automatic mesh generating algorithm will be discussed in the following section.
Figure 6.11: Aspect ratio of tetrahedra after running *Internal Flow Domain Node Generation Procedure* with $\alpha = 1.0$.

Figure 6.12: Aspect ratio of tetrahedra after running *Internal Flow Domain Node Generation Procedure* with $\alpha = 0.8$. 


Figure 6.13: Aspect ratio of tetrahedra after running *Procedure for Tetrahedron Centroid Node Addition for Skewed Element* with $\gamma_{\text{max}} = 10.0$.

Figure 6.14: Aspect ratio of tetrahedra after running *Mesh Smoothing Procedure* with $\gamma_{\text{max2}} = 5.0$. 
6.7 The Final Steps

The final steps of the three-dimensional automatic mesh generating algorithm are relatively straightforward and will be discussed briefly. At the completion of step 7 of the 3D-AMGA the domain consists of well-shaped, linear, tetrahedra. The tetrahedra are referred to as linear since they are made up of four nodes and only linear interpolations can be used to approximate functions over their volumes. In a CFD problem, Cuvelier et al. (1986) show that quadratic tetrahedra must be used for the velocity approximation. For this reason, nodes along the centres of the tetrahedra edges must be added. The difficulty associated with the addition of the nodes along the edges is that most edges are common to more than two tetrahedra. To make the addition of the quadratic nodes efficient, tree search techniques, similar to the ones used previously, were employed.

After the addition of the nodes at the mid-points of the edges of the tetrahedra, the final step requires the node data and connectivity table to be saved in the appropriate manner for the CFD software. The CFD software used to solve the flow was FIDAP (1993) and an in-house software called "tetr" (see Zhang et al. (1994)). FIDAP requires that all two-dimensional boundary "face" elements be listed separately from the three-dimensional domain connectivity table. The determination of the face elements is straightforward. Each tetrahedron is checked to see if one of its four faces is on the fibre or domain boundary. If it is, that tetrahedral face is added to the corresponding boundary face connectivity table. The results of flow simulations will be given in the next section.

6.8 Flow Simulation

A difficulty was encountered when FIDAP was used to solve the flow fields generated by the automatic three-dimensional mesh generating software. After FIDAP solved the flow, the pre-processor was used to determine the stresses acting on the faces of the cylinder. Unfortunately, the data calculated by FIDAP was completely random and unrealistic. To overcome this problem, an in-house software, called tetr (Zhang et al. (1994)), developed by the Mechanical Engineering CFD group at the University of Toronto, was used to solve the flow. The results of the CFD flow simulation presented in this section were determined by tetr. A
simple geometry was used in order that the three-dimensional results can be easily viewed.

Figure 6.15 shows elevation and plan views of the flow domain. Since tetr accepts the geometry in dimensionless form and a Reynolds number is prescribed to scale the velocity with respect to the geometry, all units given are dimensionless. A value of 0 was used for the Reynolds number, which simulates a Stokes flow, to keep the computer processing times reasonable. A rectangular box of dimensions $1 \times 3 \times 1$ surrounds a fibre of length 0.5 and diameter 0.1. A constant velocity boundary condition of 0.0001 in the $y$-direction was prescribed at the inlet plane and on all four boundary walls which run parallel to the $y$ axis. Neumann boundary conditions were specified at the outlet. The domain consisted of approximately 27 500 nodes, of which approximately 2000 were located on the fibre faces and approximately 4000 were located on the
Figure 6.16: Mesh and stress contours on the fibre.

boundary walls. After mesh refinement, there were no tetrahedra whose aspect ratio was greater than 5.0.

Figure 6.16A shows the finite element mesh on the leading face of the fibre while Figure 6.16B shows the stress contour lines on this face. The stress contour lines are consistent with what is expected - the magnitude of the stress should be symmetrical about the plane perpendicular to the $x$ axis running through the centre of the fibre, and the lines of constant stress should run parallel to the $z$ axis. Figure 6.17 plots the finite element mesh on the plane perpendicular to the $x$ axis running through the centre of the fibre and Figure 6.18 plots the velocity vectors on this plane. Similarly, Figure 6.19 plots the finite element mesh on the plane perpendicular to the $z$ axis at $z = 0.5$ and Figure 6.20 plots the velocity vectors on this plane.

As mentioned previously, Hu et al. (1992) developed a two-dimensional finite element model which simulates the motion of circular particles in a fluid. The extension of this method to three dimensions is straightforward with the limitation that an automatic three-dimensional mesh
Figure 6.17: Finite element mesh on the plane perpendicular to the $x$ axis running through the axis of the fibre.

Figure 6.18: Velocity vectors on the plane perpendicular to the $x$ axis running through the axis of the fibre.
Figure 6.19: Finite element mesh on the plane perpendicular to the $z$ axis at $z = 0.5$.

Figure 6.20: Velocity vectors on the plane perpendicular to the $z$ axis at $z = 0.5$. 
generating algorithm is required. Such an algorithm was developed here. It is expected that as computer speed and memory increase, it will become possible for a fully interactive solution using CFD and the equations of motion of a flexible fibre to be used to solve the movement of a fibre in a shear flow.
Chapter 7

Concluding Remarks

7.1 Summary and Discussion

The static and dynamic behaviour of pulp fibres was studied in this thesis. Governing equations, which represent the deflection and motion of flexible pulp fibres, were developed and numerical methods were utilized to solve the mathematical formulations. In particular, static large deflection beam theory was applied to the geometry of four existing fibre flexibility measurement devices and the advantages and shortcomings of each method were investigated. Non-linear equations representing the motion of a flexible fibre were developed. These equations were utilized to examine the ability of fibres of varying flexibility and length to enter a slot in a flow domain consisting of a channel with a slot. An automatic three-dimensional finite element mesh generating algorithm was developed, to be used with Computational Fluid Dynamics (CFD) software, to model the fluid flow around a fibre of arbitrary shape.

7.1.1 Fibre Statics

Four fibre flexibility measurement methods were analysed using small and large deflection analyses and it was shown that the small deflection analysis generally predicts stiffnesses with an error of less than 10% when compared to the large deflection analysis for fibre deflections of less than 20% of the span length. The Conforming Fibre on a Wire method was shown to have the
potential for introducing large errors in determining the stiffness, due to the difficulty in determining the span length and the sensitivity of the equations to the span length. Furthermore, the deflection of the fibre in its cross-section would result in an underestimated stiffness result. The other methods, which rely on hydrodynamic forces to deflect the fibre, were prone to error due to the difficulties associated with determining these forces. Unknown friction forces at the contact points in the Simply Supported Fibre in a Capillary method were shown to contribute to an overestimated stiffness result. The results of the small and large deflection analyses suggest that with a better prediction of the hydrodynamic forces acting on the fibre, the cantilevered fibre geometry has the most potential for predicting an accurate fibre stiffness value.

7.1.2 Fibre Dynamics
Using Hamilton's principle, the equations of motion for a flexible fibre in a flow were developed. The equations of motion were integro-differential in form and were discretized by utilizing the free modes of vibration for a beam with both ends free. The equations were simplified by removing the fibre inertia component. Since the forces acting on the fibre were assumed to vary linearly with the relative velocity of the fluid, the inertialess motion model was easily applied. The dynamic equations were shown to agree with the static beam bending theory in the steady-state limit. Two models to represent the contact between the fibre and the flow domain walls were presented. The Compliant Wall Contact Model was shown to be easily applied from a software development point of view, but was found to be inefficient and sometimes unstable. The Constraint Wall Contact Model was developed to increase the speed of the simulations. While this model requires significantly more overhead in the development, it produces a numerical model which is much more easily solved and does not possess the inherent instability of the Compliant Wall Contact Model.

The fibre motion model was used to simulate the flow of flexible and rigid fibres in a channel with a slot. While the model agreed with observed experimental results from a qualitative analysis, it did not quantitatively agree with the experimental results. The model under predicted the passage ratio.

For the first time, fractionation based on fibre flexibility was observed by using a numerical technique. It was shown, as has been observed experimentally, that fibre length has a greater
effect on fibre fractionation than flexibility. However, it was also shown that longer fibres have a
greater potential to be separated according to flexibility. Flexible fibres showed favourable slot
passage due to their ability to bend along the streamlines.

The results of the simulations substantiate experimental work which shows that
fractionation of fibres based on flexibility, while possible, is a difficult task. New processing
techniques are required to accomplish this task. It is expected that the fibre motion model could
be utilized in the development of new strategies for flexible fibre fractionation.

7.1.3 Automatic Three-Dimensional Mesh Generating Algorithm
To propose a direct method to model the flow of a fibre of any arbitrary shape and to determine
the forces acting on the fibre, an automatic three-dimensional mesh generating algorithm was
developed, which can be integrated with existing CFD packages. The algorithm was found to be
efficient and robust. Of primary importance was mesh quality. It was shown that by the addition
of nodes at the centroids of badly skewed tetrahedra, the mesh quality was greatly enhanced. In
fact, it was always possible to achieve a mesh quality with all tetrahedra having an aspect ratio of
less than 8.0. While CFD is only beginning to use unstructured tetrahedral elements, it is believed
that with better numerical methods and greater computer performance, the algorithm will be very
viable for directly simulating the motion of a flexible fibre in a shear flow.

7.2 Recommendations for Future Work
The work presented in this thesis has numerous applications. Three "numerical tools" were
developed which can be utilized in many different ways. On the subject of screening, there still
exists a fundamental question which remains unanswered: "How is it that essentially all of the
fibres in the exit layer seem to enter the slot?". The following subsections will propose future
research studies based on the work presented here.

7.2.1 Application of Large Deflection Equations
The large deflection analysis has been shown to be useful in identifying possible deficiencies in
fibre flexibility measurement methods. As new methods are proposed, the equations can be
utilized to evaluate the new geometries.
7.2.2 Flexible Fibre Motion Model

The flexible fibre motion model can be utilized in many aspects of the pulp and paper industry to study the behaviour of fibres. Primarily, it is recommended that the model be enhanced to study the effect of different screen contour surfaces. The model can also be utilized to develop a fibre flexibility measurement device in which fibres are subjected to a high shear, and their deflection is measured. The advantage of such a method is that it would not be labour intensive and would not be prone to plugging if properly designed. The predominant hurdle in developing such a method is the determination of the hydrodynamic forces.

7.2.3 Application of CFD to Model the Hydrodynamic Forces Acting on a Fibre

The automatic three-dimensional mesh generating algorithm can be used in any situation where a better estimate of the hydrodynamic forces acting on the fibre is desired. As computer power increases, the application of the mesh generating algorithm to a direct fibre flow simulation will become more feasible.

7.2.4 Pulp Screening

Since the fibre concentration gradient seems to be the primary factor affecting fibre fractionation, the effect of flexibility on the concentration gradient should be investigated. The hypothesis here is that fibre flexibility will play a negligible role on the concentration gradient. If this is shown to be true then the "turning effect" must be utilized to its maximum potential to fractionate fibres based on flexibility.

Further research is also required to better explain the unexpectedly high passage ratios observed in the experimental studies. The question: "How is it that essentially all fibres in the exit layer enter the slot?" needs further investigation.
Chapter 8

References


Steadman, R. and Luner, P., "An Improved Test to Measure the Wet Fiber Flexibility of Pulp Fibers", ESPRA Report #79, Chapter V.


Appendix A

Object Oriented Programming

The majority of the numerical work presented in this thesis was solved by software developed using the C++ Object Oriented Programming (OOP) language. The numerical work is represented by over 10,000 lines of C++ source code. To keep the programming manageable, fast and efficient, vector and matrix data structures, or classes, were developed. These C++ template classes form the basis for all the numerical programs developed. The purpose of this chapter is to give a brief description of these C++ template classes. The template classes developed in this thesis follow closely the data structures developed by Flamig (1993). The source code is given in Appendix B.

A.1 Introduction to the C++ Object Oriented Programming Language

The C++ language is based on Object Oriented Programming (OOP) that allows fast, low level coding yet provides tools for developing elegant data structures. The C language is a subset of C++, which allows all C programs to work identically in C++. The advantage of using OOP is that once a function or routine has been developed and checked for errors, it can be compartmentalized to prevent its misuse. This has the consequence of reducing program development time. Although there are many OOP languages, C++ provides the low level
capabilities that allow the development of very fast and efficient codes. The initial investment in time to develop bug-free base data structures has paid off handsomely in the final higher level numerical programming. In this section, a few key features available in C++, namely dynamic memory allocation, classes, function and operator overloading, and class templates, will be reviewed to show the advantages of using the language for scientific and engineering computing.

**Dynamic Memory Allocation**

One of the great advantages of C over FORTRAN 77 is the ability to allocate memory for data storage at run time. FORTRAN requires that all computer memory for data storage be specified at the time of program compilation. This is inefficient, since the programmer must usually overestimate the amount of memory required which leads to greater usage of computer resources. C, on the other hand, allows dynamic memory allocation; but the functions used for memory allocation are awkward and require the use of memory pointers which often lead to programming errors that are very difficult to debug. C++ allows dynamic memory to be used for any data type, including user defined data types, and, with the use of properly designed data classes, the allocation can be made efficient and safe.

**Classes**

A C++ class is an object, which is a logical entity defined by the programmer that contains data and functions that manipulate the data. Within the class, it is possible to encapsulate data and functions which are then made accessible only by functions belonging to that class. The encapsulation ensures that the encapsulated data and functions will not be accidentally manipulated or used by functions not intended to have access to the encapsulated members. This feature is extremely useful in preventing software bugs when complex programs are being developed, especially when developed by more than one programmer. A class, or object, can be considered to be a programmer-defined data type.

**Function and Operator Overloading**

C++ allows the user to overload functions and operators. For example, suppose it is desired to write a function that returns the result of the addition of three numbers and another function that returns the result of the addition of two numbers. In traditional languages such as C
and FORTRAN, the programmer would have to code two functions with different names. In
C++, on the other hand, the programmer would still code two different functions, but would be
able to use the same name for both functions. This feature is useful for performing similar tasks
with different function parameters. The programmer will not have to remember the different
names of functions.

The advantages of overloading are even more apparent when applied to operators. For
example, in mathematics the plus "+" symbol designates the sum of two entities, one on the left
hand side of the "+", the other on the right. Depending on the data type, the definition of the
summation changes, i.e. the definition for the sum of two numbers is different than that of two
vectors. With properly designed classes, it is possible to use the "+" operator to efficiently
perform vector addition with the same amount of coding effort as the addition of two integers.
The use of operator overloading will be discussed further in the following sections.

Class and Function Templates

The C++ language has a new feature which allows the programmer to use class and
function templates. Essentially, a C++ template is code which the pre-processor reads and uses
only if that particular class type was instantiated or function type called. The use of templates is
best explained with a simple example. Consider a function that has two parameters and returns
the sum of these two parameters. Assume that it is desired that the function returns an integer
value of the sum if the two parameters are integers, and returns a float\(^1\) value of the sum if the two
parameters are floats. Instead of writing two different functions, the programmer could write a
function template where the type of the parameters dictates which type of function should be
used. In the compilation stage, the pre-processor would find all the calls to the function template
with different parameter types and create the actual code for each type as needed. Thus, if all the
calls to the function template were integer types, the pre-processor would only create actual code
for the function from the template for integer type parameters. If the code had calls to the
function template of integer and float types then two functions would be created by the
pre-processor, each corresponding to its own type. Class and function templates can be used for
all data types.

---

\(^1\) A float in C and C++ is data type that represents a real number and requires 4 bytes of
memory.
By using class templates, it is possible to efficiently develop similar classes of different types. For example, a matrix class developed for integers would be almost identical to one developed for real numbers. To make the coding easy and efficient, C++ allows the use of class templates, which allows the programmer to develop a class data structure applicable to any data type. Thus the same class template for the matrix class can be used for both integer matrices and floating point matrices.

Clearly, C++ provides tools for engineering and scientific programming that are not readily available in other languages. Since most numerical programming requires the extensive use of vectors and matrices, a great deal of effort was made to develop efficient and reliable Vector and Matrix class templates. Complementing the Vector and Matrix class templates are a series of overloaded operator function templates that perform vector/matrix type multiplications. The development of the class templates is based on the data structures presented by Flamig (1993) but the coding is predominantly original work. The following sections will discuss the Vector and Matrix class templates of the header file called "matrixw.hpp". The complete listing of the file is given in Appendix B.

### A.2 The Vector Class Template

Of primary importance to the Vector class template is the VecRep class template which keeps track of memory and performs memory allocation. The VecRep class template is represented by the code in Figure A.1. The VecRep class template consists of one data type, an unsigned variable, refcount, which equals the number of objects that are pointing to the same data. an

```cpp
// VecRep class
template <class TYPE>
class VecRep {
private:
    friend class Vector<TYPE>;
    friend class VecPtr<TYPE>;
    unsigned refcount;
    void *operator new(size_t n, long i);
    void operator delete(void *p);
public:
    VecRep();
};
```

Figure A.1: VecRep class template.
overloaded operator function new which allocates memory, an overloaded operator function delete which deallocates memory, and a constructor, VecRep(), that initializes refcount to one upon the creation of a VecRep object. The keywords private and public designate the accessibility, or encapsulation, of the data or functions. Data and functions designated as private are accessible by member functions of the class, whereas public data and functions are accessible by non-member functions. When data or functions are designated as private, they are said to be "encapsulated".

The Vector class template is represented by the code in Figure A.2. The first private variable declared in the Vector class template is the pointer of type TYPE, v. It is used to point to the first TYPE data element. Here "TYPE" represents a data type specified by the programmer; for example, for an integer vector data structure, TYPE would be replaced with int\(^2\). The VecRep<TYPE> pointer, rep, is used to point to the start of the block of data allocated for a specific Vector object. Figure A.3 illustrates the data storage. The memory block designated by "VecRep data" would hold the value of refcount, where as the blocks designated by "v[0]", "v[1]", ..., "v[n]" designate the storage locations for the actual vector data.

The only other two variables of Vector are the public variables size and status of type long\(^3\) and int respectively. Size represents the number of elements of type TYPE of the particular object. The variable status is used to determine if that particular Vector object is allowed to share data. The default value is FALSE. It is advantageous to declare status to be TRUE when the Vector object is declared temporarily in a specific function and will be returned to, or passed to, another Vector object. In this way each vector data element does not need to be copied from one Vector object to another, but rather, the pointers, rep and v, are made to point to the same memory locations. In this case, refcount of VecRep would be equal to two, since two objects are pointing to the same data. This is coded in such a way that the details are hidden from the programmer/user. Using this technique produces fast, efficient and error free executable code.

The private function Allocate() initializes the variables status and size, and calls the overloaded new operator of VecRep. The new operator allocates the required memory and

---

\(^2\) A int in C and C++ is data type that represents an integer number and requires 2 or 4 bytes of memory.

\(^3\) A long in C and C++ is data type that represents an integer number and requires 4 bytes of memory. A long type and int type are equivalent on some computer platforms.
// Vector class
template <class TYPE>
class Vector {
friend class Matrix<TYPE>;
    TYPE *v;
    VecRep<TYPE> *rep;
    void Allocate(long i, int stat = FALSE);
    void Bind(const Vector<TYPE> &vec);
    void Unbind();
    void Copy(const Vector<TYPE> &vec);
    void VectorError(char *p) const;
public:
    long size; // size of vector
    int status; // if status=TRUE then treated as a temporary vector
    Vector(long i=0, int stat = FALSE);
    Vector(const Vector<TYPE> &vec);
-Vector();
    void Destruct() {Unbind();}
    Vector<TYPE>& operator=(const Vector<TYPE>& vec);
    Vector<TYPE>& operator=(TYPE value);
    TYPE& operator[](long i) const {return v[i];}
    Vector<TYPE>& operator[](long i);
    Vector<TYPE> operator-() const;
    double Norm(); // find the norm of the vector
    void Add(Vector<TYPE>& v1, Vector<TYPE>& v2); //this = v1+v2
    void Mult(TYPE cl); // this = this*cl
    void CrossProd(Vector<TYPE>& v1, Vector<TYPE>& v2); //this=v1 X v2
    // reallocate memory for a new vector and copy elements
    Vector<TYPE>& ReAlloc(long i, TYPE var=0);
    Vector<TYPE> SubVec(long start=1, long end=1, int unique=TRUE, long stride=1);
    void CopyElements(long thisi, Vector<TYPE> &vec, long il=1, long i2=0);
    void RowOfMtx(Matrix<double>& m, long i);
    void RowOfMtx(Matrix<long>& m, long i);
    void Show(long i=-1);
    friend Vector<TYPE> operator+(const Vector<TYPE>& vec1, const Vector<TYPE>& vec2);
    friend Vector<TYPE> operator-(const Vector<TYPE>& vec1, const Vector<TYPE>& vec2);
    friend Vector<TYPE> operator*(TYPE, const Vector<TYPE>& vec1);
    friend Vector<TYPE> operator*(const Vector<TYPE>& vec1, TYPE);
    friend TYPE operator*(const Vector<TYPE>& vec1, const Vector<TYPE>& vec2);
    friend ostream& operator<<(ostream& stream, const Vector<TYPE>& vec);
};

Figure A.2: Vector class template.

returns a pointer, whose value is passed to rep, pointing to the start of that memory. The pointer v is then initialized to point to the first allocated memory location of type TYPE.

Bind() is a private function which binds two vector objects to point to the same data. It
then increases the value of rep->refcount\(^4\) by one. Unbind() is a function which performs the opposite task of Bind(). It reduces the value of rep->refcount by one and checks to see if rep->refcount is equal to zero. If it is, it means that no other objects are pointing to that particular memory location and therefore the memory should be deallocated; this is performed by the overloaded delete operator of VecRep.

The private function Copy() copies one vector object to another. If the status of the Vector object to be copied is FALSE, then data sharing does not take place. In this case the Vector object to which data is being copied is unbound by utilizing the Unbind() function and memory is reallocated for the required size. Next, an element by element copy is performed. If, on the other hand, status=TRUE, then the vectors are bound by utilizing the Bind() function.

Two Vector() functions are part of the Vector class. One is the Vector constructor which calls the Allocate() function. The other function is the Vector copy constructor which is used for declarations; for example, vl can be declared to be the same as an existing v2 by the following statement:

Vector<double> vl=v2;

Here the vector data of vl will be assigned the vector data of v2.

The ~Vector() destructor is used to unbind the vector. C++ automatically calls this function when the Vector object goes out of scope. For example, if a Vector object is declared in a function, the destructor function for that object will be called upon termination of the function. Therefore, memory deallocation occurs transparently. The programmer does not have to include

\(^4\) The symbol "->" is used to designate that the VecRep pointer rep of the Vector object is pointing to refcount of the VecRep object.
a specific call to a function to free the memory. This feature is extremely useful in preventing the unintentional memory freezing bug common in C programs upon omission of the call to the function \texttt{free()}. 

The "=" operator is overloaded twice in the \textit{Vector} class: 1) to copy one vector to another, and 2) to initialize all the data to a certain value of type \texttt{TYPE}. The "[" operator is overloaded in order to return a reference pointed to by \texttt{v}. This operator is "inlined" to produce fast and efficient code and to avoid unnecessary function calling. The "()" operator is overloaded to reallocate memory for an existing \textit{Vector} object. Other overloaded operators were also written to perform specific tasks and are summarized below:

- "-" unary minus operator used to multiply all the data elements in the \textit{Vector} object by -1;
- "+" binary plus operator used to perform vector addition between two \textit{Vector} objects of the same type, \texttt{TYPE};
- "+=" binary multiplication used to multiply all the data elements of a \textit{Vector} object by a number of type \texttt{TYPE};
- "+*" binary multiplication used to perform the vector dot product between two \textit{Vector} objects of the same type, \texttt{TYPE} and size;
- "+<<" output stream operator used to output the contents of a \textit{Vector} object to the console or other output devices.

The function \texttt{ReAlloc()} was written to allocate extra memory for a vector. This function is useful in situations where the programmer may need to increase the size of a \textit{Vector} object but not lose the previous data and was used extensively in the automatic three-dimensional tetrahedral mesh generating program. A \texttt{SubVec()} function was written to allow for the use of subvectors of vectors. The subvector can be unique (default) or point to existing allocated data.

\section*{A.3 The \textit{Matrix} Class Template}

The \textit{Matrix} class template was developed for two-dimensional array data storage and manipulation. Many of the features of the \textit{Matrix} class template are similar to the \textit{Vector} class template. The \textit{Matrix} class template is represented by the code in Figure A.4. The first variable
declared in the Matrix class template is a private Vector object of type TYPE called data which contains all the elements of the two-dimensional array. Since data is a Vector object, all of the functions written for the Vector class template are available to the Matrix class template. The intricate features of memory allocation and binding did not have to be reprogrammed for Matrix. The other private variable of Matrix is the long integer colstride which determines the stride of elements along the width of the two-dimensional array. Two public variables of long integer type are present in Matrix, nrow and ncol, which represent the number of rows and columns of the two-dimensional array. By using the two variables, colstride and ncol, it is possible to have two or more different Matrix objects pointing to the same data, with one or more objects being a sub-matrix of the other object(s). It should also be pointed out that another long integer could have been used for the row stride, which would normally be equal to one. This would allow the transpose of a matrix to be easily performed by interchanging the colstride variable with the row

```cpp
// Matrix class
template <class TYPE>
class Matrix {
    Vector<TYPE> data;
    long colstride;
    void Copy(const Matrix<TYPE> &m);
  public:
    long nrow, ncol;
    Matrix(long r=0, long c=0, int stat=FALSE);
    Matrix(const Matrix<TYPE>& m);
    Matrix<TYPE>& operator=(const Matrix<TYPE>& m);
    Matrix<TYPE>& operator=(SparseSym<TYPE>& m);
    Matrix<TYPE>& operator=(TYPE value);
    TYPE *operator[](long r) {return data.v+(r-1)*colstride;}
    TYPE *operator[](long r) const {return data.v+(r-1)*colstride;}
    Matrix<TYPE> operator-() const;
    Matrix<TYPE>& operator[](long i, long j);
    Matrix<TYPE>& ReAlloc(long i, long j, TYPE var=0);
    int64 Status() {return data.status;}
    Matrix<TYPE> SubMtx(long ri, long rf, long ci, long cf);
    void CopyRows(long thisi, Matrix<TYPE> &m, long mi=1, long mj=0);
    void Show(long ii=0, long jj=0);
  friend Matrix<TYPE> operator+(const Matrix<TYPE>& m1, const Matrix<TYPE>& m2);
  friend Matrix<TYPE> operator+(TYPE value, const Matrix<TYPE>& m);
  friend Matrix<TYPE> operator*(const Matrix<TYPE>& m, TYPE value);
  friend Matrix<TYPE> operator*(const Matrix<TYPE>& m1, const Matrix<TYPE>& m2);
  friend ostream& operator<<(ostream& stream, const Matrix<TYPE>& m);
};
```

Figure A.4: Matrix class template.
stride but, since none of the programs developed in the thesis required the transpose of a matrix, the feature was not included in order to keep the code more efficient.

The private function Copy() copies one Matrix object to another by making use of the overloaded "=" operator of the Vector object data of Matrix. Since the copying is performed by the Vector functions, the same efficient features of Vector objects are available for Matrix objects. Similar to the Vector class template, Matrix also has a constructor function and a copy constructor function. The constructor function allocates the required memory for data, whereas the copy constructor allows a Matrix object to be initialized upon declaration.

Of particular importance is the overloaded operator "[]" function. This function returns a pointer of type TYPE pointing to the last element of the row r-1, where r is a parameter for a function. This provides a method of indexing the Matrix data elements in the same way that static C and C++ two-dimensional arrays are indexed. By "inlining" the code, the procedure is made fast and efficient by not producing an actual function call in the executable code. Other overloaded operators are summarized in the following list:

- "=": binary equals/assign operator used to assign a scalar value of type TYPE to all the data elements of the Matrix object of type TYPE;
- "=": binary equals/assign operator used to assign one Matrix object to another Matrix object of the same type, or to assign a SparSym object (a template class for sparse symmetric matrices) to a Matrix object both of type TYPE;
- "-": unary minus operator used to multiply all the data elements in the Matrix object by -1;
- "+": binary plus operator used to perform matrix addition between two Matrix objects of the same type, TYPE, and size;
- "*": binary multiplication used to multiply all the data elements of a Matrix object by a number of type TYPE;
- "**": binary multiplication used to perform the matrix multiplication between two Matrix objects of the same type, TYPE, and with compatible row/column sizes;
- "<<": output stream operator used to output the contents of a Matrix object
to the console or other output devices.

The function ReAlloc() was written to allocate extra memory for a matrix in a way similar to the ReAlloc() function of Vector. A SubMtx() function was written to allow for the use of submatrices of matrices. The submatrix can be unique (default) or point to existing allocated data.

A.4 Examples of the Usage of Vector and Matrix Class Templates

The purpose of this section is to show the effectiveness of using the Vector and Matrix class templates in numerical programming by providing some simple examples. Table A.1 summarizes the examples by providing the C++ code and a description of the action. The examples given below assume that the type of each object is consistent; for example, when two vectors are multiplied to give the dot product, both vectors must be of the same type, i.e. int or long or double\(^5\), etc..

Table A.1: Examples of Vector and Matrix class template usage. The examples clearly demonstrate the ease of using the Vector and Matrix class templates for programming. Although a great deal of work was required to build the class templates, the rewards of fast, efficient and error free coding are well worth the initial effort. By designing other C++ classes in appropriate ways, depending on the application, the OOP style of programming provides great advantages over traditional programming styles. The use of C++ and OOP have greatly decreased the overall effort of writing the numerical programs required for this work.

---

\(^5\) A double in C and C++ is a data type that represents a real number, similar to float, but requires 8 bytes of memory.
<table>
<thead>
<tr>
<th>Purpose or Function</th>
<th>C++ Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>Declare a long vector variable, v, with no data elements, i.e. zero size.</td>
<td>Vector&lt;long&gt; v;</td>
</tr>
<tr>
<td>Declare an int vector variable, v, of size 3.</td>
<td>Vector&lt;int&gt; v(3);</td>
</tr>
<tr>
<td>Declare a double vector, v1, and initialize it to an existing double vector, v2.</td>
<td>Vector&lt;double&gt; v1=v2;</td>
</tr>
<tr>
<td>Reallocation memory for an existing vector, v, to a size of n, where n is a previously defined long, without saving the old data of v.</td>
<td>v(n);</td>
</tr>
<tr>
<td>Reallocation memory for an existing vector, v, to a size of n, where n is a previously defined long and n is greater than the initial size of v, keeping the old data of v.</td>
<td>v.ReAlloc(n);</td>
</tr>
<tr>
<td>Reference the 3rd element of a vector, v.</td>
<td>v[3];</td>
</tr>
<tr>
<td>Perform vector addition between two vectors, v1 and v2, of the same size and assign the answer to a vector, v3. Note: if the size of v1 does not equal v2 then the overloaded &quot;+&quot; operator function will flag the error. The vector v3 can be of any size and will be reallocated as necessary.</td>
<td>v3=v1+v2;</td>
</tr>
<tr>
<td>Multiply the vector, v2, by a scalar, x, and assign the answer to v1. Note: the size of v1 will be reallocated as necessary.</td>
<td>v1=x*v2;</td>
</tr>
<tr>
<td>Perform the vector dot product between two vectors, v1 and v2, and assign the answer to a variable, x.</td>
<td>x=v1*v2;</td>
</tr>
<tr>
<td>Declare a 5 x 5 matrix, m of double.</td>
<td>Matrix&lt;double&gt; m(5,5);</td>
</tr>
<tr>
<td>Multiply two matrices, m1 and m2 and assign the answer to matrix, m3. Note: the overloaded &quot;*&quot; operator will check for consistent matrix dimensions.</td>
<td>m3=m1*m2;</td>
</tr>
<tr>
<td>Multiply a matrix m1 by a vector v1 and assign the answer to a vector, v2. Note: the overloaded &quot;*&quot; operator will check for consistent matrix/vector dimensions.</td>
<td>v2=m1*v1;</td>
</tr>
<tr>
<td>Assign a scalar value, x, to the element in the third row and second column of matrix m.</td>
<td>m[3][2]=x;</td>
</tr>
</tbody>
</table>

Table A.1: Examples of Vector and Matrix class template usage.
Appendix B

Vector and Matrix Template Classes
```
//matrixw.hpp

#include <iostream.h>
#include <stdlib.h>
#include <math.h>
#define ABS(a) (a)<0 ? -(a) : (a)
#define CHECKRANGE // if defined then subscript range checked for Vector

//define SHOW_MAKE_DELETE

const int NUMRUNS = 2;
const int TRUE = 1;
const int FALSE = 0;
const int TEMPVEC = 1;
const double TINY = 1.0e-20;
const double PI = 3.141592653589793238462643383279;
const double PRECISION = 1.0e-7;
const int MAX_ITER = 10000;

template <class TYPE> class Vector; // forward template declaration

template <class TYPE> class Matrix;

template <class TYPE> class SparSym;
template <class TYPE, class T> class SparMtx;
template <class TYPE> class VecPtr;

// VecRep class
template <class TYPE>
class VecRep {
friend class Vector<TYPE>;
friend class VecPtr<TYPE>;
int refcount;
void *operator new(size_t n, long i);
void operator delete (void *p);
public:
  VecRep();
};

// VecPtr - all purpose vector class (ie. for pointer use)
template <class TYPE>
class VecPtr{
  TYPE *v;
  VecRep<TYPE> *rep;
  void Allocate(long i, int stat = FALSE);
  void Bind(const VecPtr<TYPE> &vec);
  void Unbind();
  void Copy(const VecPtr<TYPE> &vec);
public:
  long size;
  int status;
  VecPtr<Long i=0, int stat=FALSE>;
  VecPtr(const VecPtr<TYPE> &vec);
  ~VecPtr() {Unbind();}
  VecPtr<TYPE>& operator = (const VecPtr<TYPE> & vec);
  VecPtr<TYPE>& operator = (TYPE value);
  TYPE &operator [] (long i) const {return v[i];}
  VecPtr<TYPE>& operator [] (long i);
};

//template <class TYPE> class Matrix; // forward template declaration

// Vector class
template <class TYPE>
class Vector {
  //public: // take this out after BORLAND bug is fixed
  friend class Matrix<TYPE>; //this line causes compiler error on BC3.1;
};
```
friend class SparSym<TYPE>;  
TYPE *v;  
VecRep<TYPE> *rep;  
//VecRep<TYPE> tryrep;  
void Allocate(long i, int stat = FALSE);  
void Bind(const Vector<TYPE>& vec);  
void Unbind();  
void Copy(const Vector<TYPE>& vec);  
void VectorError(char *p) const;  
public:// put this in after BORLAND bug is fixed  
long size; // size of vector  
int status; // if status=TRUE then treated as a temporary vector  
Vector(const Vector<TYPE>& vec);  
-Vector () { Unbind (); }  
void Destruct () { Unbind () ; }  
Vector<TYPE>& operator=(const Vector<TYPE>& vec);  
Vector<TYPE>& operator=(TYPE value);  
friend Vector<TYPE> operator+(const Vector<TYPE>& vecl, const Vector<TYPE>& vec2);  
friend Vector<TYPE> operator-(const Vector<TYPE>& vecl, const Vector<TYPE>& vec2);  
friend Vector<TYPE> operator*(TYPE, const Vector<TYPE>& vec1);  
friend Vector<TYPE> operator*(const Vector<TYPE>& vec1, TYPE);  
friend TYPE operator*(const Vector<TYPE>& vec1, const Vector<TYPE>& vec2);  
//friend Vector<TYPE> operator*(const SparSym<TYPE>& ml, const Vector<TYPE>& v1);  
friend ostream& operator<< (ostream& stream, const Vector<TYPE>& vec);  
}

Matrix class  
template <class TYPE>  
class Matrix {  
Vector<TYPE> data;  
long /*rowstride,*/ colstride;  
void Copy(const Matrix<TYPE>& m);  
public:  
long nrow, ncol;  
Matrix(long r=0, long c=0, int stat=FALSE);  
Matrix(const Matrix<TYPE>& m);  
Matrix<TYPE>& operator=(const Matrix<TYPE>& m);  
Matrix<TYPE>& operator=(SparSym<TYPE>& m);  
Matrix<TYPE>& operator=(TYPE value);  
TYPE *operator[](long r) {  
#ifdef CHECKRANGE  
if (r<1||r>nrow)  
  ErrorFunc("Subscript error in Matrix");  
#endif  
}  
}
return data.v + (r-1)*colstride;
Matrix<TYPE> operator [](long r) const {
    #ifdef CHECKRANGE
    if ((r<1 || r>nrow)
        ErrorFunc("Subscript error in Matrix");
    #endif
    return data.v + (r-1)*colstride;
}
public:
T n; // simulated size (ie. represented size)
SparVec(T spar_n1=0, T n1=0, int stat=FALSE);
SparVec(const SparVec<TYPE,T> &vec);
SparVec<TYPE,T> & operator ()(T i, T j);
SparVec<TYPE,T> & operator =(const SparVec<TYPE,T> &vec);
SparVec<TYPE,T> & operator =(const Vector<TYPE> &vec);
SparVec<TYPE,T> & operator -(()) const;
SparVec<TYPE,T> operator [](T ii);
void Destruct();
void Show();

friend TYPE operator *(const SparVec<TYPE,T>&svec, const Vector<TYPE>&vec);
friend TYPE operator *(const SparVec<TYPE,T>&sv1, const SparVec<TYPE,T>&sv2);
friend Vector<TYPE> operator *(const Vector<TYPE>&vec, const SparMtx<TYPE,T>&sm);

// Sparse Matrix class - vector of pointers to SparVec
template <class TYPE,class T>
class SparMtx{
private:
    VecPtr<SparVec<TYPE,T>> vp;
    void Allocate(T ro);
    void Copy(const SparMtx<TYPE,T>&m);
public:
    T row;
    SparMtx(T ro=0, int stat = FALSE);
    ~SparMtx();
    SparVec<TYPE,T> &operator [](T i){return vp[i];}
    SparVec<TYPE,T> &operator =(const SparMtx<TYPE,T> &m);
    SparVec<TYPE,T> &operator =(const Matrix<TYPE> &m);
    SparVec<TYPE,T> &operator ()(T i);
    void Destruct();
    void Show();
    void ShowStore();
    long Storage(char out=0);
    friend Vector<TYPE> operator *(const SparMtx<TYPE,T>&sm, const Vector<TYPE>&vec);
    friend Vector<TYPE> operator *(const Vector<TYPE>&vec, const SparMtx<TYPE,T>&sm);
};

// Function declarations
void KBhit(char *p); // routine for terminating program exec.
void ErrorFunc(char *p); // Error handling routine
void lubksb(Matrix<double> &a, Vector<long> &indx, Vector<double> &b);
void ludcmp(Matrix<double> &a, Vector<long> &indx, double &d);
int conj_grad(SparSym<double> &A, Vector<double> &x, Vector<double> &b, double precision=PRECISION);
int is_equal(double v1,double v2,double tol); // is v1=v2 within tol ?
int isgreater_by_tol(double v1,double v2,double tol); // is v1-v2>tol ?
int isequal_or_greater(double v1,double v2,double tol); // is v1>=v2 within tol?
long rnd_to_long(double vl); // round off vl to a long-type

//VecRep functions: ************************************************************
template <class TYPE> // VecRep constructor
VecRep<TYPE>::VecRep()
{
    refcount = 1;
}
template <class TYPE> // overloaded new operator
//n is the size of the class for which mem. alloc. is being made-it has to
// long temp = (long)n+(i+1)*(long)sizeof(TYPE);
// void *p = (void *)malloc(temp);
void *p = new char[(long)n+(i+1)*(long)sizeof(TYPE)];
if (p==NULL){
    cout << "Error in VecRep operator new \n";
    exit(0);
}
#endif SHOW_MAKE_DELETE
cout << "Created: " << p << "\n";
#endif
return p;
}

// Vector functions:

// Allocate: memory allocation

void Vector<TYPE>::Allocate(long i, int stat)
{
    status = stat;
    size = i;
    rep = new(i) VecRep<TYPE>;
    v = (TYPE *) ((char *)rep+sizeof(VecRep<TYPE>));
}

// Bind: binds this vector to data of vec

void Vector<TYPE>::Bind(const Vector<TYPE>& vec)
{
    rep = vec.rep;
    v = vec.v;
    rep->refcount++;
}

void Vector<TYPE>::Unbind()
{
    rep->refcount--;
    if(rep->refcount<0)
        VectorError("Error: refcount < 0");
    if (rep->refcount == 0)
        delete rep;
}

// Copy: copy one vector to another

void Vector<TYPE>::Copy(const Vector<TYPE> &vec)
{
    if(!vec.status){   // if one to be copied not a tempvec
        if(size != vec.size){
            Unbind();
            Allocate(vec.size);
        }
        for(long i=0;i<=size;i++)
            v[i] = vec.v[i];
}
else {
    Unbind();
    size = vec.size;
    Bind(vec);
}

} /* end class Vector */

// VectorError(prv): error handler

void Vector<TYPE>::VectorError(char *p) const
{
    cout << "Vector Error: " << p;
    exit(0);
}

// Vector public functions

template <class TYPE> // Vector constructor
Vector<TYPE>::Vector(long i, int stat)
{
    Allocate(i, stat);
}

template <class TYPE> // Vector copy constructor: copies status
Vector<TYPE>::Vector(const Vector<TYPE> &vec)
{
    Allocate(0, vec.status);
    Copy(vec);
}

template <class TYPE> // operator= : vl = v2 (doesn't copy status)
Vector<TYPE>& Vector<TYPE>::operator =(const Vector<TYPE>& vec)
{
    Copy(vec);
    return *this;
}

template <class TYPE> // operator= : vl = TYPE
Vector<TYPE>& Vector<TYPE>::operator =(TYPE value)
{
    for (long i=1; i<=size; i++)
        v[i] = value;
    return *this;
}

template <class TYPE> // operator- : -vl
Vector<TYPE> Vector<TYPE>::operator -() const
{
    Vector<TYPE> temp(size, TRUE);
    for (long i=1; i<=size; i++)
        temp.v[i] = -v[i];
    return temp;
}

template <class TYPE> // operator(): reallocate memory
Vector<TYPE>& Vector<TYPE>::operator ()(long i)
{
    if (size==i) // no need to reallocate
        return *this;
    Vector<TYPE> temp(i, TRUE);
    *this = temp;
    return *this;
}

template <class TYPE> // find the norm of the vector
double Vector<TYPE>::Norm()
{
    long i;
double ans=0.0;
for (i=1;i<=size;i++)
    ans += v[i]*v[i];
return sqrt(ans);

template <class TYPE>  //Add: this=v1 + v2
void Vector<TYPE>::Add(Vector<TYPE>& v1,Vector<TYPE>& v2)
{
    if (size!=v1.size&&size!=v2.size)
        ErrorFunc("Dimension error in Vector Add");
    long i;
    for (i=1;i<=size;i++)
        v[i]=v1.v[i]+v2.v[i];
}

template <class TYPE>  //Mult: this=c1*this
void Vector<TYPE>::Mult(TYPE c1)
{
    long i;
    for (i=1;i<=size;i++)
        v[i]*=c1;
}

template <class TYPE>  //Cross Product: this=v1 x v2
void Vector<TYPE>::CrossProd(Vector<TYPE>& v1,Vector<TYPE>& v2)
{
    if (size!=3||v1.size!=3||v2.size!=3)
        ErrorFunc("Dimension error in Vector cross product");
}

// other Vector class functions

//ReAllocate memory: all extra vars are made equal to zero
template <class TYPE>
Vector<TYPE>& Vector<TYPE>::ReAlloc(long i,TYPE var)
{
    if (size==i)  //no need to reallocate
        return *this;
    Vector<TYPE> temp(i,TRUE);
    if (i>size){  // need to copy all elements to new mem.
        long ii;
        for (ii=0;ii<=size;ii++)
            temp.v[ii]=v[ii];
        for (ii=size+1;ii<temp.size;ii++)
            temp.v[ii]=var;
    }
    *this=temp;
    return *this;
}

template <class TYPE>  //SubVec: create a subvector
Vector<TYPE> Vector<TYPE>::SubVec(long start, long end,
int unique, long stride)
{
    if (unique){
        double len = (double)(end-start)/stride;
        if (end>size || start<1 || len!=(double)((end-start)/stride))
            VectorError("Dimension error in SubVec\n");
        Vector<TYPE> temp((int)len+1,TRUE);
        for (long i=0; i<=temp.size; i++)
            temp.v[i] = v[start+(i-1)*stride];
        return temp;
    }
if(end > size)
    VectorError("Dimension error in SubVec\n");
Vector<TYPE> temp(0, TRUE);
temp.Unbind();
temp.rep = rep;
temp.v = v+start-1;
rep->refcount++;
temp.size = end-start+1;
return temp;
}

template <class TYPE> // copy element i1 to i2 from vec to this starting at thisi
    // if i2=0 then copy rest of vec to this
void Vector<TYPE>::CopyElements(long thisi, Vector<TYPE>& vec, long i1, long i2)
{
    long i, temp;
    if(i2 <= 0) // if need to copy all of vec to *this
        temp = vec.size-i1;
    else
        temp = i2-i1;
    if(temp < 0 || i1 > vec.size || i2 > vec.size || thisi + temp > size)
        ErrorFunc("Dimension error in CopyElements");
    for(i = 0; i <= temp; i++)
        *(v+thisi+i) = *(vec.v+i1+i);
}

template <class TYPE> // make a row of a matrix equal to a vector
void Vector<TYPE>::RowOfMtx(Matrix<double>& m, long i)
{
    long j;
    if(i <= 0 || i > m.nrow)
        ErrorFunc("Dimension error in RowOfMtx");
    if(size != m.ncol) // must reallocate memory
        Vector<TYPE>::operator()(m.ncol);
    for(j = 1; j <= size; j++)
        v[j] = m[i][j];
}

template <class TYPE> // make a row of a matrix equal to a vector
void Vector<TYPE>::RowOfMtx(Matrix<long>& m, long i)
{
    long j;
    if(i <= 0 || i > m.nrow)
        ErrorFunc("Dimension error in RowOfMtx");
    if(size != m.ncol) // must reallocate memory
        Vector<TYPE>::operator()(m.ncol);
    for(j = 1; j <= size; j++)
        v[j] = m[i][j];
}

template <class TYPE> // Show: display vector on console
void Vector<TYPE>::Show(long i)
{
    if(i < 0 || i > size)
        i = size;
    cout << "Vector size " << size << "\n" << "[ ";
    for (long ii = 1; ii <= i; ii++)
        cout << v[ii] << " ";
    cout << "\n";
}

// Matrix class functions

template <class TYPE> // Copy: copy one matrix to another
void Matrix<TYPE>::Copy(const Matrix<TYPE>& m)
nrow = m.nrow;
ncol = m.ncol;
//rowstride=m.rowstride;
colstride=m.colstride;
data = m.data;
}

// Matrix constructor

template <class TYPE> // Matrix constructor
Matrix<TYPE>::Matrix(long r, long c, int stat)
: data(r*c,stat)
{
  nrow = r;
  ncol = c;
  //rowstride=1;
colstride=c;
}

template <class TYPE> // Matrix copy constructor
Matrix<TYPE>::Matrix(const Matrix<TYPE>& m)
{
data.status = m.data.status;
Copy(m);
}

template <class TYPE> // Matrix operator= : m1 = m2
Matrix<TYPE>& Matrix<TYPE>::operator =(const Matrix<TYPE>& m)
{
  Copy(m);
  return *this;
}

template <class TYPE> // Matrix operator= : Matrix=SparSym
Matrix<TYPE>& Matrix<TYPE>::operator =(SparSym<TYPE>& m)
{
  int i,j;
  Matrix<TYPE> temp(m.n,m.n,TRUE);
  for(i=1;i<=temp.nrow;i++)
    for(j=1;j<=temp.ncol;j++)
      temp[i][j]=m(i,j);
  *this = temp;
  return *this;
}

// Matrix operator= : m1 = TYPE
Matrix<TYPE>& Matrix<TYPE>::operator =(TYPE value)
{
data = value;
return *this;
}

//operator- : -m1
Matrix<TYPE> Matrix<TYPE>::operator -() const
{
  Matrix<TYPE> temp(nrow, ncol, TRUE);
temp.data = -data;
return temp;
}

//operator() : reallocate memory
Matrix<TYPE>& Matrix<TYPE>::operator ()(long i, long j)
{
  if(i==nrow & j==ncol) //no need to reallocate memory
    return *this;

169
template <class TYPE>  //ReAllocate memory and copy elements
Matrix<TYPE>& Matrix<TYPE>::ReAlloc(long i,long j,TYPE var)
{
    if(i==nrow & j==ncol)  //no need to reallocate memory
        return *this;
    Matrix<TYPE> temp(i,j,TRUE);
    if(i>=nrow&&j>=ncol)
    {
        long ii,jj;
        for(ii=l;ii<=nrow;ii++)
            for(jj=ncol;jj<=ncol;jj++)
                temp[ii][jj]=Matrix<TYPE>::operator [](ii)[jj];
    }
    if(temp.ncol>ncol)
    {
        for(ii=1;ii<=temp.nrow;ii++)
            for(jj=ncol+1;jj<=temp.ncol;jj++)
                temp[ii][jj]=var;
    }
    for(ii=nrow+1;ii<=temp.nrow;ii++)
        for(jj=ncol;jj<=ncol;jj++)
            temp[ii][jj]=var;
    *this=temp;
    return *this;
}

template <class TYPE>  // return submatrix of a matrix
Matrix<TYPE> Matrix<TYPE>::SubMtx(long ri, long rf, long ci, long cf)
{
    if(ri<l || ri>nrow || ci<l || ci>ncol)
        ErrorFunc("Dimensioning error in SubMtx");
    Matrix<TYPE> m(rf-ri+1,cf-ci+l,TRUE);
    for(long i=1;i<=m.nrow; i++)
        for(j=i; j<=m.ncol; j++)
            m[i][j] = Matrix<TYPE>::operator []((ri+i-1)+(ci+j-l));
    return m;
}

template <class TYPE>  //CopyRows: copy rows mi-mj to rows thisi of *this mtx
void Matrix<TYPE>::CopyRows(long thisi, Matrix<TYPE> &m, long mi, long mj)
{
    long i,temp=mj-mi;
    if(ncol!=m.ncol||thisi>nrow||mi>m.nrow||mj>m.nrow||thisi+temp>nrow)
        ErrorFunc("Dimension error in CopyRows");
    if(mi<mi)
    {
        for(i=1;i<=ncol;i++)
            *(data.v+((thisi-1)*colstride+i)="*(m.data.v+(mi-1)*colstride+i);
    }
    else
    {
        temp=(temp+1)*ncol;
        for(i=1;i<=temp;i++)
            *(data.v+((thisi-1)*colstride+i)="*(m.data.v+(mi-1)*colstride+i);
    }
}

template <class TYPE>  //Show: display Matrix on console
void Matrix<TYPE>::Show(long ii, long jj)
{
    if(ii<=0&&jj<=0)
        cout "Matrix size " << nrow " x " << ncol << "\n"
```cpp
for (i=0; i<=nrow-1; i++)
    cout << (i+1) << " : ";
for (j=1; j<=ncol; j++)
    cout << data.v[i*colstride+j] << " ";
cout << "\n";
}
cout << "]"; 
else if(ii<nrow&&jj<ncol){
    long i, j;
    if(jj<0)
        jj=ncol;
    for (i=0; i<ii-1; i++)
        cout << (i+1) << " : ";
    for (j=1; j<=jj; j++)
        cout << data.v[i*colstride+j] << " ";
cout << "\n";
}
}
else
    cout<<"Dimension error in Show()\n";

// SparSym class functions

template <class TYPE> // Copy: copy one SparSym matrix to another
void SparSym<TYPE>::Copy(const SparSym<TYPE>& m)
{
    n = m.n;
    b = m.b;
    data = m.data;
}

template <class TYPE> // SparSym constructor
SparSym<TYPE>::SparSym(long nl, long bl, int stat)
    : data(nl*bl,stat)
{
    dummy = 0;
    n = nl;
    b = bl;
}

template <class TYPE> // SparSym copy constructor
SparSym<TYPE>::SparSym(const SparSym<TYPE>& m)
{
    dummy = 0;
    data.status = m.data.status;
    Copy(m);
}

template <class TYPE> // SparSym operator= : m1 = m2
SparSym<TYPE>& SparSym<TYPE>::operator=(const SparSym<TYPE>& m)
{
    Copy(m);
    return *this;
}

template <class TYPE> // SparSym operator= : m1 = TYPE
SparSym<TYPE>& SparSym<TYPE>::operator=(TYPE value)
{
    data = value;
    return *this;
}

template <class TYPE> // SparSym operator- : -m1
```
template <class TYPE> SparSym<TYPE>::operator -() const
    { SparSym<TYPE> temp(n, b, TRUE);
      temp.data = -data;
      return temp;
    }

template <class TYPE> // Alloc: reallocate memory
SparSym<TYPE>& SparSym<TYPE>::Alloc(long nl, long bl)
    { SparSym<TYPE> temp(nl,bl,TRUE);
      *this = temp;
      return *this;
    }

template <class TYPE> // Show: display SparSym matrix on console
void SparSym<TYPE>::Show()
    { //TYPE operator()(long i, long c);
      cout << "Matrix size " << n << " x " << n << "\n" << "[ ";
      long i, j;
      for (i=1; i<=n; i++)
      { cout << i << ": ";
        for (j=1; j<=n; j++)
        { cout << SparSym<TYPE>::operator[](i,j) << " ";
          cout << "\n";
        }
        cout << "\n";
      }
    }

// SparVec functions *****************************************

template <class TYPE,class T>
void SparVec<TYPE,T>::Copy(const SparVec<TYPE,T>& vec)
    { n = vec.n;
      sv = vec.sv;
      indx = vec.indx;
    }

template <class TYPE,class T> // SparVec constructor
SparVec<TYPE,T>::SparVec(T spar_n1, T nl, int stat)
    : sv(spar_n1,stat), indx(spar_n1,stat)
    { if(spar_n1>nl) ErrorFunc("Sizing error in SparVec");
      dummy = 0;
      n = nl;
    }

template <class TYPE,class T> // SparVec copy constructor
SparVec<TYPE,T>::SparVec(const SparVec<TYPE,T>& vec)
    { dummy = 0;
      sv.status = vec.sv.status;
      indx.status = vec indx.status;
      Copy(vec);
    }

template <class TYPE,class T> // SparVec copy constructor
void *SparVec<TYPE,T>::operator new(size_t, void *p)
    { return p;
    }

template <class TYPE,class T>
void SparVec<TYPE,T>::Show()
I
T i;
cout << "SparVec size " << n << " stored as " << sv.size << "\n" << "[ ";
for (i = 1; i <= n; i++)
    cout << SparVec<TYPE,T>::operator[](i) << " ";
cout << " ]\n";
}

template <class TYPE, class T>
TYPE & SparVec<TYPE,T>::operator[](T ii)
{
    for (T i = 1; i <= sv.size; i++)
        if (indx[i] == ii)
            return sv[i];
    return dummy;
}

template <class TYPE, class T> // operator() : reallocate memory
SparVec<TYPE,T> & SparVec<TYPE,T>::operator()(T i, T j)
{
    SparVec<TYPE,T> temp(i,j,TRUE);
    *this = temp;
    return *this;
}

template <class TYPE, class T> // copy a SparVec to a SparVec
SparVec<TYPE,T> & SparVec<TYPE,T>::operator=(const SparVec<TYPE,T> &vec)
{
    Copy(vec);
    return *this;
}

template <class TYPE, class T> // copy a Vector to a SparVec
SparVec<TYPE,T> & SparVec<TYPE,T>::operator=(const Vector<TYPE> &vec)
{
    T i, counter = 0;
    for (i = 1; i <= vec.size; i++)
        if (vec[i] != (TYPE)0)
            counter++;
    SparVec<TYPE,T> temp(counter, (T)vec.size, TRUE);
    counter = 1;
    for (i = 1; i <= vec.size; i++)
        if (vec[i] != 0)
            temp.indx[counter] = i;
        temp.sv[counter] = vec[i];
        counter++;
    *this = temp;
    return *this;
}

template <class TYPE, class T> // SparVec operator-
SparVec<TYPE,T> SparVec<TYPE,T>::operator-() const
{
    SparVec<TYPE,T> temp(sv.size,n,TRUE);
    temp.sv = -sv;
    temp.indx = indx;
    return temp;
}

template <class TYPE, class T> // Destructor used by SparMtx
void SparVec<TYPE,T>::Destruct()
{
    sv.Destruct();
}
indx.Destruct();

// Sparse Matrix class functions  **************************************************

template <class TYPE,class T>  // SparMtx allocation
void SparMtx<TYPE,T>::Allocate(T ro)
{
    T i;
    row=ro;
    vp[row];
    for(i=0;i<=ro;i++)
        new(&vp[i]) SparVec<TYPE,T>;
}

template <class TYPE,class T>  // SparMtx Copy
void SparMtx<TYPE,T>::Copy(const SparMtx<TYPE,T> &m)
{
    T i;
    if(row!=m.row)
    {
        Destruct();
        Allocate(m.row);
    }
    for(i=0;i<=row;i++)
        vp[i] = m.vp[i];
}

template <class TYPE,class T>  // SparMtx constructor
SparMtx<TYPE,T>::SparMtx(T ro, int stat)
{
    Allocate(ro);
}

template <class TYPE,class T>  // SparMtx destructor
SparMtx<TYPE,T>::~SparMtx()
{
    Destruct();
}

template <class TYPE,class T>  // copy SparMtx to SparMtx
SparMtx<TYPE,T> & SparMtx<TYPE,T>::operator =(const SparMtx<TYPE,T> &m)
{
    Copy(m);
    return *this;
}

template <class TYPE,class T>  // copy Matrix to SparMtx
SparMtx<TYPE,T> & SparMtx<TYPE,T>::operator =(const Matrix<TYPE> &m)
{
    T i,j;
    Vector<TYPE> temp(m.ncol);
    if(row!=m.nrow){
        Destruct();
        Allocate(m.nrow);
    }
    for(i=1;i<=row;i++)
        for(j=1;j<=m.ncol;j++)
        {
            temp[j]=m[i][j];
            vp[i]=temp;
        }
    return *this;
}

template <class TYPE,class T>  // reallocate SparMtx
SparMtx<TYPE,T> & SparMtx<TYPE,T>::operator ()(T i)
{
    Destruct();
}
late

template <class TYPE, class T>  // SparMtx destruction
void SparMtx<TYPE, T>::Destruct()
{
    T i;
    for (i=0; i<=row; i++)
        vp[i].Destruct();
}

template <class TYPE, class T>  // Show SparMtx
void SparMtx<TYPE, T>::Show()
{
    T i, j;
    for (i=1; i<=row; i++)
    {
        cout << i << " : ";
        for (j=1; j<=vp[i].n; j++)
            cout << vp[i][j] << " ; ";
        cout << "\n";
    }
}

template <class TYPE, class T>  // Show SparMtx SparVec at a time
void SparMtx<TYPE, T>::ShowStore()
{
    T i;
    for (i=1; i<=row; i++)
        vp[i].Show();
}

template <class TYPE, class T>  // Return # of elements being stored
long SparMtx<TYPE, T>::Storage(char out)  // if out!=0 then display
{
    T i;
    long counter=0;
    for (i=1; i<=row; i++)
        counter += vp[i].indx.size;
    if (out!=0)
        cout << "SparMtx storage = " << counter << " elements for " << row << " x " << vp[1].n << "\n";
    return counter;
}

// Overloaded operator functions

// Vector addition
template <class TYPE>  // operator+ : vl + v2
Vector<TYPE> operator+(const Vector<TYPE>& vecl, const Vector<TYPE>& vec2)
{
    if (vecl.size != vec2.size)
        ErrorFunc("Wrong dimensions in Vector addition\n");
    Vector<TYPE> temp(vecl.size, TRUE);
    for (long i=1; i<=vecl.size; i++)
        temp.v[i] = vecl.v[i] + vec2.v[i];
    return temp;
}

// Vector subtraction
template <class TYPE>  // operator- : vl - v2
Vector<TYPE> operator-(const Vector<TYPE>& vecl, const Vector<TYPE>& vec2)
{
    if (vecl.size != vec2.size)
        ErrorFunc("Wrong dimensions in Vector subtraction\n");
    Vector<TYPE> temp(vecl.size, TRUE);
// Vector*Scalar multiplication

template <class TYPE> //operator* : value*v1
Vector<TYPE> operator *(TYPE value, const Vector<TYPE>& vec)
{
    Vector<TYPE> temp(vec.size, TRUE);
    for (long i=1; i<=vec.size; i++)
    temp.v[i] = vec.v[i]*value;
    return temp;
}

// Vector dot product

template <class TYPE> //operator* : v1*v2; dot product
TYPE operator *(const Vector<TYPE>& vec1, const Vector<TYPE>& vec2)
{
    if (vec1.size != vec2.size)
        ErrorFunc("Wrong dimensions in vector operator* dot product\n");
    TYPE temp=0;
    for (long i=1; i<=vec1.size; i++)
    temp += vec1.v[i]*vec2.v[i];
    return temp;
}

// Matrix addition

template <class TYPE> //operator+ : m1 + m2
Matrix<TYPE> operator +(const Matrix<TYPE>& m1, const Matrix<TYPE>& m2)
{
    if (m1.nrow!=m2.nrow || m1.ncol!=m2.ncol)
        ErrorFunc("Wrong dimensions in Matrix addition\n");
    Matrix<TYPE> temp(0, 0, TRUE);
    temp.data = m1.data + m2.data;
    temp.nrow = m1.nrow;
    temp.ncol = m1.ncol;
    temp.colstride = m1.colstride;
    return temp;
}

// Matrix*Scalar multiplication

template <class TYPE> //operator* : value*m
Matrix<TYPE> operator *(TYPE value, const Matrix<TYPE>& m)
{
    Matrix<TYPE> temp(0, 0, TRUE);
    temp.data = value*m.data;
    temp.nrow = m.nrow;
    temp.ncol = m.ncol;
    temp.colstride = m.colstride;
    return temp;
}

template <class TYPE> //operator* : m*value
Matrix<TYPE> operator *(const Matrix<TYPE>& m, TYPE value)
{
    Matrix<TYPE> temp(0, 0, TRUE);
    temp.data = value*m.data;
    temp.nrow = m.nrow;
}
Matrix multiplication

```cpp
template <class TYPE>
Matrix<TYPE> operator *(const Matrix<TYPE>& m1, const Matrix<TYPE>& m2)
{
    if (m1.ncol != m2.nrow)
        ErrorFunc("Dimensioning error in matrix multiplication");
    Matrix<TYPE> temp(m1.nrow, m2.ncol, TRUE);
    long i, j, k;
    temp = 0;
    for (i = 1; i <= m1.nrow; i++)
    {
        for (j = 1; j <= m2.ncol; j++)
        {
            for (k = 1; k <= m1.ncol; k++)
            {
                temp[i][j] += m1[i][k] * m2[k][j];
            }
        }
    }
    return temp;
}
```

Matrix*Vector multiplication

```cpp
template <class TYPE>
Vector<TYPE> operator *(Matrix<TYPE>& m, Vector<TYPE>& vec)
{
    if (m.ncol != vec.size)
        ErrorFunc("Dimensioning error in matrix*vector multiplication");
    Vector<TYPE> temp(m.nrow, TRUE);
    long i, k;
    temp = 0;
    for (i = 1; i <= m.nrow; i++)
    {
        for (k = 1; k <= m.ncol; k++)
        {
            temp[i] += m[i][k] * vec[k];
        }
    }
    return temp;
}
```

Vector*Matrix multiplication

```cpp
template <class TYPE>
Vector<TYPE> operator *(Vector<TYPE>& vec, Matrix<TYPE>& m)
{
    if (vec.size != m.nrow)
        ErrorFunc("Dimensioning error in vector*matrix multiplication");
    Vector<TYPE> temp(m.ncol, TRUE);
    long j, k;
    temp = 0;
    for (j = 1; j <= m.ncol; j++)
    {
        for (k = 1; k <= m.nrow; k++)
        {
            temp[j] += vec[k] * m[k][j];
        }
    }
    return temp;
}
```

SparSym*Vector multiplication

```cpp
template <class TYPE>
Vector<TYPE> operator *(const SparSym<TYPE>& ml, const Vector<TYPE>& vl)
{
    if (vl.size != ml.n)
        ErrorFunc("Dimensioning error in sparsym*vector multiplication");
    Vector<TYPE> temp(ml.n, TRUE);
    long i, j;
    temp = 0;
    for (i = 1; i <= ml.n; i++)
    {
        for (j = i - ml.b + 1; j < i; j++)
        {
            if (j <= 0)
                temp[j] += ml[i][k] * vl[k];
        }
    }
    return temp;
}
```
template <class TYPE, class T> // operator* : svec*vec;
TYPE operator *(const SparVec<TYPE, T>&svl, const Vector<TYPE>&vec)
{
    Vector<TYPE> temp(svl.row, TRUE);
    T i;
    for (i = 1; i <= svl.row; i++)
    { temp[i] = svl.vp[i].n; temp[i] *= vec[i]; }
    return temp;
}
}

template <class TYPE, class T> // operator* : svec*svec:
TYPE operator *(const SparVec<TYPE, T>&svl, const SparVec<TYPE, T>&sv2)
{
    return svl.vp*sv2.vp;
}

template <class TYPE, class T> // operator* : SparMtx*vec
Vector<TYPE> operator *(const SparMtx<TYPE, T>&sm, const Vector<TYPE>&vec)
{
    Vector<TYPE> temp(sm.vp[1].n, TRUE);
    T i, j,
    for (i = 1; i <= sm.vp[1].sv.size; i++)
    { for (j = 1; j <= sm.vp[1].sv[i].sv.size; j++)
        temp[sm.vp[1].sv[i].j][j] *= sm.vp[1].sv[i][j] * vec[i];
    }
    return temp;
}

// Vector output using <<
template <class TYPE>
ostream& operator <<(ostream& stream, const Vector<TYPE>& vec)
{
    for (long i = 1; i <= vec.size; i++)
    { stream << vec[i] << " ";
        stream << "\n";
    }
    return stream;
}

// Matrix output using <<
template <class TYPE>
ostream& operator <<(ostream& stream1, const Matrix<TYPE>& m)
{
    long i, j;
for (i=1; i<=m.nrow; i++)
    for (j=1; j<=m.ncol; j++)
        streaml << m[i][j] << " ";
    streaml << "\n";
}
return streaml;

// other template functions for vectors and matrices

calculate the Euclidian norm (distance) between two points

template <class TYPE> // represented by vectors
double CalcDist(const Vector<TYPE>& v1, const Vector<TYPE>& v2)
{
    if (v1.size!=v2.size)
        ErrorFunc("Dimension error in CalcDist - 1");
    long i;
    double sum=0.0;
    for (i=1; i<=v1.size; i++)
        sum+=(v1[i]-v2[i])*(v1[i]-v2[i]);
    return sqrt(sum);
}

calculate the Euclidian norm (distance) between two points

template <class TYPE> // represented by j-th row of a mtx and a vector
double CalcDist(const Matrix<TYPE>& m, const long j, const Vector<TYPE>& v)
{
    if (m.ncol!=v.size)
        ErrorFunc("Dimension error in CalcDist - 2");
    long i;
    double sum=0.0;
    for (i=1; i<=v.size; i++)
        sum+=(m[j][i]-v[i])*(m[j][i]-v[i]);
    return sqrt(sum);
}

calculate the Euclidian norm (distance) between two points

template <class TYPE> // represented by i-th row of m1 j-th row of m2
double CalcDist(const Matrix<TYPE>& m1, const long i, const Matrix<TYPE>& m2, const long j)
{
    if (m1.ncol!=m2.ncol)
        ErrorFunc("Dimension error in CalcDist - 3");
    long ii;
    double sum=0.0;
    for (ii=1; ii<=m1.ncol; ii++)
        sum+=(m1[i][ii]-m2[j][ii])*(m1[i][ii]-m2[j][ii]);
    return sqrt(sum);
}

// Solve routine for Ax = b ie. x = A^(-1)*b

template <class TYPE>
Vector<double> SolveMtx(const Matrix<TYPE>& min, const Vector<TYPE>& b)
{
    // solves AX=B using ludcmp, and lubksb
    if (min.nrow!=min.ncol || min.nrow!=b.size)
        ErrorFunc("Dimension error in SolveMtx");
    long i;
    Vector<long> indx(b.size, TRUE);
    Vector<double> x=b;
    double d;
    Matrix<double> a=min;
    a.Status() = TRUE;          // make 'a' a temporary matrix
    x.status = TRUE;            // make 'x' a temporary vector
    ludcmp(a, indx, d);
    lubksb(a, indx, x);
    return x;
}
VecPtr functions:

template <class TYPE> // Allocate: memory allocation
void VecPtr<TYPE>::Allocate(long i, int stat)
{
    status = stat;
    size = i;
    rep = new(i) VecRep<TYPE>;
    v = (TYPE *) (char *) rep + sizeof (VecRep<TYPE>);
}

template <class TYPE> // Bind: binds this vector to data of vec
void VecPtr<TYPE>::Bind(const VecPtr<TYPE>& vec)
{
    rep = vec.rep;
    v = vec.v;
    rep->refcount++;
}

template <class TYPE> // Unbind: unbinds this vector from shared data
void VecPtr<TYPE>::Unbind()
{
    rep->refcount--;
    if (rep->refcount == 0)
        delete rep;
}

template <class TYPE> // Copy: copy one vector to another
void VecPtr<TYPE>::Copy(const VecPtr<TYPE>& vec)
{
    if (!vec.status){ // if one to be copied not a tempvec
        if (size != vec.size){
            Unbind();
            Allocate(vec.size);
        }
        for (long i=0; i<=size; i++)
            v[i] = vec.v[i];
    }
    else{
        Unbind();
        size = vec.size;
        Bind(vec);
    }
}

template <class TYPE> // VecPtr constructor
VecPtr<TYPE>::VecPtr(long i, int stat)
{
    Allocate(i, stat);
}

template <class TYPE> // VecPtr copy constructor: copies status
VecPtr<TYPE>::VecPtr(const VecPtr<TYPE> &vec)
{
    Allocate(0, vec.status);
    Copy(vec);
}

template <class TYPE> // operator= : v1 = v2 (doesn't copy status)
VecPtr<TYPE>& VecPtr<TYPE>::operator=(const VecPtr<TYPE>& vec)
{
    Copy(vec);
    return *this;
}

template <class TYPE> // operator() : reallocate memory
1334 VecPtr<TYPE>& VecPtr<TYPE>::operator ()(long i)
1335 {
1336   VecPtr<TYPE> temp(i, TRUE);
1337   *this = temp;
1338   return *this;
1339 }
1340
1341