MODELLING AND SIMULATION OF COMBINED LUMPED AND DISTRIBUTED PROCESSES

by

Min Oh

March 1995

A thesis submitted for the degree of Doctor of Philosophy of the University of London and for the Diploma of Membership of the Imperial College

Department of Chemical Engineering and Chemical Technology
Imperial College of Science, Technology and Medicine
London SW7 2BY

BIBL.
LONDIN.
UNIV.

To Ae Suk and Dong Hoon

Curiosity is life. If you're not curious you're in your coffin.

– Pierre Boulez

Abstract

Dynamic simulation serves an important role in analysing and predicting the time transient behaviour of chemical processes, and is therefore used at all stages of process design and operation. Most existing dynamic modelling and simulation tools are primarily suited to lumped parameter systems. However, a very significant number of unit operations in chemical and biochemical processes are distributed in nature, with their properties exhibiting spatial as well as temporal variations. Integrated processes can therefore be viewed as combined lumped and distributed parameter systems.

The work described in this thesis identifies a number of key issues regarding the development of general modelling languages for combined lumped/distributed parameter processes, and proposes one such language. The associated problems of solving the underlying mathematical problem and visualising the results are also considered.

The proposed language has been implemented within the *gPROMS* (general **PRO**cess Modelling System) package. It permits the mathematical modelling and dynamic simulation of arbitrarily complex unit operations that are distributed over rectangular domains and which can be described in terms of mixed systems of integral, partial and ordinary differential, and algebraic equations (IPDAEs). Following the translation, validation and analysis of process models expressed in the proposed language, our implementation employs the method of lines to convert the IPDAEs to mixed sets of ordinary differential and algebraic equations (DAEs). These are then solved simultaneously with the DAEs that model the behaviour of the lumped parameter units in the process.

Several examples are presented to illustrate the capability of the proposed modelling system to deal with a wide variety of complex combined lumped/distributed processes. In addition to spatially distributed operations, we consider processes involving distributions over other properties such as particle size.

Acknowledgements

I would like to thank Dr. C.C. Pantelides for his intellectual supervision throughout this project. His deep knowledge of mathematics and process engineering has inspired me to broaden my horizons.

Like many other big projects, lots of people have directly or indirectly been involved in the *gPROMS* project. Among those, I wish to thank Edward Smith for his valuable comments and discussions about various aspects of this project. Visualisation in terms of 3D graphics is very important to analyse the simulation results of distributed parameter processes. Therefore, I want to express my gratitude to Tom Björkholm for the implementation of *gRMS*.

It would be unthinkable to build a reliable and sophisticated software system without keen users. I, therefore, wish to acknowledge their contributions, in the form of comments and feed-back, which have played a significant role in this project. Especially, I am grateful to Nouri Samsatli and Sujinda Nilchan for their friendship as well as providing me with the mathematical models for sterilisation and pressure swing adsorption processes which are used in chapter 7.

I want to express my gratitude to Lyoong and Young for their sincere friendship which I shall always remember. Their presence made me happy even when I was in trouble.

My father had a big cancer operation during my work here, which he has tried to hide in order for me to concentrate on my work. I hope my achievement will be a little consolation for him.

Finally, I would like to thank my wife and son for their love and understanding, which always give me peace of mind and confidence in life. Without their support and sacrifice, it would have been impossible to finish my PhD work successfully.

Table of Contents

1 Introduction **7**
1.1 Thesis Outline 9

2 Literature Review **11**
2.1 Partial Differential Equations (PDEs) 12
2.1.1 Mathematical Formulation of Problems Involving PDEs 12
2.1.2 Boundary and Initial Conditions 13
2.2 Numerical Methods for PDEs 14
2.2.1 Method of Lines 14
2.2.2 Finite Difference Methods 15
2.2.3 Weighted Residual Methods 15
2.2.4 Finite Element Methods 18
2.2.5 Finite Volume Methods 18
2.2.6 Adaptive Grid Methods 21
2.3 Software Packages for PDEs 22
2.3.1 Programming Language Based PDE Packages 23
2.3.2 High-Level PDE Packages 26
2.4 CFD Packages 29
2.5 Concluding Remarks 33

3 A Modelling Language for Lumped/Distributed Parameter Systems **34**
3.1 The *gPROMS* Process Modelling Tool 35
3.1.1 Lumped Parameter System Modelling 35
3.1.2 Hierarchical Submodel Decomposition 36
3.2 Distribution Domains 37
3.3 Variable Distributions 38
3.4 Equation Distributions 40
3.4.1 Variable Sub-domains 41
3.4.2 Domain Properties for Variables and Variable Occurrences 41
3.4.3 Domain of Distributed Expressions and Equations 42
3.4.4 Explicit Declaration of Equation Distribution Domains 43
3.4.5 The Partial Differentiation Operator 43
3.4.5.1 General Considerations 43
3.4.5.2 Higher Order Partial Derivatives 45
3.4.5.3 Partial Differentiation with Respect to Time 45
3.4.6 The Integral Operator 45
3.4.6.1 Single Integrals 46
3.4.6.2 Multiple Integrals 46
3.4.6.3 Relationship between the INTEGRAL and SIGMA Operators 47
3.5 Specification of Boundary Conditions 48
3.6 Solution Method Specification 49

3.7 Concluding Remarks 51
3.7.1 Summary 51
3.7.2 Possible Extensions for Handling Some Problems with Sharp Fronts 52
3.7.3 Some Comments on Modelling of Time Variations in *gPROMS* . . . 53

4 Solution Methods for IPDAEs **57**
4.1 Finite Difference Methods 58
4.1.1 Approximations for First-Order Derivatives of $\phi(z)$ 59
4.1.2 Approximations for Second-Order Derivatives of $\phi(z)$ 61
4.1.3 Approximations for Integrals of $\phi(z)$ 62
4.1.4 Comments on Consistency, Accuracy and Stability 64
4.1.5 Avoiding the Creation of Fictitious Discretisation Points 65
4.2 Orthogonal Collocation on Finite Elements 66
4.2.1 Approximations for First-Order Derivatives of $\phi(z)$ 68
4.2.2 Approximations for Second-Order Derivatives of $\phi(z)$ 69
4.2.3 Continuity at Element Boundaries 70
4.2.4 Approximations for integrals of $\phi(z)$ 71
4.2.5 Gaussian Quadrature 72
4.3 Concluding Remarks 73

5 Implementation **75**
5.1 *gPROMS* Software Architecture 75
5.1.1 Translator 75
5.1.2 Process Manager 77
5.2 Discretisation of Variables 78
5.3 Discretisation of IPDAEs 79
5.3.1 Symbolic Preprocessing of IPDAEs during Translation 80
5.3.2 IPDAE Discretisation during Model Instantiation 82
5.3.3 First-Order Partial Derivative Approximations 82
5.3.4 Higher Order Partial Derivative Approximations 83
5.3.5 Discretisation of Integrals 84
5.3.5.1 Single Integration 84
5.3.5.2 Multiple Integration 85
5.4 Concluding Remarks 85

6 The Management of Simulation Results in *gPROMS* **88**
6.1 Selective Recording of Results 88
6.1.1 Specifying Variables to Be Monitored 88
6.1.2 Selective Monitoring of Solution Trajectory 90
6.2 The *gPROMS* Results Management System (*gRMS*) 91
6.2.1 Overview 91
6.2.2 Communication between *gPROMS* and *gRMS* 92
6.2.3 Archiving of Simulation Results 93
6.2.4 The *gRMS* User Interface 93
6.3 Concluding Remarks 95

7 Examples of Combined Lumped/Distributed Parameter System Modelling and Simulation **100**
7.1 Process with Mixed Lumped/Distributed Unit Operations 100
7.2 Fixed-bed Catalytic Reactor Immersed in Cooling Jacket 105
7.3 Rapid Pressure Swing Adsorption Process 109
7.4 Media Sterilisation in Biochemical Processes 115
7.5 CMSMPR Crystalliser 122
7.6 Concluding Remarks 126

8 Conclusions **128**
8.1 Summary of this Work 128
8.2 Needs and Directions for Future Work 129
8.2.1 Improved geometry description 129
8.2.2 Problem detection and diagnosis 131
8.2.2.1 Control of discretisation error 131
8.2.2.2 Results storage 132

References **133**

Nomenclature **139**

A The *gPROMS* Modelling Language **143**

B *gPROMS* Input Files for Examples in Chapter 7 **161**
B.1 Process with Mixed Lumped/Distributed Parameter Systems 161
B.2 Fixed-bed Catalytic Reactor Immersed in Cooling Jacket 164
B.3 Rapid Pressure Swing Adsorption 169
B.4 Continuous Media Sterilisation 171
B.5 CMSMPR Crystalliser 179

Chapter 1

Introduction

Dynamic simulation is the activity of analysing and predicting the time transient behaviour of the physical process of interest which is usually described in terms of mathematical equations. Dynamic simulation serves an important role from the early stages of process design to plant commissioning and operation. Applications of dynamic simulation include the synthesis and analysis of control structure, startup and shutdown, safety assessments, optimisation and operator training facilities (Wozny and Jeromin, 1994).

It has long been recognised (Pantelides *et al.*, 1988) that a natural mathematical description of the transient behaviour of lumped parameter processes is in terms of mixed systems of ordinary differential and algebraic equations (DAEs). However, a significant number of unit operations in chemical and biochemical processes take place in distributed parameter systems in which properties vary with respect to one or more space dimensions as well as time. Examples include packed bed tubular reactors, packed bed absorption, adsorption and distillation columns *etc.*

In other types of unit operations, some of the properties of the material are characterised by probability density functions instead of single scalar values. Examples include crystallisation units and polymerisation reactors, in which the size of the crystals and the length of the polymer chains respectively are described in terms of distribution functions. The form of the latter may also vary with both time and spatial position.

In fact, most complex processes typically involve a combination of both distributed and lumped parameter unit operations. However, currently available general-purpose softwares for the modelling and simulation of chemical and biochemical processes are primarily intended for lumped parameter systems (Marquardt, 1991). Several packages, such as *SpeedUp* (Pantelides, 1988), *DIVA* (Kroner *et al.*, 1990), *ASCEND* (Piela *et al.*, 1991) and *OMOLA* (Mattsson and Andersson, 1992) provide modelling languages that allow the transient behaviour of individual unit operations to be described in terms of DAEs. In such modelling packages, distributed parameter systems are currently modelled by manual discretisation of the distributed dimensions, which reduces the mathematical system to a set of DAEs with respect to time. This process is both difficult and error-prone, especially when advanced discretisation techniques are to be applied, and a high level of mathematical

knowledge and skill is required to perform it in a satisfactory manner.

The mathematical description of distributed unit operation models usually involves partial differential equations (PDEs) expressing the physical laws of conservation of mass, energy and momentum. In addition to these, models may involve algebraic relations that characterise phenomena, such as phase equilibria, which operate on much smaller time-scales than those described by PDEs. Algebraic equations (AEs) may also be used to express relationships between variables, such as the definition of enthalpies in terms of temperature, pressure and composition. Finally, population balances (Ramkrishna, 1985) carried out on systems involving properties characterised by probability distributions very often lead to the introduction of integral terms in some of the equations (see, *e.g.* Lee and Saleeby (1994) for a recent example). Overall, then, we are faced with mixed systems of integral, partial differential and algebraic equations (IPDAEs).

Furthermore, the range of boundary conditions encountered in typical process modelling applications is far wider than that usually studied in the mathematical literature. For instance, in addition to the well-known Dirichlet, Neumann and Robin types of boundary conditions, one encounters cyclic or multipoint conditions that establish relationships between the values of the variables at two or more points in the domain of interest.

It is clear from the above discussion that any mathematical modelling language for describing distributed parameter processes must provide effective mechanisms for the definition of general IPDAE systems subject to general boundary conditions – a rather difficult task in itself. However, a further complication arises from the fact that, even within the model of a single unit operation, not all variables necessarily have the same degree of distribution. For instance, some variables may depend on both radial and axial positions, while others may be functions of the axial position only. Therefore, the proposed modelling language must allow the model developer to define explicitly the *distribution domains* that exist within the unit under consideration, and the *degree of distribution* of every system variable with respect to these domains.

This thesis presents a general modelling language for combined lumped and distributed parameter processes which attempts to address some of the issues raised above. It also discusses the implementation of such a language and demonstrates its applicability for modelling a wide range of processes. Attention is restricted to systems that are distributed over rectangular domains; systems with irregular geometry are not considered in this thesis. The starting point for the work described here is the *gPROMS* package (Barton and Pantelides, 1994), a modelling system for complex lumped parameter processes with both

discrete and continuous characteristics. The *gPROMS* modelling language is extended to allow the definition of models of distributed parameter unit operations.

1.1 Thesis Outline

The next chapter reviews the mathematical background and the numerical methods for the solution of PDE system. A very large number of PDE numerical methods have been presented in the literature. However, in this chapter, we only examine numerical methods which are sufficiently general to be widely applicable to a wide variety of PDE problems. We also review earlier software for posing and solving PDE problems.

Process modelling tools such as *gPROMS* consist of two main parts: a modelling language which allows a declarative description of combined lumped and distributed parameter processes; and solution methods to solve such problems. Chapter 3 describes the concept and the structure of the modelling language. An examination of the characteristics of combined lumped and distributed parameter processes introduces the formalism necessary to describe a wide range of process models as well as to allow the user to specify information on the solution strategy.

After the declarative description of IPDAEs, solution methods are to be employed to solve such problems. Chapter 4 discusses in depth two families of numerical methods for IPDAEs, namely finite difference methods and orthogonal collocation methods on finite elements. All formulae of two families are derived within the unified framework using Lagrange polynomials.

Chapter 5 is concerned with the implementation of the ideas presented in chapter 3 and chapter 4 within *gPROMS*.

The simulation of combined lumped/distributed parameter processes often leads to the generation of vast amounts of numerical data, which makes the analysis of the results rather difficult. This is particularly true for problems involving two or more non-temporal dimensions. In chapter 6, a software system for managing and visualising results involving arbitrary number of dimensions is presented.

Chapter 7 demonstrates how the tool described in this thesis makes it possible to tackle complicated distributed process modelling and simulation problems from various applications. The problems cover various types of a mixed set of IPDAEs of complexity sufficient to illustrate the power and effectiveness of the package. Although the examples are mainly from the fields of chemical engineering and biotechnology, the application of the tool should not necessarily be confined to these areas.

Finally, a review of the contributions made by this work along with some suggestions for future work is given in chapter 8.

Chapter 2

Literature Review

This chapter provides a review of the main issues associated with the development of modelling environments for distributed parameter systems, and the relevant literature.

Both mathematical and software aspects are of paramount importance to the current work. This chapter begins with the review of the mathematical background relevant to the project. The next section concentrates on the mathematical formulation of combined lumped/distributed parameter processes. The current state of numerical methods for PDEs still does not provide universal solution methods capable of handling a wide spectrum of PDEs; thus section 2.2 reviews a number of different numerical methods.

The second issue is how to describe distributed problems in a software system. There are currently several software packages which support the formulation and the solution of PDE problems. However, the state of the art of this software is far from ideal. Section 2.3 surveys the current technology, considering various kinds of PDE packages. Following a general discussion, the underlying philosophy and the actual capability of some currently available packages are examined.

Traditionally, the area of Computational Fluid Dynamics (CFD) has focussed considerable attention on the solution of PDE problems representing conservation equations describing fluid flow over domains of often complex geometry. The CFD packages have been developed independently from other PDE packages. One reason for this is that improved computational efficiency may be obtained by exploiting the domain-specific form of the equations being solved, especially in dealing with difficult issues such as strong convective terms and nonlinear sinks/sources. Furthermore, ease of use is also promoted by developing generic "templates" for the limited set of equations arising in CFD applications, with the user configuring a specific problem simply by choosing the terms to be included and their precise form. Finally CFD packages devote substantial attention to the description and handling of complex geometries, an issue of particular importance to CFD applications. In section 2.4, the general concepts and the structure of CFD packages are discussed and one such commercial package is briefly examined.

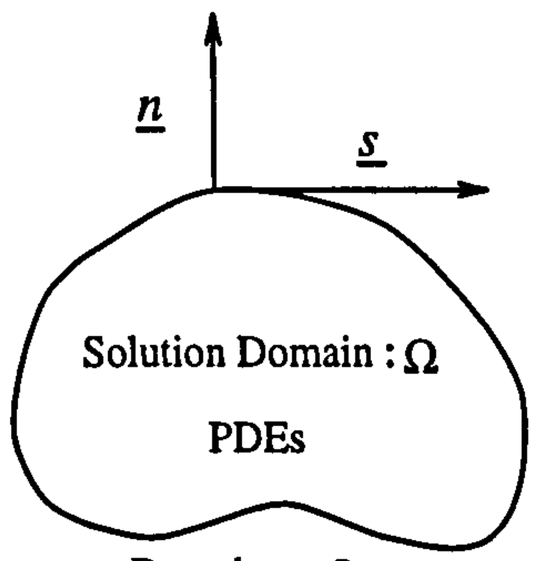


Figure 2.1: Computational domain Ω

2.1 Partial Differential Equations (PDEs)

2.1.1 Mathematical Formulation of Problems Involving PDEs

The mathematical model of a physical process in which properties vary with time as well as one or more spatial dimensions is usually described by PDEs. Consider a set of PDEs expressed over the solution domain Ω and its boundary $\delta\Omega$ as shown in Figure 2.1. If the independent variables (including both time and spatial position) are denoted by x, then the dependent variables ϕ can be written as

$$\phi = \phi(x)$$

where $\phi \in \mathcal{R}^n$, $x \in \mathcal{R}^m$. The partial derivatives are often denoted as follows:

$$(\phi_x)_{ij} = \frac{\partial \phi_i}{\partial x_j}, \qquad (\phi_{xx})_{ijk} = \frac{\partial^2 \phi_i}{\partial x_j \partial x_k}, \qquad etc.$$

Systems of PDEs can be described in terms of equations over the bounded domain Ω:

$$f(\phi, \phi_x, \phi_{xx}, \cdots, x) = 0 \qquad x \in \Omega \tag{2.1}$$

and auxiliary conditions over the boundary $\delta\Omega$:

$$g(\phi, \phi_x, \cdots, x) = 0 \qquad x \in \delta\Omega \tag{2.2}$$

The number of equations $f(\cdot)$ is equal to the number of dependent variables, n. The number of auxiliary conditions cannot generally be related to the number of PDEs. A general rule can be stated, however, for the auxiliary conditions associated with each PDE: the number of auxiliary conditions required for each independent variable equals the highest

order of any partial derivative with respect to the independent variable in the particular PDE. For instance, a PDE involving second-order spatial and first-order temporal partial derivatives requires two auxiliary conditions for the space independent variable and one auxiliary condition for the time independent variable. The former conditions are often referred to as *boundary conditions*, while the latter are called the *initial conditions*.

According to Schiesser (1991), the highest order derivative with respect to each independent variable appearing in the auxiliary condition $g(\cdot) = 0$ must be, at most, one less than the corresponding highest order derivative appearing in the equation $f(\cdot) = 0$. However, this rule is not always correct: *e.g.* in first-order ordinary differential equation (ODE) or DAE systems, the initial condition is often given in terms of the time derivatives, *e.g.* $\phi_t(0) = 0$.

2.1.2 Boundary and Initial Conditions

In general, the solution of PDEs must satisfy certain conditions (initial and boundary conditions) on the boundary of a prescribed domain as shown in equation 2.2. In the mathematical literature, common boundary conditions have traditionally been classified in three groups (Laurie, 1983):

1. Dirichlet boundary conditions : $\phi = g$ on $\delta\Omega$
2. Neumann boundary conditions : $\frac{\partial\phi}{\partial\underline{n}} = g$ or $\frac{\partial\phi}{\partial\underline{s}} = h$ on $\delta\Omega$
3. Robin boundary conditions : $\frac{\partial\phi}{\partial\underline{n}} + k\phi = g, k > 0$ on $\delta\Omega$

where $\underline{n}$ denotes an outward normal vector and $\underline{s}$ a tangential vector with respect to the boundary $\delta\Omega$ at a given point (see Figure 2.1).

Computationally, Dirichlet boundary conditions on $\delta\Omega$ can be applied exactly as long as g is analytic. However, errors are introduced in representing Neumann or Robin boundary conditions numerically. In any case, as we shall see later, typical boundary conditions encountered during modelling of industrial processes are more complicated than this classification suggests.

In some important cases, the boundary of the domain is not known in advance, but has to be determined as part of the solution. Moving boundaries are associated with time-dependent problems, and the position of the boundary has to be determined as a function of time. Moving boundaries occur mostly in heat-flow problems with phase changes and in certain diffusion processes (Crank, 1984).

2.2 Numerical Methods for PDEs

PDEs form an essential part of the mathematical description of physical systems depending on time and space, and consequently arise in an enormous number of modelling applications.

The solution of PDE systems is generally a difficult problem. Changing a parameter or one of the boundary conditions may lead to completely different behaviour from that originally anticipated. Furthermore, although some numerical methods can accurately solve a given PDE system, other numerical methods may be totally unable to do so.

Due to the lack of universally applicable solution methods, some understanding of the various kinds of numerical methods available, their applicability and limitations is essential. In the following sections, some of these numerical methods are reviewed briefly. Among them will be those which are used as the solution methods in our work. These will be discussed in more detail in chapter 4.

2.2.1 Method of Lines

The method of lines (Carver, 1981; Schiesser, 1991) converts PDEs into sets of ODEs with respect to time. It can provide numerical solutions in a wide variety of applications.

The advantage of this procedure is that sophisticated computer programs which permit fast and accurate integration of large sets of ODEs over time can be employed. In particular, integration codes based on backward differentiation formulae (*e.g.* DASSL (Petzold, 1982) and DASOLV (Jarvis and Pantelides, 1992)) can solve stiff as well as non-stiff systems of equations. They utilise sophisticated algorithms for automatic step-size adjustment and integration order selection to maintain a user-specified error tolerance.

The family of the method of lines comprises collectively a number of finite difference, finite element and weighted residual methods in which piecewise local or global approximation functions in the space dimensions are used to convert evolutionary PDE problems into initial value ODE problems.

However, despite the many benefits mentioned above, there is a crucial drawback to this method. The general ODE or DAE solver used for time integration cannot estimate and control the *space* discretisation error (Pipilis, 1990). Especially when coarse spatial grids are employed, this error can be an order of magnitude greater than the time integration error. Unfortunately, there are no general guidelines or rules for determining the number of the discretisation intervals required to obtain both computational efficiency and an accurate

solution. This can usually be established only with a trial-and-error approach.

2.2.2 Finite Difference Methods

In the finite difference approach, the continuous problem domain is discretised in terms of the values of the dependent variables at discrete points only. Derivatives are approximated by finite difference formulae resulting in an algebraic representation of the PDE. The first step to be taken in establishing a finite difference approximation for a given PDE is to map the continuous problem domain into a finite difference grid. In solving a PDE for which $\phi(z,t)$ is a dependent variable in the domain Ω with two independent variables denoted z and t, discrete grids are established on the domain, replacing $\phi(z,t)$ with $\phi(z_i,t_j)$. Discretised equations are usually written in terms of the values of the variables at the general point (z_i,t_j) and its neighbours. For instance, the first order derivative of $\phi(z,t)$ with respect to z at point (z_i,t_j) can be approximated by:

$$\frac{\partial\phi(z_i,t_j)}{\partial z} = \frac{\phi(z_{i+1},t_j)-\phi(z_{i-1},t_j)}{2h} + O(h^2) \tag{2.3}$$

where $h = z_{i+1} - z_i = z_i - z_{i-1}$ and the $O(h^2)$ term represents the truncation error.

Finite difference approximations to partial derivatives can be derived in several different ways, including (Anderson *et al.*, 1992):

1. Taylor series expansions
2. polynomial interpolation (*e.g.* based on Lagrange polynomials)
3. integral method
4. control volume approach.

It is sometimes possible to obtain exactly the same finite difference formulae by using different methods. In particular, the first two methodologies (Taylor's expansion and polynomial interpolation) always yield the same results, as noted by Pfeiffer and Marquardt (1993).

Further discussion on the concept and derivation of finite difference methods will be presented in section 4.1.

2.2.3 Weighted Residual Methods

These assume that the solution can be represented as a weighted combination of known polynomial *functions* with unknown weighting factors (Finlayson, 1980):

$$\hat{\phi}(z,t) = \sum_{k=0}^{n} a_k(t)\theta_k(z) \tag{2.4}$$

where z is the spatial independent variable, t the time independent variable, and $\hat{\phi}(z,t)$ the approximate solution of a given PDE. The approximating (trial) functions, $\theta_k(z)$, are known and the coefficients $a_j(t)$ are to be determined to satisfy the system of equations as well as boundary conditions in some "optimal" sense. The precise criterion used to choose the coefficients determines the method.

Let the partial differential equation be represented by

$$L(\phi(z,t)) = 0 \tag{2.5}$$

where L is a differential operator.

If we combine the above two equations, the resulting equation will be described in terms of the *residual* R_n as follows:

$$R_n(z,t) = L(\hat{\phi}(z,t)) \tag{2.6}$$

where $R_n(z,t)$ is a continuous function of z and t.

The next step is to form the weighted residual and choose a weighting function. The integral of the weighted residual over the domain of interest V is required to be zero, *i.e.*

$$\int_V W_k(z)\ R_n(z,t)\ dz = 0 \qquad k = 0..n \tag{2.7}$$

Different choices of weighting functions $W_k(z)$ determine the different methods in the weighted residual family of methods.

The simplest weighting function is

$$W_k(z) = \begin{cases} 1 & \text{if } z \in V_k \\ 0 & \text{elsewhere} \end{cases} \tag{2.8}$$

where the domain of interest V has been partitioned into a number of disjoint sub-domains ("control volumes") V_k such that

$$V = \bigcup_k V_k\ ;\ \ V_k \bigcap V_{k'} = \emptyset \qquad \forall k \neq k' \tag{2.9}$$

So the resulting weighted residual is,

$$\int_V R_n(z,t)\ dz = \int_{V_k} L(\hat{\phi}(z,t))\ dz = 0 \qquad k = 0..n \tag{2.10}$$

This weighted residual equation implies that the integral of the residual must be zero over each control volume V_k and can often be interpreted as a conservative principle

over $\phi(z,t)$ for the finite control volume. This variant of weighted residual methods is called the *sub-domain method.*

In the *collocation method*, the Dirac delta function is chosen as the weighting function, *i.e.*

$$W_k(z) = \delta(z - z_k) \tag{2.11}$$

where $\{z_k\}$ is a set of distinct points in V. As a result, the collocation method is obtained by setting:

$$\int_V \delta(z - z_k) R_n(z,t) dz = R_n(z_k,t) = 0 \tag{2.12}$$

at the collocation points z_k, where $k = 1..n$.

The *orthogonal collocation method* is often the most efficient way to implement the collocation method. There are three main features of this method (Finlayson, 1980):

1. the trial functions are taken as a series of orthogonal polynomials

2. the collocation points z_k are taken as the roots of one of these polynomials

3. the coefficients $\{a_k\}$ now correspond to the values of the solution at points $\{z_k\}$.

The automatic selection of collocation points avoids the possibility of an arbitrary and often poor choice by the user, and the error decreases much faster as the number of terms increases. The orthogonal collocation points for polynomials of various degrees are reported in various standard references, *e.g.* Villadsen and Michelsen (1978) and Finlayson (1980).

In the *least-squares method*, the weighting function is

$$W_k(z) = \frac{\partial R_n}{\partial a_k} \tag{2.13}$$

This is equivalent to the requirement that the squared residual integral

$$I(z,t) = \int_V R_n^2(z,t) dz \tag{2.14}$$

be minimised with respect to the coefficients a_k.

Alternatively, the trial functions may themselves be used as the weighting functions:

$$W_k(z) = \theta_k(z) \tag{2.15}$$

This so-called *Galerkin method* requires that the residual be orthogonal to each of the trial functions. The latter are usually low-order polynomials.

Spectral methods are similar to the traditional Galerkin method, but use orthogonal functions for both the approximating and the weighting functions (Fletcher, 1988). These methods have become particularly popular for large-scale computations in certain areas of computational fluid dynamics concerned with the study of homogeneous turbulence, transitions in shear flows and compressible flows (Hussaini and Zang, 1987).

Finally, in the *method of moments*, the weighting function is described as follows.

$$W_k(z) = z^{k+1} \tag{2.16}$$

2.2.4 Finite Element Methods

The weighted residual methods described in section 2.2.3 can be applied over the entire spatial domain. In this case, increased accuracy can be obtained only by increasing the order of approximation n (*cf.* equation 2.4) but this may not be appropriate if the true solution cannot be approximated well by a higher-order polynomial. This is especially true with problems exhibiting sharp fronts.

An alternative to such global solution approximations is provided by the *finite element* family of methods. These divide the domain of interest into many smaller subdomains ("elements") and apply weighted residual methods (usually of relatively low orders) within each element. Additional restrictions may be introduced to ensure various degrees of continuity of the solution at the element boundaries.

One advantage of the finite element approach is its ability to handle problems with steep gradients by concentrating many small elements in areas of sharp variation of the solution. Another advantage is its versatility in dealing with irregular domain geometries through the use of appropriate subdivisions of the domain (Zienkiewicz and Morgan, 1983).

In principle, any weighted residual method can be combined with the finite element concept to yield a corresponding finite element method. In practice, the ones most commonly used in this way are the Galerkin finite element method (Strang and Fox, 1973) and the method of orthogonal collocation on finite elements (Carey and Finlayson, 1975; Finlayson, 1980). We return to consider the latter in much more detail in section 4.2.

2.2.5 Finite Volume Methods

The basic idea of the finite volume formulation is easy to understand and lends itself to direct physical interpretation. The calculation domain is divided into a number of non-overlapping control volumes such that there is one control volume surrounding each grid point. The differential equation is integrated over each control volume. Polynomial

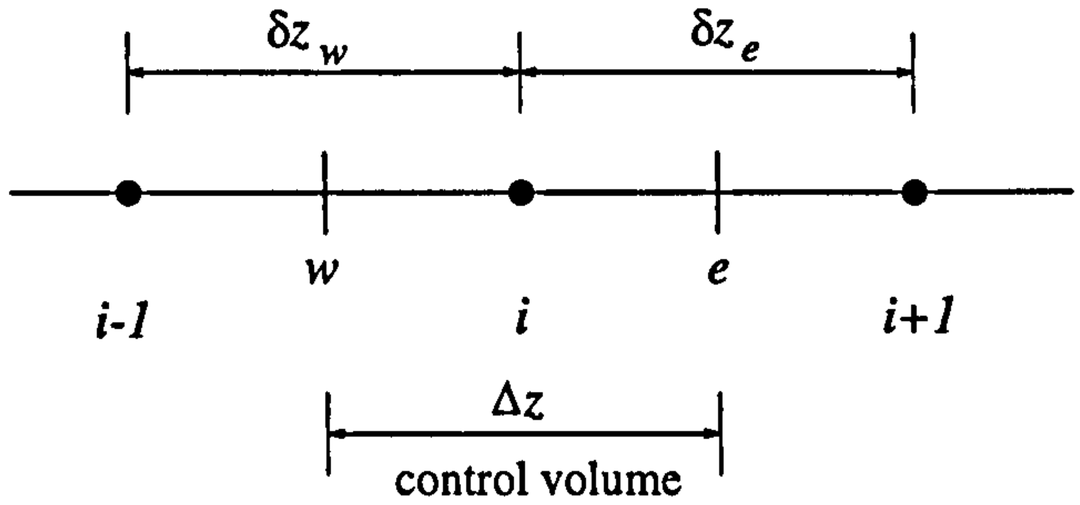


Figure 2.2: Control volume in one dimension

approximations to the dependent variables based on their values at a finite set of points are used to evaluate any required integrals. The result is a *discretisation equation* involving the values of dependent variables for a group of neighbouring points.

The discretisation equation obtained in this manner expresses the *conservation principle* for the dependent variables over the finite control volume, just as the differential equation expresses it for an infinitesimal control volume. The most attractive feature of the finite volume formulation is that the resulting solution ensures that the *integral conservation of quantities* such as mass, momentum and energy is exactly satisfied over any group of control volumes and over the whole computation domain (Patankar, 1980).

In order to illustrate the formulation of a finite volume method, consider the general equation of conservation:

$$\frac{\partial(\rho\phi)}{\partial t} + \nabla \cdot (\rho u \phi) = \nabla \cdot (\Gamma \nabla \phi) + S \tag{2.17}$$

Here $\partial(\rho\phi)/\partial t$ denotes the rate of accumulation of the conserved quantity (energy, momentum, *etc.*), and the term $\nabla \cdot (\rho u \phi)$ is the convective flux. The quantity $\nabla \cdot (\Gamma \nabla \phi)$ is the diffusive flux and S stands for the source term.

For ease of understanding, it is assumed that the variable ϕ is a function of only one independent variable z as shown in Figure 2.2. Therefore, the resulting equation is:

$$\frac{\partial(\rho\phi)}{\partial t} + \frac{\partial(\rho u \phi)}{\partial z} = \frac{\partial}{\partial z}\left(\Gamma \frac{\partial \phi}{\partial z}\right) + S \tag{2.18}$$

Integration of equation 2.18 over the control volume $[w, e]$ shown in Figure 2.2 gives:

$$\begin{aligned} \int_w^e \frac{\partial(\rho\phi)}{\partial t} dz &= -\int_w^e \frac{\partial(\rho u \phi)}{\partial z} dz + \int_w^e \frac{\partial}{\partial z}\left(\Gamma \frac{\partial \phi}{\partial z}\right) dz + \int_w^e S dz \\ &= -\{(\rho u \phi)_e - (\rho u \phi)_w\} + \{\left(\Gamma \frac{\partial \phi}{\partial z}\right)_e - \left(\Gamma \frac{\partial \phi}{\partial z}\right)_w)\} + \overline{S}\Delta z \end{aligned} \tag{2.19}$$

where $\overline{S}$ is the average value of S over the control volume.

The source term S is often a function of the dependent variable ϕ. When S is a nonlinear function of ϕ, a linear approximation is used:

$$S = S_r + \left(\frac{dS}{d\phi}\right)_r (\phi - \phi_r) \tag{2.20}$$

where S_r is the reference value of S and ϕ_r the reference value of ϕ. The reference value of ϕ is the value at the previous iteration. For instance, suppose the source term is expressed by $S = -k\phi^2$ in a second-order reaction, where ϕ is the concentration. Then, S can be approximated by

$$S = -k\phi_r^2 - 2k\phi_r(\phi - \phi_r) \tag{2.21}$$

In order to evaluate the integral on the left hand side and the average source term $\overline{S}$ on the right hand side, we need an approximation to the variation of $\phi(\cdot)$ and its spatial derivatives. These can be of the form:

$$\phi_e = \alpha\phi_i + (1-\alpha)\phi_{i+1} \tag{2.22}$$

$$\left(\frac{\partial\phi}{\partial z}\right)_e = \frac{\beta}{\Delta z}(\phi_{i+1} - \phi_i) \tag{2.23}$$

where ϕ_i and ϕ_{i+1} are the values of ϕ at the grid points (see Figure 2.2). Similar expressions hold for ϕ_w and $(\partial\phi/\partial z)_w$. The coefficients α and β determine the precise form of the finite volume method.

- *Piecewise linear schemes*: Here we assume that the dependent variable ϕ varies linearly between successive grid points. Therefore, $\alpha = 0.5$ and $\beta = 1$. The resulting discretisation is identical to a centered finite difference scheme. For convection dominant problems, this scheme may result in inaccurate, or even unrealistic, solutions.

- *Upwind schemes*: A well-known remedy for the difficulties encountered in convection dominant problems is the use of upwind schemes. This recognises that the weak point in the piecewise linear approximation is the assumption that ϕ_e is the average of ϕ_i and ϕ_{i+1}. The formulation of the diffusion term is left unchanged, but the convection term is calculated from the assumption that the value of ϕ at a control volume interface is equal to the value of ϕ at the *upwind* grid point. Typical values are $\alpha = 1$ and $\beta = 1$.

- *Exponential schemes*: This scheme is derived so as to produce the exact solution of the one-dimensional steady-state problem for any number of grid points. However, despite this highly desirable behaviour, this scheme is not widely used since it is not exact for two- or three- dimensional problems: Typical values are $\alpha = 1$ and $\beta = \frac{Pe}{exp(Pe)-1}$, where Pe is the Peclet number which denotes the ratio of the strengths of convection and diffusion.

2.2.6 Adaptive Grid Methods

The numerical solution of problems exhibiting strong convective phenomena (such as steep moving fronts and shock waves in compressible flow) is usually difficult since the nature, location, and duration of the latter are often not known in advance. Thus, the conventional numerical approaches described in the preceding sections may fail to provide adequate resolution of the fine scale phenomena or, alternatively, may incur excessive computation costs due to the use of a very fine grid covering the entire domain. In some cases, completely incorrect results may be generated.

The idea of using a numerical method in which nodes are positioned automatically only where needed is highly attractive and of increasing importance as a potential remedy for such problems.

There are two basic strategies that can be employed in dynamically adaptive grid methods, namely the spatial *redistribution of a fixed number of points* and the *local grid refinement* (Baines, 1988).

The first approach relies on algorithms for moving the nodes from regions of relatively small error (or solution gradient) to regions of large error (or solution gradient) so as to obtain the best accuracy possible for a given number of nodes.

These algorithms can further be classified into *adaptive* and *moving* grid methods. In the former category (Verwer and Trombert, 1991; Verwer *et al.*, 1989; Arney and Flaherty, 1990) the spatial grid is modified only at distinct time points when necessary. The underlying principle is one of trying to maintain approximate equality among the spatial discretisation errors incurred in different grid intervals. On the other hand, in moving grid method (Pipilis, 1990; Miller and Miller, 1981; Miller, 1981) the grid positions move continuously over time. The ODEs governing their motion are derived so as to minimise the squared residual

$$\int_V R^2(z,t)\, dz$$

with respect to the rate of change of the variable values at the nodes and the rate of change of positions.

In local grid refinement methods, nodes are activated (or de-activated) locally in a fixed grid structure in regions of relatively large error or solution gradient to provide uniform accuracy in the solution. Thus at any given time only a subset of the nodes on the grid are actually active. Since the error is locally reduced in the areas of refinement, the global error decreases. The practical advantage of this approach is that the original grid structure is preserved. The disadvantages are that the computer time and storage increase

with the refinement, and that the coding of such techniques is relatively difficult, requiring rather complex data structures (Thompson, 1988).

2.3 Software Packages for PDEs

Since the advent of powerful computing tools, solving PDE problems by numerical approximation has been one of the most challenging areas in the area of numerical analysis. Although persistent efforts have resulted in considerable progress in solving many problems, the aim of constructing reliable software capable of solving a wide spectrum of PDE problems has not yet been fulfilled, and does not look achievable in the foreseeable feature. Machura and Sweet (1980) suggest three possible reasons for this:

1. PDE problems allow a considerable freedom of formulation.

2. Certain features of PDE problems require very specific treatment.

3. There exist many numerical methods designed to deal with small classes of problems, none being truly universal.

Considering these obstacles, PDE packages have developed in two directions, namely domain-specific and general-purpose packages. Domain-specific PDE packages are designed to deal with a particular physical problem domain (*e.g.* fluid dynamics) and to allow the user to solve a specific class of problems without making reference to the mathematical formulation. In principle, they can be used by beginners and experts alike although this is not always true in practice. Despite the efficiency and the ease of use of such software within a given domain, the lack of flexibility causes some critical problems, often forcing the users to modify or transform their problems to fit the package. Most CFD packages belong to this category. They will be discussed in more detail in section 2.4.

Bearing in mind the drawbacks of domain-specific software, the natural next step is to attempt to develop general-purpose software which operates directly on PDE mathematical formulations provided by the users. These provide the user with great flexibility and, as a result, can be used in a variety of domains.

One possible way to classify these general-purpose packages is to differentiate between *programming language based* software packages and *high-level* software packages according to the structure of the user interface. In this section, two types of general-purpose packages will be discussed and some typical examples of such software will be examined briefly.

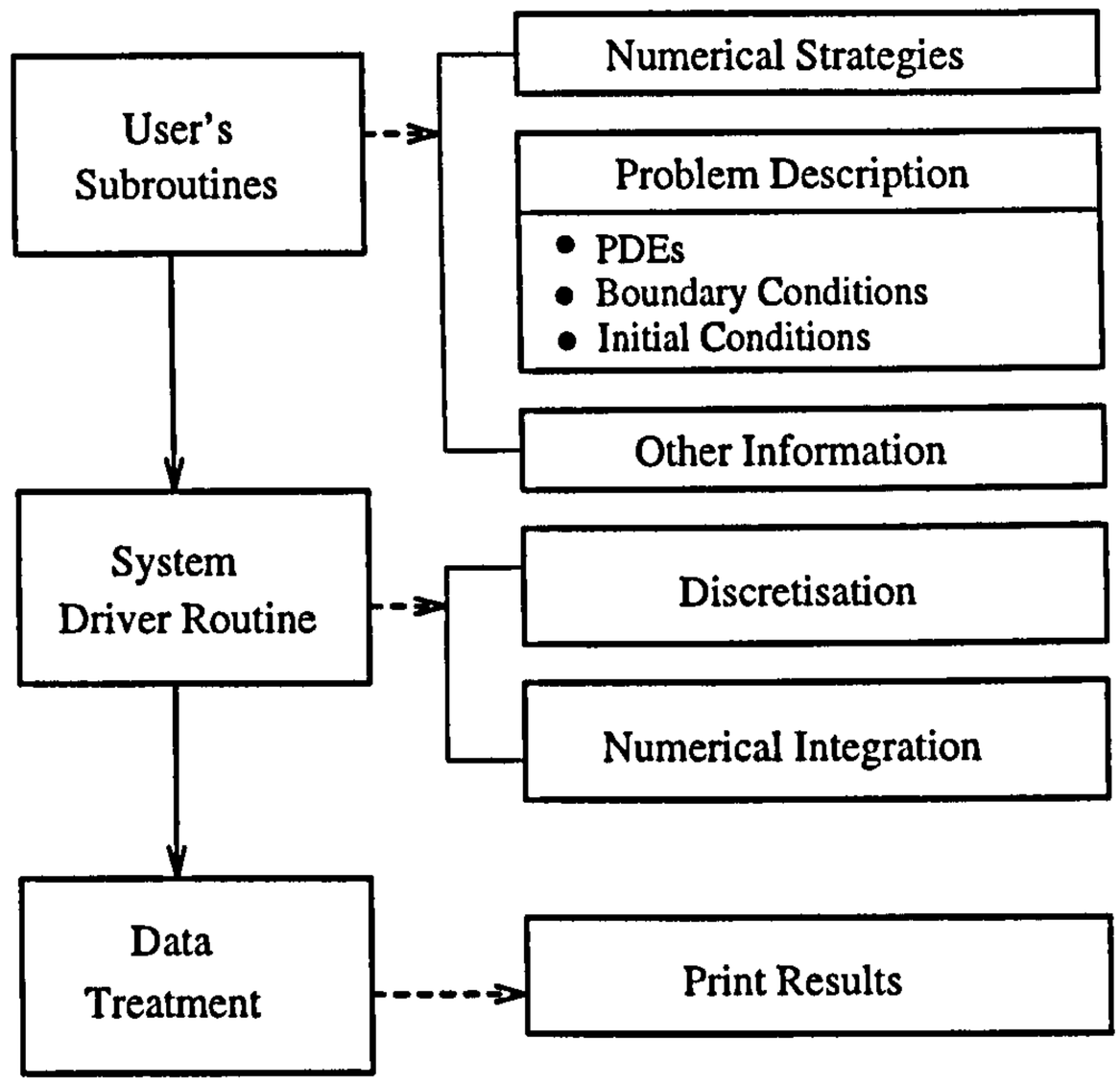


Figure 2.3: Basic structure of programming language based packages

2.3.1 Programming Language Based PDE Packages

In this type of software package, the user communicates with the solution algorithm by means of a procedural programming language, usually FORTRAN. Thus, the mathematical model of a physical system is described in terms of sub-programs written in this language. Also a library of solution methods is provided to the user in the form of callable subroutines, and each subroutine can be invoked by a driver program whenever it is needed. The typical structure of such packages is shown in Figure 2.3.

The effective use of this approach requires some understanding of the organisation of the package as well as the library of solution methods. However, the user is freed from the tedious and complicated process of understanding in detail and programming numerical algorithms for a given problem.

Another benefit of this kind of package lies in the *re-usability* of existing FORTRAN libraries of solution methods. The cost of developing reliable PDE solution codes from scratch is considerable, so it is highly desirable to re-use existing libraries. User-written numerical codes can also be incorporated within existing libraries and integrated within the framework of the software package as long as the necessary protocols are well defined. The re-usability of numerical codes in the package provides a user, especially an experienced one, with a flexible framework for handling complex computational problems.

Because of the relative simplicity of the implementation and the flexibility of such packages, most general-purpose PDE packages belong to this type, and consequently, are the most frequently used in industry as well as in academia.

PDECOL (Madsen and Sincovec, 1980) is probably the first general-purpose software package for the numerical solution of coupled systems of nonlinear parabolic PDEs. The package is based on the method of lines approach, and uses finite element collocation methods based on piecewise polynomials for the spatial discretisation. The resulting ODEs are numerically integrated by a slightly modified version of an ODE package called GEARIB (Hindmarsh, 1976).

PDECOL allows the user to describe one-dimensional PDE problems of the form:

$$\frac{\partial \phi}{\partial t} = f(\phi, \phi_z, \phi_{zz}, z, t) \tag{2.24}$$

subject to boundary conditions

$$b(\phi, \phi_z) = g(t) \tag{2.25}$$

and initial conditions

$$\phi(z, t_0) = h(z) \tag{2.26}$$

where ϕ_z and ϕ_{zz} are respectively the first- and second- order derivatives of ϕ with respect to z, t the time independent variable and z the space independent variable, and $g(\cdot)$ and $h(\cdot)$ given functions.

The PDE system described above is completely specified by defining: (a) the initial time and the boundaries of the spatial domain, (b) the number of grid elements, (c) the boundary conditions, (d) the initial conditions, and (e) the system equations in user supplied FORTRAN subroutines.

EPDCOL (Keast and Muir, 1991) is an enhanced version of PDECOL. It replaces the linear algebra modules of PDECOL code by modules which take full advantage of the special structure of the equations which arise so as to perform the solution of the linear systems in a more efficient fashion. It is reported that EPDCOL saves over 50 % in total execution time compared to PDECOL on most problems.

FIDISOL (Schönauer and Schnepf, 1987) is a package for the solution of systems of two- and three-dimensional nonlinear elliptic and parabolic PDEs with nonlinear boundary conditions in a rectangular domain. Finite difference methods with a fixed or self-adapting grid and order are used. The basic ideas for the solution methods are those developed in the earlier SLDGL package (Schönauer *et al.*, 1981) which, according to Ames (1992), has been used extensively in Europe.

The most general form of PDEs in FIDSOL is:

$$f(\phi, \phi_x, \phi_t, \phi_{xx}, x, t) = 0 \tag{2.27}$$

subject to boundary conditions

$$b(\phi_x, \phi_t, x, t) = 0 \tag{2.28}$$

and initial conditions

$$i(\phi, t_0) = 0 \tag{2.29}$$

where x is the vector of independent variables. It should be noted that the form of equations 2.27-2.29 is general enough to describe almost all elliptic and parabolic PDEs in (up to) three-dimensional rectangular domains.

DSS/2 is a package developed at Lehigh University (Schiesser, 1991). Conceptually, it can handle the formulation and the solution of nonlinear PDEs of any form, over a rectangular domain. The method of lines is used and spatial derivatives are discretised by means of finite difference methods or orthogonal collocation methods on finite elements. As with other similar packages, the user is responsible for expressing the PDEs and their auxiliary conditions as FORTRAN subroutines and for selecting appropriate solution method subroutines from the libraries in DSS/2.

Since DSS/2 utilises an initial-value ODE package to integrate the resulting equations obtained from spatial discretisation, it is not possible to solve directly steady-state boundary-value PDE problems which do not have time derivatives. In order to handle this situation, the time derivative of a dependent variable is introduced in the original PDEs, thus resulting in a pseudo-dynamic model. For instance, in the case of Poisson's equation in two-dimensional Cartesian coordinates, described by the elliptic equation:

$$\frac{\partial^2 \phi}{\partial x^2} + \frac{\partial^2 \phi}{\partial y^2} - f(x, y) = 0 \tag{2.30}$$

the introduction of an *artificial* time derivative results in the following parabolic equation:

$$\frac{\partial \phi}{\partial t} = \frac{\partial^2 \phi}{\partial x^2} + \frac{\partial^2 \phi}{\partial y^2} - f(x, y) \tag{2.31}$$

The integration of resulting ODEs is allowed to proceed sufficiently long for the solution to stabilise, thereby yielding a solution of the original elliptic PDEs[1]. This approach is called the *method of false transient.*

The method of false transient can be applied with many programming language oriented PDE packages based on the method of lines in order to solve steady-state elliptic

[1]This solution is not necessarily unique and depends on the initial condition used for the integration.

PDEs. However, the automatic introduction of an artificial time variation may cause some problems. Firstly, there is no guarantee that the resulting PDEs are stable and reach steady-state as time tends to infinity. Secondly, even if the PDEs are stable, it may take a long time to obtain the correct solution. This is especially true if the system includes many PDEs discretised over a fine spatial grid.

2.3.2 High-Level PDE Packages

In these packages, the formulation of PDE problems is done in terms of special languages which can normally be mastered in a relatively short period of time. The structure and the syntax of a language usually resemble the widely accepted mathematical notation, and therefore the process of problem formulation is much simplified and easily comprehensible. This, in turn, accelerates the process of solving PDE problems. These advantages are often obtained at the price of reduced flexibility since the language may impose certain restrictions on the type of problem that can be posed.

In contrast to the packages described in section 2.3.1, no knowledge of programming languages is necessary. Problems are described in a special language allowing the symbolic specification of PDEs, boundary conditions, initial conditions, and appropriate coefficients. As a totally automatic procedure for selecting an appropriate solution method is not yet available, some user intervention in specifying the numerical solution strategy to be adopted is also inevitable. This includes the type of numerical method, the number of discretisation intervals *etc.* The basic structure of this type of package is illustrated in Figure 2.4.

Constructing a comprehensive special language for the description of PDE problems usually involves a substantial knowledge of computer science as well as an understanding of the underlying mathematics. Building a good user interface is also a factor of great importance. There are two main facets of such an interface: input for the formulation of PDE problems and output for handling the results. In many cases, the input is handled with textual or graphical editors; graphic packages are normally used to visualise the results.

Constructing quality software of this type involves the design and implementation of special languages, user interfaces and numerical methods. Consequently, despite the many obvious advantages of high-level packages, due to the complexity of the implementation, only a very limited number of packages are reported to be reliable and practical.

ELLPACK (Rice and Boisvert, 1985) is a software system designed to solve second-order, linear elliptic PDEs in two and three dimensions with Dirichlet, von Neumann,

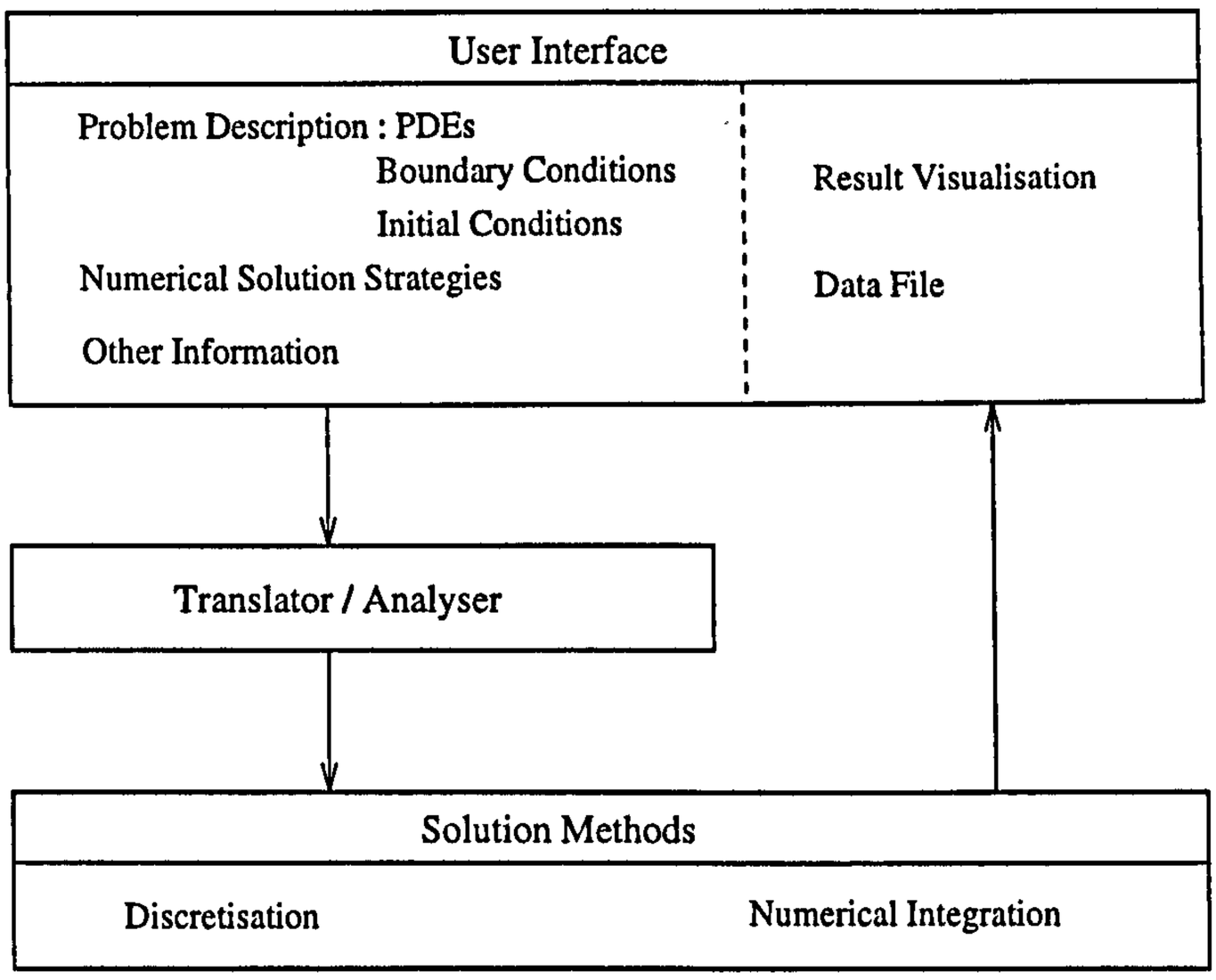


Figure 2.4: Basic structure of high-level packages

Robin, or periodic boundary conditions. It includes both a high-level problem statement language and an extensible library of problem solving modules. The language is an extension to FORTRAN that allows users to declare boundary value problems. One interesting aspect of this package is that it allows the specification of nonrectangular domains. The boundary is broken up into a series of pieces which must be connected in sequence. For instance, in Figure 2.5, the boundary comprises four pieces, *i.e.* P_1P_2, P_2P_3, P_3P_4 and P_4P_1. The boundary conditions are specified separately for each piece. In this example, the language allows a user to describe the boundary condition on P_3P_4 as follows:

```
BOUNDARY
   U = (x-0.5)*(y-4)  ON x =  4. + 1 * p *(p-4.5) **2  &
                         y = -.5 + p       FOR p = 0. TO 4.5
```

If a geometry is too complicated to be described in terms of polynomials or other simple mathematical expressions, the domain is sub-divided into a number of smaller pieces until it can be approximated in this way.

Although ELLPACK was originally designed for second-order, linear elliptic problems, its structure also allows it to be used to solve time-dependent problems, and three-dimensional problems on cylindrical domains with holes.

PDEase2 (SPDE, 1993) is a package for solving a large class of steady-state and time dependent boundary value and initial value problems. It can be used to solve most two-

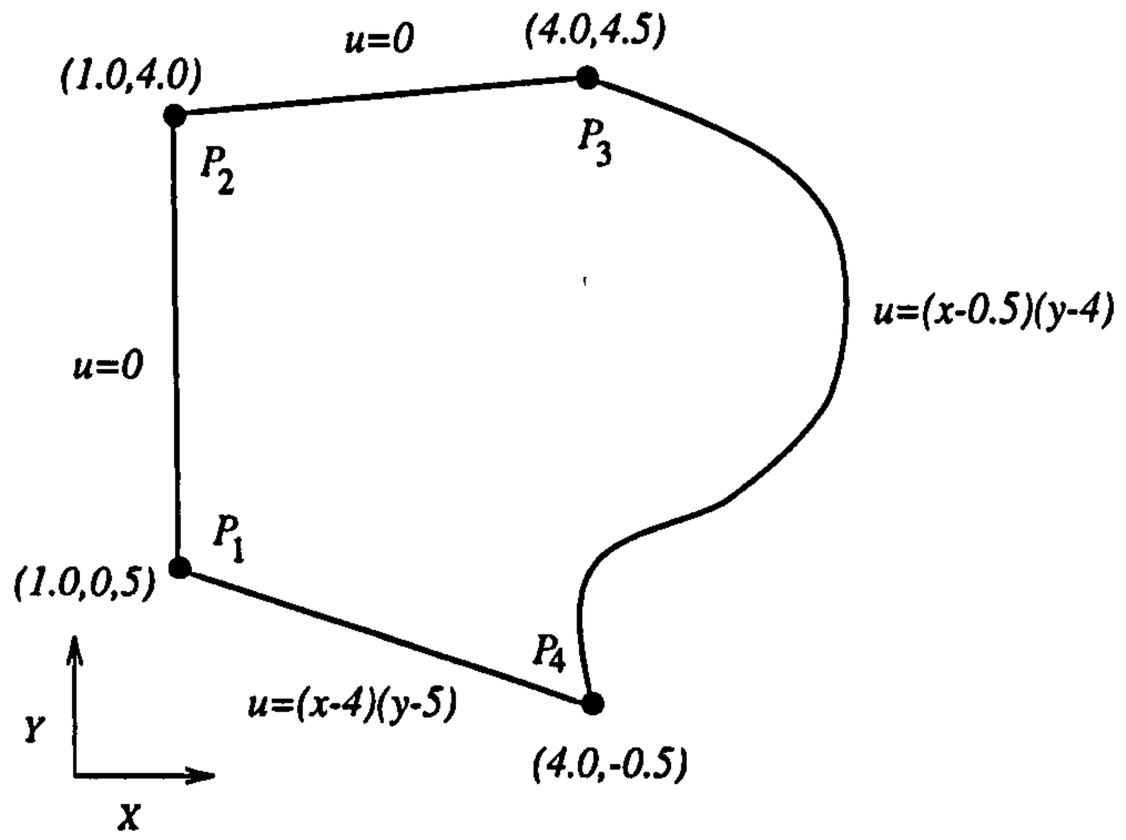


Figure 2.5: Specification of nonrectangular domain

dimensional problems of heat flow, electrostatics, diffusion, chemical reaction, streamline flow, viscous flow *etc*, described in a high-level language.

PDEase2 uses the Galerkin finite element method to convert PDEs into discrete nodal equations. One of the most difficult decisions that must be made in implementing the finite element method is to divide the area of interest into a number of elements. Small elements require excessive computational time and memory. In order to solve this problem, the package uses an iterative process to refine the grid, starting with a coarse grid of triangular elements. In this way, fine gridding is used only in areas with sharp curvatures and tight geometries.

PDEDIS (Pfeiffer and Marquardt, 1993) is a symbolic-numerical tool for the semi-discretisation of PDEs on one spatial domain in the context of a numerical method of lines solution. This package mainly aims to deal with chemical engineering problems and to enhance the capabilities of currently available modelling environments which only support DAE problems. The first prototype of a toolbox is reported to have been implemented in MACSYMA.

The current version of PDEDIS is able to deal with PDAE systems of the type:

$$n + A\frac{\partial^2\phi}{\partial z^2} + D\frac{\partial\phi}{\partial z} + E\frac{\partial\phi}{\partial t} + f(\phi) + g(z,t) = 0 \tag{2.32}$$

with the appropriate initial and boundary conditions. Here the variable t denotes time, $z \in [z_l, z_u]$ represents a single spatial coordinate in a given domain and $\phi, n, f, g \in \mathcal{R}^n$. The

coefficients A, D and E may be of the following functional forms:

$$A = A(\frac{\partial \phi}{\partial z}, \phi, z, t)$$
$$D = D(\phi, z, t)$$
$$E = E(\phi, z, t)$$

The nonlinear function n is a collection of all terms other than the ones already discussed. In particular, n is allowed to depend on higher powers of the spatial derivatives. For the purpose of spatial discretisation, finite difference method and global orthogonal collocation are used.

This general form is restricted, but includes many chemical engineering application where higher order spatial derivatives (> 2) usually do not occur. According to the authors, some basic consistency checks and a few simple rules for selecting appropriate discretisations are implemented. However, the details are not presented in the paper.

2.4 CFD Packages

Computational fluid dynamics (CFD) may be described as the activity of determining information of practical interest (*e.g.* velocity distribution and pressure distribution) about fluid mechanical problems by computational methods. CFD not only extends theoretical fluid mechanics, but also, to a certain extent, replaces experimental fluid dynamics.

The theoretical development of fluid dynamics focusses on the construction and solution of the governing equations for the different categories of fluid dynamic behaviour, and the study of various approximations to those equations. Many of the solutions that have been obtained analytically are for the case of Newtonian fluids. On the other hand, many industrially important fluids are non-Newtonian and are described in terms of highly non-linear system of equations. For these cases, only a handful of analytic solutions are available. Thus the solution of most realistic problems requires the application of numerical techniques as implemented in CFD.

Experimental fluid dynamics has been developed simultaneously with theoretical fluid dynamics to validate and determine the limits of the various approximations to the governing equations. For instance, the wind tunnel, as a piece of experimental equipment, provides an effective means of simulating real flows and plays an important role in obtaining global information, such as the drag on a body and the surface pressure distribution at key locations. However, to obtain detailed velocity and pressure distributions throughout the

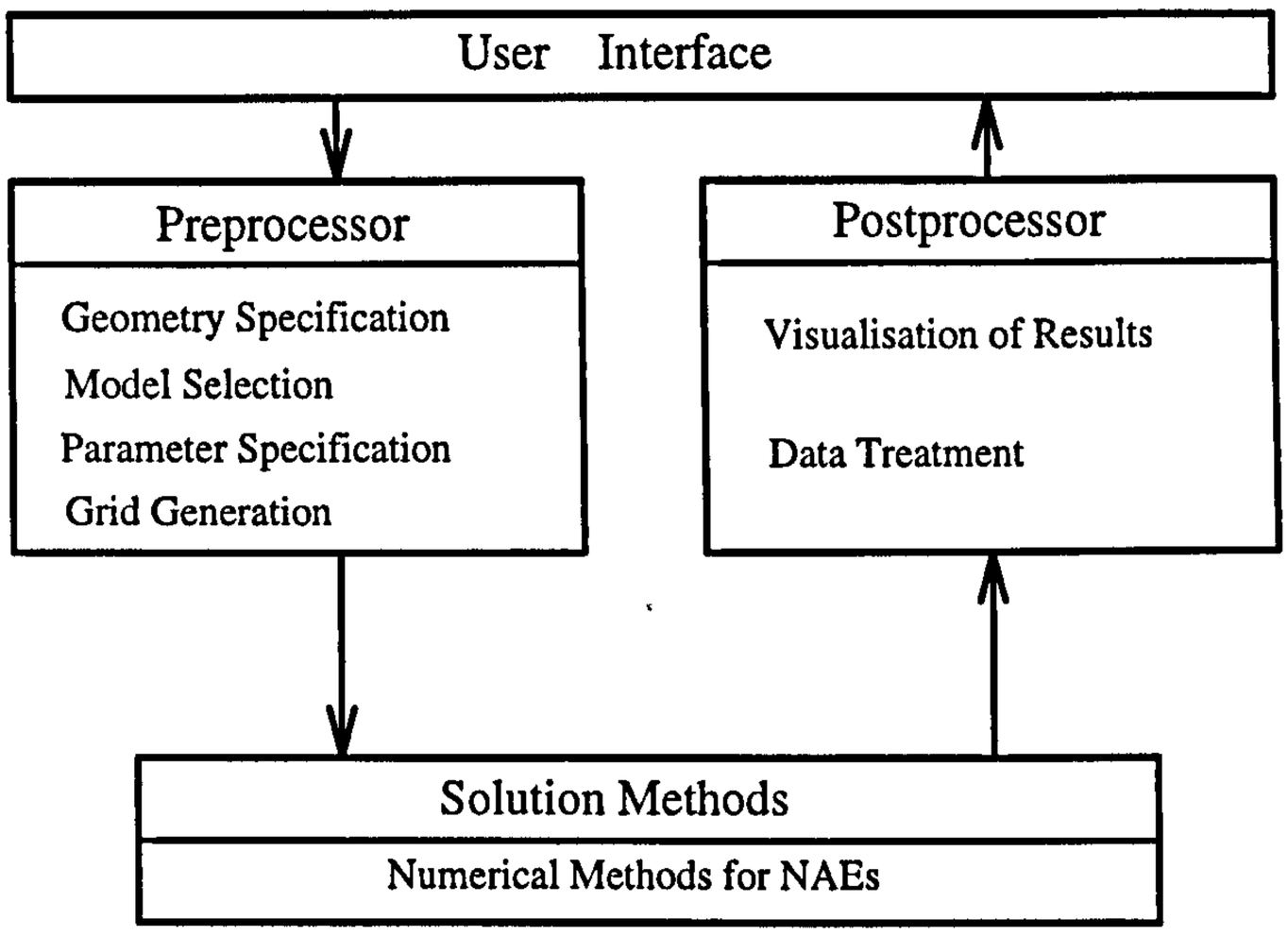


Figure 2.6: Structure of CFD package

region surrounding a body would be prohibitively expensive and very time consuming. More importantly, many experiments are limited by certain practical factors. For instance, in the case of the wind tunnel, very high temperatures associated with coupled heat transfer flow problems are beyond the scope of many experimental facilities. Even where such experiments are technically feasible, their cost can be prohibitive.

From the above discussion, it can be concluded that the major advantages of CFD compared with experimental fluid dynamics are:

- providing detailed and comprehensive information
- cost effectiveness
- simulating flow conditions not reproducible in experimental tests.

The solution of realistic problems in fluid dynamics requires a sound knowledge of fluid mechanics and a profound understanding of the numerical algorithms. CFD packages try to moderate these requirements as much as possible through appropriate user interfaces and reliable solution methods. Typical CFD packages comprise three distinct elements, namely preprocessing, solution procedure and postprocessing as shown in Figure 2.6.

Preprocessing allows the user to provide the information required for problem specification and generates some of the information required by the numerical solution methods. This includes:

1. specification of system geometry

2. equation and model selection

3. boundary and initial condition specification

4. material property specification

5. automatic grid generation.

Many CFD problems occur in complex geometries rather than simple rectangular domains and, in many cases, the influence of the boundary shape and the distribution of certain properties over the surface of a domain are of great importance. Therefore, a proper mechanism to describe the boundaries of the domain should be provided. The geometry could be determined either by specifying coordinates, or graphically or by a combination of both approaches.

The grid generation divides the solution domain into a number of sub-domains based on the built-in numerical algorithms and the geometry specification. A number of numerical methods are used at this stage. The vast majority of the packages makes use of one of three methods, namely finite differences, finite elements and finite volume techniques. Among these, the finite difference method has been the one most frequently employed. However, a major disadvantage of this method is its inability to impose correct boundary conditions for arbitrary geometries. Therefore, more recently finite element and finite volume methods have been gaining popularity in fluid dynamic analysis.

As mentioned earlier, the underlying mathematical models used in CFD packages are generally pre-determined, and consequently users have only a limited number of choices to describe their problems. Among them are:

1. selecting one of several models

2. specifying the values of parameters occurring in the model equations

3. deleting/modifying certain terms within the equations.

Because of the limited scope for user intervention, the quality and diversity of *built-in* mathematical models are of great importance. It is worth mentioning in this context that currently available CFD programs have limited support for non-Newtonian, turbulence and multiphase flow models. However, certain sophisticated CFD packages alleviate this limitation by supporting communication with user written subroutines via a set of strict

protocols. Some packages also have limited capabilities for the interfacing of user provided numerical solution methods.

The *numerical solution procedure* solves the equations of continuity, momentum and energy conservation in order to calculate the necessary variables within a flow field. Usually, generic conservation equations can be described as follows:

$$\frac{\partial(\rho\phi)}{\partial t} + \frac{\partial}{\partial x}(\rho U\phi - \Gamma_\phi \frac{\partial \phi}{\partial x}) - S_\phi = 0 \qquad (2.33)$$

where ϕ is a conserved quantity such as mass, energy or momentum, Γ_ϕ the diffusivity, S_ϕ the source/sink and x the independent variables. Most CFD packages do *not* use the method of lines approach of reducing PDEs into ODEs in time. Instead, they choose to discretise both the temporal and the spatial dimensions, thereby reducing the PDEs into sets of nonlinear algebraic equations. This choice appears to be mainly due to historical reasons.

Postprocessing enables the display and analysis of the results of the computations. The main activities are: (a) visualisation of the results, (b) data management and (c) interface with other software packages. As many CFD problems occur in multi-dimensional fields, the results are very difficult to understand without the help of sophisticated graphical tools. Most packages include graphical tools to visualise the results in three or four dimensional fields (time and three spatial dimensions).

A typical CFD package is **PHOENICS** (CHAM, 1987), a code based on finite volume techniques. Thus, the domain is broken down into a finite number of computational cells, and each PDE is transformed into a set of algebraic equations by integrating it over each cell (see section 2.2.5). This package deals with problems in three-dimensional Cartesian, cylindrical-polar and general-curvilinear coordinate.

PHOENICS provides a comprehensive library of mathematical models covering (a) compressible/incompressible flows, (b) subsonic, trans-sonic and supersonic flows, (c) flow in porous media with possibility of direction dependent porosities, (d) heat transfer by convection, conduction and radiation, (e) multi-phase flow, (f) combustion/chemical reaction, and (g) turbulent flow.

The preprocessing, processing and postprocessing elements of PHOENICS run independently, with information being communicated between them as and when necessary.

As observed earlier, the task of accurately specifying the geometry of a real system may be both formidable and tedious if the domain is irregular. In the PHOENICS package, the geometry can be specified in the physical space by using a graphic editor and the grids

are constructed interactively. This is particularly useful as the user can describe a problem and immediately inspect the results of grid generation in the physical space.

The package provides an efficient interface with FORTRAN, and experienced users can add new modelling features, such as complex boundary conditions, new turbulence models, and even alternative numerical solvers.

2.5 Concluding Remarks

This chapter started by introducing some basic terminology relating to systems of PDEs and then considering some of the numerical methods most commonly used for their solution. It may be worth remembering at this stage that, as pointed out in the Introduction, the problems of interest to process engineering are usually described by IPDAE systems rather than just PDE ones. However, the essence of PDE solution methods is to convert an *infinite* dimensional problem, the solution to which is a function $\phi(z)$ to a finite dimensional problem which can be expressed in terms of *finite* vector of parameters (such as the values of the original function at a finite number of points, $\{z_k\}$). Hence most such methods can also be used to solve IPDAE systems. We shall return to these issues in chapter 4 where we consider in detail two classes of solution method, namely those based on finite difference and orthogonal collocation on finite elements.

In the second part of this chapter, we have reviewed some of the software available for solving PDEs. The variety of problem types encountered in process engineering applications seems to suggest an approach similar to that adopted by the software reviewed in section 2.3. The benefits of using high-level languages instead of FORTRAN-defined subroutines have already been established in process engineering by packages such as SpeedUp. We therefore favour this approach; however it is worth noting that, with the exception of PDEDIS, all of the packages reviewed in section 2.3.2 are restricted to purely PDE systems and none of them actually forms part of an integrated tool that can be used for modelling processes.

Chapter 3

A Modelling Language for Lumped/Distributed Parameter Systems

As already stated in chapter 1, a significant proportion of all processes of interest involve distributed parameter systems. Usually the distribution arises from the space variation of conditions within the system (*e.g.* packed bed reactors and separation columns). In some cases, the distribution is with respect to other system properties such as the molecular weight in polymerisation reactors, or the particle size in crystallisers. Indeed, distributions with respect to several dimensions and/or other system properties occur in many systems.

The objective of the modelling language described in this chapter is to provide a means for the *direct* modelling of distributed parameter systems within the framework of the *gPROMS* package (Barton, 1992). This is consistent with the declared aim of this and other similar packages of providing the maximum possible separation between, on one hand, the model declaration, and, on the other hand, the mathematical solution.

The chapter begins with a review of the modelling formalism for lumped parameter systems and the functionality of the *gPROMS* package. This is followed by the discussion of three key elements which are indispensable for declaring distributed parameter systems in the language, namely those of distribution domains, variable distributions and equation distributions. The discussion extends to the description of the formal syntax related with those elements and the introduction of operators for partial differentiation and integration.

Boundary conditions play an important role in models of distributed systems. They define the solution domain and are the starting point to calculate the solution of interior domains. Consequently, the characteristics of boundary conditions are analysed through a simple example and the specification of boundary conditions in the language is illustrated.

As discussed in chapter 2, the current state of the art of numerical solution methods is such that no single method is able to handle a wide spectrum of distributed parameter processes. This leads to a discussion of the solution specifications necessary for performing the dynamic simulation of distributed parameter systems successfully.

3.1 The *gPROMS* Process Modelling Tool

The *gPROMS* (general **PRO**cess Modelling System) system is a software package designed for the modelling and simulation of processes with both *discrete* and *continuous* characteristics. It distinguishes two fundamental types of modelling entity. *MODEL*s describe the physical, chemical and biological laws that govern the intrinsic behaviour of a system, while *TASK*s are descriptions of the external actions and disturbances imposed on it. A third type of entity, the *PROCESS*, is formed by a *TASK* driving a *MODEL* with some additional information, such as the initial conditions and the time variation of the input variables. As a result, a simulation is defined as the execution of this process.

In this section, we illustrate some aspects of lumped parameter system modelling in *gPROMS* by using a simple example. However, chemical processes are often complex plants composed of a large number of elementary units. Naturally, it is very hard for a user to tackle the whole modelling task simultaneously. This leads to a brief discussion of the concept of hierarchical submodel decomposition.

3.1.1 Lumped Parameter System Modelling

For lumped parameter systems, the lowest-level *MODEL*s in *gPROMS* are primarily sets of variables and the DAEs relating them. The latter are declared in a high-level declarative language, as shown in Figure 3.1[1]. It is worth noting some of the features of the language. First, the modelling activity is totally decoupled from the solution method which will be used during the simulation. Secondly, the language is purely declarative, with the order in which equations are written being irrelevant. Both differential and algebraic equations may be introduced – note that the "$" symbol (see line 16 of Figure 3.1) denotes the derivative of a variable with respect to time. Finally, single or multi-dimensional arrays of both variables and equations can be declared using `ARRAY` constructs (lines 7, 8, 10, 11). Array equations may be written in implicit vector form (*e.g.* the component mass balance in line 16 of Figure 3.1), or using explicit `FOR` constructs (*e.g.* the phase equilibrium relationship in lines 19-21).

In addition to variables and equations, *gPROMS* models may involve parameters (see lines 3-4 of Figure 3.1). These are quantities that must *always* be given fixed values for the model to be used in any simulation.

[1] Note that line numbers are not a feature of the *gPROMS* language. They are used in this thesis for ease of reference only.

```
1   MODEL IsothermalFlash
2
3     PARAMETER
4       Nocomp     AS    INTEGER
5
6     VARIABLE
7       M          AS    ARRAY (Nocomp) OF Holdup
8       F          AS    ARRAY (Nocomp) OF Flowrate
9       V, L       AS                      Flowrate
10      x, y       AS    ARRAY (Nocomp) OF MoleFraction
11      K          AS    ARRAY (Nocomp) OF KValue
12                   . . .
13
14    EQUATION
15      #  Component mass balance
16      $M  =   F  -  L * x  -  V * y ;
17
18      #  Phase equilibrium relationship
19      FOR i := 1 TO Nocomp DO
20        K(i) * x(i) = y(i) ;
21      END
22
23        SIGMA(x) = SIGMA(y) = 1 ;
24                   . . .
25    END  # model isothermalflash
```

Figure 3.1: Lumped parameter model in *gPROMS*

3.1.2 Hierarchical Submodel Decomposition

Due to the complexity of chemical processes, it is often difficult to handle the whole modelling task simultaneously. The re-usability of any models which are developed and proven at substantial cost is also of great significance.

Considering these issues, hierarchical mechanisms which enable the user to construct the complex model from simpler components are introduced. The basic principle of these mechanisms is repeatedly to subdivide the modelling problem of interest until a sufficiently simple level of model is reached. In the chemical industry, such models usually correspond to elementary equipment items for unit operations, (*e.g.* pump, valve) or parts thereof (*e.g.* distillation column trays).

Because of its complexity, the activity of modelling a chemical processing system usually proceeds in three steps:

1. partitioning of the system into elementary subsystems

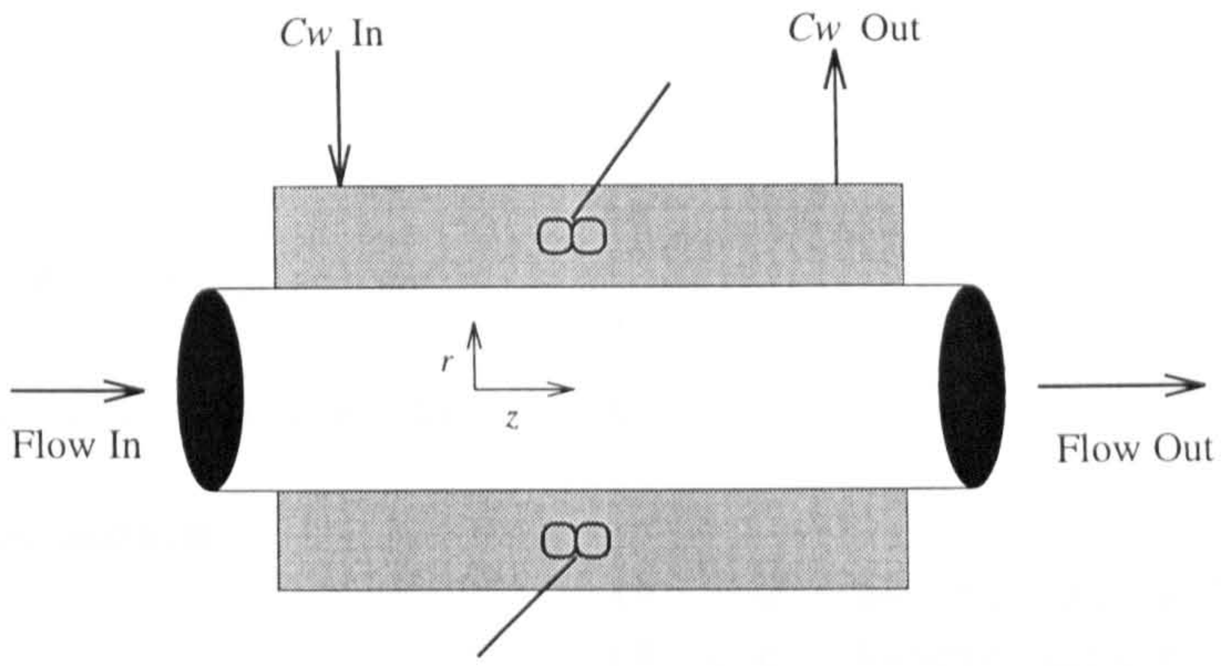


Figure 3.2: Example of distributed unit operation

2. analysing each subsystem

3. synthesising the system from the analysed subsystems by means of connectivity.

In the *gPROMS* language, the user can construct a complex model as a combination of submodels possibly connected by *streams*. Each of these submodels is an instance of another model which may, in turn, have its own submodels. In this way, a hierarchy of arbitrary depth can be created.

3.2 Distribution Domains

We now turn our attention to the issues relating to the modelling of distributed systems. We illustrate the presentation of the proposed modelling language by the system shown in Figure 3.2. This depicts a tubular reactor vessel used to carry out an exothermic chemical reaction. The intensive properties of the fluid in the tube vary with both axial and radial position, as well as with time. The reactor is surrounded by a well-stirred cooling medium bath, the temperature of which is uniform throughout, but may vary with time.

The first concern in the context of modelling a distributed parameter unit is declaring the *distribution domains* that are pertinent to this unit, and, where appropriate, associating some of the variables in the model with one or more of these distribution domains.

As already mentioned, the variation of the conditions within a distributed parameter system may be described as distributions over one or more space dimensions, molecular weight, particle size *etc.* These variations define the distribution domains for the system of interest.

All distribution domains within a given model are specified in the `DISTRIBUTION_DOMAIN` subsection of the *MODEL* entity as shown below

```
MODEL TubularReactor

   PARAMETER
      Nocomp                              AS   INTEGER
      UC1, AA, BB, GasConst, Dr,
            ....
      ReactorLength, ReactorRadius    AS   REAL

   DISTRIBUTION_DOMAIN
      Axial                               AS  ( 0 : ReactorLength )
      Radial                              AS  ( 0 : ReactorRadius )
```

Figure 3.3: Example of distribution domain declaration

```
DISTRIBUTION_DOMAIN
   <DistributionDomain>  AS  ( Expression : Expression )
```

where the `Expression` may involve real and integer constants and real and integer parameters that have been declared within the `PARAMETER` subsection of the `MODEL`.

For instance, Figure 3.3 defines two such domains for the tubular reactor model, respectively called `Axial` and `Radial`. These are declared in terms of two of the model parameters, namely the `ReactorLength` and `ReactorRadius`.

3.3 Variable Distributions

The conditions within a model are characterised by variables, some of which will be distributed over given domains. However, not all variables within the same model need have the same degree of distribution. For instance, in the tubular reactor example, the concentrations of the various chemical components and the temperature within the tube are indeed functions of radial and axial positions. However, the wall temperature is a function of axial position only, while the temperature of the cooling medium bath is not a function of spatial position at all.

Distributed variables are declared together with all others in the `VARIABLE` subsection of the `MODEL` section. As already mentioned in section 3.1, the `ARRAY` construct is used to declare distributions over one or more discrete domains. This can now be generalised to describe distributions over an arbitrary number of both discrete and continuous domains.

To allow the explicit definition of the degree of distribution of each individual model variable, we introduce the concept of *distributions*. This can be viewed as a generalisation of

```
VARIABLE
   Temp     AS  DISTRIBUTION (Axial,Radial)          OF Temperature
   Conc     AS  DISTRIBUTION (Nocomp,Axial,Radial)   OF Concentration
   TWall    AS  DISTRIBUTION (Axial)                 OF Temperature
   TBath    AS                                          Temperature
   CFeed    AS  ARRAY (Nocomp)                       OF Concentration
   . . .
```

Figure 3.4: Example of variable declaration

the mathematical concept of fields over continuous domains on one hand, and the computer science concept of arrays over discrete domains on the other. In the extended *gPROMS* language, all variables, other than those that are scalar constants or functions of time only, are distributions over one or more continuous and/or discrete domains.

Consider, for instance, the variable declarations in the tubular reactor model, as shown in Figure 3.4. Variable `Temp` representing the temperature in the tube is declared as a `DISTRIBUTION` over the two continuous domains `Axial` and `Radial` – in other words, a two-dimensional field. On the other hand, variable `Conc` representing the concentrations of the various components in the tube is a `DISTRIBUTION` over a discrete domain (ranging from 1 to `Nocomp`, the number of components in the system), as well as the two continuous domains already defined. In other words, `Conc` is an array of two-dimensional fields.

The degree of distribution of other variables in the system is declared as appropriate. No distribution is specified for the cooling water bath temperature `TBath` since this is solely a function of time. Finally, as illustrated by the case of the feed composition vector `CFeed`, the term `ARRAY` can be used as a synonym for `DISTRIBUTION`s over discrete domains only.

It is worth commenting on some of the features of this language. First, the original `ARRAY` keyword has been retained as a synonym to the newly introduced `DISTRIBUTION` keyword. This has the advantage of maintaining compatibility with earlier versions, as well as being consistent with the use of the term by most other modelling and computer programming languages.

Secondly, a continuous domain declares the functional dependence of a variable on the position in this domain. It is prohibited to specify two continuous domains of the same variable as being of the same type. For instance, a temperature field over a square domain would have to be declared as

```
Theta    AS    DISTRIBUTION (XDomain, YDomain)   OF   Temperature
```

even if both `XDomain` and `YDomain` are defined identically as

```
DISTRIBUTION_DOMAIN
   XDomain, YDomain          AS   ( 0.0 : Length )
```

This restriction is required for the unambiguous definition of partial derivatives and integrals (see section 3.4.5 and 3.4.6) as well as for the specification of solution methods described in section 3.6.

Finally, a subtler point relates to the quite different mechanisms used for declaring discrete and continuous domains. Thus, discrete domains are declared implicitly within the variable declaration itself. For instance, the use of `Nocomp` in the declaration of variable `Conc` in Figure 3.4 implicitly declares a discrete distribution domain `1..Nocomp`. On the other hand, variables can refer only to continuous domains that have already been declared *explicitly* as detailed in section 3.2. For instance the following declaration is **illegal:**

```
VARIABLE
   Twall   AS  DISTRIBUTION (ReactorLength) OF Temperature
```

This restriction is also required for the unambiguous definition of partial derivatives and integrals.

3.4 Equation Distributions

The variables in a model are related through sets of equations. It should, again, be recognised that different equations within the same model may have different degrees of distribution.

For instance, in the tubular reactor example shown in Figure 3.2, the mass and energy balances within the reactor hold over the entire interior of the two-dimensional domain. On the other hand, boundary conditions are valid over specific sub-domains (*e.g.* at the tube entrance and exit, and the wall). The energy balance on the cooling medium bath leads to a lumped equation, albeit one that is related to the reactor energy balance through an integral term describing the heat loss over the entire length of the reactor.

The definition of distributed equations over entire domains or sub-domains raises a number of issues. First, a means of defining variable sub-domains is required. In the proposed language, this is achieved through the use of "domain slice" notation. A second question concerns the determination of the degree of distribution of mathematical expressions involving two or more distributed variables – as well as the rules for checking the validity of such expressions.

Mathematical domain	*gPROMS* language
$[a, b]$	`a : b`
$[a, b)$	`a : b|-`
$(a, b]$	`a|+ : b`
(a, b)	`a|+ : b|-`

Table 3.1: Representation of open and closed domains

The ability to define variable sub-domains and the rules for determining the validity and degree of distribution of complex expressions allow the unambiguous definition of distributed equations. We examine these issues in more detail below.

3.4.1 Variable Sub-domains

The equations in a model may involve entire variable distributions, or just sub-domains (or even single points) of these distributions. To define such sub-domains, we make use of the *domain slice* notation.

For instance, in the case of the variable `Temp` which is distributed over both the axial and the radial domains, the notation `Temp(0:ReactorLength, 0)` refers to the temperature along the axis of the reactor ($r = 0$).

Furthermore, the correct declaration of models of distributed unit operations in terms of IPDAEs requires the precise definition of the domain of applicability of each equation and boundary condition. In particular, we need to be able to distinguish between the interior of the region under consideration and its boundary, and also between different parts of the boundary. The proposed modelling language allows the definition of *open* and *closed* domains in the manner shown in Table 3.1. By introducing new symbols to describe variable sub-domains, the concentrations of components 2, 3 and 4 along the central axis of the reactor in Figure 3.2 for each case may be described as shown in Figure 3.5. It will be recalled (*cf.* Figure 3.4) that the first index of variable `Comp` denotes the component, the second the axial position, and the third the radial position.

3.4.2 Domain Properties for Variables and Variable Occurrences

In order to allow variable sub-domains to be combined into mathematical expressions, we need to define some characteristics of their distributions. Here we introduce the requisite terminology.

```
Comp( 2:4, 0    : ReactorLength,    0)  #  z = [ 0, ReactorLength ]

Comp( 2:4, 0    : ReactorLength|-, 0)  #  z = [ 0, ReactorLength )

Comp( 2:4, 0|+ : ReactorLength,    0)  #  z = ( 0, ReactorLength ]

Comp( 2:4, 0|+ : ReactorLength|-, 0)  #  z = ( 0, ReactorLength )
```

Figure 3.5: Example of the representation of open and closed domains

1. Each variable is declared in the `VARIABLE` subsection to be distributed over a given number of domains.

2. Each domain of a variable is of given type. In particular, all discrete domains are considered to be of the same type ("`DISCRETE`"), while continuous domains are of the specific type explicitly declared in `DISTRIBUTION_DOMAIN`.

3. Every variable occurrence in an equation is characterised by the lower and upper bounds of each domain of the variable. For instance, in the first example shown in Figure 3.5, the lower bounds of the discrete, the axial and the radial domains are respectively `2`, `0.0`, and `0.0`; on the other hand, the corresponding upper bounds are `4`, `ReactorLength` and `0.0`.

4. The range of each domain of every variable occurrence in an equation is the difference between the upper and lower bounds of the domain, as defined above.

5. A variable occurrence with zero ranges in all its domains is called a *scalar*.

The number of domains, and the type and range of each domain are collectively known as *domain properties* of a variable occurrence.

3.4.3 Domain of Distributed Expressions and Equations

Each mathematical expression has a domain of distribution – just as for arrays, each equation has one or more dimensions. The properties of expression domains are defined recursively as follows :

1. The domain properties of an expression consisting of a single variable are the same as those of the variable itself.

2. The domain properties of an expression consisting of a unary function are the same as those of the function argument.

3. For a binary expression,

   - if both operands are scalar, then the resulting expression is scalar;
   - if only one of the two operands is scalar, then the domain properties of the resulting expression are the same as those of the other operand;
   - if neither of the two operands are scalar, then the two must be *conformant*, *i.e.* they must have the same domain properties which then also become the properties of the resulting expression; otherwise the binary expression is illegal.

The validity and the domain properties of an equation involving the equality of two expressions are determined in the same manner as those for binary expressions above.

3.4.4 Explicit Declaration of Equation Distribution Domains

As discussed in the preceding section, the distribution domain of an equation is deduced from that of the variable occurrence within it. Alternatively, an explicit declaration can be effected via the FOR construct already employed for array equations. For instance,

```
FOR z := 0 TO (ReactorLength/2) DO
   HeatLoss(z) = 0.0 ;
END
```

may be used to define that the first half of the reactor is well insulated, and therefore the axially-distributed heat loss is zero over this sub-domain. Here z is an auxiliary index variable.

In order to support the declaration of general IPDAEs, we need to define appropriate *partial differentiation* and *integration* operators. These are discussed in the next two sections.

3.4.5 The Partial Differentiation Operator

3.4.5.1 General Considerations

In the proposed language, partial differentiation of a distributed variable or expression is achieved through the use of the PARTIAL operator. In the simplest form, the syntax of this is of the form

```
PARTIAL ( Expression, DistributionDomain )
```

```
# Heat transfer relation at tube wall (r = ReactorRadius) #
FOR z := 0 TO ReactorLength DO
    - Kr * PARTIAL (Temp(z,ReactorRadius), Radial) =
      Uh * ( Temp (z,ReactorRadius) - TWall(z) )      ;
END
```

Figure 3.6: Example of first order partial differentiation

where `Expression` is a real or integer expression which may involve one or more distributed variables or nested `PARTIAL` operators, and `DistributionDomain` is one of the distribution domains in the system. The expression must normally be distributed over the specified domain (or part of it, including a single point), otherwise the result of the partial derivative becomes identically zero. In any case, the degree of distribution of the resulting partial derivative is the same as that of the expression being differentiated.

An example of the application of the `PARTIAL` operator for defining first-order partial derivatives is given in Figure 3.6 which describes the boundary condition for heat transfer at the reactor wall:

$$-K_{\mathrm{r}}\left.\frac{\partial T}{\partial r}\right|_{r=R} = U_{\mathrm{h}}(T - T_{\mathrm{w}}) \qquad \forall z \in [0, L] \tag{3.1}$$

It is worth mentioning at this stage the difference between the *conservative* and *non-conservative* forms of partial differentiation. In analytical mathematics, the result of both is the same. On the other hand, in numerical mathematics, these two descriptions of a distributed parameter system may result in different solutions. For instance, a convective term in a mass balance equation may be represented in either the conservative form

$$\frac{\partial}{\partial z}(v \cdot c)$$

or the non-conservative form

$$v\frac{\partial c}{\partial z} + c\frac{\partial v}{\partial z}$$

A centered finite difference approximation applied at point i to the partial derivatives in the above expressions respectively results in:

$$\frac{\partial (v \cdot c)_i}{\partial z} \approx \frac{(v \cdot c)_{i+1} - (v \cdot c)_{i-1}}{2\Delta z} = \frac{v_{i+1}c_{i+1} - v_{i-1}c_{i-1}}{2\Delta z} \tag{3.2}$$

$$v_i\frac{\partial c_i}{\partial z} + c_i\frac{\partial v_i}{\partial z} \approx v_i\frac{c_{i+1} - c_{i-1}}{2\Delta z} + c_i\frac{v_{i+1} - v_{i-1}}{2\Delta z} \tag{3.3}$$

which are clearly different.

The proposed modelling language supports conservative representation for both spatial and time derivative terms by allowing the `PARTIAL` and `$` (see below) operators to be applied to entire expressions rather than just simple variables.

3.4.5.2 Higher Order Partial Derivatives

As has already been mentioned, the `PARTIAL` operator can be applied to entire expressions, and these may actually involve other partial derivatives. This provides a natural mechanism for defining higher order partial derivatives in the form

```
PARTIAL ( PARTIAL ( Expression, DistributionDomain ), DistributionDomain )
```

Alternatively, the abbreviated form

```
PARTIAL ( Expression, DistributionDomain, DistributionDomain )
```

may be used.

3.4.5.3 Partial Differentiation with Respect to Time

The special operator "$" is retained in the language in order to maintain compatibility with earlier versions of *gPROMS* (see Figure 3.1). It is used to denote partial differentiation with respect to time, and can be applied to both single variables and expressions.

The combined application of these operators (`$` and `PARTIAL`) is illustrated well in Figure 3.7 which represents the energy balance equation:

$$\frac{\partial}{\partial t}(\rho_f c_p T) = -\frac{\partial}{\partial z}(v_z \rho_f c_p T) + \frac{\partial}{\partial z}(k_z \frac{\partial T}{\partial z}) + \frac{1}{r}\frac{\partial}{\partial r}(k_r r \frac{\partial T}{\partial r}) + \rho_b r(-\Delta H) \qquad \forall\, z \in (0, L),\ r \in (0, R) \qquad (3.4)$$

3.4.6 The Integral Operator

Integrals occur frequently in equations arising in a number of branches of physics and engineering, such as in the study of acoustics, optics and laser theory, fluid mechanics and statistics (Baker, 1977). In the area of process engineering, they often occur in population balance models describing, for instance, crystal size distributions, activity distributions of recycled catalyst particles, and the age and size distribution of micro-biological cultures (Himmelblau and Bischoff, 1968).

```
EQUATION
   # Energy Balance
   FOR z := 0|+ TO ReactorLength|- DO
      FOR r := 0|+ TO ReactorRadius|- DO
           $(RhoF * Cp * Temp(z,r))  =
           - PARTIAL (Vz * RhoF * Cp * Temp(z,r), Axial)
           + Kz * PARTIAL (Temp(z,r), Axial, Axial)
           + 1/r * (PARTIAL (Kr * r * PARTIAL (Temp(z,r), Radial), Radial))
           - RhoB * REACT(z,r) * ReactEnth                                  ;
      END  # for r
END  # for z
```

Figure 3.7: Example of higher order partial derivative

3.4.6.1 Single Integrals

The syntax of integrals in the proposed language is of the form:

```
INTEGRAL ( IntegralRange ; Expression )
```

where `Expression` is an expression distributed over a number of continuous or discrete domains, and `IntegralRange` represents the range of integration of the integrand. For instance, in the case of the tubular reactor model, an energy balance on the cooling medium bath yields the equation:

$$\rho_w c_w V_w \frac{dT_{cw}}{dt} = f_w c_w (T_{ci} - T_{cw}) + US \int_0^L (T(z,R) - T_{cw}) dz \tag{3.5}$$

In the proposed language, the above equation is written as shown in Figure 3.8a. In order to avoid ambiguity, it is prohibited to use the same auxiliary index in more than one domain of any variable in the integrand. Thus, it is illegal to specify an integral as follows:

```
INTEGRAL ( x := 0 : 1 ; Temp (x, x) )
```

when the variable `Temp` has been defined as

```
Temp      AS   DISTRIBUTION (Axial, Radial)  OF  Temperature
```

3.4.6.2 Multiple Integrals

By analogy to the syntax for higher-order partial derivatives, multiple integrals can be defined through either the nested application of the `INTEGRAL` operator or a short-hand notation. For example, the mean temperature, $\overline{T}$, in the entire reactor is given by

$$\overline{T} = \frac{2}{LR^2} \int_0^L \int_0^R rT(r,z) dr dz \tag{3.6}$$

which, in *gPROMS*, can be written in either of the two forms shown in Figure 3.8b.

```
RhoW * Cw * Vw * $Tcw =
    Fw * Cw * ( Tci - Tcw ) +
      U * S * INTEGRAL(z := 0 : ReactorLength ; Temp(z,ReactorRadius)-Tcw) ;
```

(a) Single integration

```
Tave = 2 / (ReactorLength * ReactorRadius^2 ) *
          INTEGRAL(z := 0:ReactorLength;
                        INTEGRAL(r := 0:ReactorRadius; r*Temp(z,r))) ;
```

or

```
Tave = 2 / (ReactorLength * ReactorRadius^2) *
          INTEGRAL(z := 0:ReactorLength; r := 0:ReactorRadius; r*Temp(z,r)) ;
```

(b) Alternative forms of multiple integrals

Figure 3.8: Examples of the use of the integral operator

```
# using SIGMA function
X(1:Nocomp,0) = C(1:Nocomp,0) / SIGMA(C(1:Nocomp,0))                ;

# using INTEGRAL operator
X(1:Nocomp,0) = C(1:Nocomp,0) / INTEGRAL(i := 1:Nocomp ; C(i,0)) ;
```

Figure 3.9: Example of SIGMA and INTEGRAL

3.4.6.3 Relationship between the INTEGRAL and SIGMA Operators

In the interests of geneality, the `INTEGRAL` operator can also be used for the integration of a given expression over a discrete domain. This can be viewed as an extension of the `SIGMA` function already used in *gPROMS* for carrying out discrete domain summations. For instance, the mole fraction of the reactant at the tubular reactor entrance described by

$$x_i = \frac{c_i}{\sum_k c_k} \tag{3.7}$$

can be represented in terms of either the `SIGMA` function or the `INTEGRAL` function, as shown in Figure 3.9. It should be noted, however, that `INTEGRAL` is more general than `SIGMA`: whereas the latter always results in a scalar by summing *all* dimensions or distribution domains of its argument, `INTEGRAL` can have a more narrowly specified summation domain.

Thus consider a 2-dimensional array variable `A(5,10)`. Then

```
SIGMA ( A(2:4, ) )
```

is the scalar $\sum_{i=2}^{4}\sum_{j=1}^{10} A_{ij}$ whereas

```
INTEGRAL ( i := 2:4 ; A(i, ) )
```

is a vector of length 10, the jth element of which is $\sum_{i=2}^{4} A_{ij}$.

3.5 Specification of Boundary Conditions

The *initial* and *boundary conditions* of a PDE system determine its unique solution from among an infinite number of solutions. From the view point of the numerical solution method, these auxiliary conditions are the starting points for obtaining the interior solution.

However, there is a subtle conceptual difference between initial conditions and boundary conditions as far as process modelling is concerned. Initial conditions define the initial state of the system of interest, and will generally differ from one simulation experiment to another. On the other hand, boundary conditions are, more fundamentally, part of the description of the physical system behaviour itself. For instance, the boundary conditions for the tubular reactor

$$\varepsilon D_z \frac{\partial c(,z,r)}{\partial z} = v_z(c(,z,r) - c_{in}) \qquad @z = 0 \tag{3.8}$$

$$\frac{\partial c(,z,r)}{\partial z} = 0 \qquad @z = L \tag{3.9}$$

are held to be true irrespective of the precise way in which the reactor is operated, provided $v_z > 0$.

On the basis of the above discussion, we conclude that the boundary conditions should be specified within the `MODEL` section describing the physical behaviour of the system. It could be argued that, from the point of view of model declaration, no distinction is really necessary between "equations" on one hand, and "boundary conditions" on the other, provided, of course, that the exact domain of definition of each such constraint be made known explicitly. However, some numerical methods for the solution of sets of PDEs handle the two kinds of relation in quite different ways. Consequently, in the proposed language, boundary conditions can be declared in a separate subsection `BOUNDARY` of the *MODEL* entity to the rest of the equations. It should be noted, however, that the same language syntax is used in both cases. Boundary conditions (equations 3.8-3.9) are described in *gPROMS* as follows.

```
BOUNDARY
   Void * Dz * PARTIAL(Conc( ,0, ), Axial) = Vz * (Conc( ,0,) - Cin) ;
   PARTIAL(Conc( ,ReactorLength, ), Axial) = 0                        ;
```

3.6 Solution Method Specification

The systems of IPDAEs defined in the *gPROMS MODEL*s are normally solved using the method of lines family of numerical methods (Schiesser, 1991). This involves discretisation of the distributed equations with respect to all spatial domains, which results in mixed sets of time-dependent DAEs. One obvious advantage of this approach is that it naturally allows for the simulation of processes involving both distributed and lumped parameter models. The latter are already described in terms of DAEs, and these can be combined with the DAEs arising from the discretisation of the IPDAEs. The resulting DAE system can be solved reliably using any of the currently available, sophisticated DAE solvers such as *DASSL* (Petzold, 1982) or *DASOLV* (Jarvis and Pantelides, 1992; Jarvis, 1993).

A number of different techniques fall within the method of lines family of methods depending on the manner chosen for discretising the spatial domains. Ideally, the method of discretising each domain should be selected automatically (see, for instance, the recent work of Pfeiffer and Marquardt, 1993) – or, indeed, a single discretisation method that can deal efficiently with all forms of equations and boundary conditions would be highly desirable. However, this is unlikely to be feasible for some time, and we therefore have to provide some mechanism for the user to specify the preferred discretisation method.

The appropriate discretisation method depends on the characteristics of the problem. For instance, finite difference methods may be appropriate for problems involving a strong convective component, whereas for predominantly dispersive system it may be possible to achieve higher accuracy at lower cost through global or local polynomial approximation methods (*e.g.* collocation on finite elements). Of course, different methods may be used for discretising different domains within the same distributed parameter system.

The experience gained from our various numerical experiments and the examination of existing PDE and CFD software packages leads to the conclusion that three specifications are essential to determine completely a number of the most common solution strategies:

- *Type of numerical method* : The proper choice of numerical method is often the critical decision for solving the system of IPDAEs. As we mentioned earlier, because no

Keyword	Type of Numerical Method
CFDM	centered finite difference method
BFDM	backward finite difference method
FFDM	forward finite difference method
UFDM	upwind-biased finite difference method
OCFEM	orthogonal collocation in finite element method

Table 3.2: Numerical methods implemented in *gPROMS*

numerical method is reliable for all IPDAE problems, the incorrect choice of numerical method may lead to physically unrealistic solutions, or even fail to obtain any results.

- *Order of approximation* : The order of approximation for partial derivatives and integrals in finite difference methods, and the degree of polynomials used in finite element methods has a great influence on the accuracy of solution. This is especially true if coarse grids or only a small number of elements are used.

- *Number of elements* : The number of discretisation intervals in finite difference methods and the number of elements in finite element methods are also of great significance in determining the solution trajectory. If a user uses coarse grids or only a small number of elements for a steep gradient problem, the result from the simulation may be far from accurate. On the other hand, if a user uses too fine a grid or too many elements, the required computational efforts will be increased drastically, leading to an inefficient solution procedure.

The proposed modelling language allows the user to specify the type of spatial approximation method (*e.g.* finite differences, finite elements *etc.*), as well as the granularity (*e.g.* number of finite difference nodes, number of finite elements *etc.*) and the order (*e.g.* first, second *etc.*) of the approximation. This information is treated as a MODEL parameter that can be specified using the SET operator:

```
SET
   DistributionDomain := [ arg1, arg2, arg3 ] ;
```

where arg1, arg2 and arg3 correspond to the type of the spatial approximation, the order of approximation and the granularity respectively. Table 3.2 illustrates the currently available numerical methods and their corresponding keywords in the language. The specification of numerical methods is done separately for each distribution domain in each instance of a distributed parameter model, thus allowing maximal flexibility in this respect. *gPROMS*

```
PROCESS StartUpSimulation

UNIT R101  IS   TubularReactor

SET
   WITHIN R101 DO
   Axial  := [ CFDM,  2, 150 ] ;
   Radial := [ OCFEM, 3,   4 ] ;
END
      ..........
```

Figure 3.10: Solution method specification

automatically uses consistent approximations for all partial derivatives and integrals over a given domain, derived from the numerical method specified for this domain. The associated issues are discussed in detail in the next chapter.

Although it is possible to specify numerical solution information within the *MODEL* itself, in the interests of model re-usability and generality, it is often better to associate it with specific *instances* of the *MODEL* entities. These are created in the *PROCESS* entities of *gPROMS*. A typical declaration of such information is shown in Figure 3.10 which illustrates the creation of an instance (*"UNIT"*) called `R101` of *MODEL* `TubularReactor`. The `Axial` domain within this instance is to be discretised using centered finite differences of second order over a uniform grid of 150 intervals (151 nodes). On the other hand, radial variations are to be handled using third-order orthogonal collocation over 4 finite elements.

3.7 Concluding Remarks

3.7.1 Summary

A significant number of chemical processes are described in term of IPDAEs. However, the current state of the art of the modelling environments does not support the direct modelling of combined lumped and distributed systems. A powerful modelling language was therefore introduced to tackle such systems, permitting the direct modelling of arbitrarily complex unit operations described in terms of mixed systems of IPDAEs over rectangular domains.

Since the work was performed within the framework of *gPROMS* language, the modelling formalism and hierarchical mechanism of *gPROMS* language were briefly reviewed.

A complication arises from the fact that, even within the model of a single unit operation, not all variables necessarily have the same degree of distribution. This has led to the introduction of the concepts of distribution domains, variable distributions and equation distributions. The proposed language allows the model developer to define explicitly the distribution domains that exist within the unit under consideration, and the degree of distribution of every system variable with respect to those domains.

Issues relating to the solution method specification were considered. It was noted that three items of information, namely the type of numerical method, the order of approximation and the number of discretisation elements, are essential to determine the solution strategies to solve IPDAE systems. The separate specification of numerical information with respect to each distribution domain guarantees maximal flexibility regarding the choice of numerical method.

3.7.2 Possible Extensions for Handling Some Problems with Sharp Fronts

The solution trajectories of some PDEs exhibit steep gradients in one or more spatial domains. In such cases, numerical methods based on fixed grids may be unsuitable, and adaptive grid methods (*cf.* section 2.2.6) may be preferable. However, as the computational cost associated with such methods may still be high and the current state-of-the-art is practically applicable to one dimensional problems only, they are not always the best way to solve steep gradient problems.

In some cases, the position of the steep gradient does not move much in time and is known a *priori.* An example of this is provided by crystallisation processes in which crystals are densely populated in certain narrow size ranges, while being sparsely distributed for all other sizes.

One possible way of tackling this kind of problem is to use fine grids for any sub-domain which contains a steep gradient and coarser grids elsewhere. The grids in each sub-domain are equally spaced and not moving in time. There is no limitation on the choice of a numerical method or the number of grids for each sub-domain.

An extension of the proposed language for providing the information of numerical solution methods is necessary in order to support this multiple sub-domain concept. Consider for instance the two dimensional domain shown in Figure 3.11 where a fine grid is desired for the middle third of the axial domain. This can be effected using the following specification:

```
SET
   Axial    :=  [ CFDM, 2, 3,  Length/3 ;
```

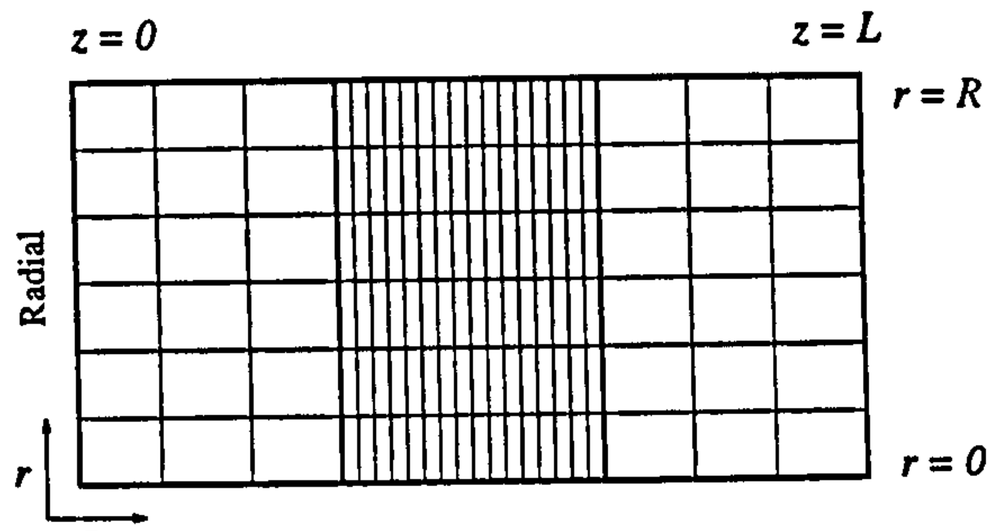


Figure 3.11: Sub-domains and their grids

```
                CFDM, 2, 16, Length/3 ;
                CFDM, 2, 3,  Length/3 ] ;

Radial  :=  [ OCFEM, 3, 6 ] ;
```

However, these extensions have not yet been implemented in *gPROMS*.

3.7.3 Some Comments on Modelling of Time Variations in *gPROMS*

We have seen on several occasions so far that time in *gPROMS* models is treated quite differently to other independent variables such as spatial position. Firstly, *all* variables are, in principle, assumed to be distributed with respect to time without this being declared explicitly in the model. A special operator ($) is used to denote partial differentiation with respect to this implicit domain. Secondly, the length of the time domain of interest is not defined within the model and neither are the initial conditions. As already explained in section 3.5, this type of information is viewed as part of the specific application (dynamic simulation) for which the model is used rather than being associated with the description of the physical system behaviour that the model attempts to capture. Thirdly, as explained in section 3.6, the numerical treatment of the time domain is different to that of the other distribution domains.

For most problem of practical interest, the above approach is well justified. However, it must be appreciated that it is not the only one possible in *gPROMS*. In fact, it is possible to treat time as just another distribution domain declared and handled explicitly within the `MODEL`.

Consider, for instance, modelling a system described by the one-dimensional hy-

perbolic PDE

$$\frac{\partial \phi}{\partial t} = -v_z \frac{\partial \phi}{\partial z} \qquad \forall z \in (0, L],\ t \in (0, T] \tag{3.10}$$

subject to the boundary condition

$$\phi(0, t) = 1 \qquad \forall t \in [0, T] \tag{3.11}$$

and the initial condition

$$\phi(z, 0) = 0 \qquad \forall z \in (0, L] \tag{3.12}$$

Two alternative ways of achieving this are shown in Figures 3.12 and 3.13.

The former adopts the standard implicit approach to handling time. Note that `phi` is declared explicitly to be a function of the spatial domain only, that the `$` operator is used to denote $\partial/\partial t$ and that the model includes the boundary but not the initial condition. The latter, together with the length of time horizon, will be introduced only when the model is used within a simulation.

On the other hand, the `MODEL` in Figure 3.13 adopts an explicit handling of time. This is now declared explicitly as a distribution domain in much the same way as the spatial domain. Both boundary and initial conditions are specified within the model.

The procedure used for the numerical solution of the two models is also different. In the case of the former, discretisation will be applied to the `ZDomain` reducing the problem to a set of ODEs. On the other hand, in the latter case both `ZDomain` and `TDomain` will be discretised resulting in a larger set of algebraic relations.

Both the computer memory and the computational cost required by the explicit handling of the time domain are likely to be higher than those associated with the implicit alternative. Also the guaranteed control of the temporal discretisation error is lost. However, explicit time discretisation may be advantageous in certain circumstances, especially for problems involving periodic or multipoint temporal conditions. One such example is presented in section 7.3.

```
MODEL Convect

PARAMETER
   ZLength        AS    Real

DISTRIBUTION_DOMAIN
   ZDomain        AS    ( 0 : ZLength )

VARIABLE
   phi            AS    DISTRIBUTION (ZDomain) OF notype
   Vz             AS                           velocity
                  . . .
BOUNDARY
   phi(0) = 1 ;

EQUATION
   FOR z := 0|+ TO ZLength DO
      $ phi(z) = -Vz * PARTIAL ( phi(z), ZDomain ) ;
   END
                  . . .
END
```

Figure 3.12: Implicit handling of time domain within *gPROMS*

```
MODEL Convect

PARAMETER
   ZLength, TLength          AS    Real

DISTRIBUTION_DOMAIN
   ZDomain                   AS    ( 0 : ZLength )
   TDomain                   AS    ( 0 : TLength )

VARIABLE
   phi                       AS    DISTRIBUTION (ZDomain,TDomain) OF notype
   Vz                        AS                                      velocity
                            . . .
BOUNDARY
   # at z = 0
   FOR t := 0 TO TLength DO
     phi(0,t) = 1
   END

   # at t = 0
   FOR z := 0|+ TO ZLength DO
     phi(z,0) = 0 ;
   END

EQUATION
   FOR z := 0|+ TO ZLength DO
      FOR t := 0|+ TO TLength DO
        PARTIAL ( phi(z,t), TDomain ) = -Vz * PARTIAL ( phi(z,t), ZDomain );
      END
   END
                            . . .
END
```

Figure 3.13: Explicit handling of time domain within *gPROMS*

Chapter 4

Solution Methods for IPDAEs

As has already been mentioned, the solution methods used in conjunction with the modelling system described in this thesis belong to the family of *methods of lines* (Schiesser, 1991). This is in fact a two-phase approach (see Figure 4.1). In the first phase, the spatial dimensions are discretised in terms of finite dimensional representations, and this reduces the IPDAEs into sets of DAEs with respect to time. In the second phase, the DAEs are integrated over the time horizon of interest using appropriate integration techniques.

Efficient and reliable techniques for the integration of DAEs over time have been considered extensively in the literature. A review of the current state of the art has recently been given by Pantelides and Barton (1993). In the rest of this chapter, we therefore concentrate on the first phase of the solution procedure, namely that concerned with the discretisation of the spatial domains.

The IPDAE systems of interest to us may involve an arbitrary number of spatial domains. However, it is worth noting at this stage that, since each of these domains is orthogonal to all others, it can effectively be discretised in isolation. The final discretisation is the cartesian product of the discretisations of all individual domains. It therefore suffices to examine discretisation methods for *single* spatial domain problems.

All discretisation schemes considered here have a number of common characteristics:

- the spatial variation of each distributed variable $\phi(z)$, $z \in [Z^L, Z^U]$ is approximated

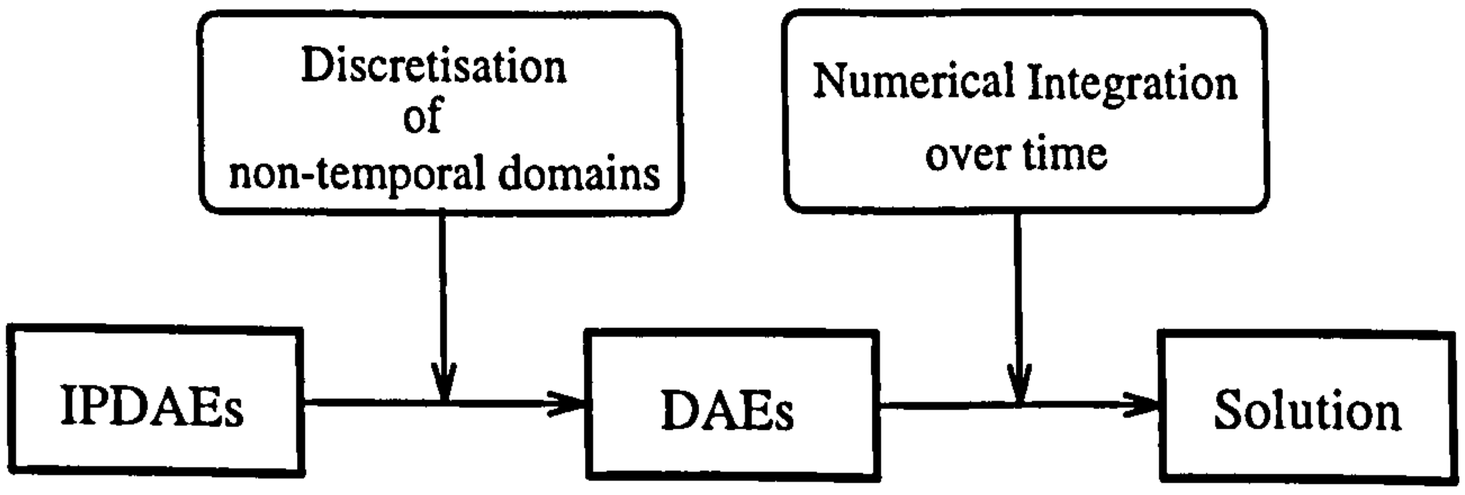


Figure 4.1: Two phases of solution strategy

in terms of the values of the variable $\phi(z_i)$ at a finite and fixed set of positions $z_i \in [Z^L, Z^U]$

- equations that are distributed over the domain $[Z^L, Z^U]$ are enforced at some of the points $\{z_i\}$ while other desirable properties of the solution (*e.g.* continuity) are enforced at others
- the partial derivatives of $\phi(z)$ at the points $\{z_i\}$ and integrals over the domain $[Z^L, Z^U]$ are approximated in terms of the values $\phi(z_i)$.

We consider two classes of method in particular: finite difference methods over uniform grids, and orthogonal collocation on finite elements. As will be seen in the next two sections, it is possible to derive these two apparently very different approaches in a fairly uniform and consistent manner. Gaussian quadrature formulae for the evaluation of integrals are also derived within the same unified framework.

4.1 Finite Difference Methods

The usual manner of deriving finite difference methods is based on Taylor expansions of the distributed variables about the grid points $\{z_i\}$. Here we prefer an approach based on polynomial approximations. This offers a more convenient route to the derivation both of expressions for spatial derivatives of arbitrary order of accuracy, and of consistent approximations for integrals.

In particular, we consider approximating the spatial variation of a distributed variable $\phi(z)$ in terms of its values at a set of $m+1$ grid points z_i, $i = 0..m$ that are uniformly spaced at a distance h apart as shown in Figure 4.2. A nth degree polynomial approximation can be constructed in terms of the values of the variable at $n+1$ consecutive points on the grid, say $\{z_i, \cdots, z_{i+n}\}$.

A particularly convenient way of expressing this approximation is in the form

$$\phi(z) \approx \sum_{j=0}^{n} \phi(z_{i+j}) L_j^{[n]}(z) \tag{4.1}$$

where $L_j^{[n]}(z)$ is a nth degree Lagrange polynomial defined as

$$L_j^{[n]}(z) \equiv \prod_{k=0, k\neq j}^{n} \frac{z - z_{i+k}}{z_{i+j} - z_{i+k}} \tag{4.2}$$

or, equivalently,

$$L_j^{[n]}(\hat{z}) \equiv \prod_{k=0, k\neq j}^{n} \frac{\hat{z} - k}{j - k} \tag{4.3}$$

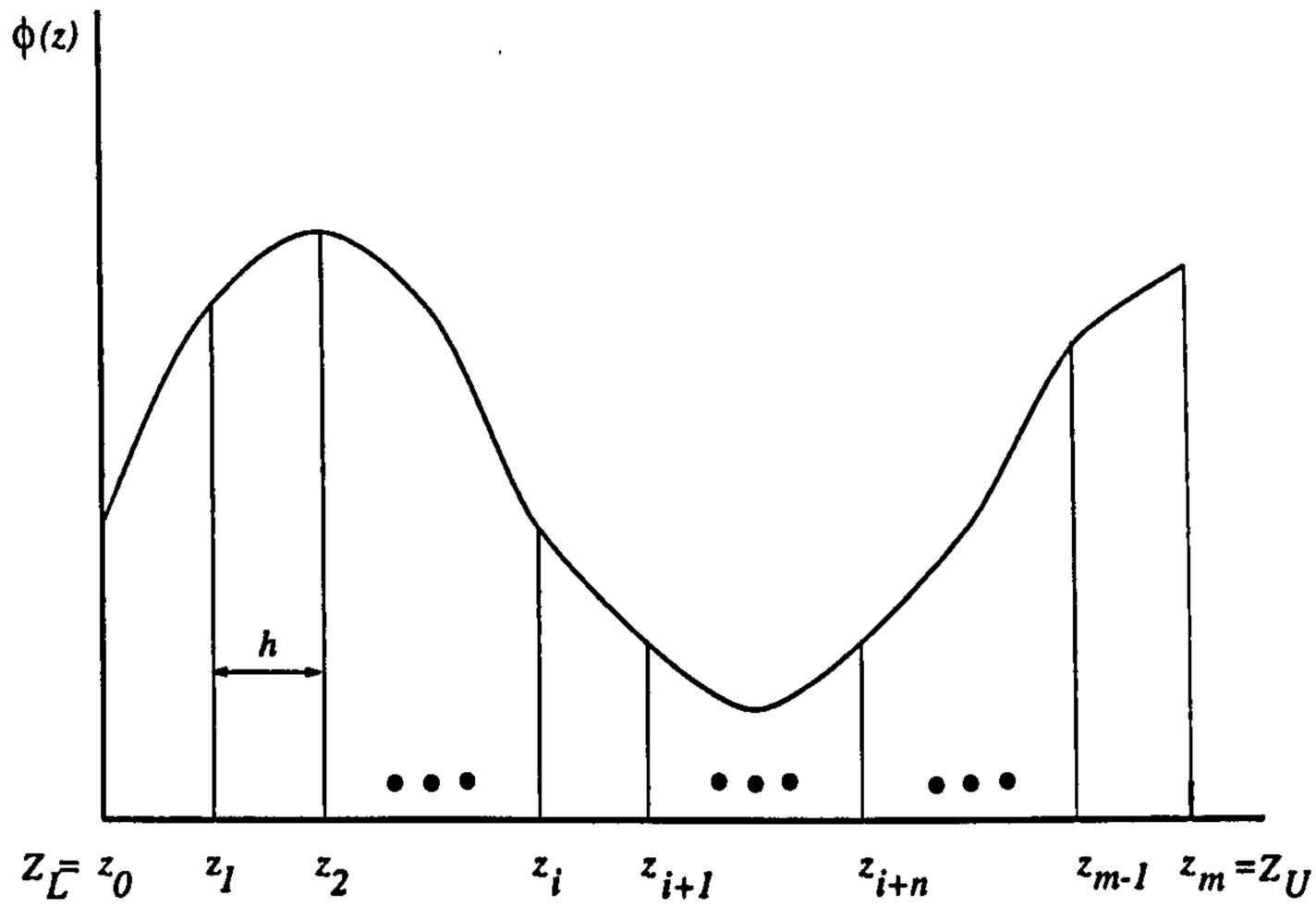


Figure 4.2: Approximation of a function with equally spaced abscissas

where $\hat{z}$ is the normalised position variable defined as

$$\hat{z} \equiv \frac{z - z_i}{h} \tag{4.4}$$

Note that, because of the uniformity of the grid, $z_k = z_i + kh$ and $z_j = z_i + jh$.

4.1.1 Approximations for First-Order Derivatives of $\phi(z)$

From equation 4.1, we can obtain expressions for the first spatial derivative of $\phi(z)$ at each of the grid points z_{i+q}, $q = 0..n$:

$$\frac{d\phi}{dz}(z_{i+q}) \approx \sum_{j=0}^{n} \phi(z_{i+j}) \frac{dL_j^{[n]}}{dz}(z_{i+q}) \qquad q = 0..n \tag{4.5}$$

By virtue of equation 4.3, the above can be written as

$$\frac{d\phi}{dz}(z_{i+q}) \approx \frac{1}{h} \sum_{j=0}^{n} A_{jq}^{[n]} \phi(z_{i+j}) \qquad q = 0..n \tag{4.6}$$

where $A_{jq}^{[n]}$ is a constant $(n+1) \times (n+1)$ matrix defined by

$$A_{jq}^{[n]} \equiv \frac{dL_j^{[n]}}{d\hat{z}}(q) \qquad j, q = 0..n \tag{4.7}$$

Typical matrices $A^{[n]}$ are listed below for $n = 1$, 2, 4 and 6.

$$A^{[1]} = \begin{bmatrix} -1 & -1 \\ 1 & 1 \end{bmatrix}; \tag{4.8a}$$

$$A^{[2]} = \frac{1}{2}\begin{bmatrix} -3 & -1 & 1 \\ 4 & 0 & -4 \\ -1 & 1 & 3 \end{bmatrix}; \tag{4.8b}$$

$$A^{[4]} = \frac{1}{24}\begin{bmatrix} -50 & -6 & 2 & -2 & 6 \\ 96 & -20 & -16 & 12 & -32 \\ -72 & 36 & 0 & -36 & 72 \\ 32 & -12 & 16 & 20 & -96 \\ -6 & 2 & -2 & 6 & 50 \end{bmatrix}; \tag{4.8c}$$

$$A^{[6]} = \frac{1}{720}\begin{bmatrix} -1764 & -120 & 24 & -12 & 12 & -24 & 120 \\ 4320 & -924 & -288 & 108 & -96 & 180 & -864 \\ -5400 & 1800 & -420 & -540 & 360 & -600 & 2700 \\ 4800 & -1200 & 960 & 0 & -960 & 1200 & -4800 \\ -2700 & 600 & -360 & 540 & 420 & -1800 & 5400 \\ 864 & -180 & 96 & -108 & 288 & 924 & -4320 \\ -120 & 24 & -12 & 12 & -24 & 120 & 1764 \end{bmatrix} \tag{4.8d}$$

Equation 4.6 gives expressions for the spatial derivatives of $\phi(z)$ at the $(n+1)$ distinct points $z_{i+q}, q = 0..n$. Alternatively, we can derive $(n+1)$ different expressions for the spatial derivative at z_i simply by translating the expression for $d\phi(z_{i+q})/dz$ over q grid positions to the left:

$$\frac{d\phi}{dz}(z_i) \approx \frac{1}{h}\sum_{j=0}^{n} A_{jq}^{[n]}\phi(z_{i-q+j}) \qquad q = 0..n \tag{4.9}$$

For instance, equations 4.8a and 4.9 lead to the following two expressions for the first spatial derivative of $\phi(z)$:

$$\frac{d\phi}{dz}(z_i) \approx \frac{-\phi(z_i) + \phi(z_{i+1})}{h} \tag{4.10a}$$

$$\frac{d\phi}{dz}(z_i) \approx \frac{-\phi(z_{i-1}) + \phi(z_i)}{h} \tag{4.10b}$$

which are immediately recognised as the first-order forward and backward finite difference approximations respectively. Similarly, three expressions can be derived for the same derivative from equations 4.8b and 4.9:

$$\frac{d\phi}{dz}(z_i) \approx \frac{-3\phi(z_i) + 4\phi(z_{i+1}) - \phi(z_{i+2})}{2h} \tag{4.11a}$$

$$\frac{d\phi}{dz}(z_i) \approx \frac{-\phi(z_{i-1}) + \phi(z_{i+1})}{2h} \tag{4.11b}$$

$$\frac{d\phi}{dz}(z_i) \approx \frac{\phi(z_{i-2}) - 4\phi(z_{i-1}) + 3\phi(z_i)}{2h} \tag{4.11c}$$

which correspond to the backward, centered and forward second-order finite difference approximations to $d\phi(z_i)/dz$.

In the same vein, five different fourth-order approximations can be derived from equations 4.8c and 4.9:

$$\frac{d\phi(z_i)}{dz} \approx \frac{1}{24h}\{-50\phi(z_i) + 96\phi(z_{i+1}) - 72\phi(z_{i+2}) + 32\phi(z_{i+3}) - 6\phi(z_{i+4})\} \tag{4.12a}$$

$$\frac{d\phi(z_i)}{dz} \approx \frac{1}{24h}\{-6\phi(z_{i-1}) - 20\phi(z_i) + 36\phi(z_{i+1}) - 12\phi(z_{i+2}) + 2\phi(z_{i+3})\} \tag{4.12b}$$

$$\frac{d\phi(z_i)}{dz} \approx \frac{1}{24h}\{2\phi(z_{i-2}) - 16\phi(z_{i-1}) + 16\phi(z_{i+1}) - 2\phi(z_{i+2})\} \tag{4.12c}$$

$$\frac{d\phi(z_i)}{dz} \approx \frac{1}{24h}\{-2\phi(z_{i-3}) + 12\phi(z_{i-2}) - 36\phi(z_{i-1}) + 20\phi(z_i) + 6\phi(z_{i+1})\} \tag{4.12d}$$

$$\frac{d\phi(z_i)}{dz} \approx \frac{1}{24h}\{6\phi(z_{i-4}) - 32\phi(z_{i-3}) + 72\phi(z_{i-2}) - 96\phi(z_{i-1}) + 50\phi(z_i)\} \tag{4.12e}$$

These include the standard forward, centered and backward finite difference approximations (equation 4.12a, 4.12c and 4.12e respectively), as well as two less commonly encountered *biased* approximations (equation 4.12b and 4.12d) involving points $\{z_{i-1}, z_i, z_{i+1}, z_{i+2}, z_{i+3}\}$ and points $\{z_{i-3}, z_{i-2}, z_{i-1}, z_i, z_{i+1}\}$ respectively.

4.1.2 Approximations for Second-Order Derivatives of $\phi(z)$

Approximations to second-order spatial derivatives of $\phi(z)$ can be derived in a manner similar and consistent to that used for first-order derivatives. They are generally of the form

$$\frac{d\phi^2}{dz^2}(z_i) \approx \frac{1}{h^2}\sum_{j=0}^{n} B_{jq}^{[n]}\phi(z_{i-q+j}) \qquad q = 0..n \tag{4.13}$$

where $B_{jq}^{[n]}$ is a *constant* $(n+1) \times (n+1)$ matrix defined by

$$B_{jq}^{[n]} \equiv \frac{d^2 L_j^{[n]}}{d\hat{z}^2}(q) \qquad j, q = 0..n \tag{4.14}$$

For $n = 1$, B corresponds to a null 2×2 matrix. The matrices $B^{[n]}$ for orders $n = 2$, 4 and 6 are listed below:

$$B^{[2]} = \begin{bmatrix} 1 & 1 & 1 \\ -2 & -2 & -2 \\ 1 & 1 & 1 \end{bmatrix}; \tag{4.15a}$$

$$B^{[4]} = \frac{1}{12}\begin{bmatrix} 35 & 11 & -1 & -1 & 11 \\ -104 & -20 & 16 & 4 & -56 \\ 114 & 6 & -30 & 6 & 114 \\ -56 & 4 & 16 & -20 & -104 \\ 11 & -1 & -1 & 11 & 35 \end{bmatrix}; \tag{4.15b}$$

$$B^{[6]} = \frac{1}{360}\begin{bmatrix} 1624 & 274 & -26 & 4 & 4 & -26 & 274 \\ -6264 & -294 & 456 & -54 & -24 & 186 & -1944 \\ 10530 & -510 & -840 & 540 & 30 & -570 & 5940 \\ -10160 & 940 & 400 & -980 & 400 & 940 & -10160 \\ 5940 & -570 & 30 & 540 & -840 & -510 & 10530 \\ -1944 & 186 & -24 & -54 & 456 & -294 & -6264 \\ 274 & -26 & 4 & 4 & -26 & 274 & 1624 \end{bmatrix} \tag{4.15c}$$

Thus equations 4.13 and 4.15a lead to the standard second-order forward, centered and backward finite difference approximations to $d^2\phi(z_i)/dz^2$:

$$\frac{d^2\phi}{dz^2}(z_i) \approx \frac{\phi(z_i) - 2\phi(z_{i+1}) + \phi(z_{i+2})}{h^2} \tag{4.16a}$$

$$\frac{d^2\phi}{dz^2}(z_i) \approx \frac{\phi(z_{i-1}) - 2\phi(z_i) + \phi(z_{i+1})}{h^2} \tag{4.16b}$$

$$\frac{d^2\phi}{dz^2}(z_i) \approx \frac{\phi(z_{i-2}) - 2\phi(z_{i-1}) + \phi(z_i)}{h^2} \tag{4.16c}$$

Also, fourth-order finite difference approximations to $d^2\phi(z_i)/dz^2$ are derived from equations 4.13 and 4.15b:

$$\frac{d^2\phi(z_i)}{dz^2} \approx \frac{1}{12h^2}\{35\phi(z_i) - 104\phi(z_{i+1}) + 114\phi(z_{i+2}) - 56\phi(z_{i+3}) + 11\phi(z_{i+4})\} \tag{4.17a}$$

$$\frac{d^2\phi(z_i)}{dz^2} \approx \frac{1}{12h^2}\{11\phi(z_{i-1}) - 20\phi(z_i) + 6\phi(z_{i+1}) + 4\phi(z_{i+2}) - \phi(z_{i+3})\} \tag{4.17b}$$

$$\frac{d^2\phi(z_i)}{dz^2} \approx \frac{1}{12h^2}\{-\phi(z_{i-2}) + 16\phi(z_{i-1}) - 30\phi(z_i) + 16\phi(z_{i+1}) - \phi(z_{i+2})\} \tag{4.17c}$$

$$\frac{d^2\phi(z_i)}{dz^2} \approx \frac{1}{12h^2}\{-\phi(z_{i-3}) + 4\phi(z_{i-2}) + 6\phi(z_{i-1}) - 20\phi(z_i) + 11\phi(z_{i+1})\} \tag{4.17d}$$

$$\frac{d^2\phi(z_i)}{dz^2} \approx \frac{1}{12h^2}\{11\phi(z_{i-4}) - 56\phi(z_{i-3}) + 114\phi(z_{i-2}) - 104\phi(z_{i-1}) + 35\phi(z_i)\} \tag{4.17e}$$

4.1.3 Approximations for Integrals of $\phi(z)$

Using the polynomial interpolant 4.1 of $\phi(z)$, we can derive expressions to the integral of the latter over any one of the sub-intervals $[z_{i+q}, z_{i+q+1}]$, $q = 0..n-1$. These are

of the form:

$$\int_{z_{i+q}}^{z_{i+q+1}} \phi(z)dz \approx \sum_{j=0}^{n} \phi(z_{i+j}) \int_{z_{i+q}}^{z_{i+q+1}} L_j^{[n]}(z)dz = h\sum_{j=0}^{n} C_{jq}^{[n]} \phi(z_{i+j}) \qquad q = 0..n-1 \quad (4.18)$$

where $C^{[n]}$ is a constant $(n+1) \times n$ matrix defined by

$$C_{jq}^{[n]} \equiv \int_{q}^{q+1} L_j^{[n]}(\hat{z})d\hat{z} \qquad j = 0..n,\ q = 0..n-1 \quad (4.19)$$

Typical matrices $C^{[n]}$ are listed below for $n = 1$, 2 and 4:

$$C^{[1]} = \begin{bmatrix} 0.50000 \\ 0.50000 \end{bmatrix}; \quad (4.20a)$$

$$C^{[2]} = \begin{bmatrix} 0.41667 & -0.08333 \\ 0.66667 & 0.66667 \\ -0.08333 & 0.41667 \end{bmatrix}; \quad (4.20b)$$

$$C^{[4]} = \begin{bmatrix} 0.34861 & -0.02639 & 0.01528 & -0.02639 \\ 0.89722 & 0.48056 & -0.10278 & 0.14722 \\ -0.36667 & 0.63333 & 0.63333 & -0.36667 \\ 0.14722 & -0.10278 & 0.48056 & 0.89722 \\ -0.02639 & 0.01528 & -0.02639 & 0.34861 \end{bmatrix}; \quad (4.20c)$$

$$C^{[6]} = \begin{bmatrix} 0.31559 & -0.01427 & -2.03719 & -13.78441 & -48.99552 & -127.61844 \\ 1.07659 & 0.41548 & 9.62103 & 65.27659 & 231.96548 & 604.27103 \\ -0.76820 & 0.77693 & -18.69876 & -129.76820 & -460.87932 & -1200.85501 \\ 0.62011 & -0.26878 & 21.28677 & 140.12011 & 495.73122 & 1292.28677 \\ -0.33418 & 0.12068 & -12.90362 & -85.83418 & -306.22307 & -800.24737 \\ 0.10437 & -0.03452 & 4.35992 & 29.20437 & 104.41548 & 271.90992 \\ -0.03510 & -0.30802 & -2.14899 & -8.98510 & -26.70177 & -63.14274 \end{bmatrix} \quad (4.20d)$$

By translating equation 4.18 over q grid positions backwards, we can derive n different approximations for the integral of $\phi(z)$ over $[z_i, z_{i+1}]$. These are of the form:

$$\int_{z_i}^{z_{i+1}} \phi(z)dz \approx h\sum_{j=0}^{n} C_{jq}^{[n]} \phi(z_{i-q+j}) \qquad q = 0..n-1 \quad (4.21)$$

For instance, as shown in Figure 4.3, the interval $[z_i, z_{i+1}]$ is covered by four different polynomial approximants. We can therefore construct four different fourth-order

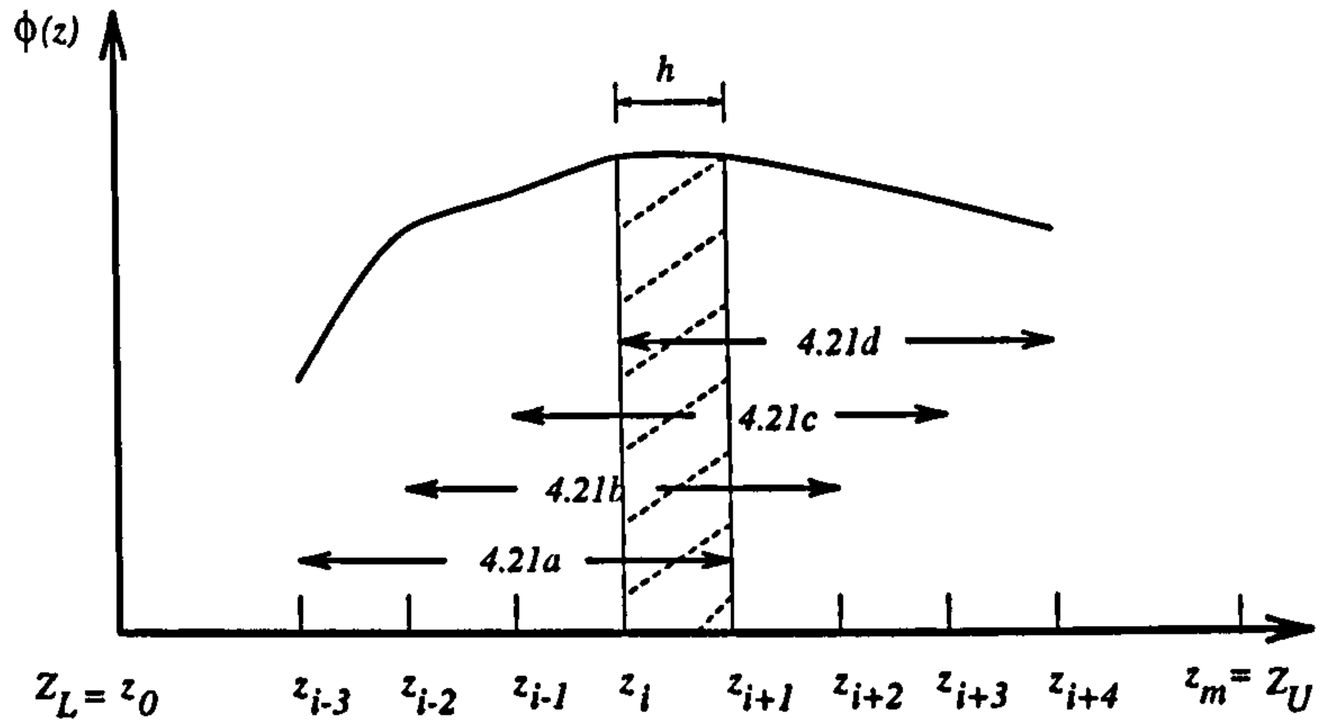


Figure 4.3: The integral of a function with equally spaced abscissas

approximations of the integral of a function $\phi(z)$ over $[z_i, z_{i+1}]$:

$$\int_{z_i}^{z_{i+1}} \phi(z)dz \approx 0.34861\phi(z_{i-3}) + 0.89722\phi(z_{i-2}) - 0.36667\phi(z_{i-1}) + 0.14722\phi(z_i) - 0.02639\phi(z_{i+1}) \quad (4.22a)$$

$$\int_{z_i}^{z_{i+1}} \phi(z)dz \approx -0.02639\phi(z_{i-2}) + 0.48956\phi(z_{i-1}) + 0.63333\phi(z_i) - 0.10278\phi(z_{i+1}) + 0.01528\phi(z_{i+2}) \quad (4.22b)$$

$$\int_{z_i}^{z_{i+1}} \phi(z)dz \approx 0.01528\phi(z_{i-1}) - 0.10278\phi(z_i) + 0.63333\phi(z_{i+1}) + 0.48056\phi(z_{i+2}) - 0.02639\phi(z_{i+3}) \quad (4.22c)$$

$$\int_{z_i}^{z_{i+1}} \phi(z)dz \approx -0.02639\phi(z_i) + 0.14722\phi(z_{i+1}) - 0.36667\phi(z_{i+2}) + 0.89722\phi(z_{i+3}) + 0.34861\phi(z_{i+4}) \quad (4.22d)$$

4.1.4 Comments on Consistency, Accuracy and Stability

The expressions derived in sections 4.1.1-4.1.3 provide $n+1$ finite difference approximations for first- and second-order spatial derivatives at point z_i (equations 4.9 and 4.13) and n approximations for integrals over interval $[z_i, z_{i+1}]$ (equation 4.21) of a given order n. *Consistent* approximation for these three quantities may be obtained by using the same value for the *translation index* q in all three expressions. We denote the approximations derived from an interpolant of order n following q leftward translations as "method (q, n)". We note that equation 4.21 is not defined for $q = n$, and in this case we adopt the convention that the expression for $q = n - 1$ is used instead.

For any one type of quantity, all expressions of the same order exhibit exactly

the same order accuracy of approximation since they were derived from the same interpolant equation 4.1. The choice among them is therefore made on the basis of *stability* considerations.

While parabolic PDEs with predominantly second-order spatial derivatives can be solved by the method of lines using centered finite difference approximations to the spatial derivatives, the substitution of a centered finite difference approximation into first-order hyperbolic PDEs introduces an *upwind numerical oscillation* which may cause serious numerical instability (Carver and Hinds, 1978). The use of first-order backward differences for a first derivative produces a stable procedure but may result in excessive *numerical diffusion* leading to inaccurate results. Of course, the numerical solution can be improved by using more grid points, but this may be inefficient.

One way to avoid numerical diffusion is using more *upwind* points, leading to mathematically higher order accuracy formulae. However, this may cause oscillations *downwind* of the point where the spatial derivatives are to be approximated.

To overcome the above drawbacks, Carver and Schiesser (1980) proposed the use of a *biased upwind* difference scheme to give a balance between accuracy and stability. An example of a five-points biased upwind approximation for $d\phi(z_i)/dz$ is given by equation 4.12b.

4.1.5 Avoiding the Creation of Fictitious Discretisation Points

The application of finite difference method (q, n) at point z_i on the grid generally involves the values of the variable $\phi(z)$ at points $z_{i-q}, \cdots, z_{i+n-q}$. We note that, if z_i lies at or near the boundaries of the domain (*i.e.* $i = 0$ or $i = m$), then some of these points may, in fact, lie outside the domain. It is difficult to arrive at a generally reasonable convention regarding the values of $\phi(z)$ at these fictitious points. We therefore consider using a different method, (q', n') specially applied at points at or near the boundary only, instead of the method (q, n) applied to all other points in the domain.

On possible approach is to maintain the same order of approximation (*i.e.* $n' = n$) and simply change the quantity q. In particular,

if $i - q < 0$,	then use method	(i, n)
if $i + n - q > m$,	then use method	$(i - n + m, n)$

We note that the necessary change in q will depend on the proximity of point i to the boundary, and that different methods may have to be used for different points

near the boundary. Consider, for instance, a centered fourth-order approximation to first-order partial derivatives (equation 4.12c). Then, according to the above rule, the following approximations must be used at various points over the domain (assuming that $m > 4$):

Point(s)	Approximation
z_0	4.12a
z_1	4.12b
$z_2..z_{m-2}$	4.12c
z_{m-1}	4.12d
z_m	4.12e

One potential problem with the above approach is that it may affect the stability of the discretised system, for instance, by using a forward finite difference scheme near the left boundary instead of the centered finite difference scheme used elsewhere. The alternative is to reduce the order ($n' < n$) while trying to maintain the nature of the approximation (*e.g.* forward, backward or centered) unchanged as far as possible. The latter requirement can simply be expressed as

$$\frac{q'}{n'} \approx \frac{q}{n}$$

We note that $q/n = 0$ corresponds to forward methods, $q/n = 1/2$ to centered methods and $q/n = 1$ to backward methods. Thus, in the above example, the centered *second*-order finite difference formula 4.11b could be used at points z_1 and z_{m-1} instead of 4.12c.

4.2 Orthogonal Collocation on Finite Elements

An orthogonal collocation method approximates the solution by weighted combinations of orthogonal polynomials of degree n, and demands that the describing equations be satisfied exactly at a finite set of points, called *collocation points.* The latter are taken as the roots of one of the orthogonal polynomials. Table 4.1 lists the normalised collocation points for the orthogonal polynomial of degree 2, 3 and 4 (Finlayson, 1980).

In many areas, such as reaction engineering, the orthogonal collocation method has proved to be a powerful method leading to accurate results (see, for example, Finlayson (1971) and Ferguson and Finlayson (1970)). However, when the solution has steep gradients, it is more beneficial to use it in conjunction with a finite element approach. As shown in Figure 4.4, the domain is divided into a number of sub-domains (elements) and an orthogonal collocation method is applied in each element. This gives rise to the *orthogonal collocation method on finite elements.*

Every individual element of the domain is numbered by its sequence index I $(1, 2, \cdots, m)$ and there exists a local numbering system $(0, 1, \cdots, n)$ for the collocation

degree of polynomial	collocation points
2	0.00000
	0.50000
	1.00000
3	0.00000
	0.21132
	0.78868
	1.00000
4	0.00000
	0.11270
	0.50000
	0.88730
	1.00000

Table 4.1: Normalised collocation points for orthogonal polynomials of degree 2, 3 and 4

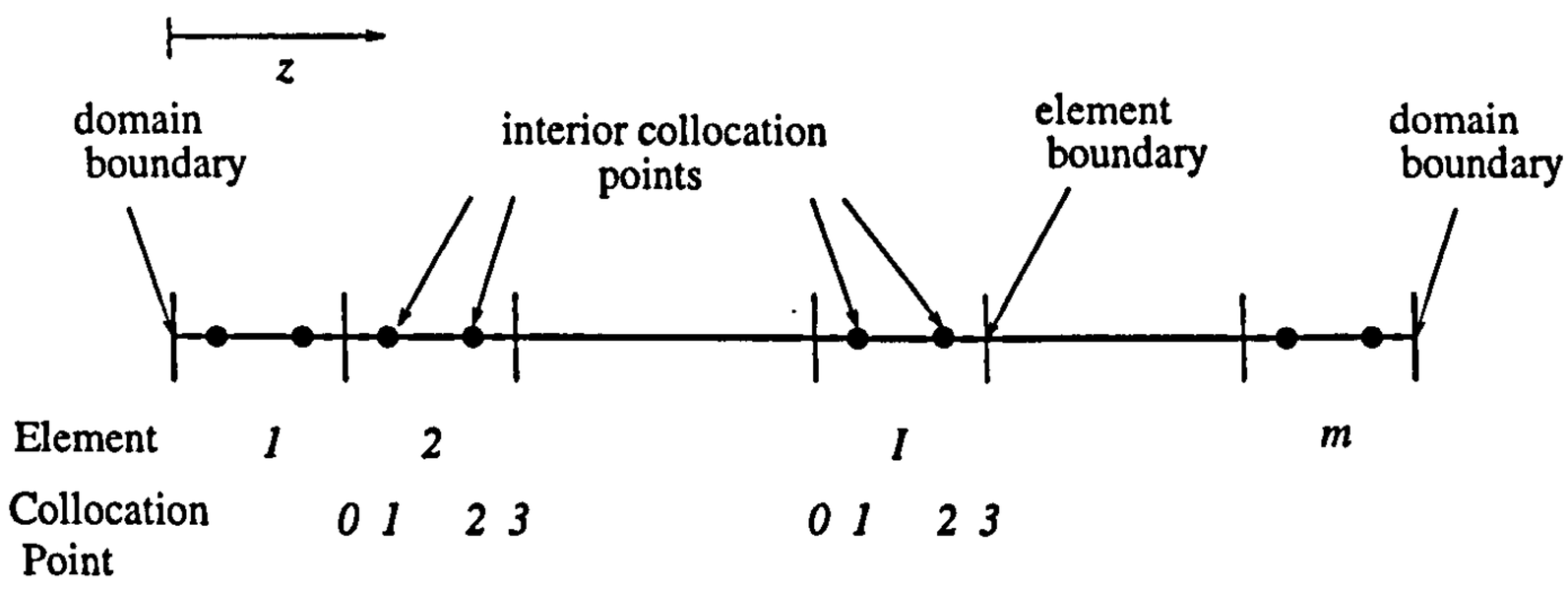


Figure 4.4: Grid numbering system of orthogonal collocation on finite elements of polynomials of degree 3

points within each element. The position of the jth point in element I is denoted by z_j^I. The function $\phi(z)$ in element I can be approximated by:

$$\phi(z^I) \approx \sum_{j=0}^{n} \phi(z_j^I) L_j^{[n]}(z^I) \qquad I = 1..m \tag{4.23}$$

where n is the degree of polynomial and m the number of elements.

The normalised form of equation 4.23 is:

$$\phi(z^I) \approx \sum_{j=0}^{n} \phi(z_j^I) L_j^{[n]}(\hat{z}) \qquad I = 1..m \tag{4.24}$$

where $\hat{z}$ is the normalised position within the element, and E^I the length of the element I, *i.e.* $z_n^I - z_0^I$.

4.2.1 Approximations for First-Order Derivatives of $\phi(z)$

From equation 4.24, the first-order derivative of the approximated solution $\phi(z)$ at position q in element I becomes

$$\frac{d\phi(z_q^I)}{dz} \approx \frac{1}{E^I} \sum_{j=0}^{n} A_{jq}^{[n]} \phi(z_j^I) \qquad I = 1..m, q = 0..n \tag{4.25}$$

where $A_{jq}^{[n]}$ is a constant $(n+1) \times (n+1)$ matrix defined by

$$A_{jq}^{[n]} \equiv \frac{dL_j^{[n]}(\hat{z}_q)}{d\hat{z}} \qquad j, q = 0..n \tag{4.26}$$

From the definition of Lagrange polynomials and the values in Table 4.1, we calculate matrices $A^{[n]}$ for $n = 2$, 3 and 4:

$$A^{[2]} = \begin{bmatrix} -3.00000 & -1.00000 & 1.00000 \\ 4.00000 & 0.00000 & -4.00000 \\ -1.00000 & 1.00000 & 3.00000 \end{bmatrix}; \tag{4.27a}$$

$$A^{[3]} = \begin{bmatrix} -7.00000 & -2.73205 & 0.73205 & -1.00000 \\ 8.19615 & 1.73205 & -1.73205 & 2.19615 \\ -2.19615 & 1.73205 & -1.73205 & -8.19615 \\ 1.00000 & -0.73205 & 2.73205 & 7.00000 \end{bmatrix}; \tag{4.27b}$$

$$A^{[4]} = \begin{bmatrix} -13.00000 & -5.32379 & 1.50000 & -0.67621 & 1.00000 \\ 14.78831 & 3.87298 & -3.22749 & 1.29099 & -1.87836 \\ -2.66667 & 2.06559 & 0.00000 & -2.06559 & 2.66667 \\ 1.87836 & -1.29099 & 3.22749 & -3.87298 & -14.78831 \\ -1.00000 & 0.67621 & -1.50000 & 5.32379 & 13.00000 \end{bmatrix} \tag{4.27c}$$

For instance, third-order approximations of first-order spatial derivatives in element I are given by:

$$\frac{d\phi(z_0^I)}{dz} \approx \frac{1}{E^I}(-7\phi(z_0^I) + 8.19615\phi(z_1^I) - 2.19615\phi(z_2^I) + \phi(z_3^I)) \tag{4.28a}$$

$$\frac{d\phi(z_1^I)}{dz} \approx \frac{1}{E^I}(-2.73205\phi(z_0^I) + 1.73205\phi(z_1^I) + 1.73205\phi(z_2^I) - 0.73205\phi(z_3^I)) \tag{4.28b}$$

$$\frac{d\phi(z_2^I)}{dz} \approx \frac{1}{E^I}(0.73205\phi(z_0^I) - 1.73205\phi(z_1^I) - 1.73205\phi(z_2^I) + 2.73205\phi(z_3^I)) \tag{4.28c}$$

$$\frac{d\phi(z_3^I)}{dz} \approx \frac{1}{E^I}(-\phi(z_0^I) + 2.19615\phi(z_1^I) - 8.19615\phi(z_2^I) + 7\phi(z_3^I)) \tag{4.28d}$$

4.2.2 Approximations for Second-Order Derivatives of $\phi(z)$

With a procedure similar to that applied in section 4.2.1, we can obtain the second-order derivative of $\phi(z)$ at position q in element I:

$$\frac{d^2\phi(z_q^I)}{dz^2} \approx (\frac{1}{E^I})^2 \sum_{j=0}^{n} B_{jq}\phi(z_j^I) \qquad I = 1..m, q = 1..n \tag{4.29}$$

where $B_{jq}^{[n]}$ is a *constant* $(n+1) \times (n+1)$ matrix defined by

$$B_{jq}^{[n]} \equiv \frac{d^2 L_j^{[n]}(\hat{z}_q)}{d\hat{z}^2} \qquad j, q = 0..n \tag{4.30}$$

The matrices $B^{[n]}$ are listed below for $n = 2$, 3 and 4:

$$B^{[2]} = \begin{bmatrix} 4.00000 & 4.00000 & 4.00000 \\ -8.00000 & -8.00000 & -8.00000 \\ 4.00000 & 4.00000 & 4.00000 \end{bmatrix}; \tag{4.31a}$$

$$B^{[3]} = \begin{bmatrix} 24.00000 & 16.39230 & -4.39230 & -12.00000 \\ -37.17691 & -24.00000 & 12.00000 & 25.17691 \\ 25.17691 & 12.00000 & -24.00000 & -37.17691 \\ -12.00000 & -4.39230 & 16.39230 & 24.00000 \end{bmatrix}; \tag{4.31b}$$

$$B^{[4]} = \begin{bmatrix} 84.00000 & 53.23790 & -6.00000 & 6.76210 & 24.00000 \\ -122.06317 & -73.33333 & 16.66667 & -13.33333 & -44.60350 \\ 58.66667 & 26.66667 & -21.33333 & 26.66667 & 58.66667 \\ -44.60350 & -13.33333 & 16.66667 & -73.33333 & -122.06317 \\ 24.00000 & 6.76210 & -6.00000 & 53.23790 & 84.00000 \end{bmatrix} \tag{4.31c}$$

For instance, third-order approximations of second-order spatial derivatives in element I are given by:

$$\frac{d^2\phi(z_0^I)}{dz^2} \approx (\frac{1}{E^I})^2(24\phi(z_0^I) - 37.17691\phi(z_1^I) + 25.17691\phi(z_2^I) - 12\phi(z_3^I)) \tag{4.32a}$$

$$\frac{d^2\phi(z_1^I)}{dz^2} \approx (\frac{1}{E^I})^2(16.3923\phi(z_0^I) - 24\phi(z_1^I) + 12\phi(z_2^I) - 4.3923\phi(z_3^I)) \tag{4.32b}$$

$$\frac{d^2\phi(z_2^I)}{dz^2} \approx (\frac{1}{E^I})^2(-4.3923\phi(z_0^I) + 12\phi(z_1^I) - 24\phi(z_2^I) - 16.3923\phi(z_3^I)) \tag{4.32c}$$

$$\frac{d^2\phi(z_3^I)}{dz^2} \approx (\frac{1}{E^I})^2(-12\phi(z_0^I) + 25.17691\phi(z_1^I) - 37.17691\phi(z_2^I) + 24\phi(z_3^I)) \tag{4.32d}$$

4.2.3 Continuity at Element Boundaries

An important facet of all finite element methods is the treatment of the boundaries between elements. In general, the solution values are assumed to be continuous at the element boundaries, and this normally corresponds to physical reality. One could also make the first spatial derivative continuous across the interface, thus resulting in C_1 continuous solution approximations throughout the domain. However, in some cases (*e.g.* for inhomogeneous domains), it may be more appropriate to enforce continuity of some other quantity at the element boundaries, for instance dispersive mass flux or conductive heat flux.

As the generalisation of the latter is rather complicated, only the former is considered in the thesis. The continuity of the first derivative at each boundary, known as C_1 *continuity*, can be written as:

$$\frac{\partial\phi(z_n^I)}{\partial z}\Big|_- = \frac{\partial\phi(z_0^{I+1})}{\partial z}\Big|_+ \qquad I = 1..m-1 \tag{4.33}$$

and can be described in the numerical domain as follows:

$$\sum_{j=0}^{n}\phi(z_j^I)\frac{\partial L_j(\hat{z}_n)}{\partial\hat{z}} = \sum_{j=0}^{n}\phi(z_j^{I+1})\frac{\partial L_j(\hat{z}_0)}{\partial\hat{z}} \qquad I = 1..m-1 \tag{4.34}$$

or, equivalently,

$$\sum_{j=0}^{n}A_{jn}^{[n]}\phi(z_j^I) = \sum_{j=0}^{n}A_{j0}^{[n]}\phi(z_j^{I+1}) \qquad I = 1..m-1 \tag{4.35}$$

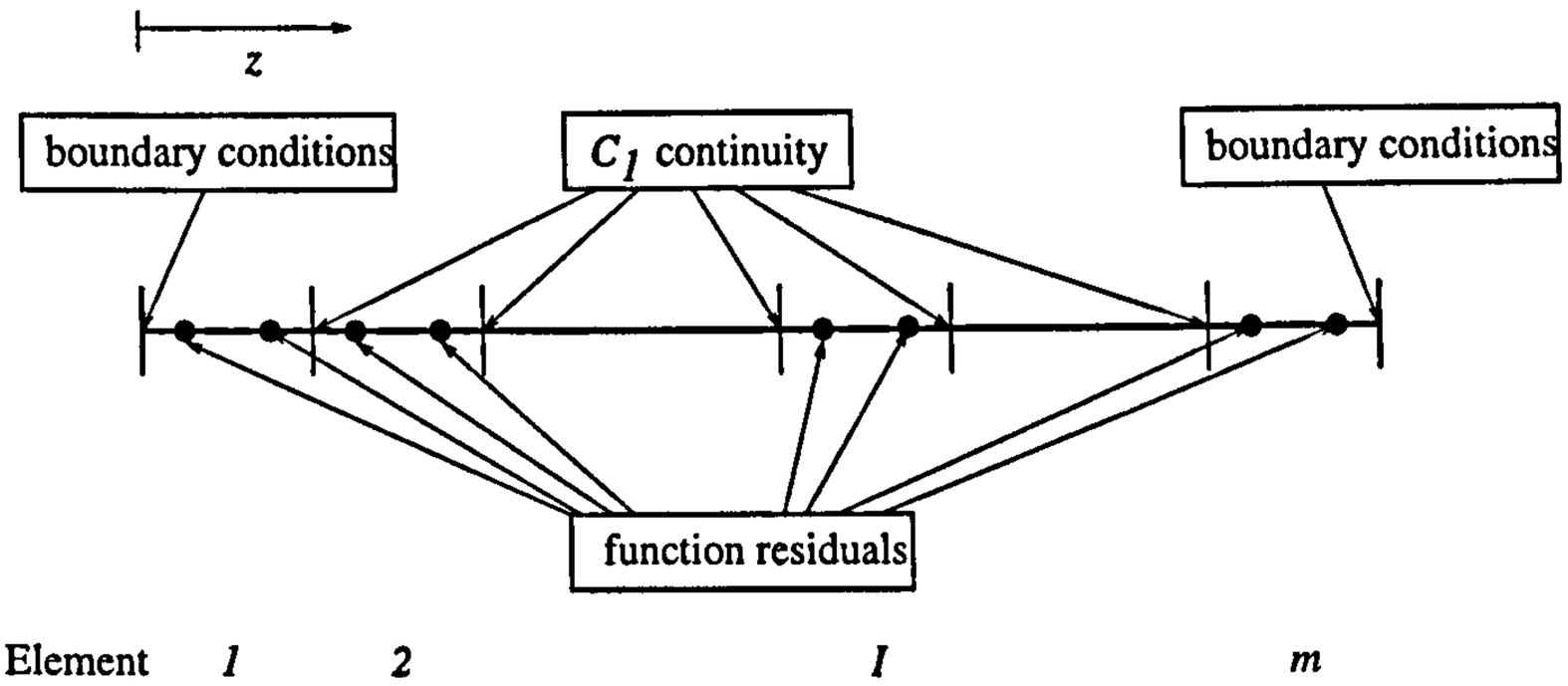


Figure 4.5: Boundary conditions, residuals and C_1 continuity on finite elements

where m is the number of elements.

In general, the discretisation of a PDE and its associated boundary conditions using the method of orthogonal collocation on finite elements results in three different classes of relation being applied at three different types of point (see Figure 4.5):

- the appropriate boundary conditions are applied at the domain boundaries
- the discretised PDE is enforced at the collocation points within each element
- C_1-continuity is enforced at the boundaries between elements.

4.2.4 Approximations for integrals of $\phi(z)$

Using the equations 4.24, the formula for the integral of $\phi(z)$ over any one of the sub-intervals $[z_q^I, z_{q+1}^I]$ can be derived as:

$$\int_{z_q^I}^{z_{q+1}^I} \phi(z)dz \approx E^I \sum_{j=0}^{n} C_{jq}^{[n]} \phi(z_j^I) \qquad I = 1..m, q = 0..n-1 \tag{4.36}$$

where m is the number of elements and $C_{jq}^{[n]}$ a constant $(n+1) \times n$ matrix described as:

$$C_{jq}^{[n]} \equiv \int_{\hat{z}_q}^{\hat{z}_{q+1}} L_j^{[n]}(\hat{z})d\hat{z} \tag{4.37}$$

From equation 4.37 and the normalised collocation points listed in Table 4.1, matrices $C_{jq}^{[n]}$ are calculated for $n = 2$, 3 and 4:

$$C^{[2]} = \begin{bmatrix} 0.20833 & -0.04167 \\ 0.33333 & 0.33333 \\ -0.04167 & 0.20833 \end{bmatrix}; \tag{4.38a}$$

$$C^{[3]} = \begin{bmatrix} 0.08978 & -0.09623 & 0.00645 \\ 0.12972 & 0.38490 & -0.01462 \\ -0.01462 & 0.38490 & 0.12972 \\ 0.00645 & -0.09623 & 0.08978 \end{bmatrix}; \tag{4.38b}$$

$$C^{[4]} = \begin{bmatrix} 0.04824 & -0.07949 & 0.03301 & -0.00176 \\ 0.06788 & 0.27272 & -0.06616 & 0.00333 \\ -0.00499 & 0.22722 & 0.22722 & -0.00499 \\ 0.00333 & -0.06616 & 0.27272 & 0.06788 \\ -0.00176 & 0.03301 & -0.07949 & 0.04824 \end{bmatrix} \tag{4.38c}$$

For instance, fourth order approximations of the integral of $\phi(z)$ over sub-intervals $[z_q^I, z_{q+1}^I]$ in the element I lead to:

$$\int_{z^I}^{z_1^I} \phi(z)dz \approx E^I(0.04824\phi(z_0^I) + 0.06788\phi(z_1^I) - 0.00499\phi(z_2^I) + 0.00333\phi(z_3^I) - 0.00176\phi(z_4^I)) \tag{4.39a}$$

$$\int_{z_1^I}^{z_2^I} \phi(z)dz \approx E^I(-0.07949\phi(z_0^I) + 0.27272\phi(z_1^I) + 0.27272\phi(z_2^I) - 0.06616\phi(z_3^I) + 0.03301\phi(z_4^I)) \tag{4.39b}$$

$$\int_{z_2^I}^{z_3^I} \phi(z)dz \approx E^I(0.03301\phi(z_0^I) - 0.06616\phi(z_1^I) + 0.22722\phi(z_2^I) + 0.27272\phi(z_3^I) - 0.07949\phi(z_4^I)) \tag{4.39c}$$

$$\int_{z_3^I}^{z_4^I} \phi(z)dz \approx E^I(-0.00176\phi(z_0^I) + 0.0333\phi(z_1^I) - 0.00499\phi(z_2^I) + 0.06788\phi(z_3^I) + 0.04824\phi(z_4^I)) \tag{4.39d}$$

4.2.5 Gaussian Quadrature

As has already been mentioned, *gPROMS* uses consistent approximation for partial derivatives and integrals according to the numerical method specified for the corresponding distribution domain. However, it is possible for a model to involve integrals of expressions which do not involve any distributed variable and therefore cannot be associated with any distribution domain (*e.g.* $\int_0^L e^{-z^2} dz$).

gPROMS employs Gaussian quadrature formulae for approximating such integrals. These are of the form:

$$\int_{z_{\min}}^{z_{\max}} \phi(z)dz \approx \sum_{j=0}^{n} w_j \phi(z_{\min} + L\hat{z}_j) \tag{4.40}$$

n	collocation points ($\hat{z}_j$)	weights ($\hat{w}_j$)
2	0.11270	0.27778
	0.50000	0.44444
	0.88730	0.27778
3	0.06943	0.17393
	0.33001	0.32607
	0.66999	0.32607
	0.93057	0.17393
4	0.04691	0.11846
	0.23077	0.23931
	0.50000	0.28444
	0.76923	0.23931
	0.95309	0.11846
5	0.03377	0.08566
	0.16940	0.18038
	0.38069	0.23396
	0.61931	0.23397
	0.83060	0.18038
	0.96623	0.08566

Table 4.2: Normalised collocation points and weight coefficients for Gaussian quadrature

where $L \equiv z_{\max} - z_{\min}$, and the weights w_j and normalised positions $\hat{z}_j$ are listed in Table 4.2 (Villadsen and Michelsen, 1978).

One interesting aspect of Gaussian quadrature is that it employs only *interior* collocation points to approximate integrals. For instance, Gaussian quadrature with degree 2 utilises three collocation points $\{0.1127, 0.5, 0.88730\}$ over the normalised domain [0, 1]. In fact, these points correspond to the interior collocation points in the orthogonal collocation method with degree 4 (see Table 4.1). This results in higher accuracy than the orthogonal collocation method of the same polynomial degree.

4.3 Concluding Remarks

In this chapter, we have considered methods for the approximation of functional variations over a single distribution domain. Two main families, namely finite difference methods and orthogonal collocation on finite elements, were examined in detail. This led to the derivation of approximations for first- and second-order partial derivatives and integrals in terms of the values of the functions at a finite set of points. Gaussian quadrature methods for the evaluation of integrals were also briefly considered.

It is interesting to note that, despite their apparent differences, the two families of method can be derived in a very similar fashion, namely by defining and manipulating interpolating polynomials over finite sets of points (see equations 4.1 and 4.24 respectively). In both cases, partial derivative approximations at a point z_k (or integral approximations over an interval $[z_k, z_{k+1}]$) involve the values of the function at a set of neighbouring points. The main difference between the two methods is in the composition of this set. For finite difference methods, this normally involves a fixed number of points to the left and right of the current point z_k. For orthogonal collocation on finite element methods, it involves all the points within the element to which z_k belongs.

The Lagrangian polynomial derivation of finite difference methods is particularly elegant in comparison to the more traditional derivations based on Taylor expansions. The introduction of the simple device of a "translation index" allows the derivation of *all* finite difference partial derivative approximations (*e.g.* forwards, backward, centered, biased) of a given order from a single formula. It also naturally leads to consistent integral and partial derivative approximations.

Finally, it should be noted that all methods in this chapter were concerned with a *single* distribution domain. The work reported in this thesis is restricted to rectangular domains. Therefore problems involving multiple domains can be handled simply by applying the appropriate discretisation method to each domain in sequence.

Chapter 5

Implementation

In the previous two chapters, the discussion has been centered on the general principles of a modelling language for distributed parameter system and the derivation of appropriate solution methods for IPDAEs. This chapter is concerned with the actual implementation of these concepts within the framework of the *gPROMS* package.

Modelling and simulation activities in *gPROMS* are carried out in two phases, namely the translation of an input file (translation phase) and the execution of process entities (execution phase). Section 5.1 surveys these two activities. Sections 5.2 and 5.3 then proceed to discuss the discretisation of variables and equations respectively in more detail.

5.1 *gPROMS* Software Architecture

gPROMS consists of three elements, namely the input translator, the process manager and the user interface environments. The overall structure is depicted in Figure 5.1. Among the elements shown, the translator and the process manager are closely associated with the discretisation of IPDAEs. Thus, we briefly review these two elements in this section. The results visualisation system (*gRMS*) is discussed in the next chapter.

5.1.1 Translator

The translator performs two major tasks: the analysis and validation of the input file containing the description of a simulation problem coded in the *gPROMS* language; and the generation of an internal model representation. As shown in Figure 5.1, the translator consists of three major elements: the scanner, the parser and the semantic routines.

The prime function of a scanner (Fischer and LeBlanc, 1988) is to analyse the input file by reading it character by character, and to group individual characters into tokens (identifiers, literals, delimiters, operators and reserved words). The tokens are encoded as integers and are fed to the parser for syntactic analysis.

The parser reads tokens from the scanner and groups them into units, as specified by the formal definition of the simulation language grammar. During operation, the parser

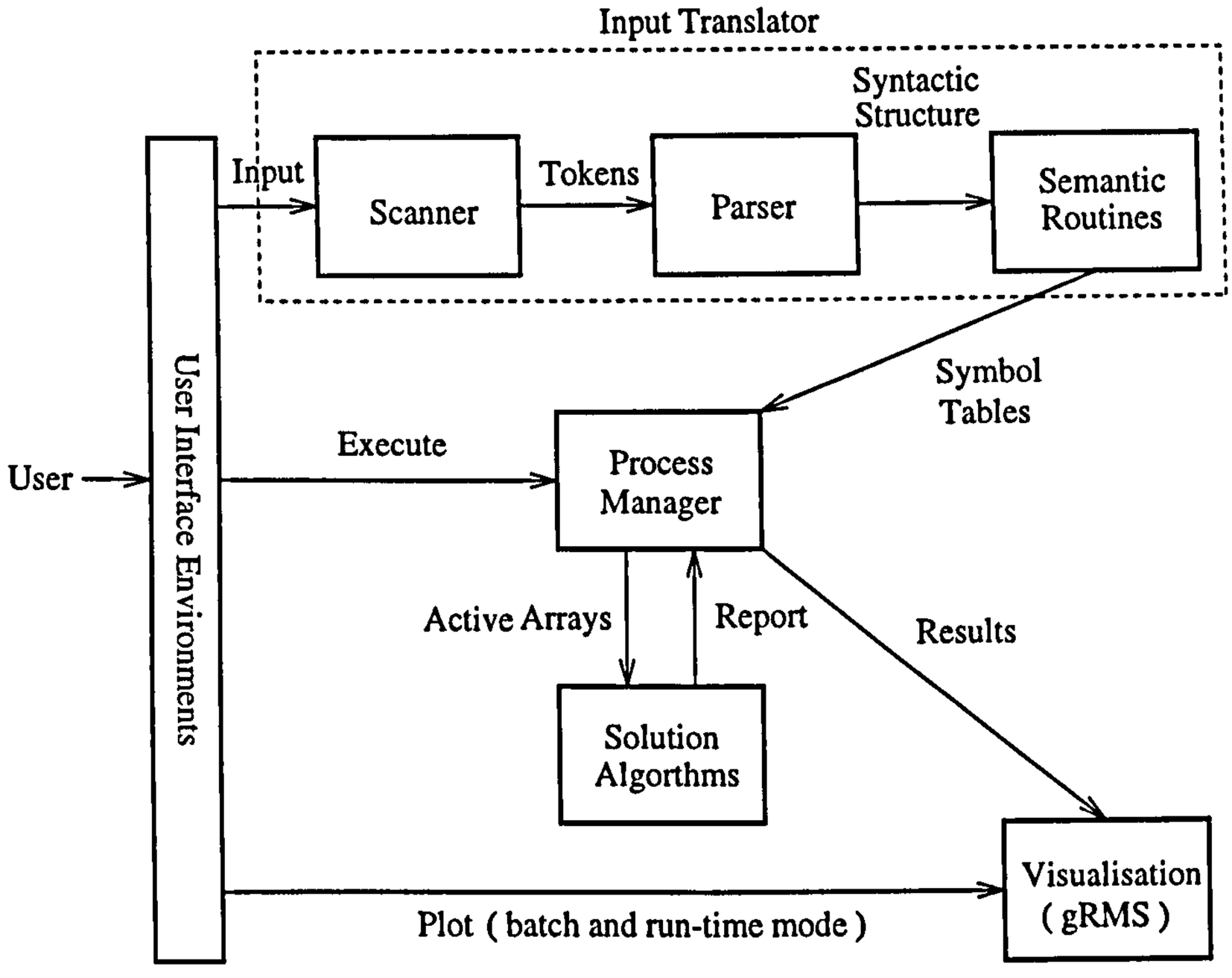


Figure 5.1: The basic structure of *gPROMS*

checks the syntax of the input file and calls the appropriate semantic routines when an action is required by the language definition. The parser is driven by the tables created from the context-free grammar by a *Modula-2* code parser generator based on *Bison* (Donnelly and Stallman, 1992). The input to *Bison* is exactly the same as that to the standard *UNIX* parser generator *Yacc* (Johnson, 1975), which takes the form of a file containing a formal declaration of context-free grammar. The declaration includes a list of tokens, the production rules as well as the calls to semantic routines as required (see Appendix A).

Semantic routines play two roles in the translation phase. Firstly, they verify that the input is legal and meaningful; for example, that the variables involved in an expression have already been declared, that the types of various entities are correct, and so on. Secondly, if the input is semantically correct, they generate the internal data structures, in the form of symbol tables, which are later utilised by the process manager. As a consequence of the adoption of an automatic parser generator, the coding of the semantic routines represents most of the effort required in the implementation of the translator.

5.1.2 Process Manager

It will be recalled (see section 3.1) that a *gPROMS* simulation experiment is described by a *PROCESS* which comprises a *TASK* (describing the externally applied manipulations and disturbances) driving a *MODEL* (describing the physical behaviour of the plant). To perform a simulation experiment, the corresponding *PROCESS* entity must be "executed".

The process manager deals with the execution of process entities, which involves the following four phases (Barton, 1992):

1. Instantiation of both the model and the task schedule.

2. Initialisation of the model.

3. Application of the task schedule to the model.

4. Recovery of the memory used during the simulation.

Among these phases, we concentrate on the issue of instantiation of the model, which is pertinent to the discretisation of IPDAEs.

Instantiation of the model is effectively concerned with generation of the data structures which the simulation algorithm employs in order to determine the time dependent behaviour of the system under consideration. During this phase, a representation of all attributes described in the model, for example parameters, variables, equations and so on, is created in memory. The data structures employed for this purpose take the form of packed arrays which allow rapid access to the information required for the evaluation of residuals and jacobian elements of the equations during the simulation. The following active arrays are required: (a) Active Parameter Array (APA) for parameter attributes, (b) Active Variable Array (AVA) for variable attributes, (c) Active Equation Array (AEA) for equation attributes, (d) Active Selector Array (ASA) for selector attributes, (e) Active Logical Condition Array (LCA) for logical conditions in `IF` and `CASE` constructs, (f) Active Scope Array (AScA) for all submodels in the overall system model and (g) Active Jacobian Array (AJA) for the jacobian elements of the AEA.

Of the above, it is the AVA and AEA which are related with the discretisation process of IPDAEs as detailed in the following sections.

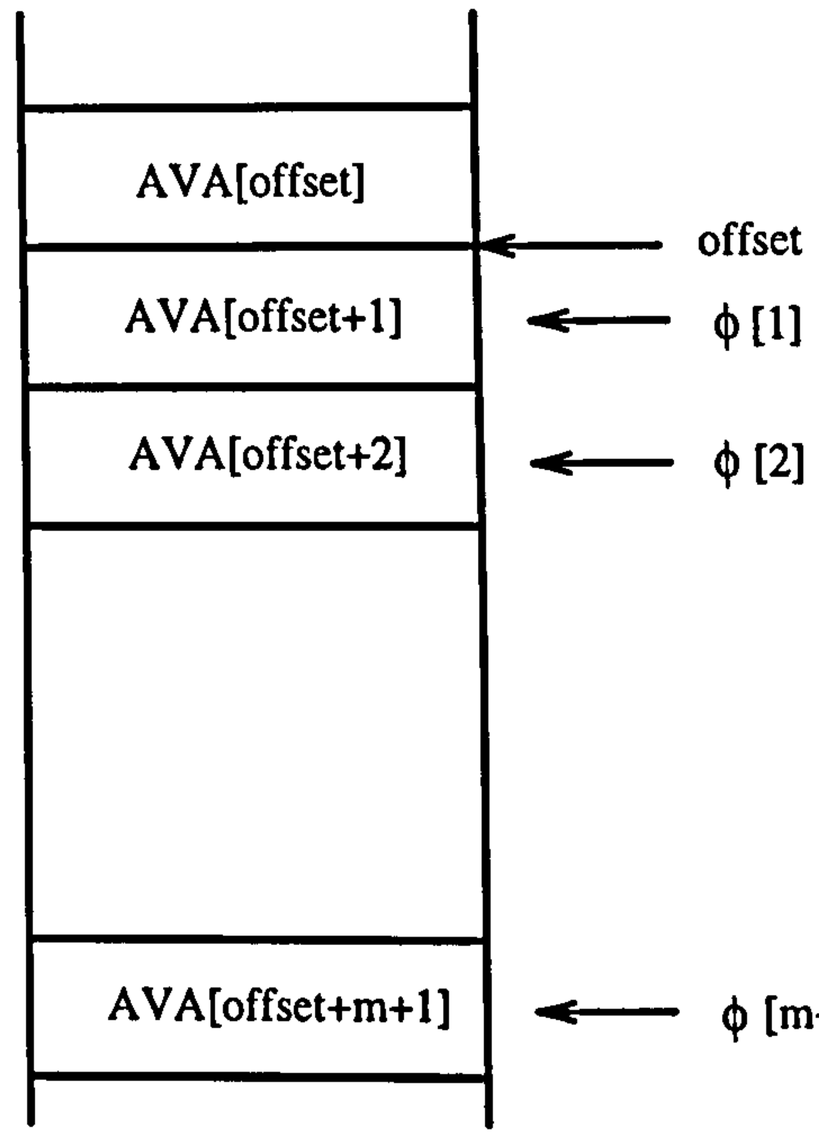


Figure 5.2: Data structure for variables

5.2 Discretisation of Variables

In general, the AVA contains one entry for each variable occurring in the DAE system resulting from the discretisation of the IPDAEs and which is to be integrated over time.

During the instantiation of the model, two types of variable are identified: *discrete variables* which are either not distributed at all (*i.e.* scalars) or distributed over discrete domains only, and *continuous variables* which are distributed over one or more continuous distribution domains. As discussed in chapter 3, the latter category of variable can also be distributed over discrete domains.

The treatment of the former is straightforward. Based on the information of dimensionality, active data structures (*active variables*) are created and stored in the AVA. For instance, if a discrete variable ϕ is distributed over the discrete domain $i \in [1..m+1]$, then $m+1$ number of active variables are created and stored in the AVA as shown in Figure 5.2.

If a continuous variable ϕ is distributed over a continuous domain $z \in [0:L]$ and is to be mapped into the discrete domain with m sub-intervals, then the discretisation process

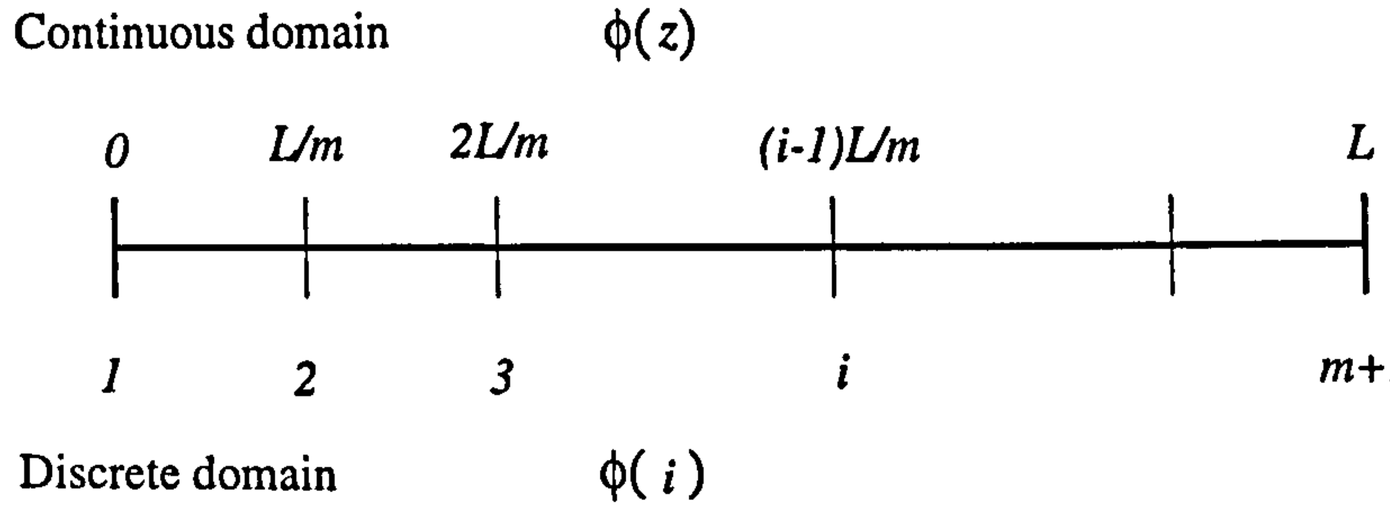


Figure 5.3: Discretisation of a continuous variable

of the continuous variable can be expressed by:

$$\phi(z) \longmapsto \phi(i) \qquad \forall z \in \mathcal{R} = [0, L],\ \forall i \in \mathcal{N} = [1..m+1] \tag{5.1}$$

The result of discretisation is shown in Figure 5.3. After the discretisation, the procedure for discrete variables is applied to the discretised variable $\phi(i)$, $i = 1..m+1$. Thus, the structure of AVA in this case is the same as that shown in Figure 5.2. Simply the entire process of discretising the continuous variable can be expressed by:

$$\phi(z) \longmapsto \phi(i) \longmapsto \text{AVA}[(n+1)..(n+m+1)]$$

5.3 Discretisation of IPDAEs

In general, the AEA contains one entry for each equation in the DAE system resulting from the discretisation of the IPDAEs and which is to be integrated over time.

Equation attributes are also classified into two types: *discrete equations* which are either scalar or distributed over discrete domains only, and *continuously distributed equations* which are distributed over one or more continuous domains (as well as, potentially, over one or more discrete domains).

The AEA entries (*active equations*) for discrete equations are created in the form of expressions that determine its residuals based on the range of the discrete domain. For example, if an equation $f(\phi) = 0$ is distributed over the discrete domain $i \in [1..m+1]$, $m+1$ active equations are created and stored in AEA as shown in Figure 5.4.

On the other hand, the treatment of the continuously distributed equations is more complicated, involving a number of steps. The first such step involves the symbolic simplification of the IPDAE system and is actually carried out during the input translation phase (see section 5.3.1). The subsequent step aims to map the continuous domain of definition

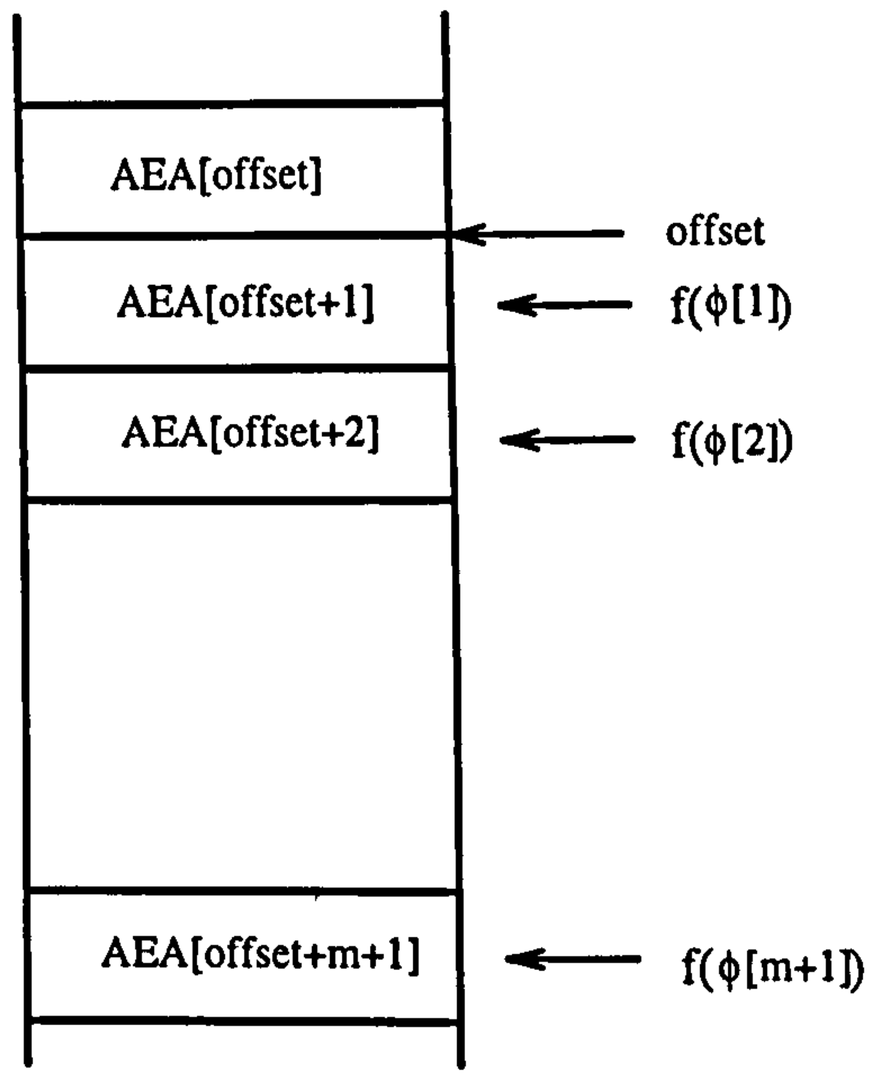


Figure 5.4: Data structure for equations

of each IPDAE onto the corresponding discretised domain and to establish precisely which condition is applied at each discretisation point (see section 5.3.2). Finally, enforcing an IPDAE at a given discretisation point may involve replacing a first or higher-order partial derivative or single or multiple integral by an appropriate approximation. These issues are dealt with in sections 5.3.3-5.3.5.

5.3.1 Symbolic Preprocessing of IPDAEs during Translation

Apart from the tasks explained in section 5.1.1, a special task associated with the transformation of partial derivatives specified by the user in a *conservative* form into an equivalent a *nonconservative* form takes place during the translation phase.

This is exemplified by the convective term in an energy balance equation which can be expressed either as

$$\frac{\partial}{\partial z}\{v(z)T(z)\} \qquad \text{or as} \qquad T(z)\frac{\partial v(z)}{\partial z} + v(z)\frac{\partial T(z)}{\partial z}$$

The language allows a user to describe this term in either of the two different ways, as illustrated in Figure 5.5.

As already discussed in section 3.4.5, there is a clear difference between analytical and numerical mathematics in this case. However, by automatically converting all partial derivatives described in a conservative form into the corresponding nonconservative forms

```
FOR z := 0|+ TO Length DO
  ... = - PARTIAL (v(z)*Temp(z),Axial) + ...
END
```

(a) Conservative form of a partial derivative

```
FOR z := 0|+ TO Length DO
  ... = - v(z)*PARTIAL (Temp(z),Axial) - Temp(z)*PARTIAL (v(z),Axial) + ...
END
```

(b) Nonconservative form of a partial derivative

Figure 5.5: Conservative and nonconservative form of partial derivatives

during translation, both expressions shown above yield the same symbolic representation, and the same results will be obtained.

In addition to the above symbolic transformations, some forms of partial derivatives and integrals can be calculated without being approximated numerically, thus providing a more accurate solution and reducing the memory and computational costs. One particularly simple case arises when either a partial differentiation or an integration operator is applied to an expression which is *not* distributed with respect to the corresponding independent variable.

In the first case, we obtain an analytically zero partial derivative. For instance, if in the above example, the variable v is not actually distributed over z, then $\partial v/\partial z = 0$. Overall, the symbolic analysis will convert the conservative form $\partial(v \cdot T(z))/\partial z$ into the simpler form $v \cdot \partial T(z)/\partial z$.

In the second case, when there is no functional relationship between an expression being integrated and the independent variable, the integral is easily calculated analytically as follows:

$$\int_{z_l}^{z_u} \phi(x)dz \equiv \phi(x)(z_u - z_l) \tag{5.2}$$

Whenever this situation is detected during the analysis, the integral expression is converted into its corresponding algebraic expression.

5.3.2 IPDAE Discretisation during Model Instantiation

As already mentioned, model instantiation takes place as the first step of executing a process describing a simulation experiment. At this point, several entries are created in the AEA for each distributed equation. The precise number of such entries depends on the domain of distribution of the equation, whether this domain is open or closed at each of its two ends, and the type of discretisation method specified for it.

Consider an equation, the domain of distribution of which (as determined by the rules of section 3.4.3) is defined by the points z_l and z_u ($z_l < z_u$). A limitation of the current implementation is that both z_l and z_u must coincide exactly with two points (say j and k) on the discretisation grid specified for the corresponding independent variable.

If the distribution domain of the equation is closed at z_l, then the first discretisation point for this equation is j, otherwise it is $j + 1$. Similarly, if the domain is closed at z_u, the last discretisation point is k, otherwise it is $k - 1$.

Once the first and last discretisation points are established, entries are created in the AEA by enforcing the given equation at these and intermediate points. If a finite difference method is to be applied to the particular distribution domain, then *all* intermediate points are used in this way. On the other hand, if an orthogonal collocation method on finite elements is selected, only collocation points in the *interior* of each element are employed. In this case, additional conditions enforcing the continuity of the first partial derivative of each variable distributed over this domain at all boundaries between elements are added to the AEA.

Some implementation issues regarding the approximation of partial derivatives and single and multiple integrals are considered below.

5.3.3 First-Order Partial Derivative Approximations

The discretisation of a first-order partial derivative at a given node is merely a process of replacing it with the approximation formulae described in sections 4.1.1 and 4.2.1, based on the specification of the solution method (equations 4.8a-4.8d and 4.6 for finite differences and equations 4.27a-4.27c and 4.25 for orthogonal collocation on finite elements).

As already pointed out in section 4.1.5, the direct application of finite difference formulae for partial derivatives at or near boundaries leads to fictitious points in the resulting expression. Two rules were suggested to avoid such points. For the purpose of our implementation, we employ the first rule, *i.e.* maintaining the same order of approximation

by changing the transition index q. The same rule is applicable to higher order partial derivatives.

5.3.4 Higher Order Partial Derivative Approximations

Higher order partial derivatives may be classified into two different types: those with respect to more than one different independent variable; and partial derivatives with respect to a single independent variable. Consider, for instance, the second-order partial derivatives of a variable distributed over two independent variables, z and r:

Type 1. $$\frac{\partial^2\phi(z,r)}{\partial z\partial r}$$

Type 2. $$\frac{\partial^2\phi(z,r)}{\partial z^2} \quad \text{or} \quad \frac{\partial^2\phi(z,r)}{\partial r^2}$$

Type 1 is a second-order derivative with respect to two independent variables, z and r. Consider its discretisation at the grid point (z_i, r_j) with a second-order centered finite difference method:

$$\begin{aligned}
\frac{\partial^2\phi(z_i,r_j)}{\partial z\partial r} &= \frac{\partial}{\partial r}\left[\frac{\partial\phi(z_i,r_j)}{\partial z}\right] \\
&\approx \frac{\partial}{\partial r}\left[\frac{\phi(z_{i+1},r_j)-\phi(z_{i-1},r_j)}{2\Delta z}\right] \\
&= \frac{\partial}{\partial r}\left[\frac{\phi(z_{i+1},r_j)}{2\Delta z}\right] - \frac{\partial}{\partial r}\left[\frac{\phi(z_{i-1},r_j)}{2\Delta z}\right] \\
&\approx \frac{\phi(z_{i+1},r_{j+1})-\phi(z_{i+1},r_{j-1})-\phi(z_{i-1},r_{j+1})+\phi(z_{i-1},r_{j-1})}{2\Delta z\cdot 2\Delta r}
\end{aligned} \tag{5.3}$$

where $\Delta z = z_{i+1} - z_i = z_i - z_{i-1}$ and $\Delta r = r_{j+1} - r_j = r_j - r_{j-1}$. Thus, the second-order derivative of *Type 1* can be converted into a first-order derivative of which the target expression recursively includes other first-order partial derivative.

Now, if we apply this recursive approach to the approximation of the partial derivative of *Type 2* at the point (z_i, r_j), we obtain:

$$\begin{aligned}
\frac{\partial^2\phi(z_i,r_j)}{\partial z^2} &= \frac{\partial}{\partial z}\left[\frac{\partial\phi(z_i,r_j)}{\partial z}\right] \\
&\approx \frac{\partial}{\partial z}\left[\frac{\phi(z_{i+1},r_j)-\phi(z_{i-1},r_j)}{2\Delta z}\right] \\
&= \frac{\partial}{\partial z}\left[\frac{\phi(z_{i+1},r_j)}{2\Delta z}\right] - \frac{\partial}{\partial z}\left[\frac{\phi(z_{i-1},r_j)}{2\Delta z}\right] \\
&\approx \frac{\phi(z_{i+2},r_j)-2\phi(z_i,r_j)+\phi(z_{i-2},r_j)}{(2\Delta z)^2}
\end{aligned} \tag{5.4}$$

On the other hand, from the direct application of equations 4.13 and 4.15a, we have:

$$\frac{\partial^2\phi(z_i,r_j)}{\partial z^2} \approx \frac{\phi(z_{i+1},r_j)-2\phi(z_i,r_j)+\phi(z_{i-1},r_j)}{\Delta z^2} \tag{5.5}$$

By comparing equation 5.5 to equation 5.4, we see that equation 5.5 is equivalent to equation 5.4 applied to a modified grid which has twice the number of points as the original and half the discretisation interval (*i.e.* $\Delta z' = \Delta z/2$, and $z'_{i+2} = z_{i+1}$, $z'_{i-2} = z_{i-1}$). Hence equation 5.5 is more accurate and should be used preferentially.

Higher than second-order partial derivatives may involve any combination of independent variables, *e.g.*

$$\frac{\partial^4\phi(z,r)}{\partial z^2\partial r^2}, \frac{\partial^4\phi(z,r)}{\partial z\partial r^3}, \frac{\partial^4\phi(z,r)}{\partial r^4}, \ldots$$

In such cases, the process manager decomposes the given partial derivative recursively until the expression which is to be differentiated is described in terms of either first-order partial derivatives, or *Type 1* or *Type 2* second-order derivatives. The discretisation then starts from this bottom level and continues being applied recursively upwards until all partial derivatives are approximated in terms of their corresponding formulae.

Overall, the combination of the methodologies for the first-order, and *Type 1* and *Type 2* second-order partial derivatives, together and the decomposition of higher order derivatives in principle provides a means of handling partial derivatives of any order.

5.3.5 Discretisation of Integrals

During instantiation, every integral which cannot be handled as described in section 5.3.1 is discretised according to the specification of the numerical method. Initially, integral of $\phi(z)$ in a given range is sub-divided into a number of piecewise integrals as follows:

$$\int_{z_l}^{z_u} \phi(z)dz = \sum_{q=0}^{p} \int_{z_{i+q}}^{z_{i+q+1}} \phi(z)dz \tag{5.6}$$

where i and p are determined by the numerical method and the range of the integral. Following this, the discretisation of integrals proceeds in two steps, namely, the decomposition of multiple integrals into a set of single integrals and the application of numerical methods for each single integral. So, integrals are categorised into single or multiple integrals, and the discretisation process is explained for each case.

5.3.5.1 Single Integration

Integrals may be of one of the following three types:

Type 1. The integrand is distributed over a *continuous domain* which is explicitly declared in the `DISTRIBUTION_DOMAIN` subsection in the language.

Type 2. The integrand is distributed over a continuous domain which is not declared in the `DISTRIBUTION_DOMAIN` subsection in the language.

Type 3. The integrand is distributed over a *discrete domain* which is implicitly declared in the `VARIABLE` subsection in the language.

The three different types of single integral are illustrated in Figure 5.6. The discretisation process is executed differently for each type.

For *Type 1* integrals, the discretisation involves the replacement of the integral with the formula consistent to the discretisation method specified for the corresponding distribution domain (see section 4.1.3 and 4.2.4).

For *Type 2* integrals, since no information regarding an appropriate numerical method is specified, the discretisation is performed using 5th order Gaussian quadrature.

Finally, when a function ϕ is distributed over a discrete domain $[i_0, i_m]$, the integration of *Type 3* can be expressed by:

$$\int_{i_0}^{i_m} \phi(i)di = \sum_{k=i_0}^{i_m} \phi(k) \tag{5.7}$$

Thus, the integration of *Type 3* is interpreted as a summation of the variables over the discrete points in the domain. In the example of Figure 5.6, the variables `I3` and `S3` have the same value.

5.3.5.2 Multiple Integration

Multiple integrals can be regarded as an extension of single integrals where the integrand includes other integrals in a recursive manner. Therefore, a recursive approach, similar to that used for the higher order partial derivatives, can be adopted for this case.

Multiple integrals are decomposed into a hierarchy of single integrals, each of which can also include other integrals. The discretisation then commences from the bottom level integral (which contains no other integrals) using the strategies discussed in section 5.3.5.1.

The next level integral is integrated with the resulting algebraic expressions from the first integral. This process will end when the outermost expression does not include any further integrals.

5.4 Concluding Remarks

We have discussed some aspects of the implementation of a modelling package which supports the direct modelling and simulation of distributed parameter systems. Since

```
MODEL IntegralExamples

      . . . . . .

DISTRIBUTION_DOMAIN
   Axial          AS    (0 : Zlength)
   Radial         AS    (0 : Rlength)

VARIABLE
   T              AS    DISTRIBUTION(Axial,Radial)  OF Temperature
   Fraction       AS    ARRAY        (Nocomp)       OF Mole_Fraction
   I1, I2, I3     AS                                  Integration
   S3             AS                                  Summation

SET
   Axial          :=  [ CFDM,  2,  20 ]   ;
   Radial         :=  [ OCFEM, 3,   4 ]   ;

EQUATION
   # Integral of Type 1
   I1  =  INTEGRAL ( z := 0 : Zlength ; T(z,Rlength) ) ;

   # Integral of Type 2
   I2  =  INTEGRAL ( z := 0 : Pi/2 ; SIN(z) )          ;

   # Integral of Type 3
   I3  =  INTEGRAL ( i := 1 : Nocomp ; Fraction(i) )   ;

      # Summation of Type 3
      S3  =  SIGMA ( Fraction(1:Nocomp) )              ;
END
```

Figure 5.6: Example of the three types of single integrals

Numerical Methods	Order(s)	Partial Derivatives	Integrals
• centered finite differences	2, 4, 6	√	√
• backward finite differences	1, 2	√	√
• forward finite differences	1, 2	√	√
• upwind-biased finite difference	2	√	†
• orthogonal collocation on finite elements	2, 3, 4	√	√
• Gaussian quadrature	5		√

† integrals evaluated using backward finite difference approximation of same order.

Table 5.1: Numerical methods for approximation of IPDAEs in *gPROMS*

the goal of the project is to construct a high-level modelling environment to deal with a variety of distributed parameter systems, a number of numerical methods are adopted for the discretisation of IPDAEs. The IPDAEs, described in terms of the modelling language introduced in chapter 3, are analysed during the translation and execution phases, and discretised based on the numerical methods discussed in chapter 4 during the execution phase. At present, the distributed system modelling capabilities in *gPROMS* are limited to rectangular geometries but can be applied, in principle, to an arbitrary number of dimensions.

The different types and orders of numerical methods currently implemented in the package are listed in Table 5.1. The implementation of further finite difference and/or orthogonal collocation on finite element methods is straightforward.

Chapter 6

The Management of Simulation Results in *gPROMS*

The incorporation of distributed system modelling capabilities in *gPROMS* often leads to dynamic simulations producing extremely large amounts of result data. Collecting, saving and displaying such results present some interesting problems, not just due to their sheer volume but also because of their complex structure. One has to deal, for instance, with model hierarchies of arbitrary depth (see section 3.1.2), with each model containing variables distributed over an arbitrary number of discrete and/or continuous domains.

The standard way of collecting results of dynamic simulations in *gPROMS* and other simulation packages involves recording the values of all variables at a regular frequency. The latter is expressed in terms of a constant monitoring time interval specified by the user.[1] Additionally, *gPROMS* records the values of the variables at the initial point of the simulation, and just before and just after every discontinuity encountered during the simulation.

One possible way of reducing the amount of data produced during a simulation is to be more selective as to which variables are recorded and/or the times at which such recording takes place. This is discussed briefly in section 6.1. Section 6.2 then proceeds to consider aspects associated with the display of results, paying particular attention to the issue of recording and displaying distributed variables.

6.1 Selective Recording of Results

6.1.1 Specifying Variables to Be Monitored

In many cases, the analysis of the result of a simulation in order to understand the behaviour of a physical system involves only a small number of variables. For instance, the fixed-bed catalytic reactor system, which will be demonstrated in section 7.2, includes many variables. However, data for the reactor temperature and concentration are often sufficient to characterise the process.

---

[1] It should be emphasised that there is no relation between this interval and the time integration step used internally by the DAE solver. The latter generally varies over the simulation, being adjusted automatically in order to control the time discretisation error.

```
PROCESS TestSimulation
  UNIT
    Reactor   AS TubularReactor
        ....

  MONITOR
     String1; String2; ... ; StringN;
        ....

  SCHEDULE
    SEQUENCE
      CONTINUE FOR 100
         ....
    END
END
```

Figure 6.1: Specification of variables to be monitored in *PROCESS* entities

Taking these considerations into account, a user can specify *a priori* which variables in a *gPROMS* simulation should be stored for later analysis. As has already been mentioned, in *gPROMS* each simulation experiment corresponds to a *PROCESS* entity. The variables to be monitored are declared in the `MONITOR` subsection of *PROCESS* entity as shown in Figure 6.1. In a sense, this is analogous to deciding which variables are to be measured in the corresponding *real* experiment.

In Figure 6.1, `String1` is either a full variable name or a partial variable name involving "wildcard" symbols (*). The latter allows variables whose name matches a given pattern to be included in the monitored set without being listed explicitly. For instance, in the tubular reactor model shown in Figure 3.4, if the monitored variables are specified as follows:

```
MONITOR
  Reactor.C* ;  Reactor.TBath ;
```

then `Conc`, `CFeed` and `TBath` will be monitored during the simulation.

If the string patterns specified in `MONITOR` subsection do not match any of the variables declared in `MODEL`, *gPROMS* issues a warning message and continues with the simulation without storing any information. On the other hand, if the `MONITOR` subsection is completely omitted from the *PROCESS* entity, all variables are assumed to be monitored during the whole period of simulation.

```
1    TASK  DoCycleReactorSimulation
2
3      VARIABLE
4         Cycle    AS    INTEGER
5
6      SCHEDULE
7         SEQUENCE
8         MONITOR OFF
9
10        Cycle  :=  0 ;
11        WHILE  Cycle < 1000  DO
12           SEQUENCE
13              Cycle  :=  Cycle + 1 ;
14              DoOneCycle
15           END
16        END
17
18        MONITOR ON
19
20        DoOneCycle
21
22     END
23   END
```

Figure 6.2: Example of selective monitoring of solution trajectory

6.1.2 Selective Monitoring of Solution Trajectory

In some simulations, some parts of the trajectory are not of primary interest and are transversed for the sole purpose of getting to more interesting regions. For instance, consider the task of studying the periodic "steady-state" of a chemical reactor subjected to cyclic feed variation. This can be achieved by carrying out a large number (often several hundreds) of cycles starting from a given initial condition; however it is only the variation of reactor conditions during the *last* cycle that is often of major interest to the modeller.

In such situations, it would be desirable to be able to switch off results monitoring during parts of the simulation. To allow this, we have introduced two elementary tasks, `MONITOR ON` and `MONITOR OFF`, that can be used within the simulation schedule. An example of their use is shown in Figure 6.2 which shows a *gPROMS TASK* implementing the periodic reactor simulation discussed above. We note that results monitoring is initially switched off (line 8) before the reactor is taken through 1000 cycles of its operation (lines 10-16).[2]

[2]Here `DoOneCycle` is assumed to be a lower-level *TASK* entity that drives the reactor through a single cycle.

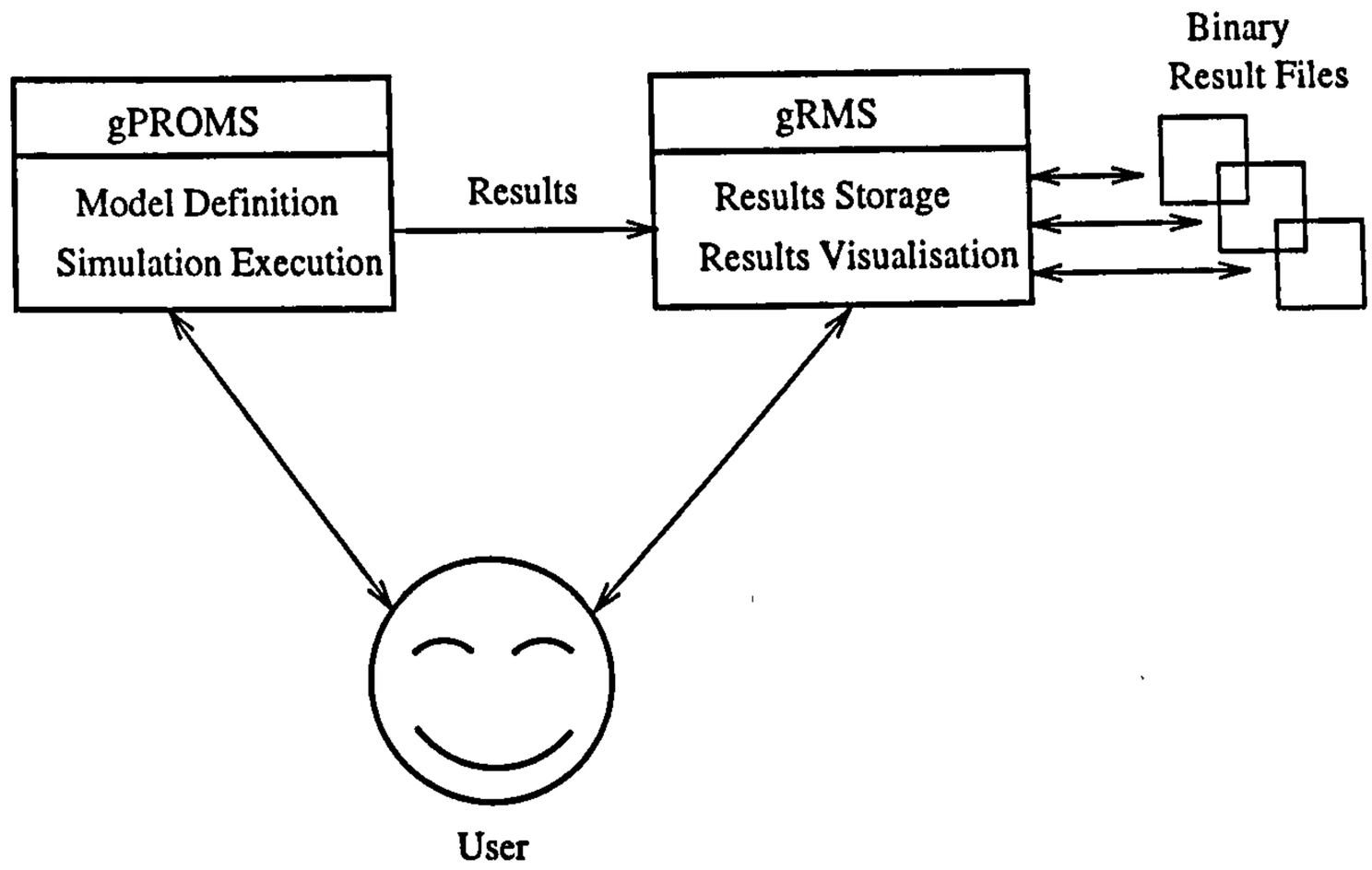


Figure 6.3: *gPROMS* and *gRMS* communication

We then switch result monitoring on (line 18) before taking one last cycle (line 20) for the purpose of collecting results on the periodic steady-state of the reactor.

6.2 The *gPROMS* Results Management System (*gRMS*)

6.2.1 Overview

The task of actually collecting and physically archiving results produced by a complex simulation, as well as handling their display to the user both during the simulation and after it, can itself be quite complex and computationally demanding. It was therefore decided to design and implement a separate software package, the *gPROMS* Results Management System (*gRMS*), to which to delegate this task.[3]

An overview of the interaction between *gPROMS*, *gRMS* and the user is shown in Figure 6.3. The user interacts with *gPROMS* to define the process model and to execute the simulation. Results are transmitted from *gPROMS* to *gRMS* which has sole responsibility for recording them physically in persistent computer storage. Furthermore, the user can interact directly with *gRMS* to request the display of the results of a currently executing simulation, or of one which has already been completed.

In the rest of this section, we first consider the communication between *gPROMS*

[3]The detailed software design and implementation of *gRMS* was carried out by Mr T.O.Björkholm (1994) in the Centre for Process Systems Engineering, Imperial College.

and *gRMS*, paying particular attention to the information that needs to be communicated in the context of distributed system modelling. We then describe briefly the physical implementation of the results files, before considering some aspects of the *gRMS* user interface; again particular emphasis is given to the issue of displaying results arising from the simulation of distributed systems.

6.2.2 Communication between *gPROMS* and *gRMS*

gPROMS and *gRMS* are implemented as separate programs communicating through appropriate mechanisms.[4] It is therefore possible to execute them on different computers, thereby dividing the computational load and potentially taking advantage of different machine characteristics (*e.g.* using a machine with a powerful floating-point processor to run the *gPROMS* simulation and one with a graphics accelerator for *gRMS*).

It is also worth mentioning that *gRMS* can communicate concurrently with more than one executing *gPROMS* simulation, recording their results in separate physical files while allowing, for instance, their comparative display.

At the start of a simulation, *gPROMS* registers with *gRMS* all the variables that are going to be monitored during the simulation. This could include all the variables in the model, or, if a `MONITOR` subsection has been included in the `PROCESS` being executed (see section 6.1.1), just the variables that match one or more of the specified patterns.

For scalar variables, this registration process is straightforward, involving little more than their name. However, it is rather more complicated for variables that are distributed over one or more discrete or continuous distribution domains. Firstly, *gPROMS* registers with *gRMS* all information concerned with continuous and discrete domains in each model in the entire model hierarchy for the present problem. This includes the names and ranges of all discrete and continuous domains as well as the numerical solution information associated with the latter. In the next step, the names of variables and the domains over which they are distributed are also registered.

Once the initial variable registration is completed, the simulation is allowed to proceed. Assuming that variable monitoring is switched on (see section 6.1.2), then variable values are transmitted from *gPROMS* to *gRMS* at the specified monitoring intervals, or whenever a discontinuity is detected. A separate communication call is initiated for each monitored variable, transmitting all entries in the AVA (see section 5.2) pertaining to this

[4] The precise form of this interaction depends on the type of computer and operating system used; for instance, socket communication is used on UNIX-based systems.

variable.

6.2.3 Archiving of Simulation Results

One of the main functions of *gRMS* is to maintain a permanent record of any information transmitted to it by *gPROMS*. This is achieved by recording the results of an executing *gPROMS PROCESS* in a file as they are made available to it by *gPROMS*.

For reasons of both computational and storage utilisation efficiency, such files are binary in nature, thus avoiding the overhead involved in converting real numbers to their *ASCII* representation.

It is worth noting that, in addition to recording any results being received in a file, *gRMS* also holds a copy of them in memory. This allows rapid display of the results of any currently executing simulation. If a number of *gPROMS* simulations are executed, *gRMS* will normally hold the results of *all* of them in memory until either the *gRMS* run is terminated, or the user explicitly requests that some of these in-memory result sets be closed and the memory be recovered. In either case, the permanent file archive remains intact and can be recovered at a later stage.

6.2.4 The *gRMS* User Interface

gRMS is implemented as a user-friendly, window-based system.[5] The user typically selects and displays the required results by navigating through a hierarchy of menus.

If *gRMS* is not already running on a specified computer, a *gRMS* run is initiated automatically on starting the *gPROMS* package. This results in the appearance of a separate *gRMS* window as shown in the top left corner of Figure 6.4.

Initiating the execution of a *PROCESS* entity within *gPROMS* automatically creates a new result dataset in *gRMS* and this is listed in the *gRMS* window (see Figure 6.5). It will be seen that the dataset adopts the name of the process, from the execution of which it resulted. Thus, the command "`execute CT`" in the *gPROMS* window has created the dataset `CT.gRMS` in the *gRMS* window.

From this point onwards, the user can request the creation of 2- and 3-dimensional plots of any data in the currently open dataset(s) through the Graph menu.[6] This results in the creation of an initially blank list of variables to be displayed. Variables can be added to

[5] The current implementation is based on the OSF/MOTIF set of X-Windows widgets. Two- and three-dimensional graphical displays are created using the commercially available XRT/graph and XRT/3d widgets (KL Group Inc., 1993a,b).

[6] Other data sets created and archived during earlier simulations can also be opened using the File menu.

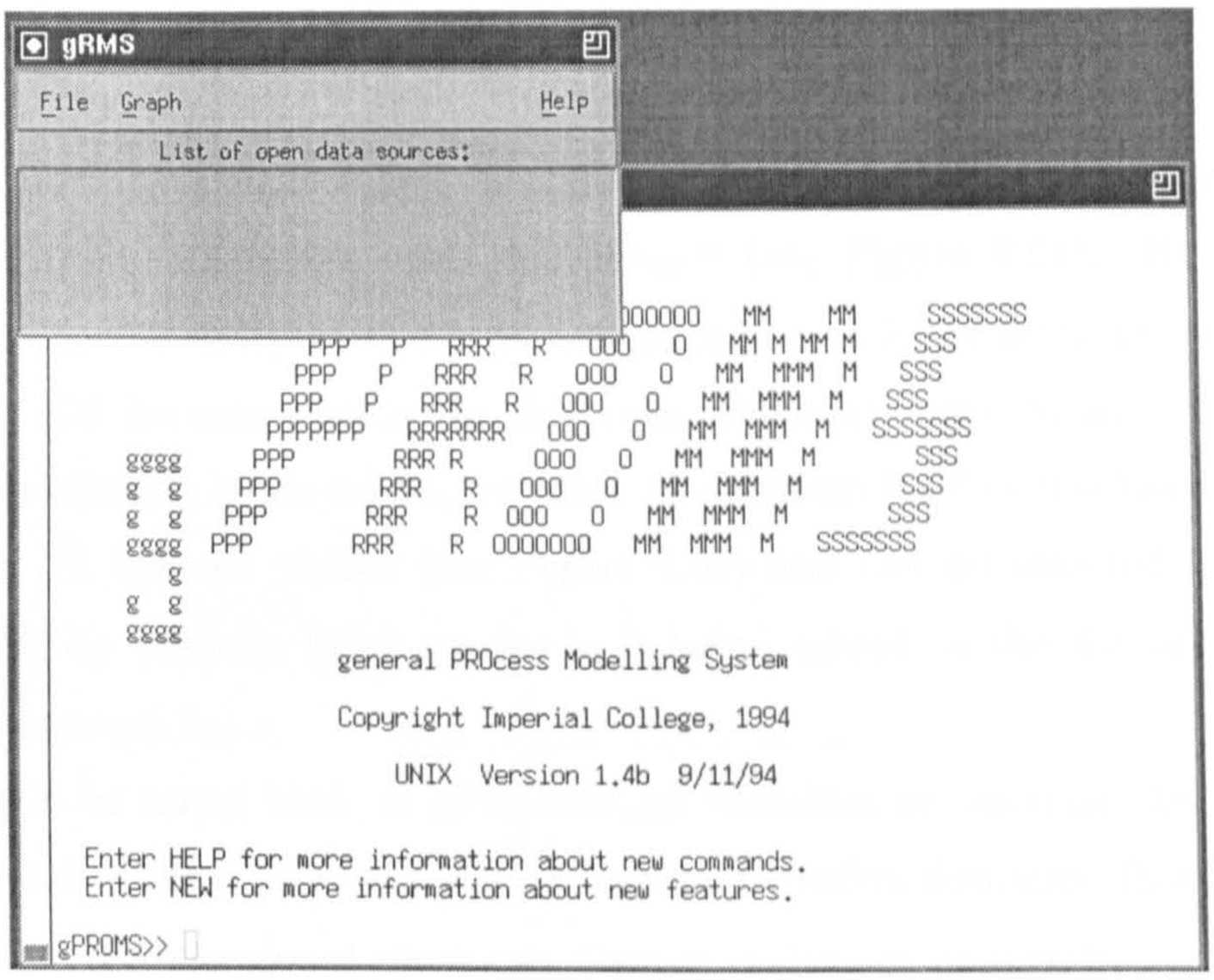


Figure 6.4: Starting *gPROMS* and *gRMS*

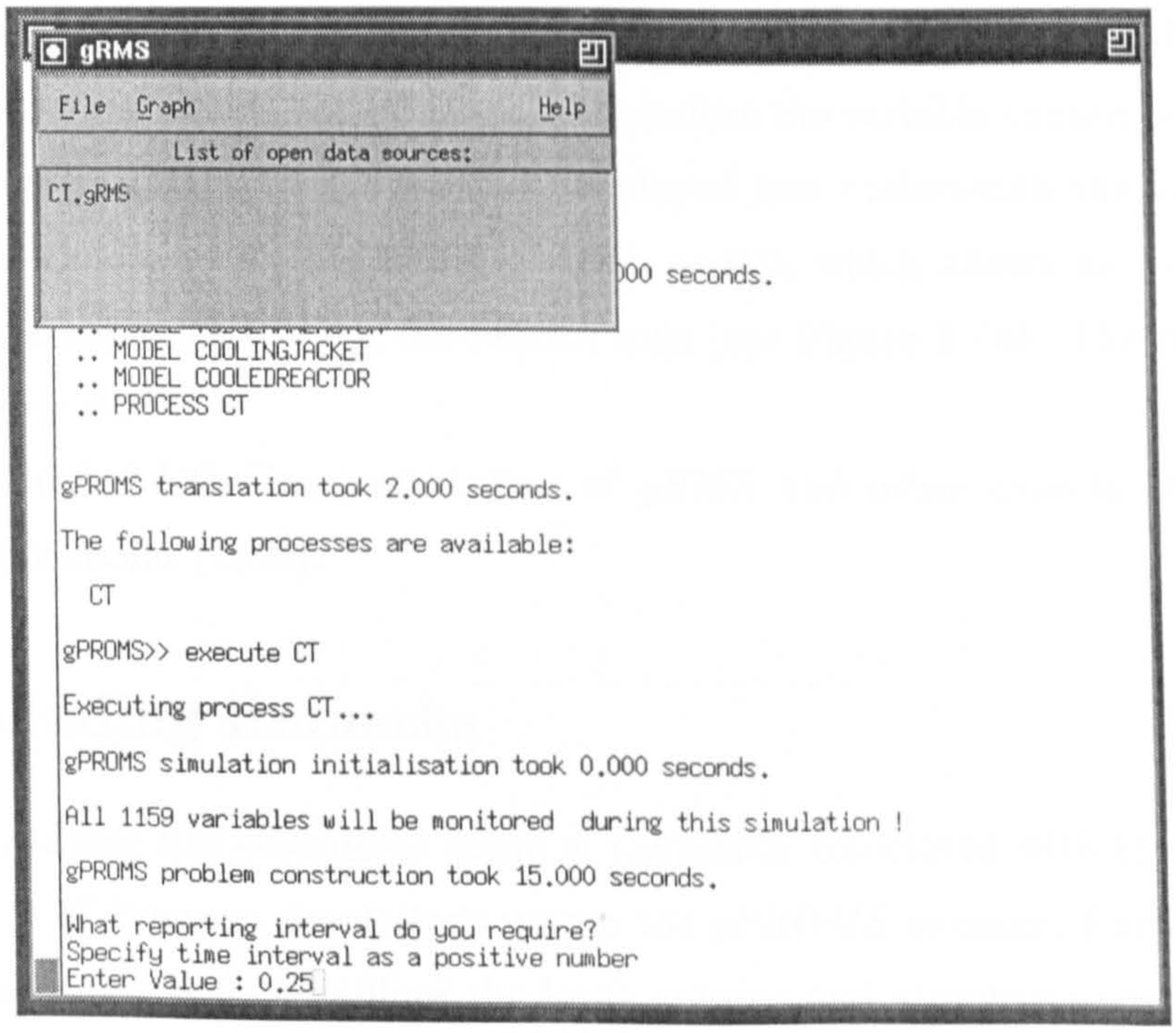


Figure 6.5: The creation of a dataset by the execution of *PROCESS* entity

this list through another window (see Figure 6.6) that allows navigation through a hierarchy of models of arbitrary depth.

The particular example shown in Figure 6.6 is discussed in detail in section 7.2. The top level model instance is called `CS` and this is the only entity visible at the top-most level of the hierarchy associated with this dataset (see Figure 6.6a). By "clicking" the mouse button on the `CS` entry at this level, we descend to a lower level where the submodel instances within `CS` become visible; in this case, there are two, namely `CJ` and `TR` (see Figure 6.6b). By clicking again on `TR`, we reach the bottom level of the hierarchy where the variables within `TR` become visible (see Figure 6.6c) and can be selected for display. For instance, clicking on variable `TEMP` results in it being added to the list of variables to be displayed (see Figure 6.7a).

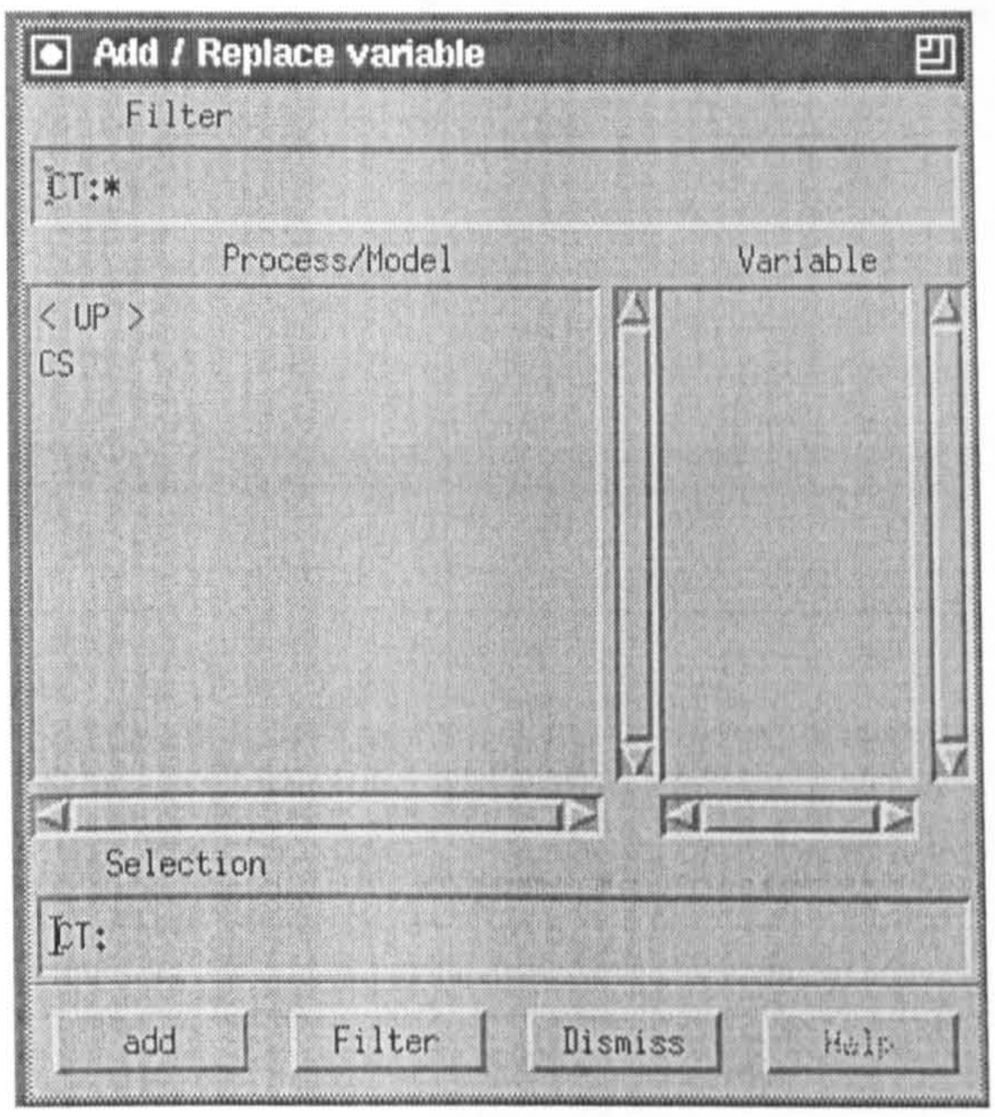
It should be noted that, in *gPROMS*, all variables are assumed to vary over time. Furthermore, some of them vary over one or more distribution domains. However, a problem arises since 2- and 3-dimensional plots can display variations over only 1 or 2 independent variables respectively. In many cases, we are therefore forced to restrict the variation of a certain variable so that it can be displayed in a given type of plot. This is done by fixing the position in time and/or one or more of the distribution domains. For instance, the variable selected in Figure 6.7 is a temperature that is distributed over an axial and a radial domain as well as being a function of time. This variation over 3 independent variables cannot be captured in a standard 3-dimensional plot and therefore the variable cannot be displayed (as already indicated by *gRMS* by the message displayed just underneath the variable name). In this case, we choose to fix the radial position at 0.0, which allows us to plot the time variation of the temperature along the reactor axis (see Figure 6.7b). The resulting graph is shown in Figure 6.8.

The details of the implementation of *gRMS* and other aspects of its usage are described by Björkholm (1994).

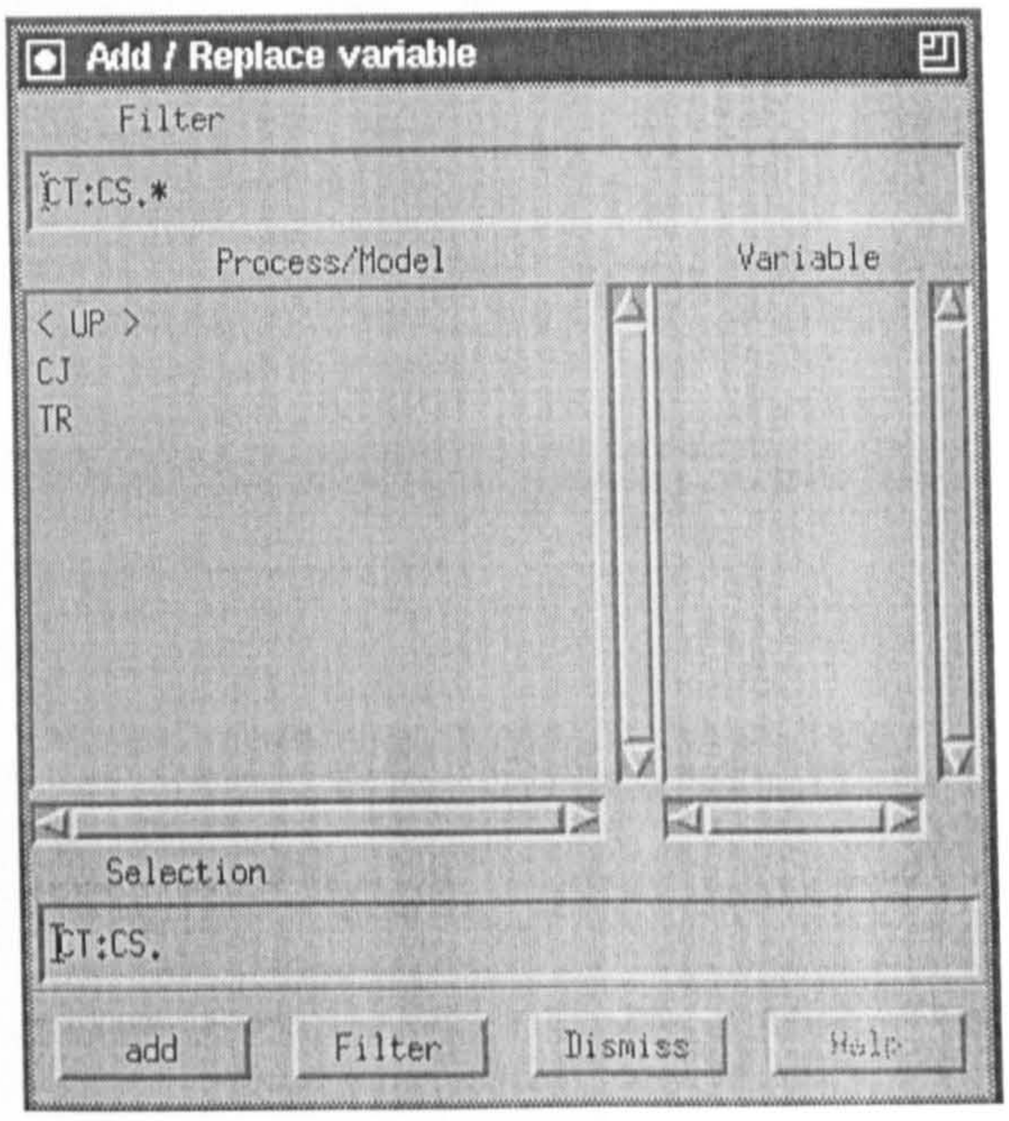
6.3 Concluding Remarks

This chapter has considered some of the issues associated with the collection and display of results of dynamic simulations within the *gPROMS* package. Particular emphasis was paid to mechanisms for handling the large volume and structural complexity of such results.

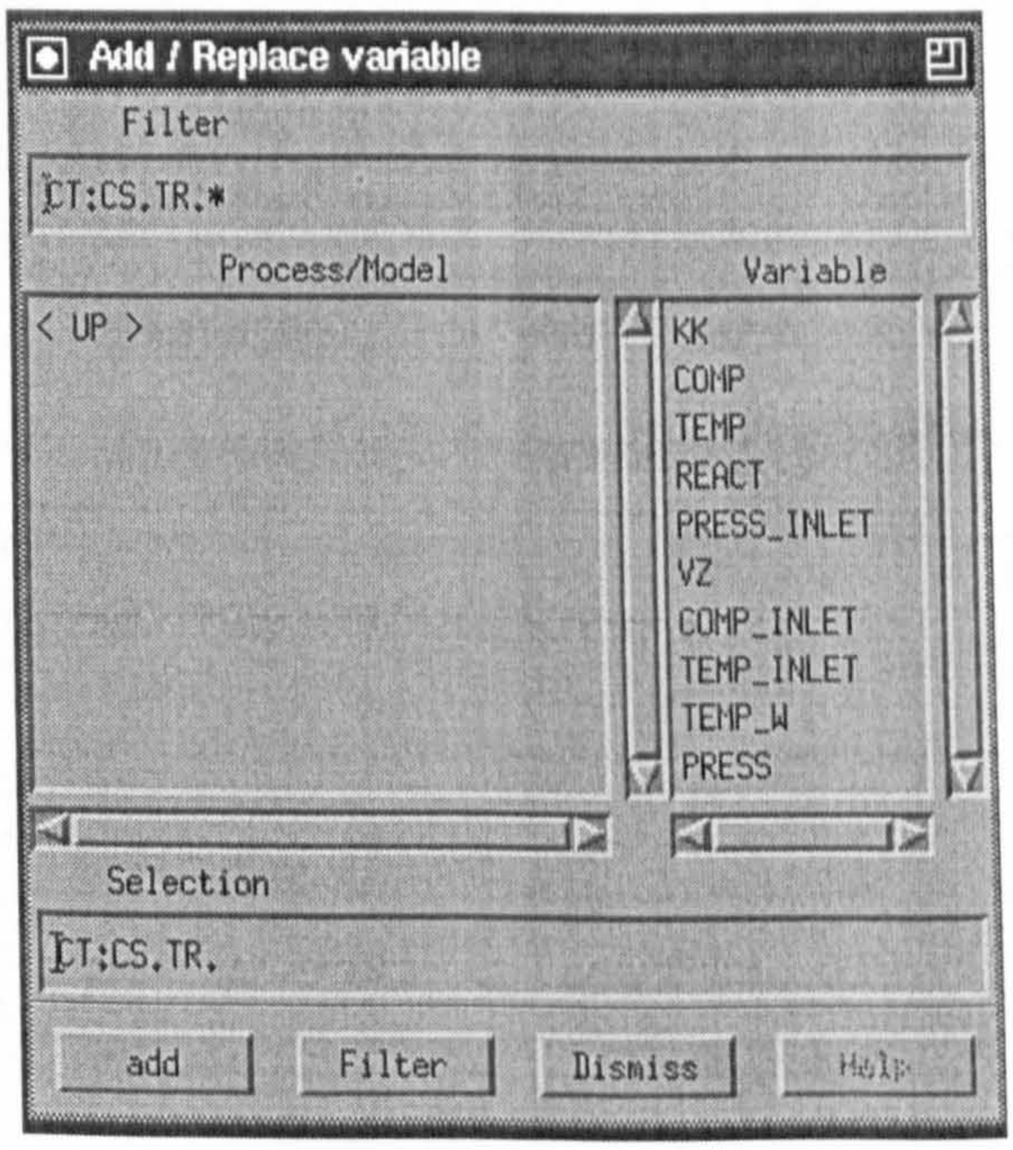
As described in the first part of this chapter, selective monitoring of results, restricted to a subset of the problem variables and/or part of the time horizon, is a convenient



(a) Top level : dataset CT

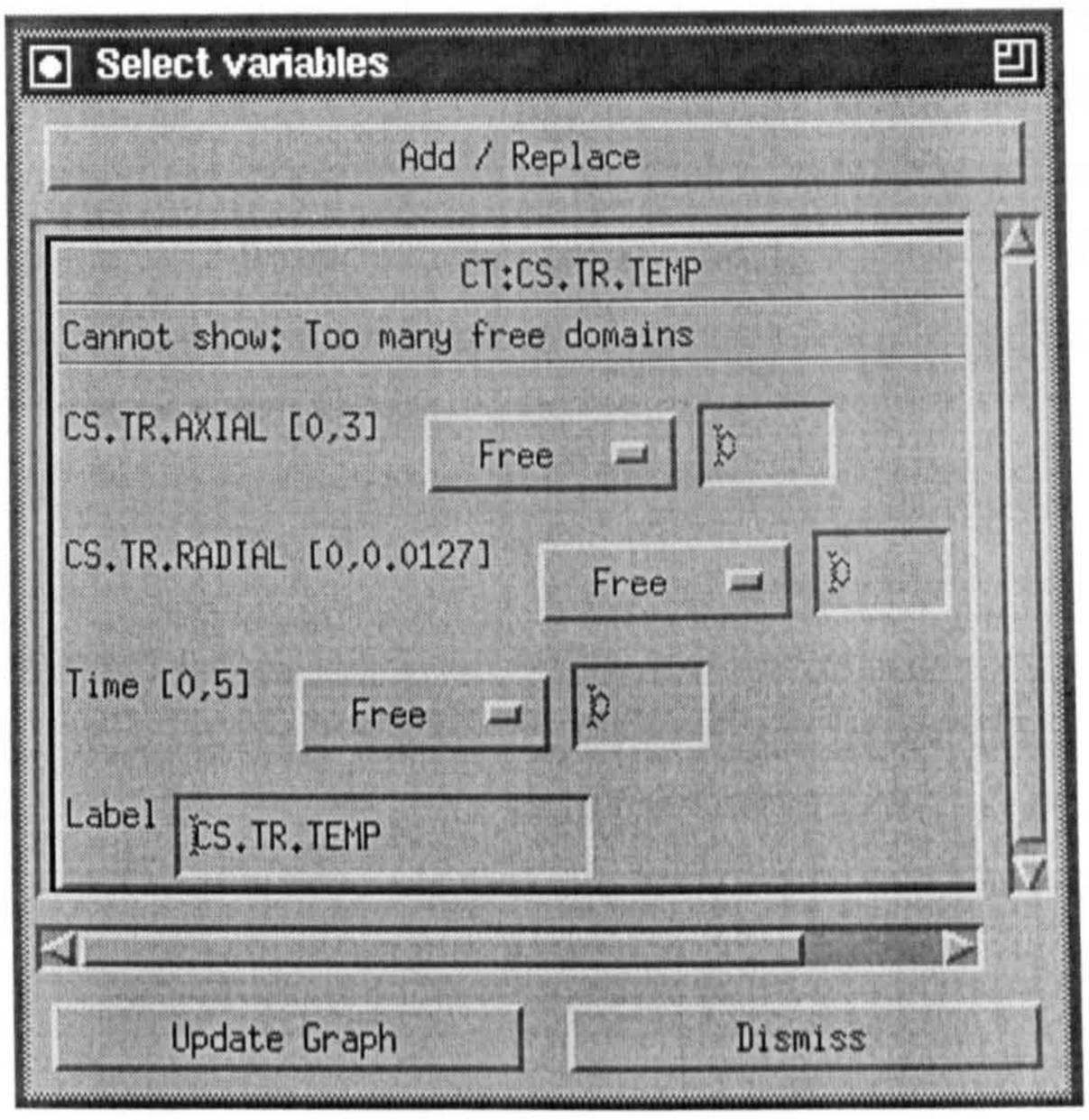


(b) Intermediate level : model CS within CT

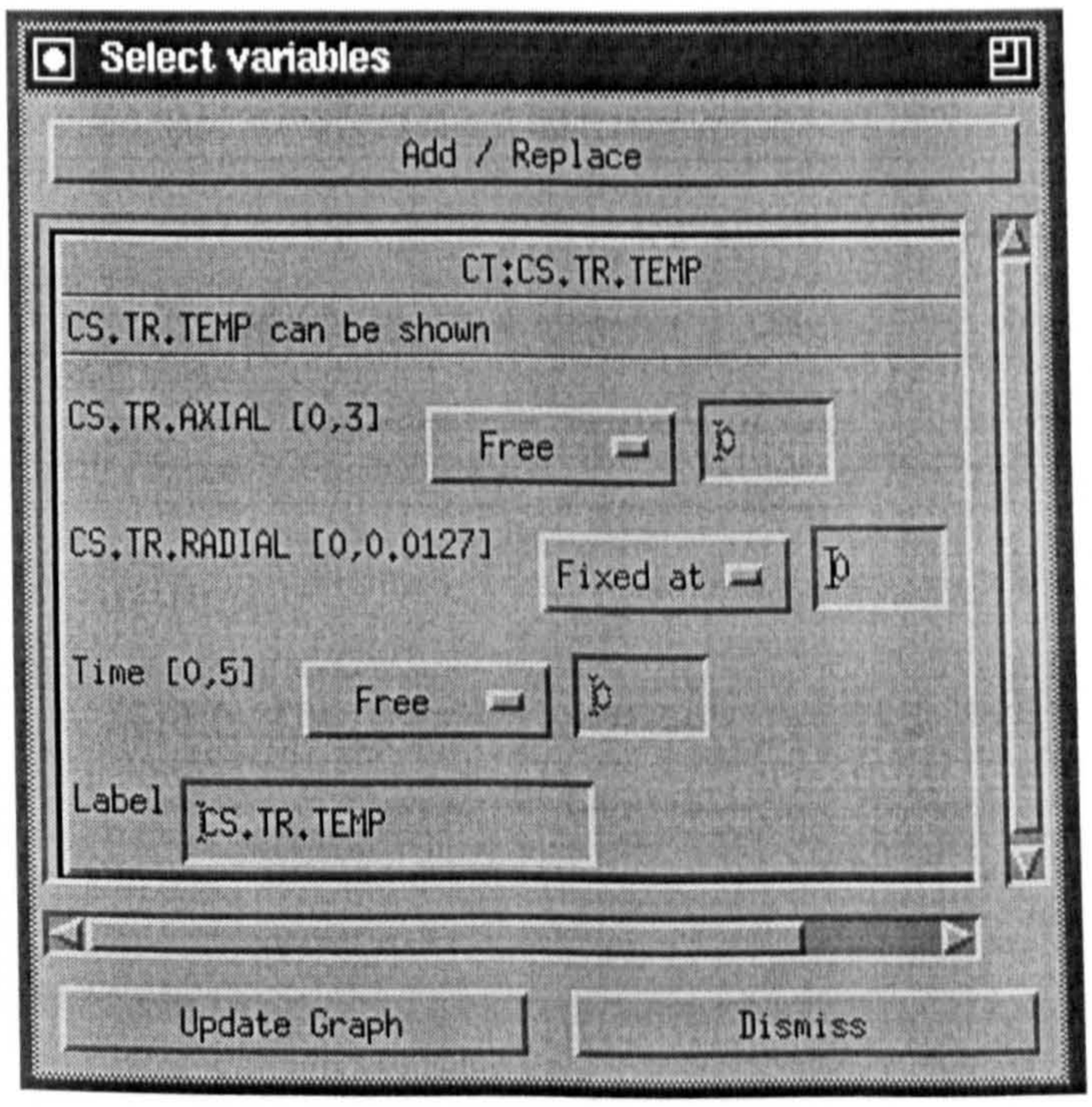


(c) Bottom level : model TR within CS

Figure 6.6: Display of results in model hierarchy



(a) Before domain restriction



(b) After domain restriction

Figure 6.7: List of variables to be plotted

way of reducing the volume of the results collected. Needless to say, this relies entirely on the user being able to decide *a priori* on the important variables and/or parts of the solution trajectory, and may therefore have the undesirable effect of significant results being lost.

The *gRMS* system described in the second part of this chapter provides an effective and user-friendly means of archiving, retrieving and displaying results of simulations comprising arbitrary hierarchies of models, each involving variables distributed over arbitrary numbers of domains.

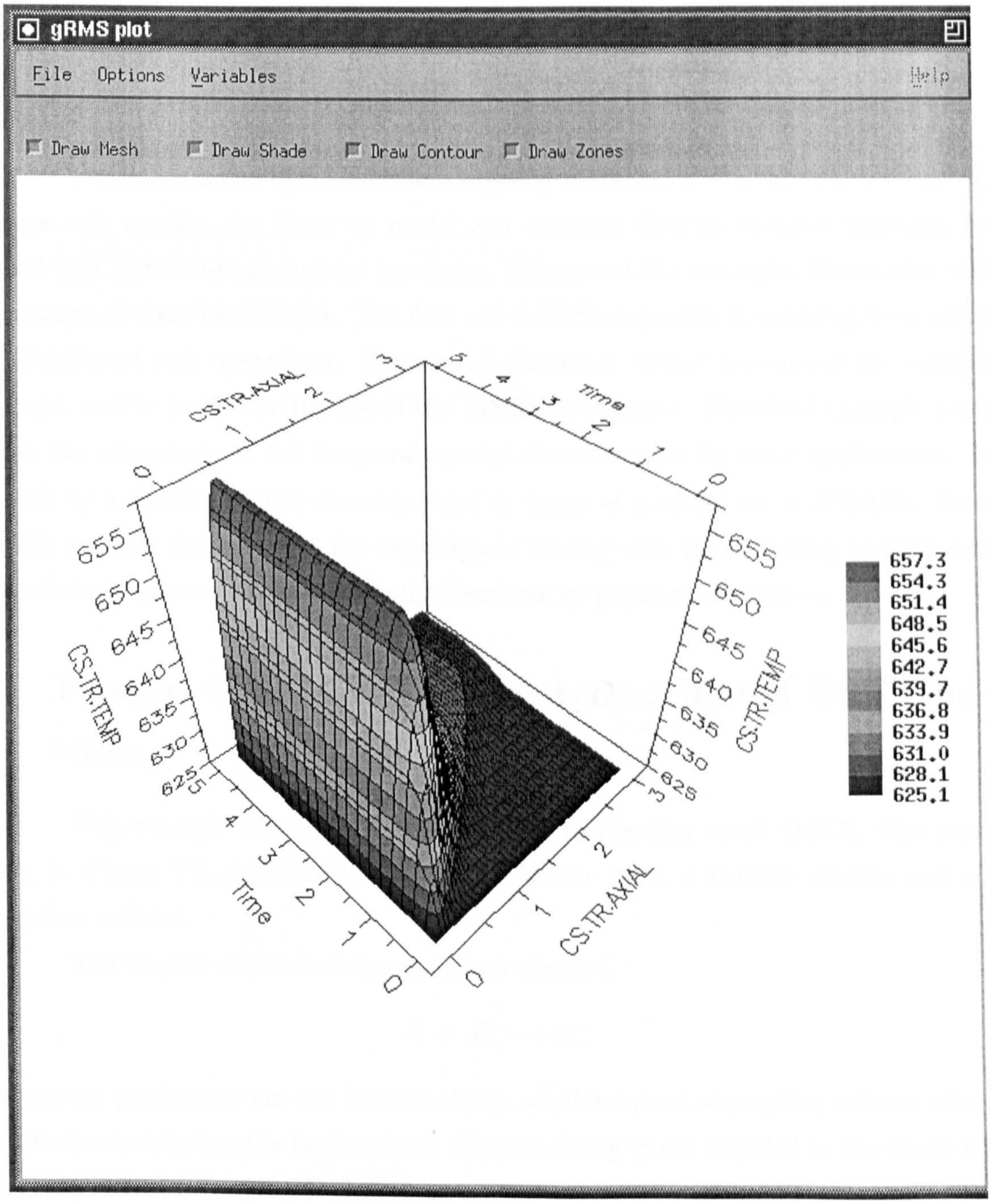


Figure 6.8: Visualisation of results in 3D graphics

Chapter 7

Examples of Combined Lumped/Distributed Parameter System Modelling and Simulation

The introduction of the modelling language described in this thesis into the *gPROMS* package now enables the latter to model and simulate directly systems involving both lumped and distributed parameter processes. We present five examples illustrating different aspects of these capabilities. The first one involves a process comprising both lumped and distributed unit operations. The second illustrates further features of the modelling language, and in particular the use of the `INTEGRAL` operator. The third example demonstrates the advantages of full temporal/spatial discretisations for some applications. It is followed by a fourth example also expressed in terms of a mixed set of IPDAEs. Finally, the fifth example demonstrates the capability of dealing with the modelling and simulation of particulate systems which are usually described by population balances.

7.1 Process with Mixed Lumped/Distributed Unit Operations

This example was originally proposed by Heydweiller *et al.* (1977). The process, shown in Figure 7.1, comprises a well-stirred mixer tank, a tubular reactor and a gas absorption column.

The reactor carries out the gas-phase reaction

$$A + B \longrightarrow 2C \tag{7.1}$$

The reactor product enters the bottom of the countercurrent absorption column where C is partially absorbed in the liquid phase. The remaining gas is recycled to the mixer where it is combined with fresh feed.

The dynamic response of this system is determined by simultaneously solving the ordinary differential equations describing the mixer, the parabolic differential equations for the tubular reactor and the hyperbolic differential equations for the gas absorption column.

These equations are shown below in dimensionless form. Because an isothermal process is assumed, only mass balance equations are considered here.

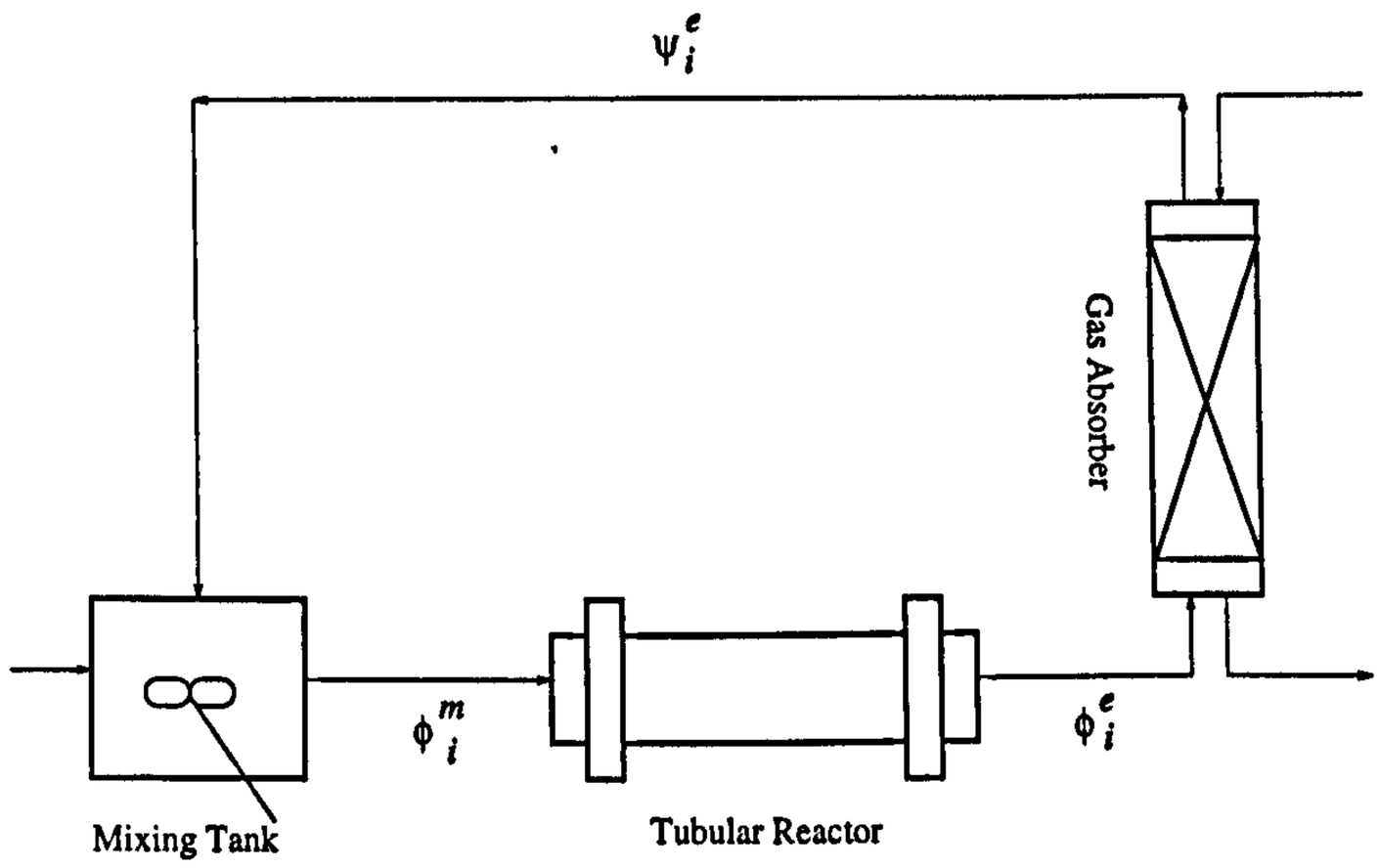


Figure 7.1: Flowsheet of a mixer, a tubular reactor and a gas absorber

Tubular reactor :

Mass balance:

$$\frac{\partial \phi_i}{\partial \tau} = K_1 \frac{\partial^2 \phi_i}{\partial \lambda^2} - \frac{\partial \phi_i}{\partial \lambda} + \nu_i K_2 \phi_A \phi_B \quad \forall \lambda \in (0,1),\ i = A, B, C \tag{7.2}$$

Boundary conditions:

$$\frac{1}{Pe}\frac{\partial \phi_i}{\partial \lambda} = \phi_i - \phi_i^m \qquad @\ \lambda = 0,\ i = A, B, C \tag{7.3}$$

$$\frac{\partial \phi_i}{\partial \lambda} = 0 \qquad @\ \lambda = 1,\ i = A, B, C \tag{7.4}$$

where $\lambda \in [0,1]$ is the dimensionless axial position and τ the dimensionless time, while $\phi_i(\lambda, \tau)$ represents the dimensionless concentration of component i in the reactor and ϕ_i^m the corresponding quantity in the reactor feed. The stoichiometric coefficients used in the mass balance are $\nu_A = -1$; $\nu_B = -1$; $\nu_C = +2$.

Gas absorber :

Mass balance:

$$\frac{\partial \psi_A}{\partial \tau} = -K_3 \frac{\partial \psi_A}{\partial \zeta} \qquad \forall \zeta \in (0,1] \tag{7.5}$$

$$\frac{\partial \psi_B}{\partial \tau} = -K_3 \frac{\partial \psi_B}{\partial \zeta} \qquad \forall \zeta \in (0,1] \tag{7.6}$$

$$\frac{\partial \psi_C}{\partial \tau} = -K_3 \frac{\partial \psi_C}{\partial \zeta} - K_4(\psi_C - K_5 \theta_C) \qquad \forall \zeta \in (0,1] \tag{7.7}$$

$$\frac{\partial \theta_C}{\partial \tau} = K_6 \frac{\partial \theta_C}{\partial \zeta} + \frac{K_4}{K_7}(\psi_C - K_5 \theta_C) \qquad \forall \zeta \in [0,1) \tag{7.8}$$

Constant	Value	Constant	Value
K_1	0.1	K_6	0.003
K_2	3.0	K_7	75.0
K_3	0.1	K_8	30.7
K_4	20.0	K_9	0.5
K_5	10.0		

Table 7.1: Constants for models of section 7.1

Boundary conditions:

$$\psi_i = \phi_i^e \qquad @\,\zeta = 0,\ i = A, B, C \tag{7.9}$$

$$\theta_C = 0 \qquad @\,\zeta = 1,\ i = A, B, C \tag{7.10}$$

where $\zeta \in [0, 1]$ is the dimensionless axial position of gas absorber, $\psi_i(\zeta, \tau)$ and $\theta_i(\zeta, \tau)$ the dimensionless concentrations in the gas and liquid phases respectively, and ϕ_i^e the dimensionless concentrations at the reactor exit.

Mixing tank :

$$\frac{d\phi_i^m}{d\tau} = K_8[\phi_i^f + K_9\psi_i^e - (1 + K_9)\phi_i^m] \qquad i = \mathrm{A, B, C} \tag{7.11}$$

where $\phi_i^m(\tau)$ is the dimensionless concentration of component i in the mixing tank, ϕ_i^f the dimensionless concentration of component i in the feed stream and ψ_i^e the dimensionless concentration of component i in the recycle.

The values of the various constants in the above models are shown in Table 7.1.

The mathematical models of the three units as described in *gPROMS* are shown in Appendix B.1. It should be mentioned that the omission of a distribution index in a distributed variable occurring in an equation signifies that the index is taken over the entire domain. Thus `C( ,z)` (used on several occasions in Appendix B.1) is an abbreviated form of `C(1:NoC,z)`. It should also be noted that two different axial domains, called `Forward` and `Reverse`, are declared in the `Absorber` model. This allows different discretisation methods to be applied to each one of them (see below).

Figure 7.2 illustrates a higher level *MODEL* describing the entire process by creating (lines 3-6) and linking together (lines 31-33) instances of the lower level *MODEL*s. The higher level *MODEL* also specifies the numerical methods to be used for the discretisation of the space dimensions. In particular, orthogonal collocation on finite elements is used for the discretisation of the `Axial` domain in the tubular reactor (line 13), while backward and

```
1   MODEL Flowsheet
2
3     UNIT
4       M101                        AS   Mixer
5       R101                        AS   Reactor
6       A101                        AS   Absorber
7
8     SET
9       WITHIN R101 DO
10        K1          :=   0.1                  ;
11        K2          :=   3.0                  ;
12        Length      :=   1.0                  ;
13        Axial       :=   [ OCFEM, 3, 15 ]     ;
14      END
15      WITHIN A101 DO
16        K3          :=   0.1                  ;
17        K4          :=   20.0                 ;
18        K5          :=   10.0                 ;
19        K6          :=   0.003                ;
20        K7          :=   75.0                 ;
21        Length      :=   1.0                  ;
22        Forward     :=   [ BFDM, 1, 40 ]      ;
23        Reverse     :=   [ CFDM, 2, 40 ]      ;
24      END
25      WITHIN M101 DO
26        K8          :=   30.7                 ;
27        K9          :=   0.5                  ;
28      END
29
30    EQUATION
31      M101.Outlet   IS   R101.Inlet           ;
32      R101.Outlet   IS   A101.Inlet           ;
33      A101.Outlet   IS   M101.Inlet           ;
34
35  END  # model flowsheet
```

Figure 7.2: Higher level *MODEL* for example of section 7.1

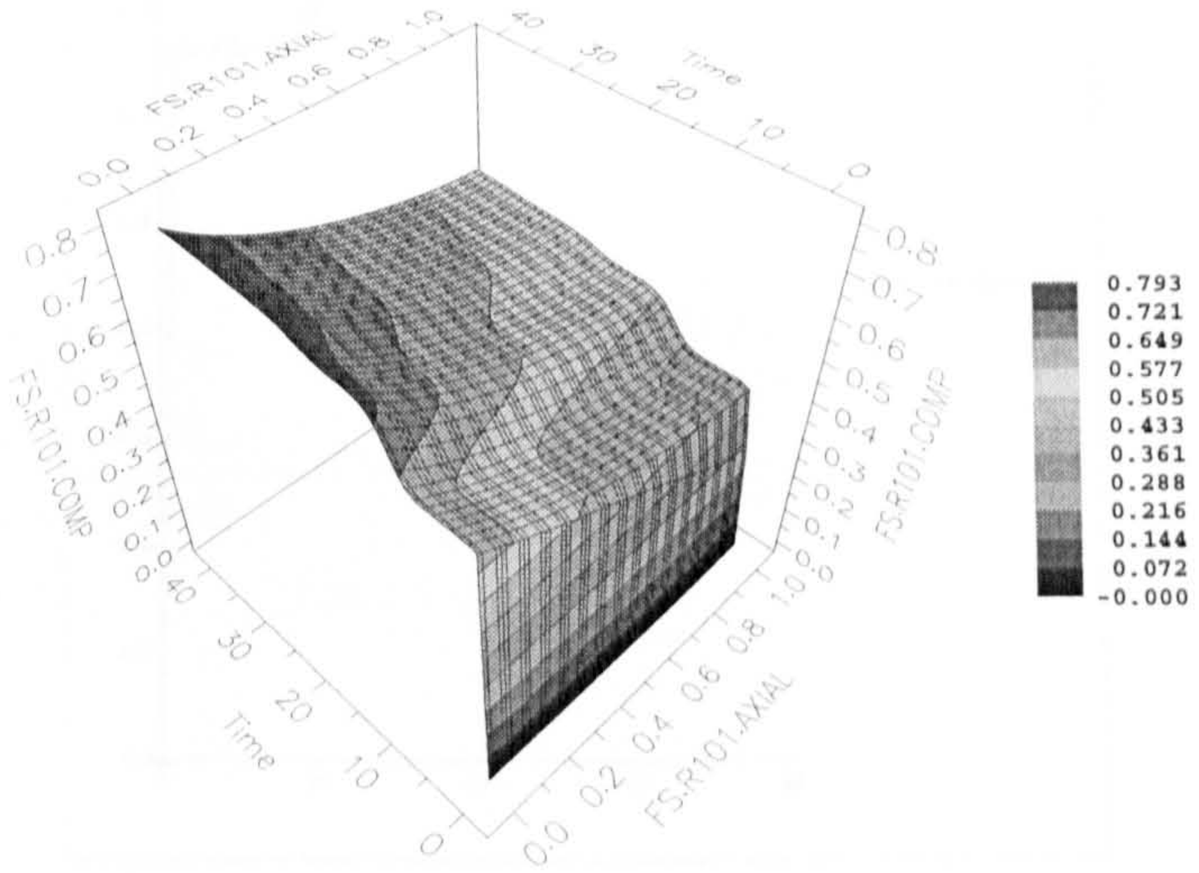


Figure 7.3: Concentration profile of component A in reactor in section 7.1

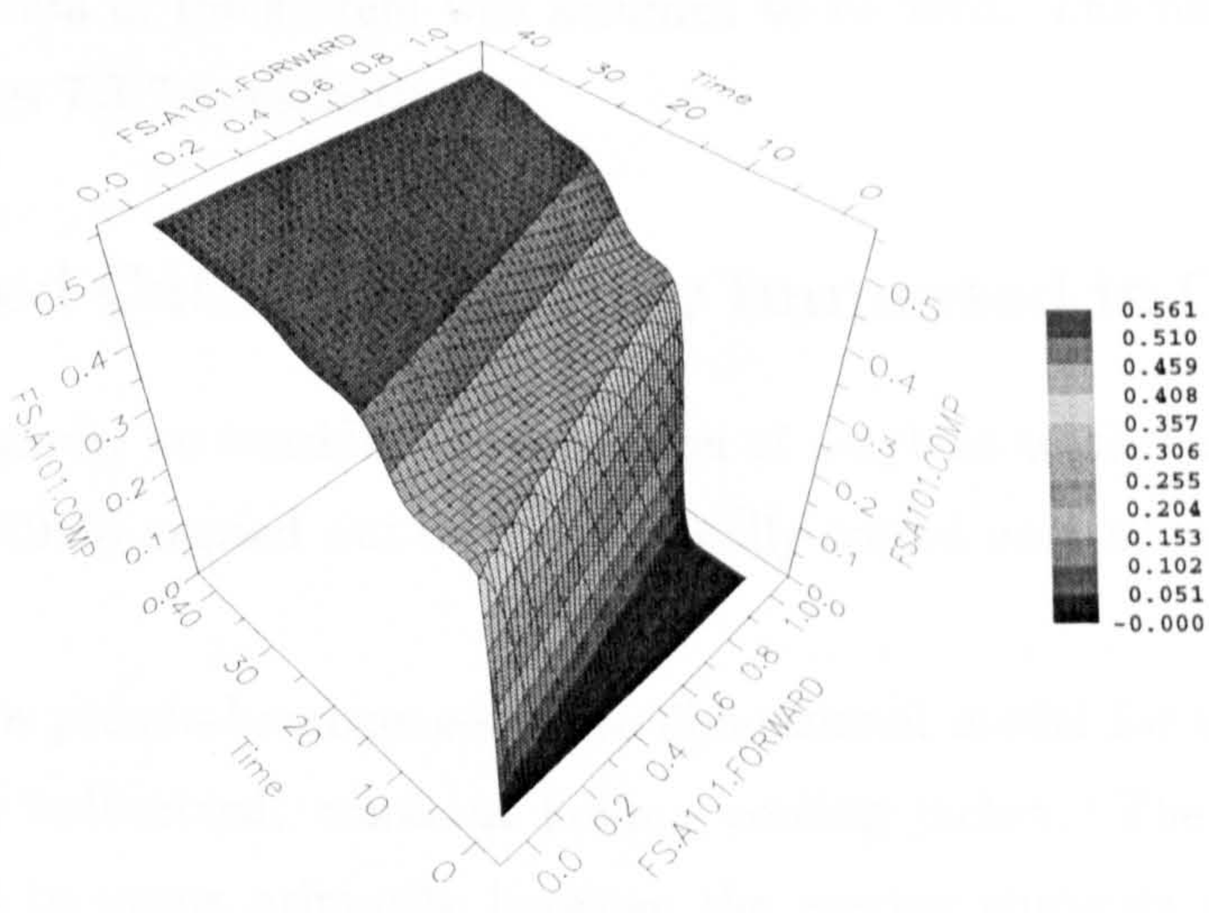


Figure 7.4: Concentration profile of component A in gas absorber in section 7.1

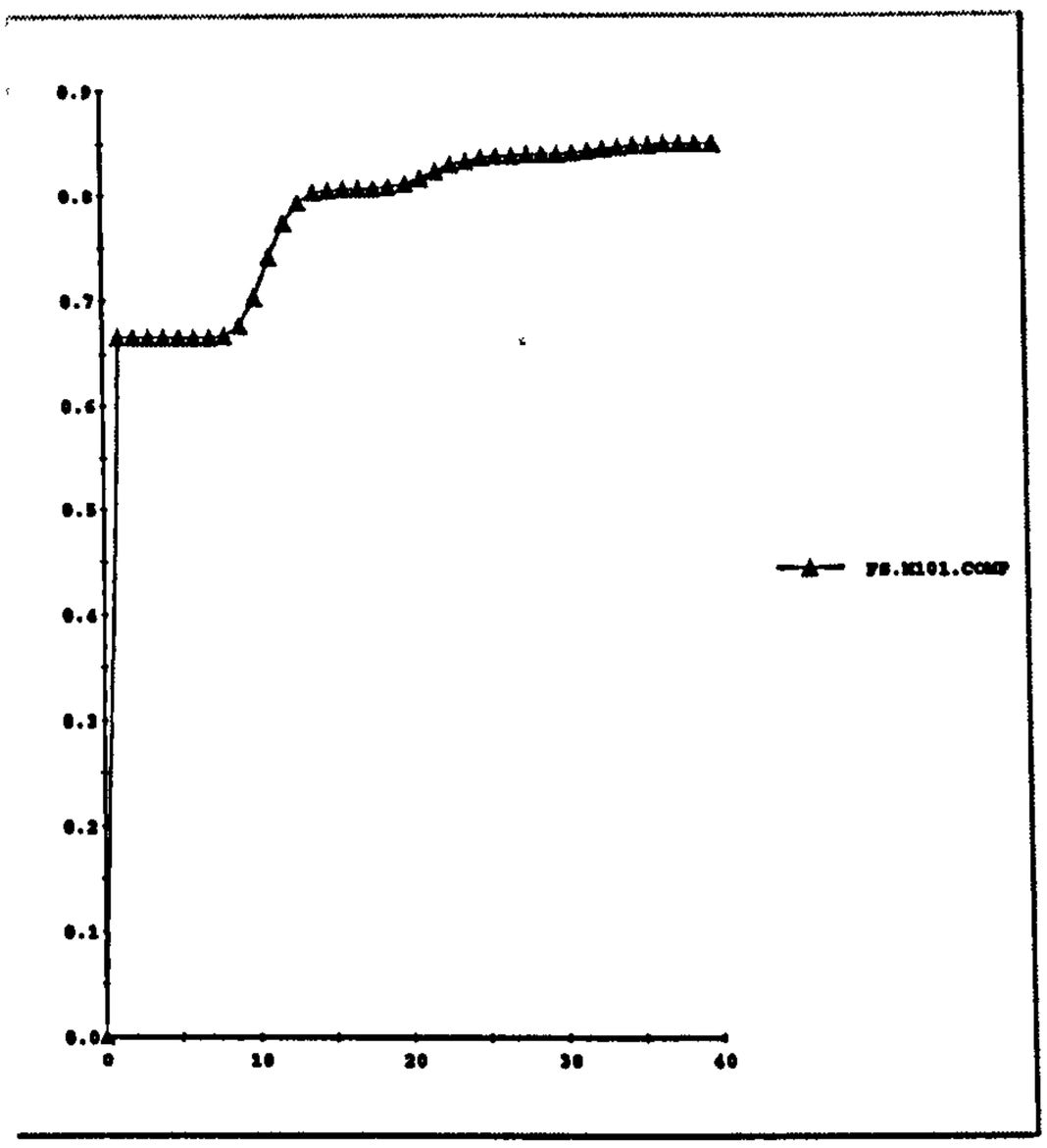


Figure 7.5: Time transient concentration of component A in mixer in section 7.1

centered finite difference approximations are used for the `Forward` and `Reverse` domains in the gas absorber unit (lines 22 and 23).

Dynamic simulation of the above system was carried out with a feed concentration of A of 1.0 $kmol/m^3$ and B of 0.9 $kmol/m^3$. The initial concentration of all three components in the three units in the system was assumed to be zero. The results are presented graphically in Figures 7.3-7.5 respectively.

7.2 Fixed-bed Catalytic Reactor Immersed in Cooling Jacket

In this example, we consider the oxidation of *o*-xylene to phthalic anhydride (Froment and Bischoff, 1990) carried out in the externally-cooled reactor system illustrated in Figure 3.2.

We adopt a pseudo-homogeneous two-dimensional model for the tubular reactor, while we assume a well-mixed, constant holdup cooling jacket. The resistance to heat transfer is assumed to occur primarily between the reactor contents and the wall of the tube, the latter being effectively at the cooling medium temperature. For simplicity, physical properties are assumed to be constant over the range of conditions in the system.

Fixed-bed catalytic reactor:

Mass balance for component i:

$$\frac{\partial c_i}{\partial t} = -\frac{\partial}{\partial z}(v_z c_i) + \frac{\partial}{\partial z}(\varepsilon D_z \frac{\partial c_i}{\partial z}) + \frac{1}{r}\frac{\partial}{\partial r}(\varepsilon D_r r \frac{\partial c_i}{\partial r}) + \nu_i \mathrm{r} \rho_\mathrm{b} \qquad \forall\, z \in (0, L_\mathrm{a}),\, r \in (0, L_\mathrm{r}) \tag{7.12}$$

Boundary conditions:

$$-\varepsilon D_z \frac{\partial c_i}{\partial z} = v_z(c_{i0} - c_i) \qquad @\, z = 0,\, r \in [0, L_\mathrm{r}] \tag{7.13}$$

$$\frac{\partial c_i}{\partial z} = 0 \qquad @\, z = L_\mathrm{a},\, r \in [0, L_\mathrm{r}] \tag{7.14}$$

$$\frac{\partial c_i}{\partial r} = 0 \qquad @\, r = 0,\, z \in (0, L_\mathrm{a}) \tag{7.15}$$

$$\frac{\partial c_i}{\partial r} = 0 \qquad @\, r = L_\mathrm{r},\, z \in (0, L_\mathrm{a}) \tag{7.16}$$

where $c_i(z, r, t)$ is the concentration of component i, v_z the superficial velocity, ε the voidage fraction, D_z and D_r the diffusivities in the axial and radial directions respectively, r the rate of reaction, ρ_b the bulk density, L_a the reactor length, L_r the reactor radius, c_{i0} the inlet concentration of component i, ν_i the stoichiometric coefficient of component i, and z and r the axial and radial coordinates respectively.

Energy balance:

$$\frac{\partial}{\partial t}(\rho_\mathrm{f} c_\mathrm{pf} T) = -\frac{\partial}{\partial z}(v_z \rho_\mathrm{f} c_\mathrm{pf} T) + \frac{\partial}{\partial z}(k_z \frac{\partial T)}{\partial z}) + \frac{1}{r}\frac{\partial}{\partial r}(k_r r \frac{\partial T}{\partial r}) + \rho_\mathrm{b} \mathrm{r}(-\Delta H) \qquad \forall\, z \in (0, L_\mathrm{a}),\, r \in (0, L_\mathrm{r}) \tag{7.17}$$

Boundary conditions:

$$-k_z \frac{\partial T}{\partial z} = \rho_\mathrm{f} c_\mathrm{pf} v_z (T_\mathrm{fi} - T) \qquad @\, z = 0,\ \ r \in [0, L_\mathrm{r}] \tag{7.18}$$

$$\frac{\partial T}{\partial z} = 0 \qquad @\, z = L_\mathrm{a},\ \ r \in [0, L_\mathrm{r}] \tag{7.19}$$

$$\frac{\partial T}{\partial r} = 0 \qquad @\, r = 0,\ \ z \in (0, L_\mathrm{a}) \tag{7.20}$$

$$-k_r \frac{\partial T}{\partial r} = h_\mathrm{w}(T - T_\mathrm{w}) \qquad @\, r = L_\mathrm{r},\ \ z \in (0, L_\mathrm{a}) \tag{7.21}$$

where $T(z, r, t)$ is the temperature in the reactor, ρ_f the fluid density, c_pf the fluid specific heat capacity, k_z and k_r the thermal conductivities in the axial and radial directions respectively, ΔH the heat of reaction, h_w the wall heat transfer coefficient, T_fi the inlet temperature of fluid and T_w the temperature of the reactor wall.

Quantity	Value	Units
c_{pf}	992.0	$J/kg \cdot K$
c_{pw}	4200	$J/kg \cdot K$
D_r	0.01	m^2/s
D_z	0.0001	m^2/s
k_r	0.5	$W/m \cdot K$
k_z	0.05	$W/m \cdot K$
L_{a}	3.0	m
L_{r}	0.00254	m
T_{cw}	625.15	K
U	96.0	$W/m^2 \cdot K$
v_z	1.0	m/s
α	19.837	
β	13636.0	K
ΔH	-1200000	J/mol
ϵ	0.35	
ρ_{b}	1300.0	kg/m^3
ρ_{f}	1.293	kg/m^3

Table 7.2: Physical properties and conditions in section 7.2

Reaction kinetics:

We assume a second-order irreversible reaction with the following rate equation:

$$r_{\mathrm{A}} = kP_{\mathrm{A}}P_{\mathrm{B}} \tag{7.22}$$

$$k = \exp(\alpha - \frac{\beta}{T}) \tag{7.23}$$

where P_{A} and P_{B} are the partial pressures of component A and B respectively, k the reaction rate constant, and α, β kinetic constants.

Cooling jacket:

Energy balance:

$$\rho_{\mathrm{w}} V_{\mathrm{w}} c_{\mathrm{pw}} \frac{dT_{\mathrm{cw}}}{dt} = f_{\mathrm{w}} c_{\mathrm{pw}} (T_{\mathrm{ci}} - T_{\mathrm{cw}}) + h_{\mathrm{w}} A \int_0^L (T(z, L_{\mathrm{r}}) - T_{\mathrm{cw}}) dz \tag{7.24}$$

where $T_{\mathrm{cw}}(t)$ is the temperature of the coolant, ρ_{w} is the density of coolant, V_{w} the volume of the jacket, f_{w} the mass flow rate, c_{pw} the coolant heat capacity, and A the area of the tube in contact with the cooling medium.

The above two mathematical models expressed in the *gPROMS* language are illustrated in Appendix B.2. As shown there, the higher level model, `MODEL CooledReactor`, combines instances of the two low level *MODEL*s and specifies the solution method. It also

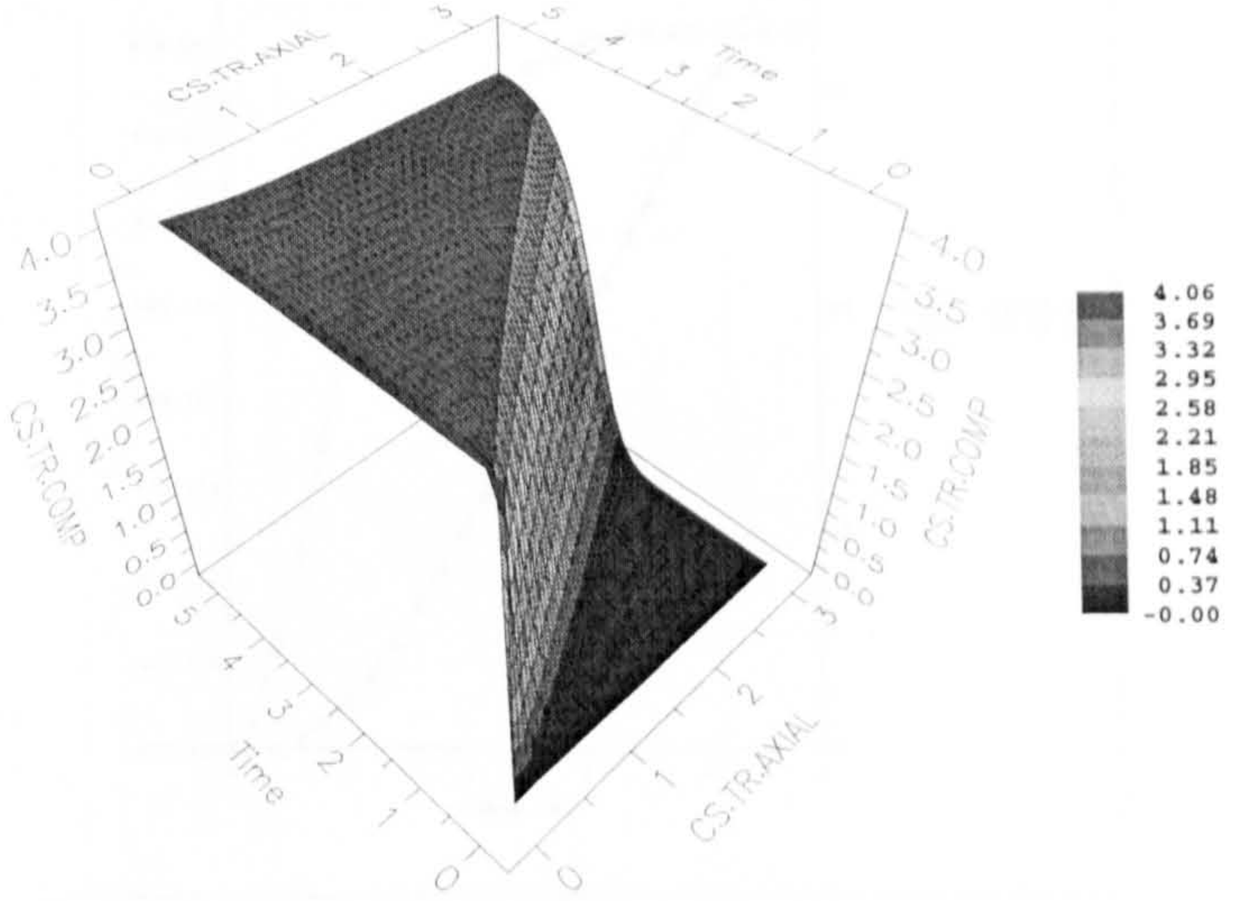


Figure 7.6: Concentration profile of component B at the centre of fixed-bed catalytic reactor in section 7.2

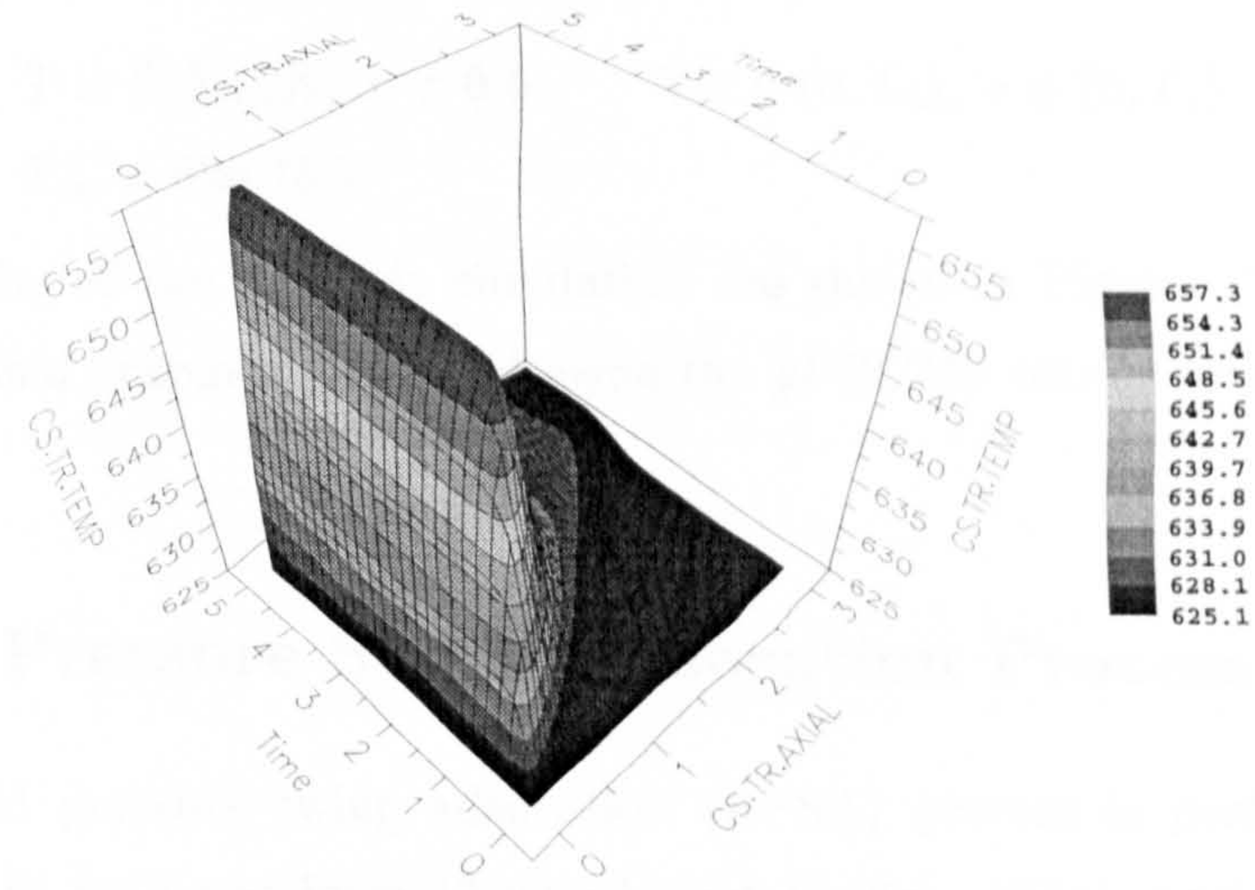


Figure 7.7: Temperature profile at the centre of fixed-bed catalytic reactor in section 7.2

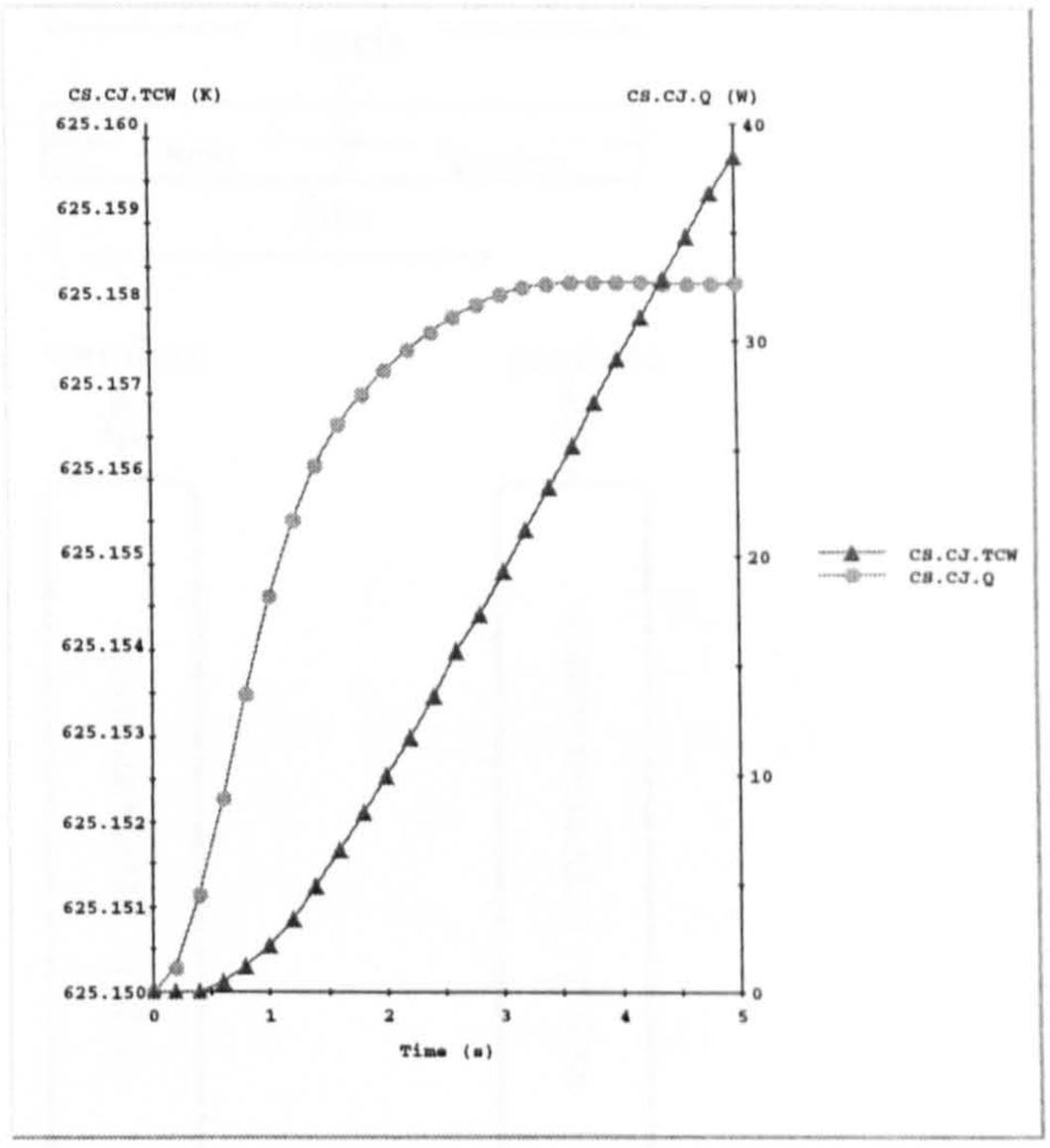


Figure 7.8: Temperature of cooling medium and cooling load in section 7.2

links the two sub-models by equating the reactor wall temperature to that of the cooling medium, and expressing the total heat input to the jacket, `Q`, as an integral of the heat loss from the tube surface over the entire tube length.

The parameter values used for the dynamic simulation are shown in Table 7.2.

The initial conditions ($t = 0$) were as follows:

$$T = 625.15K,\ c = 0.0 \qquad \forall\ z \in (0, L_{\mathrm{a}}),\ r \in (0, L_{\mathrm{r}})$$
$$T_{\mathrm{cw}} = 625.15K$$

The results of the dynamic simulation are shown in Figures 7.6-7.8 in terms of 2- and 3- dimensional graphics generated using the *gPROMS* results management (*gRMS*) facilities.

7.3 Rapid Pressure Swing Adsorption Process

The rapid pressure swing adsorption (RPSA) process is performed in a single adsorption bed. In its basic form (Turnock and Kadlec, 1971), each operational cycle comprises a pressurisation and a depressurisation phase as shown in Figure 7.9.

The durations of the two phases are approximately equal, and of the order of a few seconds. During the pressurisation phase, high-pressure feed is introduced at the bed inlet,

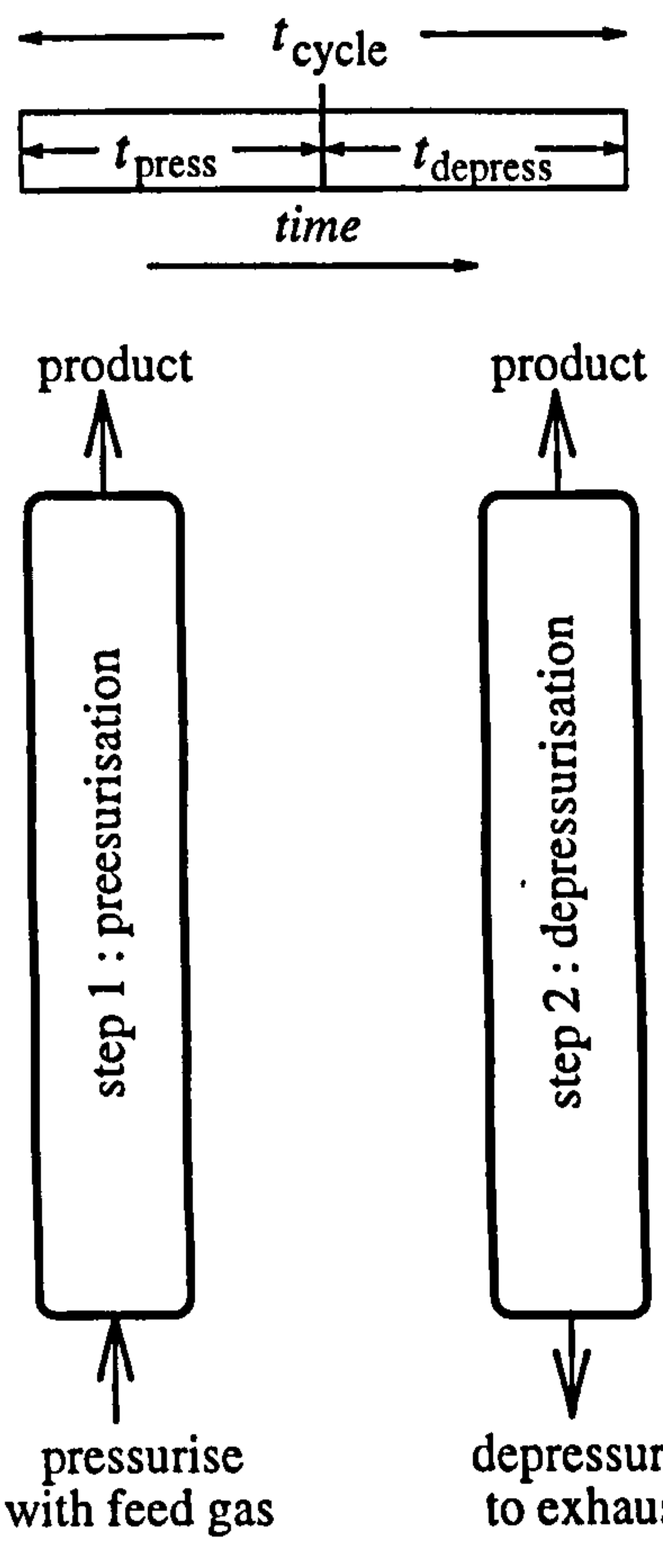


Figure 7.9: Basic RPSA process of section 7.3

and product rich in the less strongly adsorbed component(s) is withdrawn at the outlet. Following this, the feed supply is interrupted, and the bed inlet is switched to a low-pressure exhaust while still withdrawing product at the other end. This process is then repeated.

After an initial start-up period, the system reaches a periodic steady-state, and it is this state which is of primary interest in terms of assessing the process performance. Traditionally, the periodic conditions have been computed by carrying out a dynamic simulation of the system, starting from a given initial condition, over a large number of cycles. However, a more direct way is to replace the initial condition specification by a periodicity condition demanding that the process states at the start and end of the cycle be identical (Nilchan and Pantelides, 1994).

In *gPROMS*, the periodicity condition is readily taken into account by treating time explicitly as a distribution domain over a single cycle (*cf.* section 3.7.3). For simplicity, we assume isothermal bed operation and ideal gas behaviour, and ignore any radial variation of properties within the bed. This leads to the following mathematical model:

Mass balance on component i:

$$\frac{\partial \tilde{c}_i}{\partial t} = -\frac{\partial}{\partial z}(v_z c_i) + D_z \frac{\partial^2 c_i}{\partial z^2} \qquad \forall\, z \in (0, L_a),\ t \in (0, t_{cycle}] \tag{7.25}$$

where t and z denote time and axial position respectively, v_z the superficial velocity of gas in the bed, c_i the gas phase molar concentration of component i, and D_z the effective dispersion coefficient. The combined gas/solid molar concentration $\tilde{c}_i$, is given by

$$\tilde{c}_i = \varepsilon_t c_i + \rho_b q_i \tag{7.26}$$

where ρ_b is the bed density and q_i the solid phase concentration. The total bed void fraction, ε_t, is given by

$$\varepsilon_t = \varepsilon_b + \varepsilon_p(1 - \varepsilon_b) \tag{7.27}$$

where ε_b and ε_p are the interparticle and intraparticle void fractions respectively.

Adsorption rate:

We assume a simple linear driving force adsorption process leading to the rate equation

$$\frac{\partial q_i}{\partial t} = k_i^*(q_i^* - q_i) \qquad \forall\, z \in [0, L_a],\ i = 1..N \tag{7.28}$$

where q_i^* is the equilibrium adsorbed phase concentration, and k_i^* the overall mass transfer coefficient for component i.

Adsorption isotherm:

Assuming a linear isotherm for the gas-solid equilibrium and ideal gas behaviour, we have

$$q_i^* = m_i c_i RT \qquad \forall\, z \in [0, L_a],\ t \in [0, t_{cycle}],\ i = 1..N \tag{7.29}$$

$$\frac{P}{RT} = \sum_{i=1}^{N} c_i \qquad \forall\, z \in (0, L_a],\ t \in [0, t_{cycle}] \tag{7.30}$$

where m_i is the adsorption isotherm gradient, T the bed temperature, R the universal gas constant, and P the pressure in the bed.

Pressure drop equation:

Assuming spherical particles and laminar flow, the pressure drop equation is of the form:

$$\frac{\partial P}{\partial z} = -\frac{\alpha \mu (1 - \varepsilon_b)^2}{d_p^2 \varepsilon_b^3} v_z \qquad \forall\, z \in [0, L_a),\ t \in [0, t_{cycle}] \tag{7.31}$$

where μ is the gas viscosity, d_p the particle diameter, and α a constant.

Quantity	Value	Units
A	1.963×10^{-3}	m^2
d_{p}	1.0×10^{-6}	m
D_z	1.0×10^{-6}	m^2/s
k_1^*	25.32	s^{-1}
k_2^*	55.23	s^{-1}
L_{a}	1.0	m
m_1	3.0×10^{-6}	$m^2 \cdot mol/N \cdot kg$
m_2	1.43×10^{-6}	$m^2 \cdot mol/N \cdot kg$
P_{atm}	1.0×10^5	Pa
P_{f}	2.4×10^5	Pa
P_{w}	1.0×10^5	Pa
Q	1.0×10^{-5}	m^3/s
t_{cycle}	3.0	s
t_{press}	1.5	s
T_{f}	290	K
y_1	0.79	
y_2	0.21	
α	180	
ε_{b}	0.35	
ε_{p}	0.55	
μ	1.8×10^{-5}	$N \cdot s/m^2$
ρ_{b}	800	kg/m^3

Table 7.3: Physical properties and conditions in section 7.3

Boundary conditions:

The above equations describe the physical behaviour of the bed under all conditions. The boundary conditions, however, depend on the phase of operation. In particular, during the pressurisation phase, the inlet ($z = 0$) boundary conditions are

$$c_i = y_i \frac{P_{\mathrm{f}}}{\mathrm{R}T} \qquad @\, z = 0,\ \forall t \in (0, t_{\mathrm{press}}],\ i = 1..N \tag{7.32}$$

$$P = P_{\mathrm{f}} \qquad @\, z = 0,\ \forall t \in (0, t_{\mathrm{press}}] \tag{7.33}$$

where y_i is the mole fraction of component i in the feed, P_{f} the feed pressure and t_{press} the pressurisation time. Note that, for simplicity, the above form of the boundary condition on concentration is used instead of the more usual Danckwerts condition.

During the depressurisation phase, these conditions are replaced by

$$\frac{\partial c_i}{\partial z} = 0 \qquad @\, z = 0,\ \forall t \in (t_{\mathrm{press}}, t_{\mathrm{cycle}}],\ i = 1..N \tag{7.34}$$

$$P = P_{\mathrm{w}} \qquad @\, z = 0,\ \forall t \in (t_{\mathrm{press}}, t_{\mathrm{cycle}}] \tag{7.35}$$

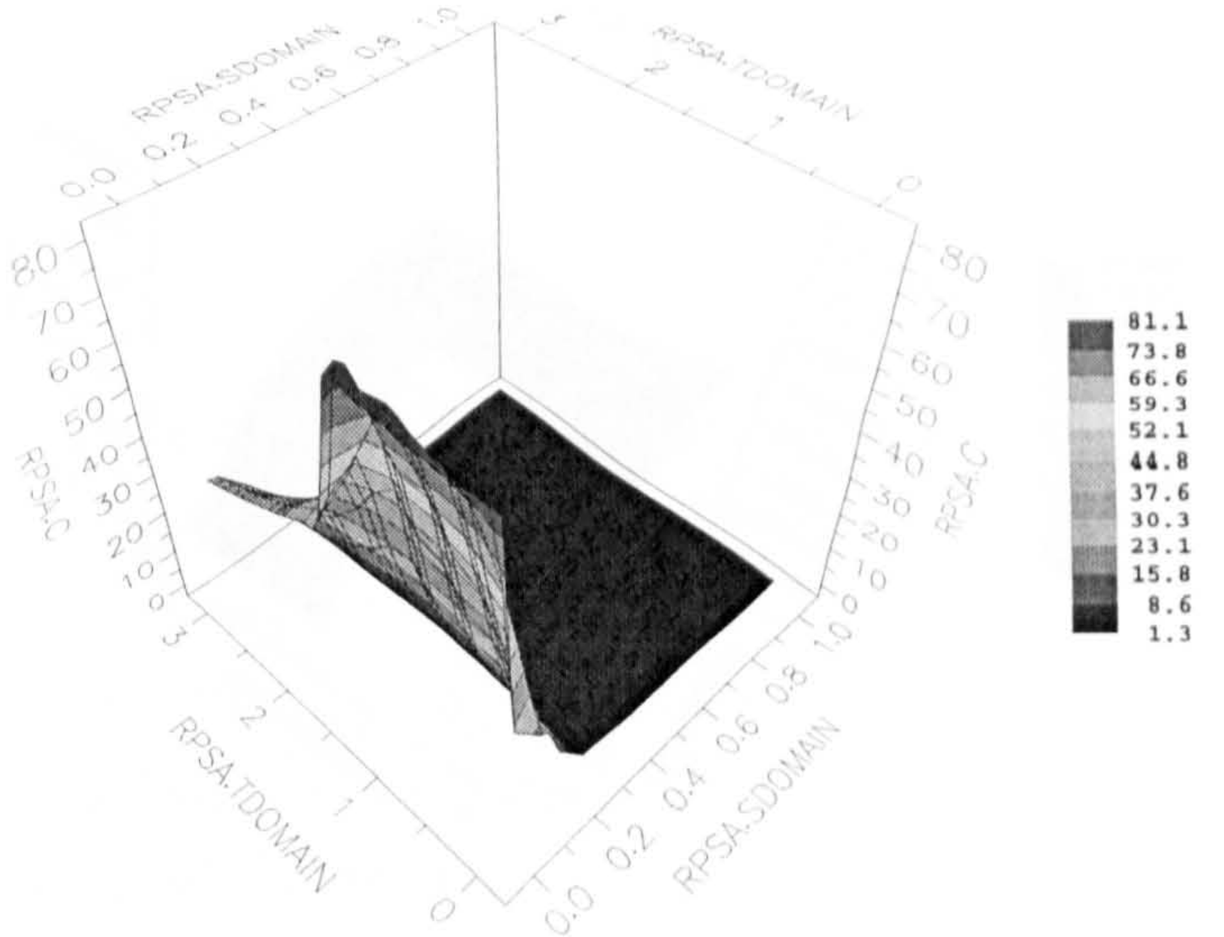


Figure 7.10: Gas phase concentration of nitrogen in section 7.3

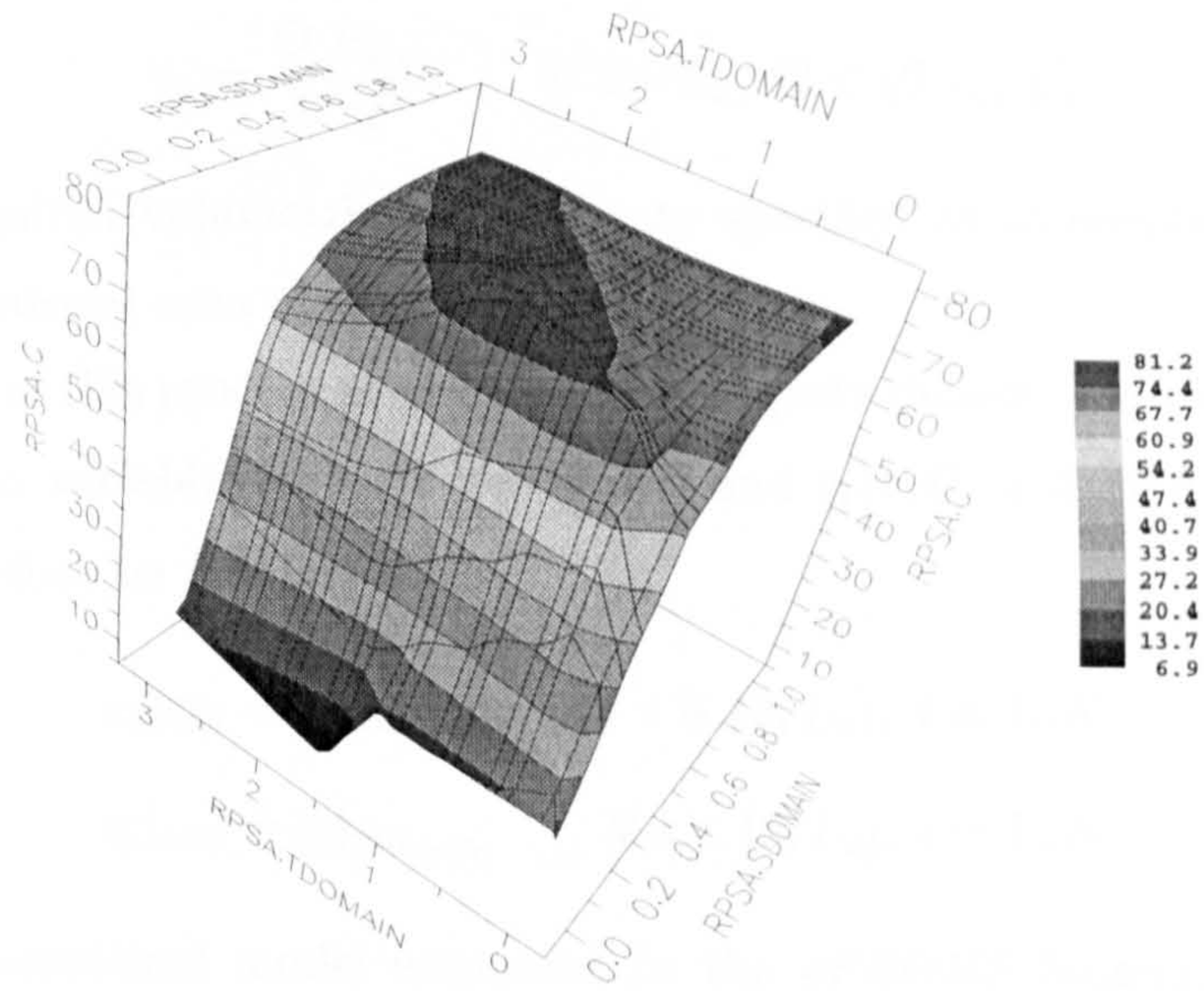


Figure 7.11: Gas phase concentration of oxygen in section 7.3

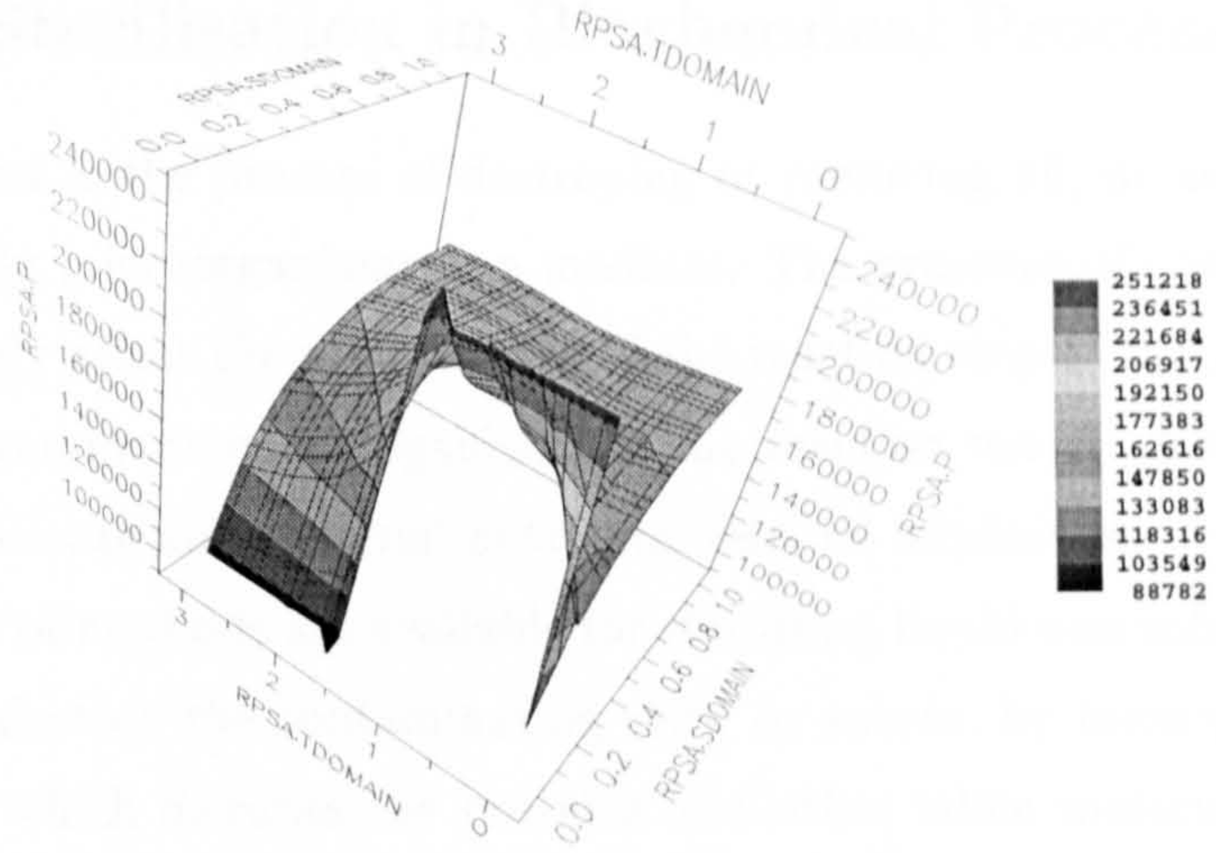


Figure 7.12: Pressure in RPSA column in section 7.3

where t_{cycle} is the cycle time and P_{w} the external pressure.

During both phases, the outlet boundary conditions are

$$\frac{\partial c_i}{\partial z} = 0 \qquad @\ z = L_{\text{a}},\ \forall t \in (0, t_{\text{cycle}}] \tag{7.36}$$

$$v_z = \frac{Q}{A}\frac{P_{\text{atm}}}{P} \qquad @\ z = L_{\text{a}},\ \forall t \in (0, t_{\text{cycle}}] \tag{7.37}$$

where Q is the required volumetric gas flow rate specified at atmospheric pressure P_{atm}, and A the cross-sectional area of the bed.

The state of this process at any time t is uniquely determined by the fluid and solid phase concentration variables $c_i(z,t)$, $z \in (0, L_{\text{a}})$ and $q_i(z,t)$, $z \in [0, L_{\text{a}}]$. The periodicity condition can therefore be written as

$$c_i|_{t=0} = c_i|_{t=t_{\text{cycle}}} \qquad \forall\ z \in (0, L_{\text{a}}),\ i = 1..N \tag{7.38}$$

$$q_i|_{t=0} = q_i|_{t=t_{\text{cycle}}} \qquad \forall\ z \in [0, L_{\text{a}}],\ i = 1..N \tag{7.39}$$

The mathematical model expressed in the *gPROMS* language is shown in Appendix B.3. Both temporal and spatial domains are explicitly declared (`SDomain`, `TDomain`) in the model. The latter was used to simulate the oxygen-enrichment of air using Zeolite 5A. The operating conditions and model parameters are shown in Table 7.3.

A third-order orthogonal collocation method on 7 finite elements was employed for both the spatial and temporal domains. Typical results are shown in Figures 7.10-7.12.

7.4 Media Sterilisation in Biochemical Processes

Sterilisation is the process of destroying or removing all, or as many as possible, of the contaminating microorganisms in a medium. The presence of contaminant microorganisms can seriously affect the performance of the reaction step of the process. Therefore, the number of contaminant microorganisms in the reaction vessels, interconnecting pipes and the feed of substrate and essential nutrients must be minimised.

A number of methods are available for sterilising liquid and solid media. The most common one is to destroy the contaminating cells, or spores, by heating the medium to a high temperature, which disrupts the proteins and other labile molecules within the cell, rendering it unable to function. This is simple, reliable and economical.

In continuous heat sterilisation, a quantity of medium is heated up and subsequently held at a constant high temperature for a period before being cooled down to the fermentation temperature. The sterilisation time is of the order of minutes and takes place at high temperature, typically 140 to 150℃. A typical flowsheet for a continuous heat steriliser is shown in Figure 7.13. Spiral heat exchangers are used in order to minimise solids

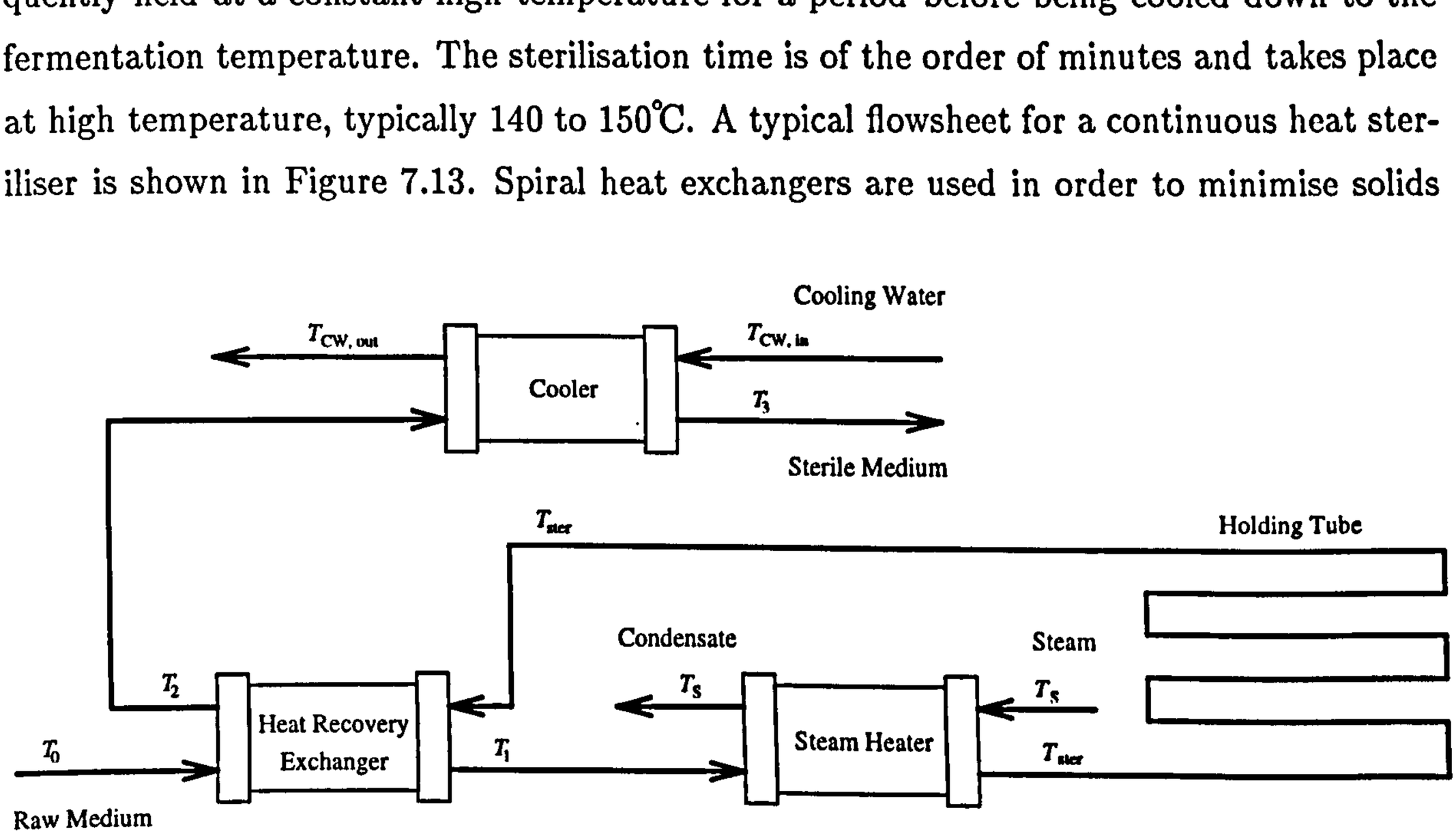


Figure 7.13: Continuous sterilisation network in section 7.4

accumulation in dead spaces.

Here we study the *steady-state* behaviour of the system using a model proposed by Samsatli (1994). The cooler heat exchanger was not modelled because it was found that

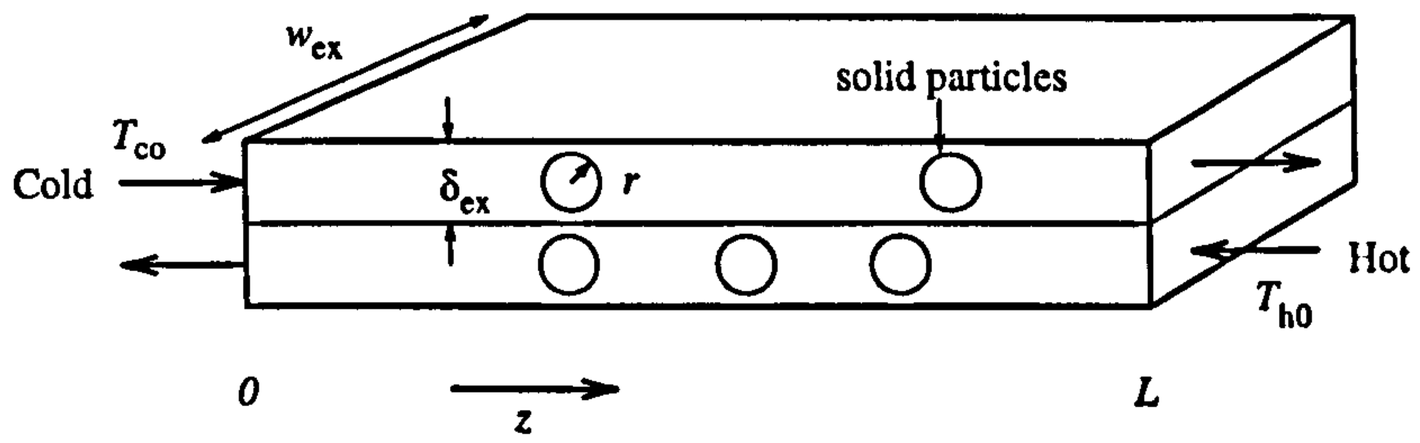


Figure 7.14: Schematic diagram of the recovery heat exchanger in section 7.4

the death rate of cells already drops to an insignificant value within the hot side of the recovery heat exchanger.

A schematic diagram of the recovery heat exchanger is shown in Figure 7.14. The cold and hot fluids flow countercurrently in two channels of rectangular cross-section of width w_{ex}, height δ_{ex} and length L. The fluids on both sides carry spherical solid particles of radius R_p.

Heat transfer takes place from the hot fluid to the cold fluid, and from each fluid to the solid particles within it. Temperatures and concentrations are assumed to be uniform in each fluid over any cross-section, but gradients exist within each solid particle. However, all solid particles at any cross-section of each fluid are assumed to be at identical conditions.

Below we present the model for the cold side of the recovery heat exchanger (Samsatli, 1994). The hot side model is very similar.

Liquid medium:

Mass balance:

$$\frac{\dot{v}}{w_{ex}\delta_{ex}}\frac{dN}{dz} = -K_{d0}\exp\left[\frac{-E_d}{RT_c}\right]N \qquad \forall z \in (0, L] \tag{7.40}$$

$$\frac{\dot{v}}{w_{ex}\delta_{ex}}\frac{dC}{dz} = -K_{d0,nut}\exp\left[\frac{-E_{d,nut}}{RT_c}\right]C \qquad \forall z \in (0, L] \tag{7.41}$$

Boundary conditions:

$$N = N_0 \qquad @\, z = 0 \tag{7.42}$$

$$C = C_0 \qquad @\, z = 0 \tag{7.43}$$

where $\dot{v}$ is the volumetric flow rate, $N(z)$ and $C(z)$ the cell and nutrient concentrations respectively, K_{d0} and $K_{d0,nut}$ the first-order death rate constants for micro-organisms and nutrients respectively, E_d and $E_{d,nut}$ the corresponding activation energies, R the universal gas constant and $T_c(z)$ the temperature of the fluid on the cold side.

We also introduce two performance measures, namely *st* the sterility of the liquid and *deg* the degradation of the liquid defined as follows:

$$st = \log_{10}\left[\frac{N(0)}{N(L)}\right] \tag{7.44}$$

$$deg = \log_{10}\left[\frac{C(0)}{C(L)}\right] \tag{7.45}$$

Energy balance:

$$\varepsilon\rho\dot{v}\frac{dT_c}{dz} = 2Uw_{ex}(T_h - T_c) - 3(1-\varepsilon)w_{ex}\delta_{ex}\frac{h}{R_p}(T_c - T_s(z, R_p)) \qquad \forall z \in (0, L] \tag{7.46}$$

Boundary Conditions:

$$T_c = T_{c,0} \qquad @\, z = 0 \tag{7.47}$$

where ε is the voidage, ρ the fluid density, U the overall heat transfer coefficient, $T_h(z)$ the temperature of fluid on the hot side, h the heat transfer coefficient between the solid particles and the liquid medium, R_p the radius of the particles and $T_s(z, r)$ the temperature of the solid particles.

Solid particles:

Mass balance:

$$\frac{\dot{v}}{w_{ex}\delta_{ex}}\frac{\partial N_s}{\partial z} = -K_{d0}\exp\left[\frac{-E_d}{RT_s}\right]N_s \qquad \forall z \in (0, L],\ r \in [0, R_p] \tag{7.48}$$

Boundary Condition:

$$N_s = N_{s,0} \qquad @z = 0,\ \forall r \in [0, R_p] \tag{7.49}$$

where $N_s(z, r)$ is the local concentration of micro-organisms in the solid particles. We also define the solid particle sterility measure, st_s:

$$st_s = \log_{10}\left(\frac{\int_0^{R_p} N_s(0, r)4\pi r^2\, dr}{\int_0^{R_p} N_s(L, r)4\pi r^2\, dr}\right) \tag{7.50}$$

Energy balance:

$$\frac{\dot{v}}{w_{ex}\delta_{ex}}\frac{\partial T_s}{\partial z} = \frac{k_s}{\rho_s c_{ps}}\left(\frac{\partial^2 T_s}{\partial r^2} + \frac{2}{r}\frac{\partial T_s}{\partial r}\right) \qquad \forall z \in (0, L],\ r \in (0, R_p) \tag{7.51}$$

Boundary Conditions:

$$T_s = T_{c,0} \qquad @z = 0,\ \forall r \in [0, R_p] \tag{7.52}$$

$$\frac{\partial T_s}{\partial r} = 0 \qquad @r = 0,\ \forall z \in (0, L] \tag{7.53}$$

$$k_s\frac{\partial T_s}{\partial r} = h(T_c - T_s) \qquad @r = R_p,\ \forall z \in (0, L] \tag{7.54}$$

Quantity	Values	Unit
C_0	10^{11}	m^{-3}
c_p	4229	$J/kg \cdot K$
c_{ps}	1880	$J/kg \cdot K$
E_d	283446.4	J/mol
$E_{d,nut}$	92109.6	J/mol
h	9650	$W/m^2 \cdot K$
k_s	0.445	$W/m \cdot K$
K_{d0}	exp(83.81)	s^{-1}
$K_{d0,nut}$	exp(21.2)	s^{-1}
L_h	90.432	m
L_s	48.974	m
L_t	3.53	m
N_0	10^{11}	m^{-3}
$N_{s,0}$	10^{11}	m^{-3}
$T_{c,0}$	298.15	K
T_{st}	423.15	K
$\dot{v}$	0.0111	m^3/s
w_{ex}	1.0	m
δ_{ex}	0.012	m
ε	0.9	
ρ	962	kg/m^3
ρ_s	790	kg/m^3

Table 7.4: Physical properties and conditions in section 7.4

where k_s is the thermal conductivity of the solid particle, ρ_s the solid density and c_{ps} the solid specific heat capacity.

The magnitudes of the values of N and C are extremely large, and in order to avoid numerical problems, it may be desirable to express the mass balance equations in terms of the natural logarithms of such variables as follows:

Liquid medium:

Mass balance:

$$\frac{\dot{v}}{w_{ex}\delta_{rx}}\frac{d\log N}{dz} = -K_{d0}\exp\left[\frac{-E_d}{RT_c}\right] \qquad \forall z \in (0, L] \tag{7.40'}$$

$$\frac{\dot{v}}{w_{ex}\delta_{ex}}\frac{d\log C}{dz} = -K_{d0,nut}\exp\left[\frac{-E_{d,nut}}{RT_c}\right] \quad \forall z \in (0, L] \tag{7.41'}$$

Boundary conditions:

$$\log N = \log N_0 \qquad @\ z = 0 \tag{7.42'}$$

$$\log C = \log C_0 \qquad @\ z = 0 \tag{7.43'}$$

Solid particles:

Mass balance:

$$-\frac{\dot{v}}{w_{ex}\delta_{ex}}\frac{\partial \log N_s}{\partial z} = K_{d0}\exp\left[\frac{-E_d}{RT_s}\right] \qquad \forall z \in (0, L],\ r \in [0, R_p] \qquad (7.48')$$

Boundary Condition:

$$\log N_s = \log N_{s,0} \qquad @z = 0,\ \forall r \in [0, R_p] \qquad (7.49')$$

Similar equations have been used for the steam heater, holding tube and the hot side of the recovery heat exchanger.

Based on the mathematical models and the flowsheet (Figure 7.13), we can now proceed with the simulation of continuous heat sterilisation. It is assumed that the physical properties of the system are independent of process conditions. Table 7.4 shows the values used for the simulation (Armenante and Leskowicz, 1990).

The *gPROMS* input file for this problem is given in Appendix B.4, and some results of the simulation are shown in Figures 7.15-7.19.

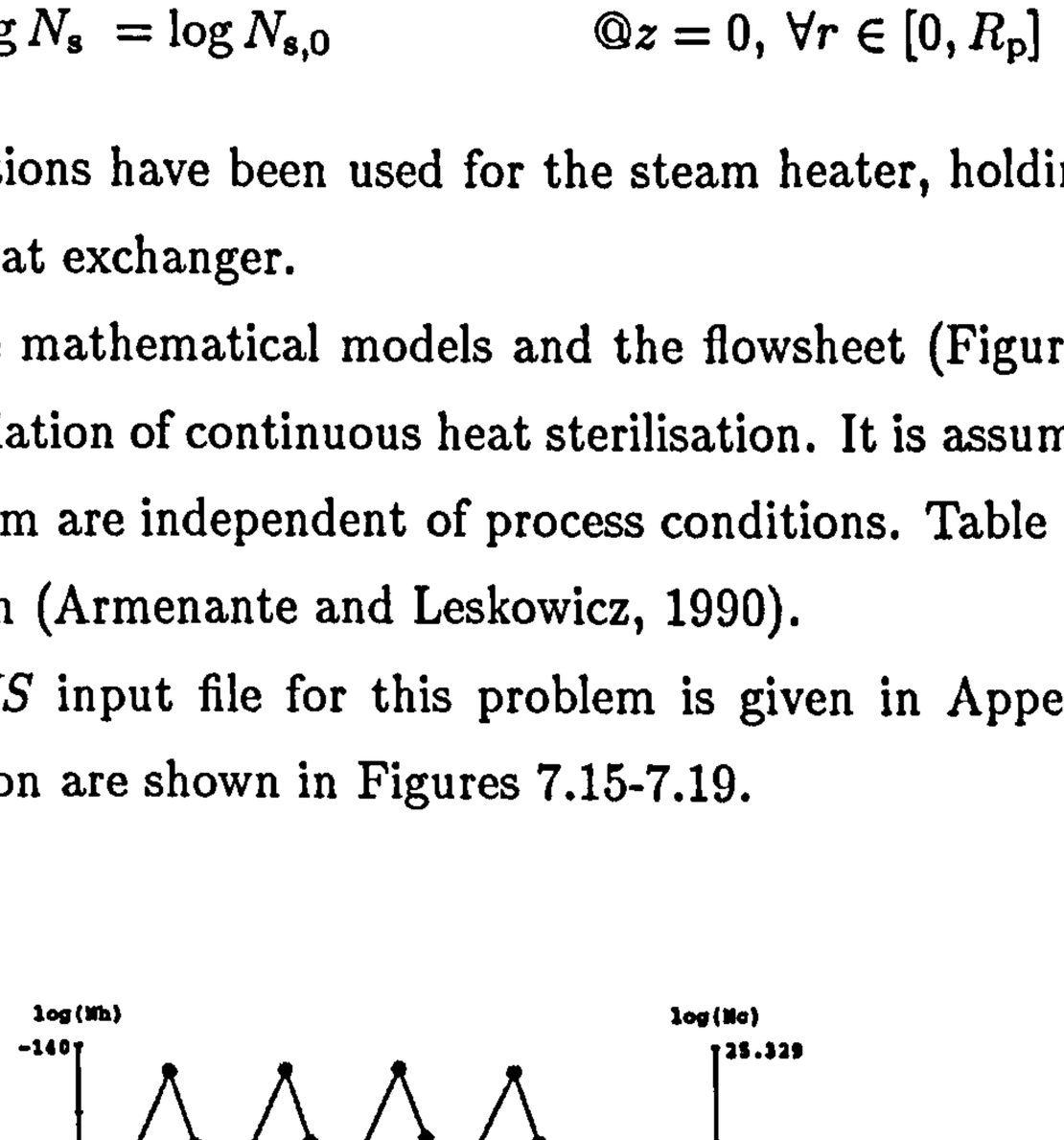


Figure 7.15: Cell concentration on hot and cold side of recovery exchanger in section 7.4

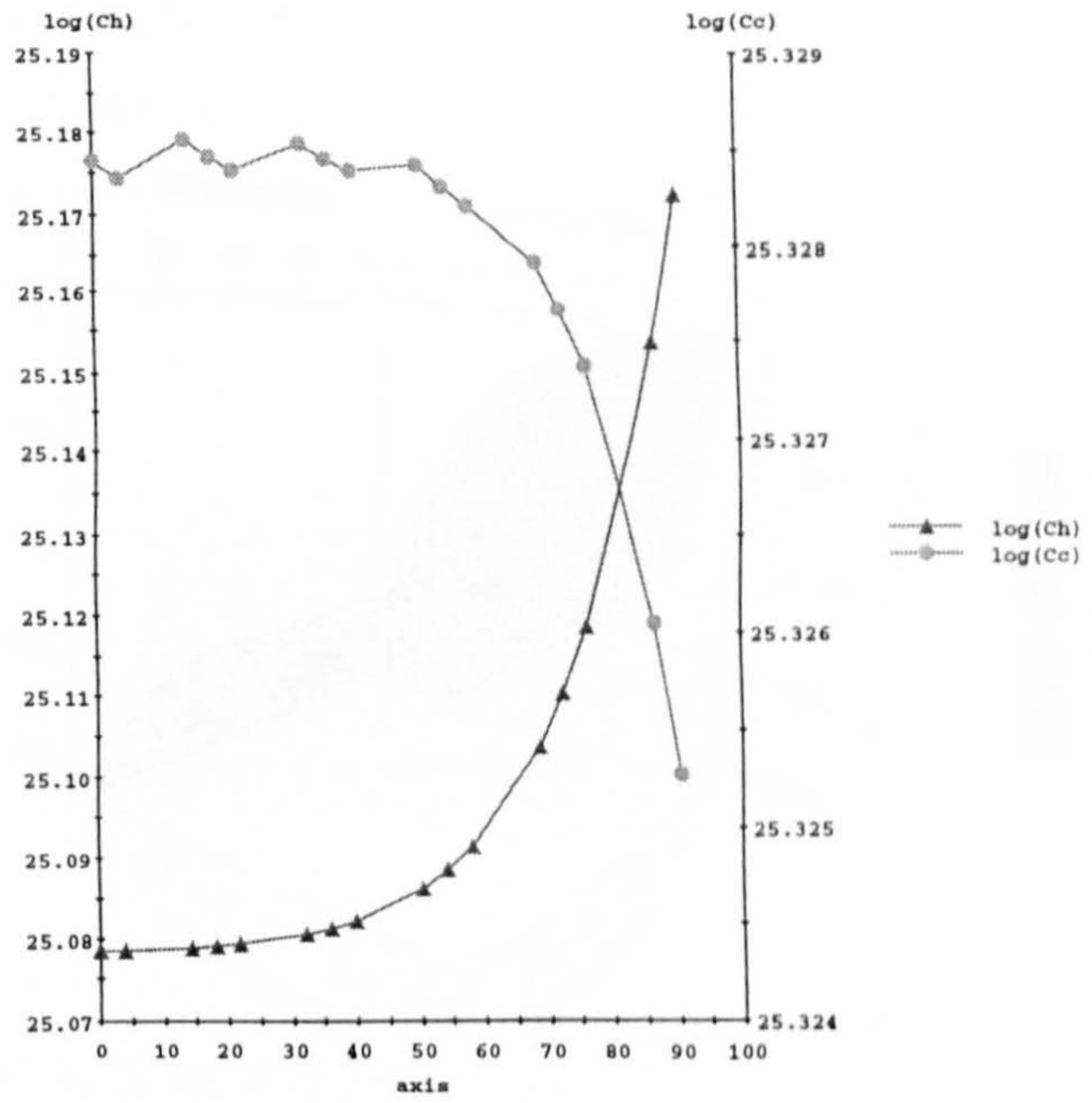


Figure 7.16: Nutrients concentration on hot and cold side of recovery exchanger in section 7.4

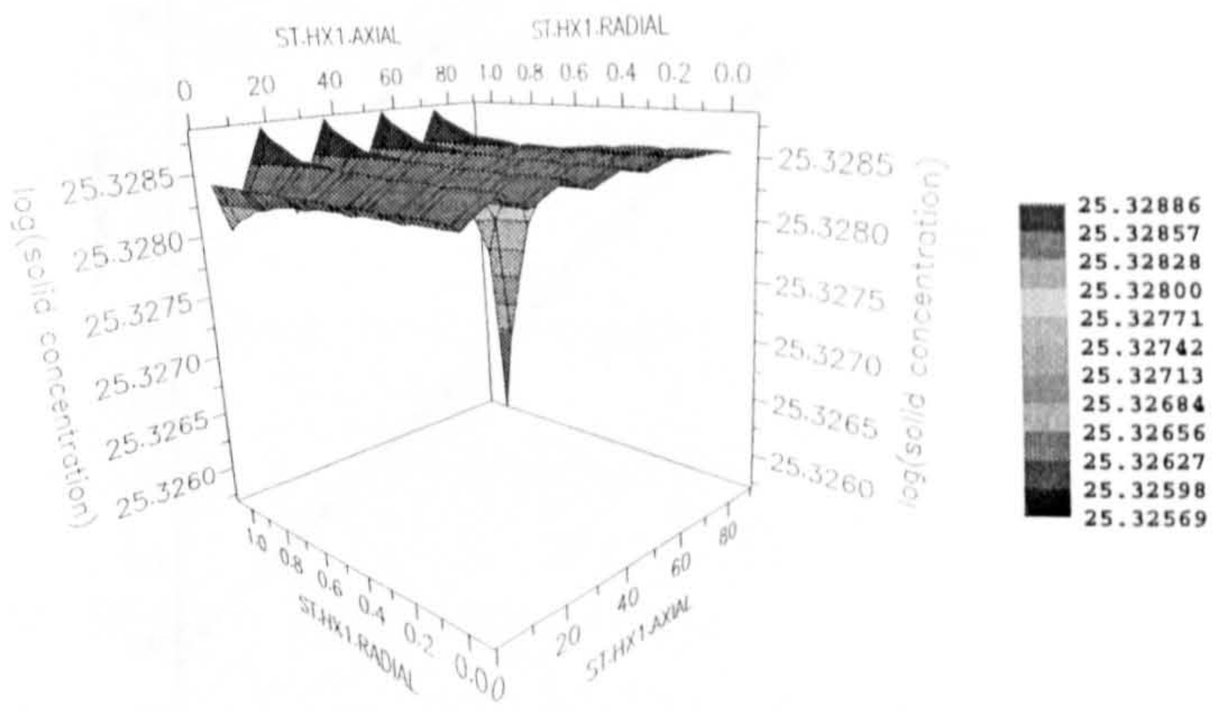


Figure 7.17: Cell concentration in solid on cold side of recovery exchanger in section 7.4

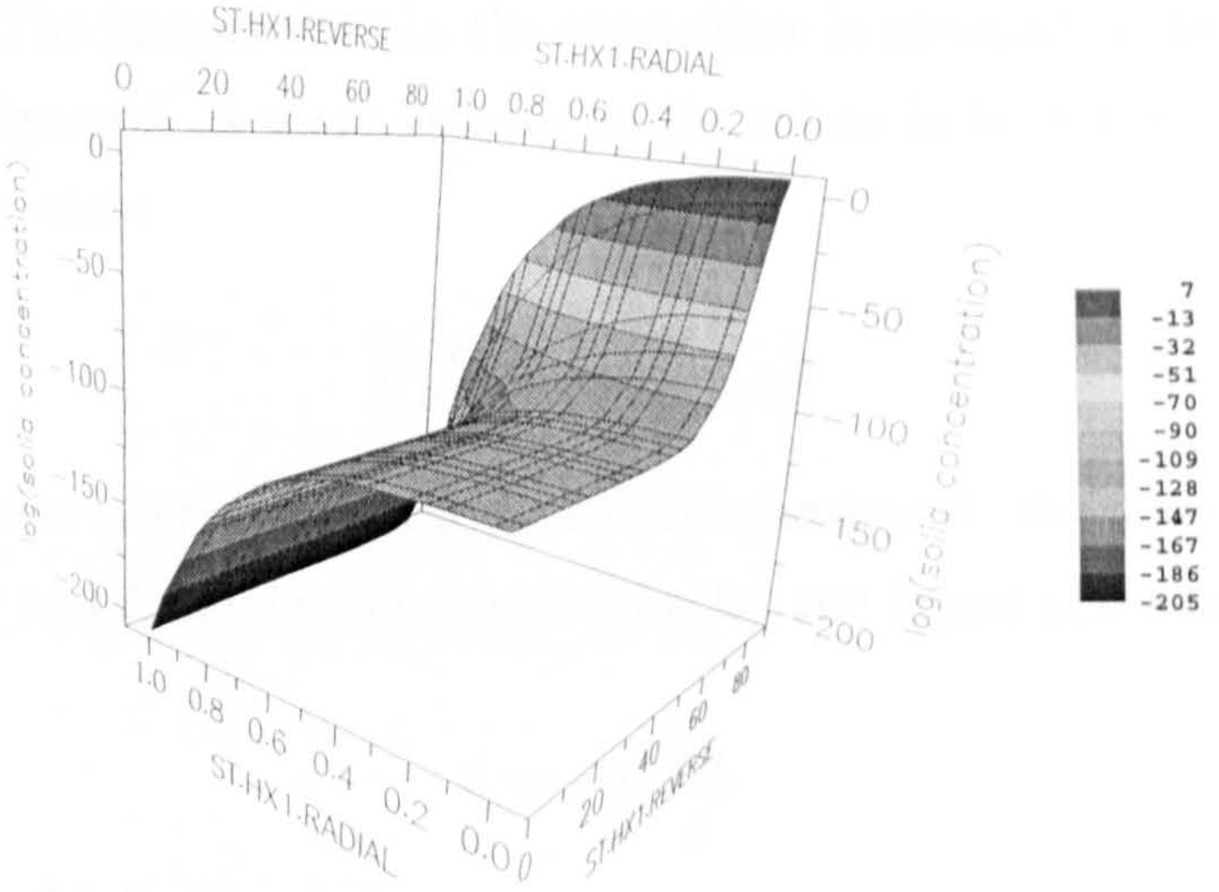


Figure 7.18: Cell concentration in solid on hot side of recovery exchanger in section 7.4

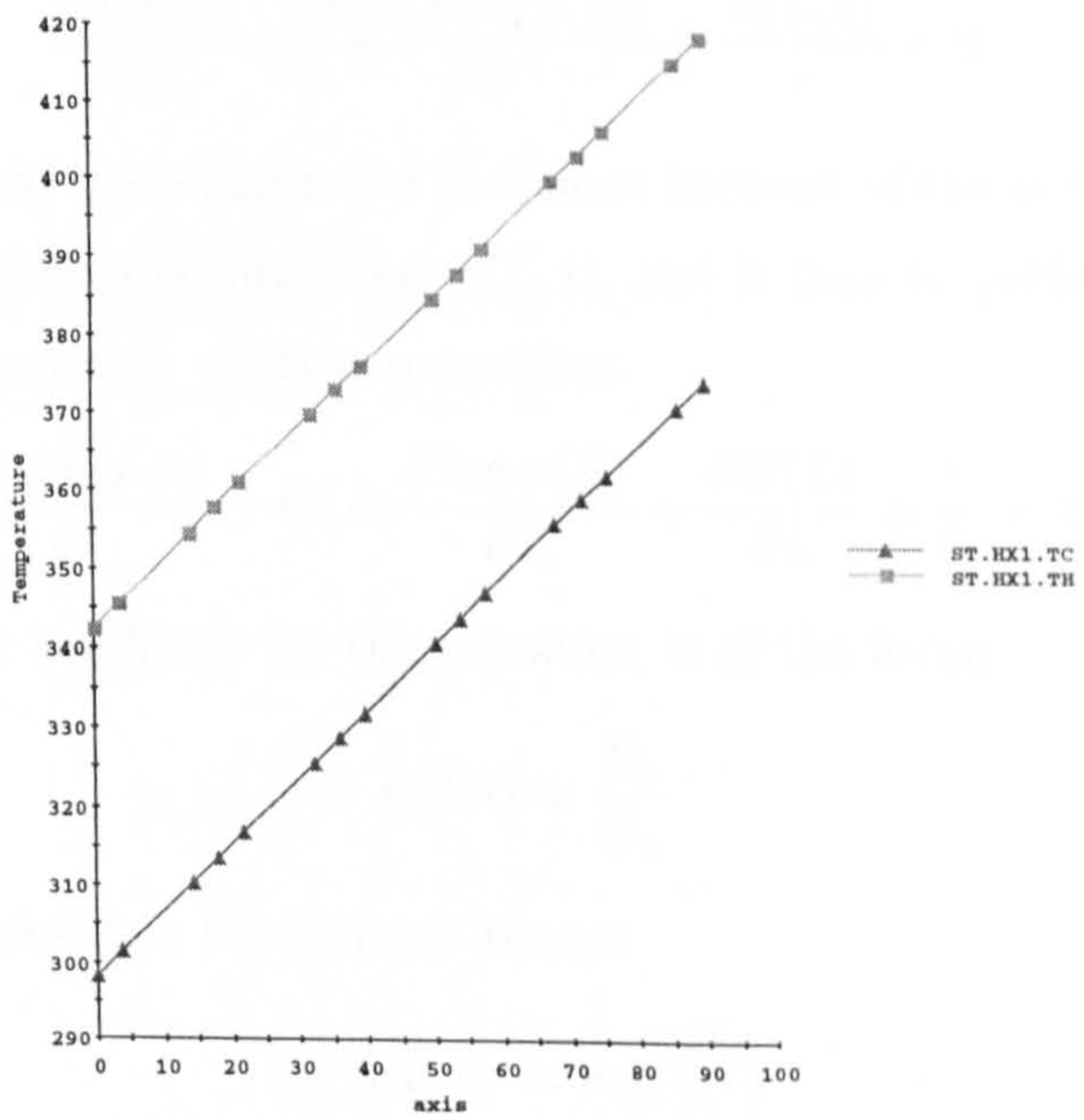


Figure 7.19: Temperature on hot and cold side of recovery exchanger in section 7.4

7.5 CMSMPR Crystalliser

We consider a continuous mixed suspension mixed product removal (CMSMPR) crystallisation unit operating in cooling mode to produce potassium sulphate crystals from aqueous solutions. The temperature in the crystalliser is assumed to be constant and any crystal breakage is ignored. A mass balance on the solute in both the liquid and the solid phase yields the equation:

$$V\rho\frac{\partial\tilde{c}}{\partial t} = W(c_{\text{in}} - \tilde{c}) \tag{7.55}$$

where V is the volume of crystalliser, ρ the solvent density, W the mass flow rate of liquid through crystalliser and $\tilde{c}$ denotes the *combined* solid and liquid phase concentration of the solute, defined as:

$$\tilde{c} = c + \frac{M_{\text{t}}}{\rho} \tag{7.56}$$

The magma density M_{t} is given by:

$$M_{\text{t}} = \rho_{\text{s}} \int_0^\infty K_{\text{v}}(L) L^3 n(L,t) dL \tag{7.57}$$

where ρ_{s} is the solid density.

A population balance on the crystals yields the partial differential equation

$$\frac{\partial n(L,t)}{\partial t} + \frac{\partial(G(L)n(L,t))}{\partial L} + \frac{n(L,t)}{\tau} = 0 \tag{7.58}$$

In fact, this equation may pose numerical problems because of the potentially large magnitude of the crystal number concentrations $n(L,t)$, and it may be preferable to express it in terms of the natural logarithm of these quantities:

$$\frac{\partial \log n(L,t)}{\partial t} + G(L)\frac{\partial \log n(L)}{\partial L} + \frac{\partial G(L)}{\partial L} + \frac{1}{\tau} = 0 \tag{7.58'}$$

The boundary condition for this equation is of the form:

$$n(0,t) = \frac{B_o}{G_o} \tag{7.59}$$

which can also be expressed in logarithmic terms:

$$\log n(0,t) = \log(\frac{B_o}{G_o}) \tag{7.59'}$$

The residence time τ in the system is related to the mass flowrate W through

$$\tau = \frac{V\rho}{W} \tag{7.60}$$

Parameter	Value	Units
c_{in}	0.15	*kg solid/kg solvent*
V	0.05	m^3
W	100	kg/hr
T	25	$\deg C$
ρ	1000	kg/m^3
ρ_s	2660	kg/m^3

Table 7.5: Parameter values in section 7.5

The crystal growth rate $G(L)$, the nucleation rate B_o and the nuclei growth rate G_o are all functions of the relative supersaturation σ of the solution given by

$$\sigma = \max(c - c_{eq}, 0) \tag{7.61}$$

where the equilibrium saturation concentration of the solute, c_{eq}, is generally a function of temperature. For the purposes of our model, we use the following correlations obtained by Chianese *at al.* (1987) from experimental data on the crystallisation of potassium sulphate from aqueous solutions:

$$G(L) = 892\sigma^2(1 + 5.87L) \tag{7.62}$$

$$B_o = 4.12 \times 10^{14} M_t \sigma^{3.4} \tag{7.63}$$

$$G_o = 847\sigma^2 \tag{7.64}$$

where L (mm) is the crystal size, G (mm/hr) the crystal growth rate, M_t (kg/m^3) the magma density, σ (kg/kg) the relative supersaturation and B_o $(n/m^3 \cdot hr)$ the nucleation rate.

The crystal volumetric shape factor, $K_v(L)$ is given by the following discontinuous correlation (Budz *et al.*, 1987):

$$K_v(L) = \begin{cases} 0.898e^{0.168\sqrt{1000L} - 8.234L} & \text{for } L \leq 0.1mm \\ 4.460e^{-0.0797\sqrt{1000L} + 6.76\times10^{-1}} & \text{for } L > 0.1mm \end{cases}$$

Finally the saturation concentration of potassium sulphate in water is taken as (Perry and Chilton, 1973)

$$c_{eq} = 0.0735 + 1.675 \times 10^{-3} T \tag{7.65}$$

where C_{eq} (kg/kg) is the equilibrium solute concentration and T (K) the temperature.

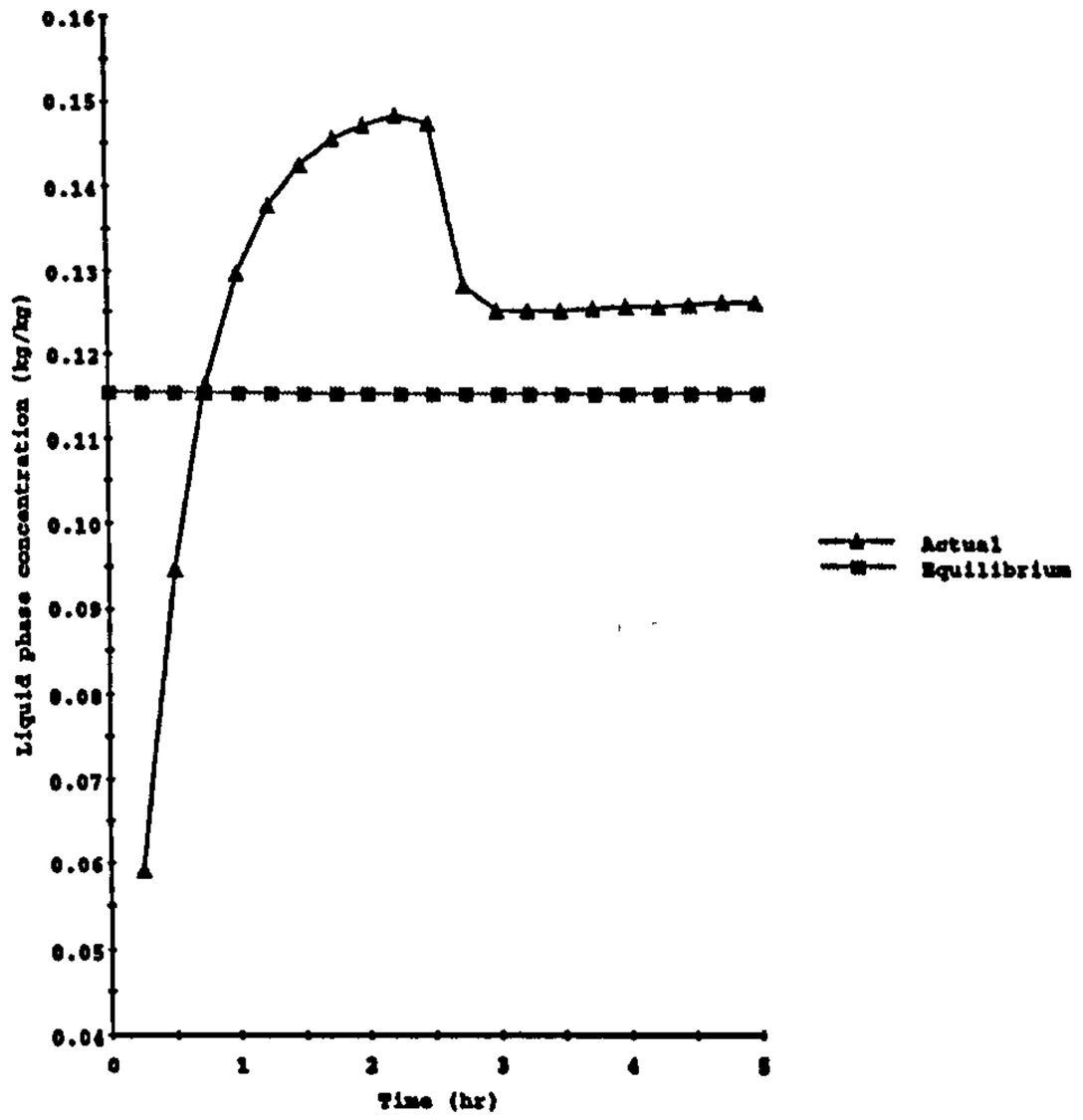


Figure 7.20: Variation of liquid phase solute concentration with time in section 7.5

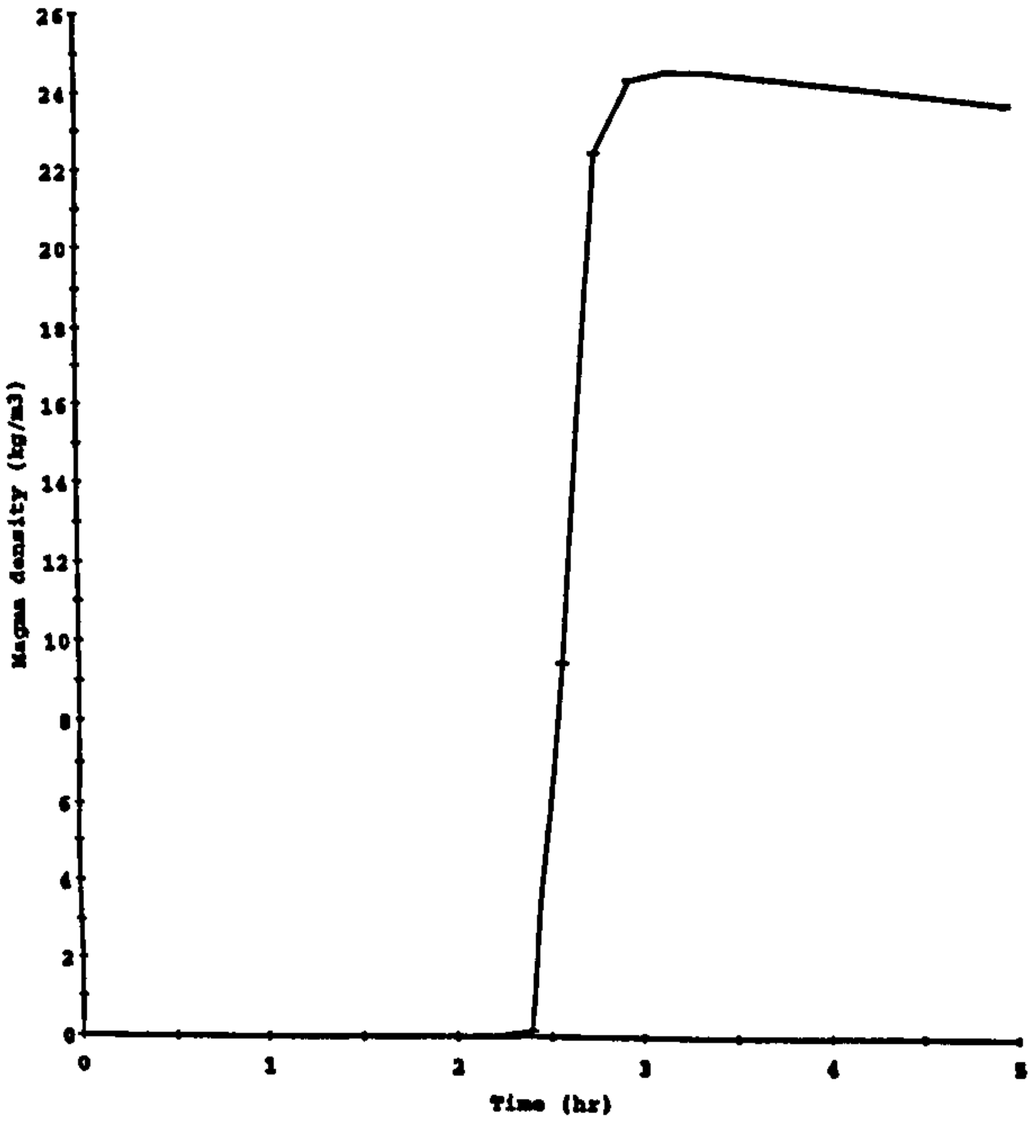


Figure 7.21: Variation of magma density in section 7.5

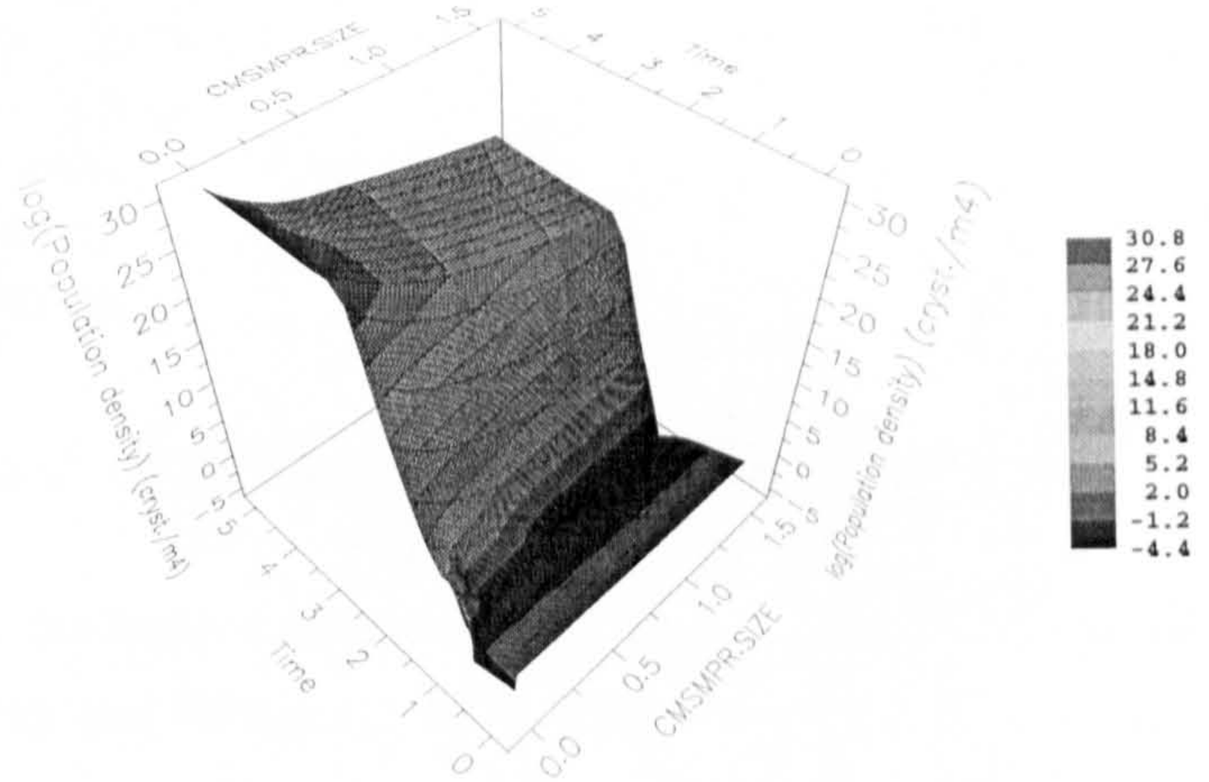


Figure 7.22: Variation of crystal population density (in natural logarithm terms) in section 7.5

Overall, the model of the crystalliser comprises the partial differential equation 7.58, the ordinary differential equation 7.55, the integral equation 7.57 and several algebraic relations. It is worth noting that equation 7.61 introduces an implicit discontinuity in the system as, in general, it is not known a *priori* if and when the liquid will reach its saturation point.

The values of the various parameters used for the simulation are shown in Table 7.5. Crystals of sizes up to 1.5 mm (corresponding to parameter `MaxSize` in the *gPROMS* model listed in Appendix B.5) were considered. A first-order backward finite difference method with 151 nodes was specified for the discretisation of the crystal size domain.

The dynamic simulation considers the start-up of the crystalliser, with the unit initially being full of pure water. Thus the initial liquid concentration of solute in the unit is set to zero, and so is the logarithmic number density for all non-zero crystal sizes. The latter specification actually corresponds to a population density of 1 crystal/m^4, but this is equivalent to zero for all practical purposes.

The results of the simulation are shown in Figures 7.20-7.22. Figure 7.20 shows the actual liquid phase concentration, together with the equilibrium value. The system stays below the saturation line for the first 0.735 hr of its operation. Even after crossing

Example	Process	Model type	Highest order partial derivative	Integral	Dimension
1	mixing, reaction and absorption	dynamic	2		1
2	catalytic reaction	dynamic	2		2
3	pressure swing adsorption	dynamic (fully discretised)	2		2
4	sterilisation	steady-state	2	√	2
5	crystallisation	dynamic	1	√	1

Table 7.6: Summary of examples illustrated in chapter 7

the saturation line, the liquid phase concentration continues to rise towards the feed value (0.15 kg/kg) until the rate of crystallisation increases sufficiently to reverse this trend. This is also evident in Figure 7.21 as the point at which the magma density starts becoming appreciable.

Figure 7.22 shows the variation of crystal population density (in natural logarithm terms) with respect to crystal size and time.

7.6 Concluding Remarks

In this chapter we have demonstrated the modelling and simulation of combined lumped and distributed processes in the *gPROMS* package using five examples representing typical chemical and biochemical processes. These are summarised in Table 7.6. Each example has its own characteristics and sufficient complexity to test various aspects of the capability and the usefulness of the package developed.

A comparison of the *gPROMS* input files in Appendix B with the models presented here will demonstrate that the model description in the *gPROMS* environment is straightforward since the syntax of the language follows standard mathematical convention. Multi-dimensional problems can easily be expressed in terms of cartesian products of distribution domains declared in `DISTRIBUTION_DOMAIN` subsection. Although the examples shown here involved up to 2 such domains, there is no limitation (other than increasing computational cost) to the number of dimensions that can be handled by the package.

The separation of the numerical solution from the underlying mathematical models is clearly beneficial from both the user's and the *gPROMS* developer's point of view. The user can quickly apply and test different numerical methods to the same model without any

change to the latter. On the other hand, from the developer's point of view, it is relatively easy to add new numerical methods as this activity can be pursued independently of the characteristics of the individual problem.

gPROMS currently provides six different built-in numerical methods for IPDAEs as discussed in chapter 5. As the examples shows, a variety of problems described in terms of IPDAEs can be handled with the package using these methods.

Finally, it is important to note that all the facilities for handling distributed parameter systems have been completely integrated within the structure of *gPROMS*, thus preserving all its other capabilities such as the ability to construct models hierarchically (see examples 1, 2, and 4) and to handle implicit discontinuities (example 5).

Chapter 8

Conclusions

The range of what is both desirable and practically feasible in process modelling has been expanding significantly in recent years. This trend, which is partly due to the realisation of the potential benefits of increased modelling realism and partly to rapid advances in computer hardware and numerical software, has provided the impetus for the work described in the present thesis.

In reality, spatial variations occur in *all* physical systems to a lesser or greater extent, and increased realism dictates that some account be taken of them. With earlier generations of process modelling packages, it was often considered both necessary and appropriate to use rather simple models, often based on some lumped approximation of the distributed nature of the real system. This was particularly true for the purposes of studying the behaviour of integrated processes involving several interacting unit operations. When more detailed models were deemed necessary, these were usually developed and studied in isolation, using different types of software, such as the CFD tools discussed in section 2.4.

This work aimed to produce a new generation process modelling tool that could be used directly for the construction and solution of general distributed unit operation models, both in isolation and within integrated processes. In the next section, we summarise the key aspects of our work as presented in this thesis. Then in the subsequent section, we consider some of the limitations of the present work, and conclude with some ideas and comments on potential avenues for further progress in this area.

8.1 Summary of this Work

The basis of this work is a formal mathematical modelling language for process systems described by mixed sets of integral, partial and ordinary differential, and algebraic equations (IPDAEs) subject to general boundary conditions. The three main concepts of the language are those of distribution domains and variable and equation distributions. The language imposes no limitation on the number of distribution domains over which a variable and/or equation may be distributed, nor does it assume the same degree of distribution for all such entities within the same model. Moreover, the introduction and use of the operators

PARTIAL and INTEGRAL allows the representation of partial derivatives and integrals of any order.

The solution of general sets of IPDAEs is an important aspect of the modelling and simulation of distributed systems. For the purposes of this work, we have adopted a method-of-lines approach which discretises all non-temporal variations, thereby reducing the IPDAEs to mixed sets of ordinary differential and algebraic equations (DAEs) to be integrated over time.

Discretisation methods based on both finite differences and orthogonal collocation on finite elements were developed. The general derivation of both of these classes of method using interpolating polynomials led to consistent approximations for first and second-order partial derivatives and integrals, and indicated generally-applicable ways for handling various implementation issues.

All of the above ideas have been incorporated within *gPROMS*, an existing process modelling tool. Significant attention has been paid to the efficient implementation of the new concepts while preserving all the existing capabilities of *gPROMS* such as its ability to describe processes with combined discrete and continuous characteristics of arbitrary complexity. Mechanisms for handling the large volumes of complex data potentially produced by dynamic simulations of combined lumped and distributed processes have also been devised and implemented.

Five examples demonstrating the application of the resulting tool to the modelling and simulation of chemical and biochemical processes have been presented. The examples have been selected to encompass a wide variety of mathematical models with respect to the numbers of distribution domains and the type of equations and boundary conditions, and range from single unit operations to integrated processes. To the author's knowledge, no other process modelling tool of comparable generality has been reported to date.

8.2 Needs and Directions for Future Work

Despite the capabilities of the extended *gPROMS* package, it is clear that further work is required in a number of areas. We consider some of them below:

8.2.1 Improved geometry description

The current package is restricted to modelling systems distributed over a number N of dimensions, the co-ordinate x_i in each dimension $i = 1..N$ being free to vary within a

fixed range $[x_i^L, x_i^U]$. This covers most common regular geometries including the cartesian, cylindrical and spherical ones.

One disadvantage of the definition of the system geometry as the cartesian product of a number of distribution domains is the fact that the precise form of the equations (in particular, the partial derivatives and integrals in them) depends on the overall shape of the domain. For instance, the conductive component in an energy conservation equation must be written as

$$k\left(\frac{\partial^2 T}{\partial x^2}+\frac{\partial^2 T}{\partial z^2}+\frac{\partial^2 T}{\partial z^2}\right)$$

in cartesian co-ordinates, while in cylindrical co-ordinates, it could be

$$k\left[\frac{1}{r}\frac{\partial}{\partial r}\left(r\frac{\partial T}{\partial r}\right)+\frac{\partial^2 T}{\partial z^2}\right]$$

Moreover, the number of conductive terms to be included in the equation would depend on whether there is variation with respect to other co-ordinates (*e.g.* θ in the cylindrical one).

One alternative which avoids the need for multiple models describing essentially the same physics is to define the equations in terms of general operators such as ∇ and ∇^2. The geometry could then be specified as an additional parameter directly in terms of the shape of the domain (*e.g.* `CARTESIAN`, `CYLINDRICAL`, `SPHERICAL`), its size and assumptions regarding the variation of properties actually encountered in the system. Since *gPROMS* already allows deferred specification of any model parameter, the details of the geometry could be omitted from low-level models and only be specified in higher-level ones or in the simulation experiment (`PROCESS`) itself.

Perhaps a more serious deficiency of the current package concerns the description and handling of irregular geometries. This is, fortunately, an area where much work has been reported elsewhere. Thus, for the simpler irregular geometries, an approach such as that adopted by ELLPACK (see section 2.3.2) might be adequate. On the other hand, the graphical tools incorporated in many CFD packages (see section 2.4) seem more appropriate. In any case, the discretisation grid could be generated automatically (see, for instance, Thompson (1988)).

An interesting question in this context regards the distinction between process modelling tools such as *gPROMS* on one hand and CFD packages on the other. Although this distinction was fairly clear in the past, with the developments described in this thesis there is already some overlap in capability between the two classes of software. If process modelling tools evolve further in the directions outlined above, this overlap will become much more significant.

8.2.2 Problem detection and diagnosis

Process modelling tools afford the user considerable flexibility both in model construction and in the specification of degrees of freedom and initial conditions. The possibility therefore exists for the definition of problems which are either badly posed (in the sense that they do not possess a well-defined mathematical solution), or impossible or difficult to solve using the current state of solution techniques.

The detection and diagnosis of badly posed problems is already a difficult task for systems of nonlinear algebraic equations and mixed ordinary differential and algebraic equations. For the IPDAE systems of interest to the present work, this task is much more complex. Given the lack of the necessary mathematical framework, we probably have to rely mainly on analysis performed at the level of the DAE system resulting from discretisation, checking for instance, the well-posedness of the system and its initial conditions, and the index of the DAE system. An additional complication in this context is that, since the discretisation is carried out automatically, the user is not necessarily aware of the form (or even the existence) of the discretised equations. The results of any such analysis would therefore have to be expressed in terms of the original IPDAE system for presentation to the user.

8.2.2.1 Control of discretisation error

One limitation of the numerical methods currently implemented in *gPROMS* for the discretisation of variations over non-temporal domains is the fact that they make no attempt at estimating and controlling the error incurred from such discretisation. Instead, they rely entirely on the user to select an appropriate method and grid. The additional restriction of the latter being uniform and fixed in time may have severe implications regarding the efficiency of solving some classes of problem, in particular those exhibiting sharp moving fronts.

The specification of non-uniform but fixed grids has already been considered in section 3.7.2. More radically, moving or adaptive grid techniques, of the type discussed in section 2.2.6 could be used to estimate the space discretisation error and adjust the grid accordingly. Most of the effective demonstrations of this approach (Pipilis, 1990) have been limited to one-dimensional problems. However, this may not be a major deficiency as many multidimensional problems exhibit sharp fronts in only one dimension. In such cases, the use of the moving front technique could be limited to this one dimension, with fixed grid discretisation being applied to the other dimensions.

8.2.2.2 Results storage

The powerful modelling capabilities of the extended *gPROMS* package together with ever increasing computational speeds imply that the software is used to build and study increasingly complex problems involving variations over relatively large numbers of dimensions. Despite the clear benefits of the developments described in chapter 6, it is already clear that much more needs to be achieved in reducing further the volume of results being stored without reducing the amount of information.

The collection and storing results at a *fixed* frequency over long time horizons may be particularly wasteful. We note that the DAE integration packages used for solving the discretised IPDAE system utilise sophisticated step-length adjustment algorithms for traversing a given time horizon in the least number of steps. Moreover, it is possible to reconstruct the entire solution trajectory from the values of the variables at these steps together with the order of integration at each step. It may therefore be advantageous to store results only at the steps used internally by the integrator. This will have the desirable side-effect of shifting some of the computational overhead (that associated with interpolating the results of the integration to produce the values of the variables at the fixed reporting intervals) from the integrator to the results visualisation system (in this case, *gRMS*).

The use of data compression techniques (see, for example, Hale and Sellars (1981)) is another possible means of reducing the volume of stored results.

References

W. F. Ames. *Numerical Methods for Partial Differential Equations.* Academic Press, New York, 1992.

D. A. Anderson, J. C. Tannehill, and R. H. Pletcher. *Computational Fluid Mechanics and Heat Transfer.* McGraw-Hill, New York, 1992.

P. M. Armenante and M. A. Leskowicz. Design of continuous sterilization systems for fermentation media containing suspended solids. *Biotechnol. Prog.*, 6(4):292–306, 1990.

D. C. Arney and J. E. Flaherty. An adaptive mesh-moving and local refinement method for time-dependent partial differential equations. *ACM Trans. Math. Soft.*, 16(1):48–71, 1990.

M. J. Baines. Moving element methods for time dependent problems. In K. W. Morton and M. J. Baines, editors, *Numerical Methods for Fluid Dynamics—III*, pages 513–518. Clarendon Press, Oxford, U.K., 1988.

C. T. H. Baker. *Numerical Treatment of Integral Equations.* Clarendon Press, Oxford, 1977.

P. I. Barton and C. C. Pantelides. The modelling of combined discrete/continuous processes. *AIChE J.*, 40:966–979, 1994.

P. I. Barton. *The modelling and simulation of combined discrete/continuous processes.* PhD thesis, University of London, 1992.

T. Björkholm. *gRMS* program documentation. Technical report, Centre for Process Systems Engineering, Imperial College, 1994.

J. Budz, A. G. Jones, and J. W. Mullin. On the shape-size dependence of potassium sulfate crystals. *Ind. Eng. Chem. Res.*, 26:820–824, 1987.

G. F. Carey and B. A. Finlayson. Orthogonal collocation on finite elements. *Chem. Engng Sci.*, 30:587–596, 1975.

M. B. Carver and H. W. Hinds. The method of lines and the advective equation. *Simulation*, pages 59–69, August 1978.

M. B. Carver and W. E. Schiesser. Biased upwind difference approximations for first-order hyperbolic (convective) partial differential equations. *AIChE Annual Meeting*, November 1980.

M. B. Carver. Method of lines solution of differential equations — fundamental principles and recent extensions. In R. S. H. Mah and W. D. Seider, editors, *Foundations of Computer-Aided Process Design*, pages 369–402. Engineering Foundation, New York, New Hampshire, 1981.

CHAM Ltd., Wimbledon, U.K. *The PHOENICS beginner's guide Release 1.4*, 1987.

A. Chianese, S. di Cave, and B. Mazzarotta. Crystallization kinetics of potassium sulphate in a MSMPR cooling crystallizer. *Ing. Chim. Ital.*, 26:28–38, 1987.

J. Crank. *Free and Moving Boundary Problems.* Clarendon Press, Oxford, 1984.

C. Donnelly and R. Stallman. Bison Version v1.20: The Yacc – compatible Parser Generator. Technical report, Free Software Foundation, 1992.

N. B. Ferguson and B. A. Finlayson. Transient chemical reaction analysis by orthogonal collocation. *Chemial Engineering Journal*, 1:327–336, 1970.

B. A. Finlayson. Packed bed reactor analysis by orthogonal collocation. *Chem. Engng Sci.*, 26:1081–1091, 1971.

B. A. Finlayson. *Nonlinear Analysis in Chemical Engineering.* McGraw-Hill, New York, 1980.

B. A. Finlayson. *Numerical Methods for Problems with Moving Fronts.* Ravenna Park Publishing, Inc, 1992.

C. N. Fischer and R. J. LeBlanc. *Crafting a Compiler.* Benjamin/Cummings, 1988.

C. A. Fletcher. *Computational Techniques for Fluid Dynamics I – Fundamental and General Techniques.* Springer Verlag, New York, 1988.

G. F. Froment and K. B. Bischoff. *Chemical Reactor Analysis and Design.* John Wiley and Sons, New York, 1990.

J. C. Hale and H. L. Sellars. Historical data recording for process computers. *Chem. Engng Prog.*, November:38–43, 1981.

J. C. Heydweiller, R. F. Sincovec, and L. T. Fan. Dynamic simulation of chemical processes described by distributed and lumped parameter models. *Computers chem. Engng*, 1:125–131, 1977.

D. M. Himmelblau and K. B. Bischoff. *Process Analysis and Simulation.* John Wiley and Sons, New York, 1968.

A. C. Hindmarsh. Preliminary documentation of GEARIB: Solution of implicit systems of ordinary differential equations with banded jacobian. Technical report, Lawrence Livermore Lab., Livermore, Calif., 1976.

N. Y. Hussaini and T. A. Zang. Spectral methods in fluid dynamics. *Ann. Rev. Fluid. Mech.*, 19:339–367, 1987.

R. B. Jarvis and C. C. Pantelides. DASOLV — a differential-algebraic equation solver. Technical report, Centre for Process Systems Engineering, Imperial College, 1992.

R. B. Jarvis. *Robust dynamic simulation of chemical engineering processes.* PhD thesis, University of London, 1993.

S. C. Johnson. Yacc - yet another compiler compiler. c.s.#32. Technical report, Bell Telephone Laboratories, 1975.

P. Keast and P. H. Muir. EPDCOL : A more efficient PDECOL code. *ACM Trans. Math. Soft.*, 17:153–166, 1991.

KL Group Inc., Toronto, Ontario, Canada. *XRT/3d for Motif: Programmer's guide & reference manual, version 2.0*, 1993a.

KL Group Inc., Toronto, Ontario, Canada. *XRT/graph for Motif: Widget programmer's guide & reference manual, version 2.3*, 1993b.

A. Kroner, P. Holl, W. Marquardt, and E. D. Gilles. DIVA — An open architecture for dynamic simulation. *Computers chem. Engng*, 14:1289–1295, 1990.

D. P. Laurie. Basic principles of discretization methods. In D.P.Laurie, editor, *Numerical Solution of Partial Differential Equations : Theory, Tools and Case Studies.* ISNM 66, Basel, Switzerland, 1983.

H. W. Lee and E. G. Salleeby. Mathematical behaviour of the population balance of a MSMPRC with agglomeration, feed and size-dependent growth rate. *Computers chem. Engng*, 18:899–907, 1994.

M. Machura and R. A. Sweet. A survey of software for partial differential equations. *ACM Trans. Math. Soft.*, 6:461–488, 1980.

N. K. Madsen and R. F. Sincovec. PDECOL, general collocation software for partial differential equations. *ACM Trans. Math. Soft.*, 5:326–351, 1980.

W. Marquardt. Dynamic process simulation - recent progress and future challenges. In *Proceedings CPC IV*, Texas, USA, 1991.

S. E. Mattsson and M. Andersson. OMOLA — An object-oriented modeling language. In M. Jamshidi and C. J. Herget, editors, *Recent Advances in Computer Aided Control Systems Engineering*. Elsevier, New York, 1992.

K. Miller and R. N. Miller. Moving finite elements. I. *SIAM J. Numer. Anal.*, 18:1019–1032, 1981.

K. Miller. Moving finite elements. II. *SIAM J. Numer. Anal.*, 18:1033–1057, 1981.

S. Nilchan and C. C. Pantelides. Modelling, simulation and optimisation of periodic adsorption processes. In Proc. IChemE Research Event, London, U.K., 1994.

C. C. Pantelides and P. I. Barton. Equation-oriented dynamic simulation : current status and future perspectives. *Computers chem. Engng*, 17S:263–285, 1993.

C. C. Pantelides, D. Gritsis, K. R. Morison, and R. W. H. Sargent. The mathematical modelling of transient systems using differential-algebraic equations. *Computers chem. Engng*, 12:449–454, 1988.

C. C. Pantelides. SpeedUp — recent advances in process simulation. *Computers chem. Engng*, 12:745–755, 1988.

S. V. Patankar. *Numerical Heat Transfer and Fluid Flow.* McGraw-Hill, New York, 1980.

R. H. Perry and C. H. Chilton. *Chemical Engineer's Handbook, 5th edition.* McGraw-Hill, New York, 1973.

L. R. Petzold. A description of DASSL: a differential/algebraic system solver. In *Proceedings IMACS World Congress, Montreal, Canada, August, 1982*, pages 65–68, 1982.

B. M. Pfeiffer and W. Marquardt. Symbolic semi-discretization of partial differential equation systems. In *Proc. Sc'93, International IMACS Symposium on Symbolic Computation*, Vileneuve d'Ascq, France, February 23–25, 1993.

P. C. Piela, T. G. Epperly, K. M. Westerberg, and A. W. Westerberg. ASCEND: An object-oriented computer environment for modeling and analysis: the modeling language. *Computers chem. Engng*, 15:53–72, 1991.

K. G. Pipilis. *Higher order moving finite element methods systems described by partial differential-algebraic equations.* PhD thesis, University of London, 1990.

D. Ramkrishna. The status of population balances. *Rev. in Chem. Engng*, 3:49–90, 1985.

J. R. Rice and R. F. Boisvert. *Solving Elliptic Problems Using ELLPACK.* Springer Verlag, New York, 1985.

N. J. Samsatli. A review of media sterilisation in biochemical processes. Technical report, Centre for Process Systems Engineering, Imperial College, 1994.

W. E. Schiesser. *The Numerical Method of Lines.* Academic Press, New York, 1991.

W. Schönauer and E. Schnepf. Software considerations for the black box solver FIDISOL for partial differential equations. *ACM Trans. Math. Soft.*, 13:333–349, 1987.

W. Schönauer, K. Routh, and G. Glotz. The SLDGL program package for the self-adaptive solution of nonlinear systems of elliptic and parabolic pde's. In R. Vichnevetsky and R. S. Stepleman, editors, *In Advances in Computational Methods in Partial Differential Equations IV*, pages 117–125. IMACS, 1981.

SPDE, Inc, Arlington, Massachusetts. *PDEase reference manual release2.4*, 1993.

G. Strang and G. J. Fox. *An Analysis of the Finite Element Method.* Prentice-Hall, 1973.

J. F. Thompson. Some current trends in numerical grid generation. In K. W. Morton and M. J. Baines, editors, *Numerical Methods for Fluid Dynamics — III*, pages 87–100. Clarendon Press, Oxford, U.K., 1988.

P. H. Turnock and R. H. Kadlec. Separation of nitrogen and methan via periodic adsorption. *AIChE J.*, 17:335–342, 1971.

J. G. Verwer and R. A. Trombert. Analysis of an adaptive finite-difference method for time dependent pdes. In D. F. Griffiths and G. A. Watson, editors, *Numerical Analysis 1 — Proceedings of 14th Dundee Conference*, pages 267–284. Dundee, U.K., 1991.

J. G. Verwer, J. G. Blom, R. M. Furzeland, and P. A. Zegeling. A moving grid method for one-dimensional pdes based on the method of lines. In J. E. Flaherty, P. J. Paslow, M. S. Shephard, and Vasilakis J. D, editors, *Adaptive Methods for Partial Differential Equations*, pages 160–175. Clarendon Press, Oxford, 1989.

J. Villadsen and M. L. Michelsen. *Solution of Differential Equation Models by Polynomial Approximation.* Prentice-Hall, 1978.

G. Wozny and L. Jeromin. Dynamic process simulation in industry. *Intl. chem. Engng*, 34:159–177, 1994.

O. C. Zienkiewicz and K. Morgan. *Finite Elements and Approximation.* John Wiley and Sons, New York, 1983.

Nomenclature

A	area (sections 7.2 and 7.3)
B_o	nucleation rate (section 7.5)
c	concentration (section 7.5)
$\tilde{c}$	combined solute concentration in both liquid and solid phase (section 7.5)
c_0	inlet concentration (section 7.2)
c_{eq}	equilibrium solute concentration (section 7.5)
c_{in}	solute concentration in feed stream (section 7.5)
c_i	concentration of component i (sections 7.2 and 7.3)
$\tilde{c}_i$	combined gas/solid molar concentration of component i (section 7.3)
c_{pf}	fluid heat capacity (section 7.2)
c_{ps}	solid heat capacity (section 7.4)
c_{pw}	heat capacity of coolant (section 7.2)
C	nutrient concentration (section 7.4)
C_0	nutrient concentration in liquid inlet (section 7.4)
C_{L}	nutrient concentration in liquid outlet (section 7.4)
deg	degradation of the liquid (section 7.4)
d_{p}	particle diameter (section 7.3)
D_r	diffusivities in radial direction (section 7.2)
D_z	diffusivities in axial direction (sections 7.2 and 7.3)
E_{d}	activation energy for the death of micro-organism (section 7.4)
$E_{\mathrm{d,nut}}$	activation energy for the degradation of nutrients (section 7.4)
f_{w}	mass flow rate of coolant (section 7.2)
G	crystal growth rate (section 7.5)
G_o	nuclei growth rate (section 7.5)
h	heat transfer coefficient between the solid particle and the liquid medium (section 7.4)
h_{w}	wall heat transfer coefficient (section 7.2)
k	reaction rate constant (section 7.2)
k_i^*	mass transfer coefficient of component i (section 7.3)
k_r	radial thermal conductivity ((section 7.2)
k_{s}	thermal conductivity of the solid particle (section 7.4)
k_z	axial thermal conductivity (section 7.2)

K_{d0} first-order death rate constants for micro-organisms (section 7.4)

$K_{d0,nut}$ first-order death rate constants for nutrients (section 7.4)

K_j constants, $j = 1..9$ (section 7.1)

K_v crystal volumetric shape factor (section 7.5)

L length of process equipment (section 7.4)
or crystal size (section 7.5)

L_a reactor length (section 7.2)
or column length (section 7.3)

L_h length of heat exchanger (section 7.4)

L_r reactor radius (section 7.2)

L_s length of steam heater (section 7.4)

L_t length of tube (section 7.4)

m_i adsorption isotherm gradient of component i (section 7.3)

$\dot{m}$ mass flow rate of liquid (section 7.4)

M_t magma density (section 7.5)

n crystal population density (section 7.5)

N cell concentration in liquid (section 7.4)

N_0 cell concentration in liquid inlet (section 7.4)

N_L cell concentration in liquid outlet (section 7.4)

N_s concentration of micro-organisms in solid particle (section 7.4)

P pressure (section 7.3)

P_{atm} atmospheric pressure (section 7.3)

Pe Peclet number

P_f feed pressure (section 7.3)

P_i partial pressure of component i (section 7.2)

P_w external pressure (section 7.3)

q heat transfer rate per unit length (section 7.4)

q_i solid phase concentration of component i (section 7.3)

q_i^* equilibrium adsorbed phase concentration of component i (section 7.3)

Q volumetric flow rate (section 7.3)

r radial coordinate

r rate of reaction (section 7.2)

R gas constant

R_p radius of particle (section 7.4)

st	sterility of the liquid (section 7.4)
st_s	sterility of the solid particle (section 7.4)
t	time coordinate
t_{cycle}	cycle time (section 7.3)
t_{press}	pressurisation time (section 7.3)
T	temperature
T_c	temperature of fluid in cold side (section 7.4)
T_{ci}	inlet temperature of coolant (section 7.2)
T_{cw}	coolant temperature in cooling tank (section 7.2)
T_{fi}	inlet temperature of fluid (section 7.2)
T_h	temperature of fluid in hot side (section 7.4)
T_s	solid temperature (section 7.4)
T_{st}	saturated steam temperature (section 7.4)
T_w	temperature of reactor wall (section 7.2)
U	overall heat transfer coefficient (sections 7.2 and 7.4)
v	average velocity of the liquid (section 7.4)
v_z	superficial velocity (sections 7.2 and 7.3)
V	volume of crystalliser (section 7.5)
V_w	volume of cooling jacket (section 7.2)
$\dot{v}$	volumetric flow rate (section 7.4)
w_{ex}	width of the spiral heat exchanger (section 7.4)
W	mass flowrate of liquid through crystalliser (section 7.5)
x	vector of independent variables
y_i	mole fraction of component i in the feed (section 7.3)
z	axial coordinate

Greek Symbols

α	constant (sections 7.2 and 7.3)
α_z	turbulent axial thermal diffusivity (section 7.4)
β	constant (section 7.2)
δ_{ex}	clearance between the plates of the heat exchanger (section 7.4)
ΔH	heat of reaction (section 7.2)
ε	voidage fraction (sections 7.2 and 7.4)
ε_b	interparticle void fraction (section 7.3)

ε_p intraparticle void fraction (section 7.3)

ε_t total void fraction (section 7.3)

ζ dimensionless axial coordinate for absorber (section 7.1)

θ_i dimensionless concentration of component i in liquid phase of absorber (section 7.1)

θ_i^e dimensionless concentration of component i in liquid phase of absorber exit (section 7.1)

λ dimensionless axial coordinate for reactor (section 7.1)

μ gas viscosity (section 7.3)

ν_i stoichiometric coefficient of component i in reaction (sections 7.1 and 7.2)

ρ density

ρ_b bulk density (sections 7.2 and 7.3)

ρ_f fluid density (section 7.2)

ρ_s solid density (section 7.4)

ρ_w coolant density (section 7.2)

σ relative supersaturation (section 7.5)

τ dimensionless time (section 7.1)
or residence time (section 7.5)

ϕ vector of distributed variables

ϕ_i dimensionless concentration of component i in reactor (section 7.1)

ϕ_i^e dimensionless concentration of component i at reactor exit (section 7.1)

ϕ_i^f dimensionless concentration of component i in feed stream (section 7.1)

ϕ_i^m dimensionless concentration of component i in completely mixing tank (section 7.1)

ψ_i dimensionless concentration of component i in gas phase of absorber (section 7.1)

ψ_i^e dimensionless concentration of component i at absorber exit (section 7.1)

Appendix A

The *gPROMS* Modelling Language

Below we provide a formal definition of the *gPROMS* modelling language in terms of a *YACC* input file. The language listed is consistent with version 1.4b of the package (9/11/94) and supersedes that provided by Barton (1992). The semantic routines associated with the various syntactic constructs have been omitted.

```
%token StringNQ          1 ILITERAL       2 RLITERAL    3 STRLITERAL  4
%token '^'               5 '+'            6 '-'         7 '*'         8 '/'    9
%token '$'              10 ';'           11 '='        12 '('        13 ')'   14
%token ASGN             15 ':'           16 ','        17 '.'        18 '>'   19
%token '<'              20 GTEQ          21 LTEQ       22 NTEQ       23 '['   24
%token ']'              25 BARP          26 BARM       27 '|'        28 '%'   29
%token ABSOLUTEACCURACY          30
%token ADD                       31
%token AND                       32
%token ARRAY                     33
%token AS                        34
%token ASSIGN                    35
%token BFDM                      36
%token BOUNDARY                  37
%token CARTESIAN                 38
%token CASE                      39
%token CFDM                      40
%token CONNECTION                41
%token CONTINUE                  42
%token CYLINDRICAL               43
%token DECLARE                   44
%token DEFAULT                   45
%token DEL                       46
%token DISTRIBUTION              47
%token DISTRIBUTION_DOMAIN       48
%token DIV                       49
%token DO                        50
%token EFFECTIVEZERO             51
%token ELSE                      52
%token END                       53
%token EQUATION                  54
%token FALSE                     55
%token FDPERTURBATION            56
%token FFDM                      57
%token FILE                      58
%token FOR                       59
%token FREQUENCY                 60
%token GAUSSIAN                  61
%token GEOMETRY                  62
%token GET                       63
%token GLOBAL_TIME               64
%token GPLOT                     65
```

```
%token GRMS                    66
%token IF                      67
%token INHERITS                68
%token INITACCURACY            69
%token INITIAL                 70
%token INTEGER                 71
%token INTEGER_EXPRESSION      72
%token INTEGRAL                73
%token IRREPRODUCIBLE          74
%token IS                      75
%token LAPLACIAN               76
%token LINEARISE               77
%token LOGICAL                 78
%token LOGICAL_EXPRESSION      79
%token MAXINITITERATIONS       80
%token MAXITERNOIMPROVE        81
%token MESSAGE                 82
%token MFDM                    83
%token MFEM                    84
%token MOD                     85
%token MODEL                   86
%token MONITOR                 87
%token NOT                     88
%token NSTEPREDUCTIONS         89
%token OCFEM                   90
%token OF                      91
%token OFF                     92
%token OLD                     93
%token ON                      94
%token OR                      95
%token OUTPUTDESTINATION       96
%token OUTPUTLEVEL             97
%token PARALLEL                98
%token PARAMETER               99
%token PARTIAL                 100
%token PAUSE                   101
%token PRESET                  102
%token PROCESS                 103
%token PSFACTOR                104
%token RANDOM_STREAM           105
%token REAL                    106
%token REAL_EXPRESSION         107
%token REINITIAL               108
%token RELATIVEACCURACY        109
%token REMOVE                  110
%token REPLACE                 111
%token REPORTINGINTERVAL       112
%token RESET                   113
%token SCHEDULE                114
%token SCREEN                  115
%token SEED                    116
%token SELECTOR                117
%token SEND                    118
%token SEQUENCE                119
%token SET                     120
%token SINGPERTFACTOR          121
%token SLRFACTOR               122
%token SOLUTIONPARAMETERS      123
```

```
%token SPHERICAL                124
%token STATEEVENTACCURACY       125
%token STATUS                   126
%token STEADY_STATE             127
%token STEP                     128
%token STOP                     129
%token STREAM                   130
%token SWITCH                   131
%token TASK                     132
%token TERMINATE                133
%token THEN                     134
%token TIME                     135
%token TIMEEVENTACCURACY        136
%token TO                       137
%token TRUE                     138
%token TYPE                     139
%token UFDM                     140
%token UNDEFINED                141
%token UNIT                     142
%token UNTIL                    143
%token VARIABLE                 144
%token WHEN                     145
%token WHILE                    146
%token WITH                     147
%token WITHIN                   148

%%
/*      GRAMMAR DEFINITION      */

Simulation      : Block
                | Simulation Block
                ;

/*      DEFINITION OF A BLOCK   */

Block           : DeclareBlock
                | ModelBlock
                | TaskBlock
                | ProcessBlock
                ;

/*      SYNTAX FOR A DECLARE BLOCK      */

DeclareBlock    : DECLARE VarTypeSection StrmTypeSection END
                ;

/*      SYNTAX FOR VARIABLE TYPE DECLARATIONS   */

VarTypeSection  : TYPE VarTypeDec
                | VarTypeSection VarTypeDec
                ;

VarTypeDec      : Identifier '=' Real ':' Real ':' Real OptionalUnits
                ;

OptionalUnits   : UNIT '=' Units
                | Empty
                ;
```

```
Units             : STRLITERAL
                  | StringNQ
                  ;

/*      SYNTAX FOR STREAM TYPE DECLARATIONS     */

StrmTypeSection : STREAM StrmTypeList
                  | Empty
                  ;

StrmTypeList      : StreamTypeDec
                  | StrmTypeList StreamTypeDec
                  ;

StreamTypeDec     : Identifier IS IdentifierList
                  ;


/*      SYNTAX FOR A MODEL TYPE DECLARATION     */

ModelBlock        : MODEL Identifier Parent
                    ModelEntities
                    SetSection BoundarySection EquationSection
                    END
                  ;

Parent            : Empty
                  ;

ModelEntities     : EntityList
                  | Empty
                  ;

EntityList        : ModelAttributes
                  | EntityList ModelAttributes
                  ;

ModelAttributes : ParameterSection
                  | DomainSection
                  | UnitSection
                  | VariableSection
                  | StreamSection
                  | SelectorSection
                  ;


/*      PARAMETER SECTION       */

ParameterSection : PARAMETER ParameterList
                   ;

ParameterList    : ParameterDec
                 | ParameterList ParameterDec
                 ;

ParameterDec     : IdentifierList AS ParameterType
                 ;
```

```
ParameterType   : BasicParameter
                | ARRAY '(' ExpList ')' OF BasicParameter
                ;

BasicParameter  : BasicType DEFAULT Expression
                | BasicType
                | MODEL Identifier
                ;

BasicType       : REAL
                | INTEGER
                | LOGICAL
                ;

/*      DISTRIBUTION SECTION    */

DomainSection   : DISTRIBUTION_DOMAIN DomainList
                ;

DomainList      : DomainDec
                | DomainList DomainDec
                ;

DomainDec       : IdentifierList  AS DomainType
                ;

DomainType      : ARRAY '(' ExpList ')' OF BasicDomain
                | BasicDomain
                ;

BasicDomain     : BasicDomainType
                ;

BasicDomainType : '(' SimpleExp ':' SimpleExp ')'
                ;

/*      UNIT SECTION    */

UnitSection     : UNIT UnitList
                ;

UnitList        : UnitDec
                | UnitList UnitDec
                ;

UnitDec         : IdentifierList AS UnitType
                ;

UnitType        : BasicUnit
                | ARRAY '(' ExpList ')' OF BasicUnit
                ;

BasicUnit       : Identifier
                ;
```

```
/*      VARIABLE SECTION        */

VariableSection : VARIABLE VariableList
                ;

VariableList    : VariableDec
                | VariableList VariableDec
                ;

VariableDec     : IdentifierList AS VariableType
                ;

VariableType    : BasicVariable
                | ARRAY '(' ExpList ')' OF BasicVariable
                | DISTRIBUTION '(' ExpList ')' OF BasicVariable
                ;

BasicVariable   : Identifier
                ;

/*      STREAM SECTION  */

StreamSection   : STREAM StreamList
                ;

StreamList      : StreamDec
                | StreamList StreamDec
                ;

StreamDec       : Identifier IS PathName
                | Identifier ':' PathNameList AS StreamType
                ;

StreamType      : BasicStream
                | ARRAY '(' ExpList ')' OF BasicStream
                ;

BasicStream     : Identifier
                | CONNECTION
                ;

/*      SELECTOR SECTION        */

SelectorSection : SELECTOR SelectorList
                ;

SelectorList    : SelectorDec
                | SelectorList SelectorDec
                ;

SelectorDec     : IdentifierList AS SelectorType
                ;

SelectorType    : BasicSelector
                | ARRAY '(' ExpList ')' OF BasicSelector
                | DISTRIBUTION '(' ExpList ')' OF BasicSelector
                ;
```

```
BasicSelector   : '(' IdentifierList ')'
                | '(' IdentifierList ')' DEFAULT Identifier
                ;

/*      SET  SECTION    */

SetSection      : SET SetList
                | Empty
                ;

SetList         : SetEntity
                | SetList SetEntity
                ;

SetEntity       : PathName ASGN SetValues        ';'
                | PathName ASGN SolutionMethods ';'
                | SetWithin
                | SetFor
                ;

SetWithin       : WITHIN PathName DO SetList END
                ;

SetFor          : FOR Identifier ASGN SimpleExp TO SimpleExp OptStep DO
                  SetList END
                ;

SolutionMethods : '['  Method
                   ',' Expression
                   ',' Expression ']'
                ;

Method          : CFDM
                | BFDM
                | FFDM
                | UFDM
                | OCFEM
                ;

/*      BOUNDARY SECTION        */

BoundarySection : BOUNDARY EquationList
                | Empty
                ;

/*      EQUATION SECTION        */

EquationSection : EQUATION EquationList
                | Empty
                ;

/*      SYNTAX FOR A TASK TYPE DECLARATION      */

TaskBlock       : TASK Identifier
                  TaskParaSection TaskVarSection ScheduleSection END
                ;
```

```
TaskParaSection : PARAMETER TaskParaList
                | Empty
                ;

TaskParaList    : TaskParaDec
                | TaskParaList TaskParaDec
                ;

TaskParaDec     : IdentifierList AS TaskParaType
                ;

TaskParaType    : BasicParaType
                | MODEL Identifier
                ;

BasicParaType   : REAL
                | INTEGER
                | LOGICAL
                | REAL_EXPRESSION
                | INTEGER_EXPRESSION
                | LOGICAL_EXPRESSION
                ;

TaskVarSection  : VARIABLE TaskVarList
                | Empty
                ;

TaskVarList     : TaskVarDec
                | TaskVarList TaskVarDec
                ;

TaskVarDec      : IdentifierList AS TaskVarType
                ;

TaskVarType     : BasicVarType
                ;

BasicVarType    : REAL
                | INTEGER
                | LOGICAL
                ;

/*      SCHEDULE SECTION        */

ScheduleSection : SCHEDULE Schedule
                ;

/*      SYNTAX FOR A PROCESS TYPE DECLARATION   */

ProcessBlock    : PROCESS Identifier
                  ProcessEntities MonitorSection SetSection
                  EquationSection AssignSection PresetSection
                  IntSelector IntInitial
                  Options IntSchedule
                  END
                ;
```

```
Options              : SOLUTIONPARAMETERS  ExecutionOptions
                     | Empty
                     ;

ExecutionOptions     : ExecutionParameters
                     | ExecutionOptions  ExecutionParameters
                     ;

ExecutionParameters : ABSOLUTEACCURACY     ASGN  RealValue  Optcol
                     | EFFECTIVEZERO       ASGN  RealValue  Optcol
                     | FDPERTURBATION      ASGN  RealValue  Optcol
                     | MAXINITITERATIONS   ASGN  RealValue  Optcol
                     | MAXITERNOIMPROVE    ASGN  RealValue  Optcol
                     | NSTEPREDUCTIONS     ASGN  RealValue  Optcol
                     | OUTPUTDESTINATION   ASGN  Target     Optcol
                     | OUTPUTLEVEL         ASGN  RealValue  Optcol
                     | PSFACTOR            ASGN  RealValue  Optcol
                     | RELATIVEACCURACY    ASGN  RealValue  Optcol
                     | REPORTINGINTERVAL   ASGN  RealValue  Optcol
                     | SINGPERTFACTOR      ASGN  RealValue  Optcol
                     | SLRFACTOR           ASGN  RealValue  Optcol
                     | INITACCURACY        ASGN  RealValue  Optcol
                     | STATEEVENTACCURACY  ASGN  RealValue  Optcol
                     | TIMEEVENTACCURACY   ASGN  RealValue  Optcol
                     | GPLOT               ASGN  OnOff      Optcol
                     | GRMS                ASGN  OnOff      Optcol
                     ;

Optcol               : ';'
                     | Empty
                     ;

OnOff                : ON
                     | OFF
                     ;

Target               : FILE
                     | SCREEN
                     ;

ProcessEntities      : ProcessEntityList
                     | Empty
                     ;

ProcessEntityList    : ProcessAttributes
                     | ProcessEntityList ProcessAttributes
                     ;

ProcessAttributes    : ParameterSection
                     | RandomSection
                     | UnitSection
                     ;

/*       RANDOM_STREAM SECTION    */

RandomSection        : RANDOM_STREAM RanStreamList
                     ;
```

```
RanStreamList    : RanStreamDec
                 | RanStreamList RanStreamDec
                 ;

RanStreamDec     : IdentifierList AS SEED Expression
                 | IdentifierList AS IRREPRODUCIBLE
                 ;

/*       MONITOR SECTION          */

MonitorSection   : MONITOR
                    PatternList
                 | Empty
                 ;

PatternList      : PatternEntity
                 | PatternList PatternEntity
                 ;

PatternEntity    : PatternType ';'
                 ;

PatternType      : PatternBase
                  | PatternType  PatternBase
                  ;

PatternBase      :  Identifier
                 |  ILITERAL
                 | '('
                 | ')'
                 | '*'
                 | '%'
                 | '.'
                 | ','
                 ;

/*       ASSIGN SECTION  */

AssignSection    : ASSIGN AssignmentList
                 | Empty
                 ;

/*       PRESET SECTION  */

PresetSection    : PRESET PresetList
                 | Empty
                 ;

PresetList       : PresetEntity
                 | PresetList PresetEntity
                 ;

PresetEntity     : Name ASGN Real ';'
                 | Name ASGN OptReal ':' OptReal ':' OptReal ';'
                 | PresetWithin
                 | PresetFor
                 ;
```

```
PresetWithin      : WITHIN PathName DO PresetList END
                  ;

PresetFor         : FOR Identifier ASGN SimpleExp TO SimpleExp OptStep DO
                    PresetList END
                  ;

OptReal           : Real
                  | Empty
                  ;

/*       SELECTOR SECTION         */

IntSelector       : SELECTOR AssignmentList
                  | Empty
                  ;

/*       INITIAL SECTION */

IntInitial        : INITIAL EquationList
                  | INITIAL STEADY_STATE
                  | Empty
                  ;

/*       SCHEDULE SECTION         */

IntSchedule       : ScheduleSection
                  | Empty
                  ;

/*       UTILITY SYNTAX CONSTRUCTS        */

/*       A LIST OF IDENTIFIERS    */

IdentifierList    : Identifier
                  | IdentifierList ',' Identifier
                  ;

/*       A LIST OF PATHNAMES      */

PathNameList      : PathName
                  | PathNameList ',' PathName
                  ;

/*       A LIST OF EQUATIONS      */

EquationList      : Equation
                  | EquationList Equation
                  ;

Equation          : SimpleEquation  ';'
                  | WithinEquation
                  | ForEquation
                  | IfEquation
                  | CaseEquation
                  ;
```

```
/*      STRUCTURED EQUATIONS    */

WithinEquation  : WITHIN PathName  DO
                  EquationList END
                ;

ForEquation     : FOR Identifier ASGN SimpleExp TO SimpleExp OptStep DO
                    EquationList END
                ;

OptStep         : STEP SimpleExp
                | Empty
                ;

IfEquation      : IF Expression THEN
                    EquationList
                    ELSE EquationList END
                ;

CaseEquation    : CASE PathName OF CaseList END
                ;

CaseList        : CaseClause
                | CaseList CaseClause
                ;

CaseClause      : WHEN PathName ':' EquationList OptSwitchList
                ;

OptSwitchList   : SwitchList
                | Empty
                ;

SwitchList      : Switch
                | SwitchList Switch
                ;

Switch          : SWITCH TO PathName IF Expression ';'
                ;

/*      A SIMPLE EQUATION       */

SimpleEquation  : PathName ':' SimpleEqtn
                | SimpleEqtn
                ;

SimpleEqtn      : SimpleExp Equality SimpleExp
                | SimpleEqtn Equality SimpleExp
                ;

/*      A LIST OF ASSIGNMENTS   */

AssignmentList  : AssignEntity
                | AssignmentList AssignEntity
                ;
```

```
AssignEntity     : PathName ASGN SetValues ';'
                 | AssignWithin
                 | AssignFor
                 ;

AssignWithin     : WITHIN PathName  DO AssignmentList END
                 ;

AssignFor        : FOR Identifier ASGN SimpleExp TO SimpleExp OptStep DO
                   AssignmentList END
                 ;

/*       A SCHEDULE OF TASKS      */

ScheduleList     : Schedule
                 | ScheduleList Schedule
                 ;

Schedule         : TaskInstance
                 | SequenceTask
                 | ParallelTask
                 | WhileTask
                 | IfTask
                 | ContinueTask
                 | ResetTask
                 | ReinitialTask
                 | ReplaceTask
                 | ForeignTask
                 | MessageTask
                 | SwitchTask
                 | AssignTask
                 | MonitorTask
                 | LineariseTask
                 ;

TaskInstance     : Identifier '(' ParaAssignList ')' ';'
                 | Identifier ';'
                 ;

SequenceTask     : SEQUENCE ScheduleList END
                 ;

ParallelTask     : PARALLEL ScheduleList END
                 ;

WhileTask        : WHILE Expression DO Schedule END
                 ;

IfTask           : IF Expression THEN Schedule END
                 | IF Expression THEN Schedule ELSE Schedule END
                 ;

ContinueTask     : CONTINUE FOR SimpleExp
                 | CONTINUE UNTIL Expression
                 | CONTINUE FOR SimpleExp LogicalOp UNTIL Expression
                 ;
```

```
ResetTask       : RESET  AssignmentList END
                ;

ReinitialTask   : REINITIAL PathNameList WITH EquationList END
                ;

ReplaceTask     : REPLACE PathNameList WITH AssignmentList
                  END
                ;

ForeignTask     : GetTask
                | SendTask
                | PauseTask
                ;

AssignTask      : PathName ASGN SetValues  ';'
                ;

MonitorTask     : MonitorOnOffTask
                | MonitorFrequencyTask
                | MonitorAddRemoveTask
                ;

LineariseTask   : LINEARISE PathName Optcol
                ;

/*      LIST OF GET TASK ASSIGN */

GetTask         : GET Status GetList END
                ;

GetList         : GetEntity
                | GetList GetEntity
                ;

GetEntity       : PathName ASGN PathName  ';'
                ;

Status          : STATUS  PathName
                | Empty
                ;

/*      LIST OF SEND TASK ASSIGN        */

SendTask        : SEND Status SendList END
                ;

SendList        : SendEntity
                | SendList SendEntity
                ;

SendEntity      : PathName  ASGN  SetValues ';'
                ;

SendWithin      : WITHIN PathName DO SendList END
                ;
```

```
PauseTask       : PAUSE PathName Status
                ;

MessageTask     : MESSAGE STRLITERAL
                ;

/*      LISTS OF MONITOR TASK   */

MonitorOnOffTask : MONITOR ON
                 | MONITOR OFF
                 ;


MonitorAddRemoveTask : MONITOR ADD  PatternList END
                     | MONITOR REMOVE PatternList END
                     ;

MonitorFrequencyTask : MONITOR FREQUENCY SimpleExp
                     ;

/*      LISTS OF SWITCH TASK ASSIGN     */

SwitchTask      : SWITCH AssignmentList END
                ;


/*      LISTS OF PARAMETER VALUE ASSIGNMENTS    */

ParaAssignList  : Identifier IS SetValues
                | ParaAssignList ',' Identifier IS SetValues
                ;

/*      LISTS OF EXPRESSIONS    */

SetValues       : '[' ExpList ']'
                | Expression
                ;

ExpList         : Expression
                | Empty
                | SimpleExp ':' SimpleExp
                | ExpList ',' Expression
                | ExpList ',' Empty
                | ExpList ',' SimpleExp ':' SimpleExp
                ;

/*      AN EXPRESSION   */

Expression      : Relation
                | Expression LogicalOp Relation
                ;

Relation        : SimpleExp
                | Relation RelationOp SimpleExp
                ;
```

```
SimpleExp       : UnaryTerm
                | SimpleExp AddOp Term
                ;

UnaryTerm       : UnaryOp Term
                | Term
                ;

Term            : Factor
                | Term  MultOp  Factor
                ;

Factor          : Primary Power Primary
                | NOT Primary
                | Primary
                | Primary DomainOp
                ;

Primary         : ILITERAL
                | RLITERAL
                | Logical
                | TIME
                | GLOBAL_TIME
                | Name
                | '(' Expression ')'
                | OLD '(' Expression ')'
                | Derivatives
                ;

/*      A PATHNAME      */

Name            : PathName
                | '$' Identifier IdSuffix
                | PathName '.' '$' Identifier IdSuffix
                ;

Derivatives     : '$' '(' SimpleExp ')'
                | PARTIAL '(' SimpleExp ',' PathNameList ')'
                | INTEGRAL '('  IntegralOperandList ';' SimpleExp ')'
                ;

IntegralOperandList : IntegralOperand
                    | IntegralOperandList ',' IntegralOperand
                    ;

IntegralOperand : Identifier
                   ASGN SimpleExp ':' SimpleExp
                ;

PathName        : Identifier IdSuffix
                | PathName '.' Identifier IdSuffix
                ;

IdSuffix        : '(' ExpList ')'
                | Empty
                ;
```

```
/*      REAL, INTEGER OR BOOLEAN VALUES */

Real            : RealValue
                | UnaryOp RealValue
                ;

Logical         : TRUE
                | FALSE
                ;

RealValue       : RLITERAL
                | ILITERAL
                ;

/*      OPERATOR DEFINITIONS    */

LogicalOp       : AND
                | OR
                ;

RelationOp      : '='
                | NTEQ
                | '<'
                | LTEQ
                | '>'
                | GTEQ
                ;

AddOp           : '+'
                | '-'
                ;

MultOp          : '*'
                | '/'
                | DIV
                | MOD
                ;

DomainOp        : BARP
                | BARM
                | '|'
                ;

Power           : '^'
                ;

UnaryOp         : '+'
                | '-'
                ;

Equality        : '='
                | IS
                ;
```

```
/*      DEFINE IDENTIFIER     */

Identifier      : STRLITERAL
                | StringNQ
                ;

/*      THE EMPTY SET    */

Empty           :
                ;
```

Appendix B

gPROMS Input Files for Examples in Chapter 7

B.1 Process with Mixed Lumped/Distributed Parameter Systems

```
# ================================================================
#
#     Flowsheet consists of mixing tank, tubular reactor and
#                          gas absorption column.
#
# ================================================================

DECLARE
   TYPE
      Concentration              =   0.5   : -1E-3   :  1.0

   STREAM
      Mainstream                 IS   Concentration

END   # Declare

# ================================================================

MODEL Reactor

  PARAMETER
    NoC               AS                      INTEGER
    K1, K2            AS                      REAL
    Length            AS                      REAL
    StoichCoeff       AS   ARRAY (NoC) OF INTEGER

  DISTRIBUTION_DOMAIN
    Axial             AS  ( 0 : Length )

  VARIABLE
    C                 AS DISTRIBUTION(NoC,Axial) OF Concentration
    C_In, C_Out       AS ARRAY(NoC)              OF Concentration

STREAM
  Inlet  : C_In   AS Mainstream
  Outlet : C_Out  AS Mainstream

SET
  StoichCoeff       := [ -1, -1, 2 ] ;

BOUNDARY
  K1 * PARTIAL(C( ,0),Axial) = C( ,0) - C_In   ;
  PARTIAL(C( ,Length),Axial) = 0.0             ;
```

```
  EQUATION
    FOR z := 0|+ TO Length|- DO
      $C( ,z) = K1*PARTIAL(C( ,z),Axial,Axial) - PARTIAL(C( ,z),Axial)
                + StoichCoeff*K2*C(1,z)*C(2,z)  ;
    END

    C_Out   = C( ,Length) ;
END  # model reactor

# ============================================================

MODEL Absorber

  PARAMETER
    NoC              AS  INTEGER
    Length, K3, K4,
    K5, K6, K7       AS  REAL

  DISTRIBUTION_DOMAIN
    Forward          AS  ( 0 : Length )
    Reverse          AS  ( 0 : Length )

  VARIABLE
    C                AS  DISTRIBUTION(NoC,Forward) OF Concentration
    C_In, C_Out      AS  ARRAY       (NoC)         OF Concentration
    Theta            AS  DISTRIBUTION(Reverse)     OF Concentration

  STREAM
    Inlet  : C_In    AS  Mainstream
    Outlet : C_Out   AS  Mainstream

  BOUNDARY
    C( ,0)           = C_In                ;
    Theta(Length)    = 0.0                 ;

  EQUATION
    # Governing equations - gas phase
    FOR z := 0|+ TO Length DO
      $C(1:2,z) = -K3*PARTIAL(C(1:2,z),Forward)                              ;
      $C(3,z)   = -K3*PARTIAL(C(3,z),Forward)-K4*(C(3,z)-K5*Theta(z)) ;
    END

    # Governing equations - liquid phase
    FOR z := 0 TO Length|- DO
      $Theta(z) = K6 * PARTIAL(Theta(z),Reverse)
                  + K4/K7*(C(3,z)-K5*Theta(z))  ;
    END

    C ( ,Length)  =   C_Out                   ;
END   # model absorber

# ============================================================

MODEL Mixer

  PARAMETER
    NoC                  AS INTEGER
    K8, K9               AS REAL
```

```
VARIABLE
  C, C_Inlet,
  C_recycle              AS ARRAY(NoC) OF Concentration

STREAM
  Inlet  : C_recycle  AS Mainstream
  Outlet : C          AS Mainstream

  EQUATION
    $C = K8 * (C_Inlet + K9*C_Recycle-(1+K9)*C);
END  # model mixer

# ============================================================================

MODEL Flowsheet

  UNIT
    R101                        AS   Reactor
    A101                        AS   Absorber
    M101                        AS   Mixer

  SET
    WITHIN R101 DO
      K1        :=   0.1                ;
      K2        :=   3.0                ;
      Length    :=   1.0                ;
      Axial     :=   [ OCFEM, 3, 15 ]   ;
    END
    WITHIN A101 DO
      K3        :=   0.1                ;
      K4        :=   20.0               ;
      K5        :=   10.0               ;
      K6        :=   0.003              ;
      K7        :=   75.0               ;
      Length    :=   1.0                ;
      Forward   :=   [ BFDM, 1, 40 ]    ;
      Reverse   :=   [ CFDM, 2, 40 ]    ;
    END
    WITHIN M101 DO
      K8        :=   30.7               ;
      K9        :=   0.5                ;
    END

  EQUATION
    M101.Outlet   IS   R101.Inlet       ;
    R101.Outlet   IS   A101.Inlet       ;
    A101.Outlet   IS   M101.Inlet       ;
END  # model flowsheet

# ============================================================================

PROCESS SM

  UNIT
    Fs      AS   Flowsheet

  SET
```

```
    WITHIN Fs DO
      R101.NoC    :=  3 ;
      A101.NoC    :=  3 ;
      M101.NoC    :=  3 ;
    END

  ASSIGN
    WITHIN Fs DO
      M101.C_Inlet  :=  [ 1.0, 0.5 , 0.0 ] ;
    END

  INITIAL
    WITHIN Fs DO
      M101.C                        =  0.0 ;
      WITHIN R101 DO
        For z := 0|+ TO Length|- DO
          C( ,z)                    =  0.0 ;
        END
      END
      WITHIN A101 DO
        FOR z := 0|+ TO Length DO
          C( ,z)                    =  0.0 ;
        END
        A101.Theta(0:Length|-) =  0.0 ;
      END
    END

  SCHEDULE
    CONTINUE FOR 40

END   # Process Simulation
```

B.2 Fixed-bed Catalytic Reactor Immersed in Cooling Jacket

```
# ============================================================
#
#      Fixed-bed catalytic reactor with cooling jacket
#
# ============================================================

DECLARE
   TYPE
      Concen_molar         =  1E1  : -1E-3 :  1E8       UNIT = "mole/m3"
      Pressure             =  1E5  : -1E-3 :  1E11      UNIT = "Pa"
      Reaction             =  1E1  : -1E5  :  1E11      UNIT = "mole/kg-s"
      Reaction_coeff       =   -1  : -1E4  :  1E4       UNIT = "mole/kg-Pa2-s"
      Temperature          =  620  : -2E-4 :  1E4       UNIT = "K"
      Velocity             =    1  :    0  :  1E5       UNIT = "m/s"
      Mass_FlowRate        =   20  :    0  :  1E3
      Heat                 =  1E2  : -1E5  :  1E5
END   # Declare

# ============================================================

MODEL TubularReactor
```

```
PARAMETER
  # system dependent parameters
  NoC                       AS                    INTEGER
  Uc1, Void                 AS                    REAL

  # geometrical parameters
  ReactorRadius,
  ReactorLength             AS                    REAL

  # physical properties
  Db, Df, U, Cpf            AS                    REAL

  # ideal gas constant
  GasConst                  AS                    REAL

  # reaction
  Nu                        AS   ARRAY(NOC) OF INTEGER
  Aa, Bb, React_enth        AS                    REAL

  # transport properties
  Dr, Dz, Kr, Kz            AS                    REAL

DISTRIBUTION_DOMAIN
  Axial             AS   ( 0 : ReactorLength )
  Radial            AS   ( 0 : ReactorRadius )

VARIABLE
  Comp              AS   DISTRIBUTION(NoC,Axial,Radial)    OF Concen_molar
  Comp_inlet        AS   ARRAY(NoC)                        OF Concen_molar
  Press             AS   DISTRIBUTION(NoC,Axial,Radial)    OF Pressure
  Press_inlet       AS   ARRAY(NoC)                        OF Pressure
  Temp              AS   DISTRIBUTION(Axial,Radial)        OF Temperature
  Temp_inlet        AS                                        Temperature
  Temp_w            AS                                        Temperature
  Kk                AS   DISTRIBUTION(Axial,Radial)        OF Reaction_coeff
  React             AS   DISTRIBUTION(Axial,Radial)        OF Reaction
  Vz                AS                                        Velocity

BOUNDARY
  # @ z = 0
  FOR r := 0 TO ReactorRadius DO
    -Void * Dz * PARTIAL(Comp(,0,r),Axial)     =
              Vz * (Comp_inlet-Comp(,0,r))                  ;

    -Kz * PARTIAL(Temp(0,r),Axial)             =
            Df * Cpf * Vz * (Temp_inlet-Temp(0,r) ) ;
END

# @ z = L
FOR r := 0 TO ReactorRadius DO
  PARTIAL(Comp(,ReactorLength,r),Axial)     = 0    ;
  PARTIAL(Temp(ReactorLength,r),Axial)      = 0    ;
END

# @ r = 0
FOR z := 0|+ TO ReactorLength|- DO
  PARTIAL(Comp(,z,0),Radial)                = 0    ;
  PARTIAL(Temp(z,0),Radial)                 = 0    ;
```

```
END

# @ r = R
FOR z := 0|+ TO ReactorLength|- DO
  PARTIAL(Comp(,z,ReactorRadius),Radial)       = 0   ;
  -Kr * PARTIAL(Temp(z,ReactorRadius),Radial) =
            U*(Temp(z,ReactorRadius)-Temp_w)         ;
END

EQUATION

# ============================
#       Material Balance
# ============================

  FOR z := 0|+ TO ReactorLength|- DO
    FOR r := 0|+ TO ReactorRadius|- DO
      $Comp(,z,r) =
       - Vz * PARTIAL(Comp(,z,r),Axial)
       + Void * Dz * PARTIAL(Comp(,z,r),Axial,Axial)
       + Void * Dr * ( PARTIAL(Comp(,z,r),Radial,Radial)
                     + (1/r)*PARTIAL(Comp(,z,r),Radial) )
       - Nu *  React(z,r) * Db                                    ;
    END
  END

# ============================
#       Heat Balance
# ============================

  FOR z := 0|+ TO ReactorLength|- DO
    FOR r := 0|+ TO ReactorRadius|- DO
      Df * Cpf * $Temp(z,r) =
       - Vz * Df * Cpf * PARTIAL(Temp(z,r),Axial)
       + Kz * PARTIAL(Temp(z,r),Axial,Axial)
       + Kr * ( PARTIAL(Temp(z,r),Radial,Radial)
            + (1/r)*PARTIAL(Temp(z,r),Radial) )
       - React(z,r) * Db * React_enth                             ;
    END
  END

# ============================
#       Reaction
# ============================

    FOR z := 0 TO ReactorLength DO
      FOR r := 0 TO ReactorRadius DO
        Kk(z,r)       =  EXP( Aa - Bb/Temp(z,r) )                          ;
        React(z,r)    =  Kk(z,r) * Press(2,z,r) * Press(1,z,r) / Uc1       ;
        Press(,z,r)   =  Comp(,z,r) * GasConst * Temp(z,r)                 ;
      END
    END

    Press_inlet(1:NoC)  =  Comp_inlet(1:NoC) * GasConst * Temp_inlet  ;

END   # model tubularreactor

# ============================================================================
```

```
MODEL CoolingJacket

  PARAMETER
    RhoW, Vw, Cpw          AS                  REAL

  VARIABLE
    Fw                     AS                  Mass_FlowRate
    Tin, Tcw               AS                  Temperature
    Q                      AS                  HEAT

  EQUATION
    # Energy balance
    RhoW * Vw * Cpw * $Tcw = Fw * Cpw * (Tin -Tcw) + Q ;

END   # model coolingjacket

# ===================================================

MODEL CooledReactor

  PARAMETER
    U, PI, S    AS  REAL

  UNIT
    TR          AS  TubularReactor
    CJ          AS  CoolingJacket

  SET
     WITHIN TR DO
        Axial   := [ BFDM , 1, 40 ] ;
        Radial  := [ OCFEM, 3,  1 ] ;
     END

  EQUATION
    CJ.Tcw = TR.Temp_w                                                    ;
    CJ.Q = U * S * INTEGRAL ( z := 0 : TR.ReactorLength ;
                              TR.Temp(z,TR.ReactorRadius) - CJ.Tcw ) ;

  END # model cooledreactor

# ===================================================

PROCESS CT

UNIT
  CS                AS   CooledReactor

SET
  WITHIN CS DO
    WITHIN TR DO
      NoC                   :=   2          ;
      Uc1                   :=   3.6E10     ;
      ReactorRadius         :=   0.0254/2   ;   # [ m ]
      ReactorLength         :=   3.0        ;   # [ m ]
      GasConst              :=   8.314      ;
      Aa                    :=   19.837     ;
      Bb                    :=   13636.0    ;
```

```
    Cpf                 :=   992.0          ;    # [ J/kg-K ]
    Void                :=   0.35           ;
    React_enth          :=   307000*4.183   ;    # [ J/mol ]
    Dr                  :=   1E-2           ;    # [ m2/s ]
    Dz                  :=   1E-4           ;    # [ m2/s ]
    Db                  :=   1300.0         ;    # [ kg/m3 ]
    Df                  :=   1.293          ;    # [ kg/m3 ]
    Nu                  :=   -1             ;
      U                 :=   96.0           ;    # [ J/m2-K-s ]
      Kr                :=   5E-1           ;    # [ J/m-K-s]
      Kz                :=   5E-2           ;    # [ J/m-K-s]
    END # with TR

    WITHIN CJ DO
      RhoW := 1000                                              ;
      Vw   :=  3*(0.0254^2)*(4-3.141592^2/4)                    ;
      Cpw  :=  4200                                             ;
    END # with CJ
    U := 9.6                                                    ;
    S := 2 * 3.141593 * TR.ReactorRadius * TR.ReactorLength     ;
  END

ASSIGN
  WITHIN CS DO
    WITHIN  TR DO
      Vz    :=  4*1.6/(3.14159*0.0254^2)/3600 ;    # [ m/s ]
      # Inlet conditions
      Press_inlet(1)  :=  0.011E5             ;    # [ pa ]
      Press_inlet(2)  :=  0.211E5             ;    # [ pa ]
      Temp_inlet      :=  352.0 + 273.15      ;    # [ K ]
    END   # Within TR

    WITHIN CJ DO
      Fw      := 0.1            ;
      Tin     := 273.15 + 352.0 ;
    END   # within CJ
  END # within cs

INITIAL
  WITHIN CS DO
    WITHIN TR DO
      FOR z := 0|+ TO ReactorLength|- DO
        FOR r := 0|+ TO ReactorRadius|- DO
          Comp(,z,r)   = 0.0                    ;    # [ mol/m3 ]
          Temp(z,r)    = 352 + 273.15           ;    # [ K ]
        END # for r
      END # for z
    END   # Within TR
    CJ.Tcw = 273.15 + 352.0 ;
  END  # within cs

  SCHEDULE
     CONTINUE FOR 5.0

END   # Process Simulation
```

B.3 Rapid Pressure Swing Adsorption

```
# ================================================================
#
#      Rapid pressure swing adsorption
#
# ================================================================

DECLARE
  TYPE
    Conc            = 8.6       :  -1E-3     :  1E10
    Loading         = 0.07      :  -1E10     :  1E10
    Molefrac        = 0.21      :  -1E-3     :  1E10
    Pressure        = 1E5       :   1E4      :  1E10
    Temperature     = 290       :   1E1      :  1E6
    Velocity        = 4E-3      :  -1E10     :  1E10
    notype          = 100       :   0        :  5E2
END

# ================================================================

MODEL AdsorptionBed

  PARAMETER
    Nocomp                      AS                      INTEGER
    Dz, Eb, Ep, Et,
    dp, Db, alpha,
    BedLength,  CycleTime,
    PressTime,
    mu, Area, Q, R, Patm        AS                      REAL
    m, y                        AS  ARRAY(Nocomp) OF REAL
    k1, k2                      AS  ARRAY(Nocomp) OF REAL
    ks                          AS  ARRAY(Nocomp) OF REAL

DISTRIBUTION_DOMAIN
    SDomain                     AS  ( 0 : BedLength )
    TDomain                     AS  ( 0 : CycleTime )

VARIABLE
    C               AS  DISTRIBUTION(Nocomp,Sdomain,Tdomain)  OF Conc
    P               AS  DISTRIBUTION(Sdomain,Tdomain)         OF Pressure
    Pf, Pw          AS                                           Pressure
    qi              AS  DISTRIBUTION(Nocomp,Sdomain,Tdomain)  OF Loading
    Tf              AS                                           Temperature
    V               AS  DISTRIBUTION(Sdomain,Tdomain)         OF Velocity

BOUNDARY
  FOR t:= 0 TO PressTime DO
    C(1:Nocomp,0,t) = y(1:Nocomp) * P(0,t)/(R * Tf) ;
    P(0,t)          = Pf                            ;
  END

  FOR  t:= PressTime|+ TO CycleTime  DO
    PARTIAL(C( ,0,t),SDomain)  =  0.0               ;
    P(0,t)                     =  Pw                ;
  END

  FOR  t:= 0 TO CycleTime DO
```

```
  PARTIAL(C(1:Nocomp,BedLength,t),SDomain) =  0.0                ;
  V(BedLength,t)              =  (Q/Area) * (Patm/P(BedLength,t)) ;
END

FOR z:=  0|+  TO  BedLength|-  DO
  C( ,z,0)   =  C( ,z,CycleTime)  ;
END
```

```
  FOR z:= 0 TO BedLength DO
    qi( ,z,0)  =  qi( ,z,CycleTime) ;
  END

EQUATION
  FOR z := 0|+ TO BedLength|- DO
    FOR t:= 0|+ TO CycleTime DO
      PARTIAL(Et * C( ,z,t) + Db * qi( ,z,t),Tdomain) =
           - PARTIAL(V(z,t) * C( ,z,t),SDomain)
           + Dz * PARTIAL(C( ,z,t),SDomain,SDomain) ;
    END
  END

  FOR z := 0 TO BedLength DO
    FOR t:= 0|+ TO CycleTime DO
      FOR i := 1 TO Nocomp DO
        PARTIAL(qi(i,z,t),TDomain) =
            ks(i)* ( m(i)*C(i,z,t)*R*Tf - qi(i,z,t)) ;
      END
    END

    FOR z := 0|+ TO BedLength DO
      FOR t:= 0 TO CycleTime DO
        P(z,t) / (R*Tf) = SIGMA( C(1:Nocomp,z,t) )      ;
      END
    END

    FOR z := 0  TO BedLength|-  DO
      FOR t := 0 TO CycleTime DO
        PARTIAL(P(z,t),Sdomain) =
             - (alpha * mu * (1-Eb)^2 * V(z,t)) / (dp^2 * Eb^3) ;
      END
    END

END # model adsorptionbed
```

```
#=============================================================

PROCESS PSA

UNIT rpsa  AS adsorptionbed
  SET # for parameters and distribution domain
    WITHIN rpsa  DO
      Nocomp        :=  2              ;   # [N2, O2]
      BedLength     :=  1              ;   # Column length m
      Dz            :=  1E-6           ;   # Axial dispersion m2 s-1
      R             :=  8.314          ;   # Gas constant J mol-1 K-1
      Eb            :=  0.35           ;
      Ep            :=  0.55           ;
```

```
  Et              :=  Eb + Ep * (1 - Eb) ;
  dp              :=  200E-6             ;   # Particle diameter m
  Db              :=  800.               ;   #   kg m-3
  alpha           :=  180                ;
  mu              :=  1.8E-5             ;   # Gas viscosity N s m-2
  y               :=  [ 0.79, 0.21 ]     ;   # mole of N2, O2
  PressTime       :=  1.5                ;   # sec
  CycleTime       :=  3.0                ;   # sec
  Area            :=  1.963E-3           ;   # X-area of bed m2
  m               :=  [ 3.0E-6, 1.43E-6 ]
                                         ;   # isother gradients [N2,O2]
  Q               :=  1E-5               ;   # Constant gas vol. flow rate
                                             # at ambient conditions m3 s-1
  Patm            :=  1E5                ;   # Pressure at ambient Pa
  Ks              :=  [25.32060, 55.22957]    ;   #
  SDomain         := [ OCFEM, 3, 7 ] ;
  Tdomain         := [ OCFEM, 3, 7 ] ;
  k1              := [251680724.24174, 785158330.10596] ;
  k2              := [47.68482, 69.06734] ;
END

ASSIGN
  WITHIN rpsa DO
    Pw            :=  1E5                ;   # Pa
    Pf            :=  2.4E5              ;   # Pa
    Tf            :=  290                ;   # Feed Temperature K
  END
END # process
```

B.4 Continuous Media Sterilisation

```
# ================================================================
#
#  Flowsheet model for entire sterilisation system, with plug flow.
#                          Parameters are from Armenante, 1990.
#
# ================================================================

DECLARE
  TYPE
    Temperature    = 348.15  :    0    : 473.15   UNIT = "K"
    Power          = 5E4     : -1E6    : 1E6      UNIT = "J/s"
    logConc        = 5       : -1E3    : 1E3      UNIT = "/m3"
    Integ          = 4E3     :  1E-14  : 1E10     UNIT = "/m3"

END

# ================================================================

MODEL recovery

  PARAMETER
    cp, rho, xi                          AS REAL
    cps, rhos, ks, h                     AS REAL
    logKd0, logKd0_nut, Ed, Ed_nut       AS REAL
    wex, dex, U, R, pi, vf               AS REAL
```

```
  Radius, Length                  AS REAL

DISTRIBUTION_DOMAIN
  Axial                           AS (0 : Length)
  Reverse                         AS (0 : Length)
  Radial                          AS (0 : 1)

VARIABLE
  Tc              AS DISTRIBUTION(Axial)          OF Temperature
  Th              AS DISTRIBUTION(Reverse)        OF Temperature
  Q               AS DISTRIBUTION(Axial)          OF Power

  logNc           AS DISTRIBUTION(Axial)          OF logConc
  logNh           AS DISTRIBUTION(Reverse)        OF logConc

  logCc           AS DISTRIBUTION(Axial)          OF logConc
  logCh           AS DISTRIBUTION(Reverse)        OF logConc

  Tcs             AS DISTRIBUTION(Axial,Radial)   OF Temperature
  Ths             AS DISTRIBUTION(Reverse,Radial) OF Temperature

  logNcs          AS DISTRIBUTION(Axial,Radial)   OF logConc
  logNhs          AS DISTRIBUTION(Reverse,Radial) OF logConc

BOUNDARY
  FOR z:= 0|+ TO Length DO
    PARTIAL(Tcs(z,0),Radial)    =  0                                    ;
    ks*PARTIAL(Tcs(z,1),Radial) =  Radius*h*( Tc(z) - Tcs(z,1) ) ;
  END

  FOR z:= 0 TO Length|- DO
    PARTIAL(Ths(z,0),Radial)    =  0                                    ;
    ks*PARTIAL(Ths(z,1),Radial) =  Radius*h*( Th(z) - Ths(z,1) ) ;
  END

EQUATION
  # ============================================
  #                 Liquid medium
  # ============================================

  ##      mass balances
  #  mass balances for cell and nutrients : cold side
  FOR z := 0|+ TO Length DO
    (vf/wex/dex)*PARTIAL(logNc(z),Axial) =
      -EXP(logKd0 - Ed/(R*Tc(z)))          ;

    (vf/wex/dex)*PARTIAL(logCc(z),Axial) =
      -EXP(logKd0_nut - Ed_nut/(R*Tc(z))) ;
END # of FOR z statement

#  mass balances for cell and nutrients : hot side
FOR z := 0 TO Length|- DO
  (vf/wex/dex) * PARTIAL(logNh(z),Reverse) =
            EXP(logKd0 - Ed/(R*Th(z))) ;

  (vf/wex/dex)*PARTIAL(logCh(z),Reverse) =
```

```
    EXP(logKd0_nut - Ed_nut/(R*Th(z))) ;
END # of FOR statement

##      eneregy balance
#  energy balance : cold side
FOR z := 0|+ TO Length DO
  (1-xi)*vf*rho*cp*PARTIAL(Tc(z),Axial)
    = Q(z) - 3*xi*wex*dex/Radius*h*( Tc(z) - Tcs(z,1) )    ;
END # of FOR z statement

#  energy balance : hot side
FOR z := 0 TO Length|- DO
  (1-xi)*vf*rho*cp*PARTIAL(Th(z),Reverse) =
    Q(z) + 3*xi*wex*dex/Radius*h*( Th(z) - Ths(z,1) ) ;
END # of FOR statement

FOR z := 0 TO Length DO
  Q(z) = 2*U*wex*(Th(z) - Tc(z)) ;
END

# ============================================
#               Solid particles
# ============================================
##      mass balance
#  mass balance : cold side
FOR z := 0|+ TO Length DO
  FOR rad := 0 TO 1 DO
    (vf/wex/dex)*PARTIAL(logNcs(z,rad),Axial) =
        -EXP (logKd0 - Ed/(R*Tcs(z,rad))) ;
  END
END

#  mass balance : hot side
FOR z := 0 TO Length|- DO
  FOR rad := 0|+ TO 1|- DO
    (vf/wex/dex)*PARTIAL(Ths(z,rad),Reverse) =
        -ks/(rhos*cps*Radius^2)*
                    (  PARTIAL(Ths(z,rad),Radial,Radial) +
                    (2/rad)*PARTIAL(Ths(z,rad),Radial)   ) ;
  END # FOR rad
END # FOR z

##      energy balance
#  energy balance : cold side
FOR z := 0|+ TO Length DO
  FOR rad := 0|+ TO 1|- DO
    (vf/wex/dex)*PARTIAL(Tcs(z,rad),Axial) =
        ks/(rhos*cps*Radius^2)*
            ( PARTIAL(Tcs(z,rad),Radial,Radial) +
              (2/rad)*PARTIAL(Tcs(z,rad),Radial) ) ;
  END # FOR rad
END # FOR z

#  energy balance : hot side
FOR z := 0 TO Length|- DO
  FOR rad := 0 TO 1 DO
    (vf/wex/dex)*
      PARTIAL(logNhs(z,rad),Reverse) =
```

```
                EXP(logKd0 - Ed/(R*Ths(z,rad))) ;
      END
    END

END # of recovery model

# ================================================================
MODEL steam_heater

  PARAMETER
    cp, rho, xi                          AS REAL
    cps, rhos, ks, h                     AS REAL
    logKd0, logKd0_nut, Ed, Ed_nut       AS REAL
    wex, dex, U                          AS REAL
    R, pi, vf, Tsteam                    AS REAL
    Radius, Length                       AS REAL

  DISTRIBUTION_DOMAIN
    Axial               AS (0 : Length)
    Radial              AS (0 : 1)

  VARIABLE
    T                   AS DISTRIBUTION(Axial)         OF Temperature
    Q                   AS DISTRIBUTION(Axial)         OF Power

    logN                AS DISTRIBUTION(Axial)         OF logConc
    logC                AS DISTRIBUTION(Axial)         OF logConc

    Ts                  AS DISTRIBUTION(Axial,Radial) OF Temperature
    logNs               AS DISTRIBUTION(Axial,Radial) OF logConc

  BOUNDARY
    FOR z := 0|+ TO Length DO
      PARTIAL(Ts(z,0),Radial)    = 0                                    ;

      ks*PARTIAL(Ts(z,1),Radial) = Radius*h*( T(z) - Ts(z,1) ) ;
    END

  EQUATION
    # ============================================
    #                 Liquid medium
    # ============================================

    ##     mass balance
    FOR z := 0|+ TO Length DO
      (vf/wex/dex)*PARTIAL(logN(z),Axial) =
        -EXP(logKd0 - Ed/(R*T(z))) ;

      (vf/wex/dex)*PARTIAL(logC(z),Axial) =
        -EXP(logKd0_nut - Ed_nut/(R*T(z))) ;
    END

    ##    energy balance
    FOR z := 0|+ TO Length DO
      (1-xi)*vf*rho*cp*PARTIAL(T(z),Axial) =
        Q(z) - 3*xi*wex*dex/Radius*h*( T(z) - Ts(z,1) );
    END
```

```
FOR z := 0 TO Length DO
  Q(z) = 2*U*wex*(Tsteam - T(z)) ;
END

# ============================================
#              Solid particles
# ============================================

##   mass balance
FOR z := 0|+ TO Length DO
  FOR rad := 0 TO 1 DO
    (vf/wex/dex)*PARTIAL(logNs(z,rad),Axial) =
              -EXP(logKd0 - Ed/(R*Ts(z,rad))) ;
  END
END

##   energy balance
FOR z := 0|+ TO Length DO
  FOR rad := 0|+ TO 1|- DO
    (vf/wex/dex)*PARTIAL(Ts(z,rad),Axial) =
      ks/(rhos*cps*Radius^2)*
      (  PARTIAL(Ts(z,rad),Radial,Radial) +
         (2/rad)*PARTIAL(Ts(z,rad),Radial)  ) ;
  END # FOR rad
END # FOR z

END # of steam_heater model

# ============================================================

MODEL holding

  PARAMETER
    cp, rho, xi                          AS REAL
    cps, rhos, ks, h                     AS REAL
    logKd0, logKd0_nut, Ed, Ed_nut       AS REAL
    R, pi, vf, Radius, Length            AS REAL
    Diameter                             AS REAL

  DISTRIBUTION_DOMAIN
    Axial                AS (0 : Length)
    Radial               AS (0 : 1)

  VARIABLE
  T                      AS DISTRIBUTION(Axial)          OF Temperature
  logN                   AS DISTRIBUTION(Axial)          OF logConc
  logC                   AS DISTRIBUTION(Axial)          OF logConc
  Ts                     AS DISTRIBUTION(Axial,Radial) OF Temperature
  logNs                  AS DISTRIBUTION(Axial,Radial) OF logConc

BOUNDARY
  FOR z := 0|+ TO Length DO
    PARTIAL(Ts(z,0),Radial)    = 0                                  ;

    ks*PARTIAL(Ts(z,1),Radial) = Radius*h*( T(Z) - Ts(z,1) ) ;
  END
```

```
EQUATION
  # ===========================================
  #                 Liquid medium
  # ===========================================

  ##   mass balances
  FOR z := 0|+ TO Length DO
    (vf*4/pi/Diameter^2)*PARTIAL(logN(z),Axial) =
      -EXP(logKd0 - Ed/(R*T(z))) ;

    (vf*4/pi/Diameter^2)*PARTIAL(logC(z),Axial) =
      -EXP(logKd0_nut - Ed_nut/(R*T(z))) ;
END # of FOR statement

##   energy balance
FOR z := 0|+ TO Length DO
  (1-xi)*vf*rho*cp*PARTIAL(T(z),Axial) =
    - 3/4*xi*pi*Diameter^2/Radius*h*( T(z) - Ts(z,1) );
END # of FOR statement

# ===========================================
#             Solid particles
# ===========================================

##   mass balance
FOR z := 0|+ TO Length DO
  FOR rad := 0 TO 1 DO
      (vf*4/pi/Diameter^2)*PARTIAL(logNs(z,rad),Axial) =
        -EXP(logKd0 - Ed/(R*Ts(z,rad))) ;
    END
  END

  ##   energy balance
  FOR z := 0|+ TO Length DO
    FOR rad := 0|+ TO 1|- DO
      (vf*4/pi/Diameter^2)*PARTIAL(Ts(z,rad),Axial) =
        ks/(rhos*cps*Radius^2)*
        (  PARTIAL(Ts(z,rad),Radial,Radial) +
           (2/rad)*PARTIAL(Ts(z,rad),Radial)  ) ;
    END # FOR rad
  END # FOR z

END # of holding  model

# ===========================================================================

MODEL flowsheet

  PARAMETER
    cp, cps                                  AS REAL
    rho, rhos, ks, h, xi                     AS REAL
    logKd0, logKd0_nut, Ed, Ed_nut           AS REAL
    R, pi, vf, Tsteam, Radius                AS REAL
    Tin, logNin, logCin, Tins, logNins       AS REAL

  UNIT
    HX1        AS  recovery
    HX2        AS  steam_heater
```

```
  Pipe      AS  holding

DISTRIBUTION_DOMAIN
  Radial                          AS (0 : 1)

VARIABLE
  I1                              AS   Integ
  logSTs                          AS   logConc
  logST                           AS   logConc
  logDeg                          AS   logConc
  FS                              AS   Integ
  H1_COST                         AS   Integ
  H2_COST                         AS   Integ
  P_COST                          AS   Integ
  STEAM_COST                      AS   Integ
  TA_COST                         AS   Integ

EQUATION

  #  Boundary conditions: for cold side of recovery exchanger
  HX1.Tc(0)    = Tin    ;
  HX1.logNc(0) = logNin ;
  HX1.logCc(0) = logCin ;

  FOR rad := 0 TO 1 DO
    HX1.Tcs(0,rad)    = Tins    ;
    HX1.logNcs(0,rad) = logNins ;
  END

  #  for steam heater
  HX2.T(0)    = HX1.Tc(HX1.Length)    ;
  HX2.logN(0) = HX1.logNc(HX1.Length) ;
  HX2.logC(0) = HX1.logCc(HX1.Length) ;

  FOR rad := 0 TO 1 DO
    HX2.Ts(0,rad)    = HX1.Tcs(HX1.Length,rad)    ;
    HX2.logNs(0,rad) = HX1.logNcs(HX1.Length,rad) ;
  END

  #  for holding section
  Pipe.T(0)    = HX2.T(HX2.Length)    ;
  Pipe.logN(0) = HX2.logN(HX2.Length) ;
  Pipe.logC(0) = HX2.logC(HX2.Length) ;

  FOR rad := 0 TO 1 DO
    Pipe.Ts(0,rad)    = HX2.Ts(HX2.Length,rad)    ;
    Pipe.logNs(0,rad) = HX2.logNs(HX2.Length,rad) ;
  END

  #  for hot side of recovery exchanger
  HX1.Th(HX1.Length)    = Pipe.T(Pipe.Length)    ;
  HX1.logNh(HX1.Length) = Pipe.logN(Pipe.Length) ;
  HX1.logCh(HX1.Length) = Pipe.logC(Pipe.Length) ;

  FOR rad := 0 TO 1 DO
    HX1.Ths(HX1.Length,rad)    = Pipe.Ts(Pipe.Length,rad)    ;
    HX1.logNhs(HX1.Length,rad) = Pipe.logNs(Pipe.Length,rad) ;
  END
```

```
#  sterility for solids
I1  = INTEGRAL(rad := 0 : 1 ;
                    4*pi*rad^2*Radius^2*EXP(HX1.logNcs(0,rad)))
    / INTEGRAL(rad := 0 : 1 ;
                    4*pi*rad^2*Radius^2*EXP(HX1.logNhs(0,rad) + 150 ))  ;

logSTs*2.30259 = 150 + log(I1) ;

logST*2.30259   = HX1.logNc(0) - HX1.logNh(0) ;
logDeg*2.30259  = HX1.logCc(0) - HX1.logCh(0) ;

FS*(2436.26823 - 1.14949*TSTEAM - 6.63754E-3*TSTEAM^2) =
    rho*vf*cp*(HX2.T(HX2.Length) - HX2.T(0)) ;

# objective functions for optimisation
STEAM_COST = 4*FS/3600*
             (-2.28332 + 0.0748554*TSTEAM - 1.19130E-4*TSTEAM^2) ;

H1_COST    = 8640/3*(HX1.wex*HX1.Length)^0.65 ;

H2_COST    = 8640/3*(HX2.wex*HX2.Length)^0.65 ;

P_COST     = 10515/3*Pipe.Diameter^1.066*Pipe.Length^0.802 ;

TA_COST    = STEAM_COST + H1_COST + H2_COST + P_COST ;

END # of flowsheet model

# ================================================

PROCESS SS

  UNIT
    Steriliser    AS    flowsheet

  SET
    WITHIN Steriliser DO
      Tin               := 273.15 + 25      ; #[K]
      Tins              := 273.15 + 25      ; #[K]
      logNin            := log(1E11)        ; #[/m3]
      logNins           := log(1E11)        ; #[/m3]
      logCin            := log(1E11)        ; #[/m3]
      Tsteam            := 273.15 + 150     ; #[K]
      cp                := 4229             ; #[J/kg.K]
      cps               := 1880             ; #[J/kg.K]
      rho               := 962              ; #[kg/m3]
      rhos              := 790              ; #[kg/m3]
      ks                := 445E-3           ; #[J/s.m.K]
      h                 := 9650             ; #[J/s.m2.K]
      xi                := 0.1              ; #[-]
      vf                := 40/3600          ; #[m3/s]
      logKd0            := 83.81            ; #[/s]
      logKd0_nut        := 21.2             ; #[/s]
      Ed                := 67700*4.1868     ; #[J/mol]
      Ed_nut            := 22000*4.1868     ; #[J/mol]
      R                 := 8.31441          ; #[J/mol.K]
      pi                := 3.14159          ;
```

```
Radius              := 0.006                ; #[m]
Radial              := [ OCFEM, 3, 5]       ; #
WITHIN HX1 DO
  wex               := 1                    ; #[m]
  dex               := 0.012                ; #[m]
  U                 := 400                  ; #[J/s.m2.K]
  Axial             := [ OCFEM, 3, 5 ]      ; #
  Reverse           := [ OCFEM, 3, 5 ]      ; #
  Length            := 90.432               ; #[m]
  Radial            := [ OCFEM, 3, 5]       ; #
END # within HX1
WITHIN HX2 DO
  wex               := 1                    ; #[m]
  dex               := 0.012                ; #[m]
  U                 := 1000                 ; #[J/s.m2.K]
  Axial             := [ OCFEM, 3, 5 ]      ; #
  Length            := 48.974               ; #[m]
  Radial            := [ OCFEM, 3, 5]       ; #
END # within HX2
WITHIN Pipe DO
  Diameter          := 0.123                ; #[m]
  Axial             := [ OCFEM, 3, 5 ]      ; #
  Length            := 3.53                 ; #[m]
END # within Pipe
END # within Steriliser

END # of fluidi process
```

B.5 CMSMPR Crystalliser

```
# ================================================================
#
#    Crystallisation process described by population balances
#
# ================================================================

DECLARE
  TYPE
    number_density     =  1E4     :  1E-10  :   1E15    unit = "n/m4"
    mass_density       =  20.0    :  1E-10  :   50.0    unit = "kg/m3"
    population_density =  15.0    :    0    :   1E8     unit = "kg/m4"
    growth_rate        =   0.5    :  1E-10  :   20.0    unit = "mm/h"
    flow_rate          =  78.75   :    0    :   1E3     unit = "kg/h"
    residence_time     =   0.165  :    0    :    1.0    unit = "h"
    temperature        =  25.0    :   20    :   40.0    unit = "C"
    concentration      =   0.0    :    0    :    1.0    unit = "kg/kg"
    supersaturation    =  1E-5    :  1E-5   :    1.0    unit = "kg/kg"
    birth_rate         =  1.7E7   : -1E-3   :   1E15    unit = "nucl/m3h"
    shape_factor       =  1.0     :    0    :   1E3
    notype             =  1.0     :  -10    :   21.0
    lognumber_density  =  1E1     : -1E2    :   2E2
END

MODEL Crystalliser
```

```
PARAMETER
  size1, max_size     AS   REAL
  rho, rhoS           AS   REAL
  V                   AS   REAL

DISTRIBUTION_DOMAIN
  size               AS   ( 0 : max_size )

VARIABLE
  logn          AS   DISTRIBUTION (size) OF lognumber_density
  g             AS   DISTRIBUTION (size) OF growth_rate
  Kv            AS   DISTRIBUTION (size) OF shape_factor
  c, cc,
  cin, ceq      AS                        concentration
  Mt            AS                        mass_density
  sigma         AS                        supersaturation
  tau           AS                        residence_time
  temp          AS                        temperature
  W             AS                        flow_rate


BOUNDARY
  logn(0) = log (4.12e14 * Mt * sigma^3.4 / (847 * sigma^2 * 1E-3)) ;

EQUATION
  # Total mass balance on solute
  V * rho * $cc  = W * (cin-c) - Mt * V / tau                        ;

  # Definition of combined concentration
  cc = c  + Mt / rho                                                 ;

  # Definition of magma density
  Mt = rhoS * 1E-12 *
       INTEGRAL ( L := 0 : max_size ; Kv(L) * L^3 * EXP(logn(L)))    ;

  # Crystal population balance
  FOR L := 0|+ TO max_size DO
   $logn(L) + PARTIAL( g(L), size )
            + g(L) * PARTIAL(logn(L), size) + 1 / tau = 0            ;
  END

  # Residence time
  tau  =  V * rho / W                                                ;

  # Supersaturation
  IF c - ceq < 1E-5 THEN
      sigma  = 1E-5     ;
  ELSE
      sigma = c - ceq ;
  END

  # Crystal growth rate correlation (Chianese et al., 1987)
  FOR L := 0 TO max_size DO
      g(L) = 892 * sigma^2 * (1+5.87*L)                                    ;
  END

  # Volumetric shape factor correlation (Butz et al., 1987)
  FOR L := 0 TO size1 DO
```

```
  Kv(L) = 0.898 * EXP(0.168*(L*1e3)^0.5 - 8.234e-3*(L*1e3) )      ;
END
FOR L := size1|+ TO max_size DO
  Kv(L) = 4.46 * EXP(-0.0797*(L*1e3)^0.5 + 6.76E-4*(L*1e3) )      ;
END

# Equilibrium concentration (Perry,
ceq = 0.0735 + 1.675E-3 * temp                                    ;
END  # model crystalliser

# ================================================

PROCESS crystallisation

  UNIT
    CMSMPR   AS  Crystalliser

  SET
    WITHIN CMSMPR DO
      size1       := 0.1                ;
      max_size    := 1.5                ;
      rho         := 1000.0             ;
      rhoS        := 2660               ;
      V           := 0.050              ;

      size        := [ bfdm, 1, 150 ]   ;
    END

  ASSIGN
    WITHIN CMSMPR DO
      cin    :=   0.15        ;
      temp   :=   25.0        ;
      w      :=  100.0        ;
    END

  INITIAL
    WITHIN CMSMPR DO
      cc  = 0 ;
      FOR L := 0|+ TO max_size DO
        logn(L) = 0 ;
      END
    END

  SCHEDULE
     CONTINUE FOR 5
END
```

BIBL.
LCNBIN.