A SPACE-TIME REACTOR SIMULATION WITH INTEGRAL THERMALHYDRAULICS

By

GERALD DENIS GABOURY, M.Sc.

A Thesis

Submitted to the School of Graduate Studies

in Partial Fulfilment of the Requirements

for the Degree

Doctor of Philosophy

McMaster University

DOCTOR OF PHILOSOPHY (1993) McMASTER UNIVERSITY

(Engineering Physics) Hamilton, Ontario

TITLE:

A Space-Time Reactor Simulation

with Integral Thermalhydraulics

AUTHOR:

Gerald Denis Gaboury, B.Sc. (University of Alberta)

M.Sc. (Queens University)

SUPERVISOR:

Doctor W.J. Garland

NUMBER OF PAGES: xi, 241

Abstract

Exploration of the processes involved in fission reactor dynamics and the solution techniques required for such a problem is the objective of this research. Reactor behavior is governed by neutron kinetics and thermalhydraulics, and the spatial and temporal interaction of the two processes. The analytical solution of such a problem in all but the simplest of cases, is impossible. Hence, numerical methods are required. Simulation software is developed to model the space-time behavior of nuclear systems and this software is then used to study the behavior of these systems. This investigation involves the mathematical modeling and numerical simulation of the phenomena encountered. The desired result is an accurate depiction of reactor behavior.

The majority of contemporary simulations involve only a loose coupling between the neutronics and thermalhydraulics both spatially and temporally. It is more expedient to solve reactor kinetics and thermalhydraulic simulation problems separately. A point kinetics model is frequently added to a thermalhydraulic simulation to model reactor power but this would ignore spatial effects. The simulation developed here is a space-time reactor kinetics simulation with an integral thermalhydraulics module. This integral thermalhydraulics module provides for good spatial and temporal coupling of the two processes and is the primary distinguishing feature of the software.

The rate form of the equation of state is used in the solution of the pressure field in the thermalhydraulic problem. A rate equation is better suited to the solution procedure for the thermalhydraulics than the use of an equation of state.

Efforts were made to take advantage of the structure of the problem to reduce computational effort. Partitioned matrices are used in the time integration of both the neutron kinetics and thermalhydraulics. This helped reduce computational effort by facilitating the use of routines that exploit the sparsity of the Jacobian. A second order semi-implicit Runge-Kutta method is used for the time integration of the neutron kinetics problem. This method requires matrix inversion but is a robust and reliable method for solving a stiff system of ordinary differential equations.

The two component modules of the software, the neutronics and the thermalhydraulics, were developed and tested separately before they were integrated. The results of these tests and simulations using the completed software are given.

Acknowledgements

I have many people to thank for proof reading this and related work, notably Chantal Guertin and Harold Smith. I would like to thank Professor A.A. Harms for my initiation into the study of Nuclear Engineering. Mostly, I would like to thank William J. Garland for his support, guidance and friendship throughout this endeavor.

The sciences do not try to explain, they hardly even try to predict, they mainly make models. By a model is meant a mathematical construct which, with the addition of verbal interpretations, describes observed phenomena. The justification of such a mathematical construct is solely and precisely that it is expected to work.

John von Neumann

to Denis

Table of Contents

Table of Figures

Table of Tables

Introduction

Fission reactor behavior is governed by the spatial and temporal interaction of the two main processes, neutron kinetics and thermalhydraulics. The analytical solution of such a problem, in all but the simplest of cases, is impossible. The numerical solution of such a problem is not trivial either. In order to properly model the behavior of such a system, the governing equations of the two processes must somehow be solved simultaneously or nearly simultaneously. If the processes interaction is sufficiently frequent, this would require effectively simultaneous solution.

To model the passive behavior of a reactor was the objective. A reactor with passive cooling and no external reactivity control is such a reactor. A pool type reactor with vertical flow tubes cooled by natural convection is the simplest implementation of passive cooling and it is not an uncommon design. The dynamical behavior of such a reactor, without reactivity control or with disabled reactivity control, is modeled here.

The majority of the energy released from nuclear fission is manifested as kinetic energy of the fission products which results in thermal energy production in the fuel. This thermal energy is transferred by conduction to the fuel-coolant boundary where it is transported away by convective flow. Density variation within the coolant due to thermal expansion is the forcing term in the conservation of momentum equation. Temperature dependence of the multigroup cross sections within the fuel is modeled using thermal feedback coefficients. This is the most direct thermal reactivity feedback effect. The multigroup cross sections within the coolant are directly dependent on the coolant density. This primarily affects the neutron thermalization characteristics is therefore a less direct form of feedback.

Since their inception, reactor kinetics and thermalhydraulic simulations have existed separately. When the simulation of a reactor transient is desired, it is more expedient to solve the neutron kinetics and the thermalhydraulics separately. It is not uncommon for an iterative approach to be used where successive thermalhydraulic and neutron kinetic simulations are run for the same transient until the changes between the successive simulations decrease to an acceptable level. Fuel temperature, coolant temperature and void fraction are output from the thermalhydraulic simulation and become input into the neutron kinetics simulation. A spatial power distribution, as a function of time, an output of the reactor kinetics simulation would then be used as input into another thermalhydraulic simulation.

Increasing interest has been shown of late in software that simulates these two processes as interacting nearly simultaneously. The majority of these simulations involve only a loose coupling between the neutronics and thermalhydraulics both spatially and temporally. A point neutron kinetics model is relatively easy to implement and is frequently added to a thermalhydraulic simulation to model reactor power. Such

a simulation would ignore spatial effects but would give good temporal coupling of the two processes. The addition of a simple thermalhydraulic model to a spatial kinetics simulation would also give good temporal coupling but again spatial effects would not be modelled very well.

The simulation developed here is a reactor kinetics simulation involving spatial effects and is therefore classified as a space-time reactor kinetics simulation. However, this simulation has an integral thermalhydraulic module which is its primary distinguishing feature. In the design of the simulation software, good spatial and temporal coupling of the neutron kinetics and thermalhydraulics was sought. A finite difference approximation was chosen because it was the simplest and most direct method of solving the required equations. This allowed for very tight temporal coupling with thermal feedback being felt at each spatial point and good temporal coupling, depending on the time step size used in the simulation. Unfortunately, finite difference methods are computationally expensive. Other less computationally expensive methods are available but the penalty would be complexity and greater problems of implementation. A large body of literature is available on the finite difference method which makes it attractive and expedites development.

Efforts were made to take advantage of the structure of the problem to reduce computational effort. The rate of change of the delayed neutron precursors used in the neutron kinetics is much less than the rate of change of the neutron flux. A partitioned matrix inversion is used in the time integration of the neutron kinetics where the neutron flux and the delayed neutron precursors are integrated separately. Although a more expensive implicit method is required for the neutron flux, a less expensive explicit method may be used for the delayed neutron precursors. This approach helps reduce the Jacobian matrix complexity as well as the computational effort. A partitioned matrix solution is also used in the thermalhydraulic problem. The Jacobian of the conservation of momentum equations has a block tridiagonal structure but the solution of the pressure equation (the rate form of the equation of state) must be separated in order to allow the usage of a tridiagonal matrix inversion routines. The use of routines that exploit the sparsity of the Jacobian will significantly reduce computational effort.

The rate form of the equation of state is used in the solution of the pressure field in the thermalhydraulic problem. A rate equation is better suited to the solution procedure for the thermalhydraulics than the use of an equation of state, where the pressure is calculated given the density and enthalpy (or temperature in single phase) of the fluid. The use of an equation of state would require that an iterative procedure be used for the calculation of pressure in order to satisfy the conservation of mass equation.

The time integration of the neutronics and heat transfer equations is separate from the hydraulics equations. The desired accuracy of the time integration will determine maximum step size that can be used in the integration. The greater the rate of change, the smaller the maximum step size. The rate equations for the multigroup neutron flux, delayed neutron precursor concentration and temperature in the fuel and coolant are integrated first, and the hydraulics equations, the fluid velocity and pressure, are integrated second.

While the time integration is in two sections, the temporal interaction is sufficiently frequent that this is effectively a simultaneous solution. In a transient simulation, the rapidly changing neutron kinetics will determine the upper limit of the time step used, and hence the maximum period between interaction of the two processes. This time step is generally well below the upper limit required by the hydraulics and hence the solution of the problems is effectively simultaneous from that perspective. The fuel temperature can change quite rapidly, depending on the fuel type used, and can have a large feedback effect on the neutronics but it is integrated simultaneously with the neutronics eliminating any problems. The solution of the two problems is therefore considered to be effectively simultaneous. Should greater temporal detail be required in order to confirm this, simply reducing the time step will provide that greater detail.

A second order semi-implicit Runge-Kutta method is one of the methods that can be used for time integration. This method was found to be particularly useful in the solution of the neutron kinetics problem. This method requires matrix inversion and is therefore more computationally demanding than an explicit method but it is a robust and reliable method for solving a stiff system (widely varying rates of change) of ordinary differential equations, especially the neutron kinetics problem. A second order semi-implicit Runge-Kutta method requires that matrix inversion be used twice during one time integration. Due to the nature of the particular algorithm used, the matrix is decomposed once and only the solver is used twice. This results in significant savings of computational effort.

The general structure of this document is as follows. The theory outlining the governing equations for the problem described above is presented and developed following a literature review. The required numerical methods and their use are examined. A description of the overall simulation and problems encountered in the computer coding is given. The results of various tests used in the design of the software including tests of the two component simulations, the neutron kinetics and the thermalhydraulics, are presented and discussed. The results of simulated transients using the completed software are given. Some derivations and the software listings are given in the appendices.

2 Literature

The two major areas of classification of reactor simulation programs are reactor physics and thermalhydraulics. There are a few programs that combine the two processes but these are relatively recent. The more important codes in these areas are presented with an emphasis placed on codes that are in general usage. A large volume of information is available on general numerical methods therefore only the algorithms of direct concern are presented.

2.1 Reactor Physics

Reactor physics codes solve the problem of neutron behavior in reactors. These codes have evolved over time to a level where they reliably calculate the flux distribution in the reactor core for a given configuration. There are several classifications used to describe these codes and some fall into more that one category. The reactor physics codes can be divided into three main areas: general physics, which includes lattice and superlattice codes; reactor codes, which include static criticality calculations, fuel management and, xenon transients; and the kinetics codes, which include both direct and approximate methods. The list presented here is not intended to be exhaustive but merely to give examples of some codes that are commonly used in Canada. Unfortunately, information for a number of commercial codes is restricted and hence can not be referenced. However, brief descriptions will be given in these cases. More detailed descriptions of some of the methods mentioned above are presented in Chapter 3.

The most general equation for describing neutron behavior is the time-dependent neutron transport equation. This equation has directional, spatial and time dependence. To attempt to solve this equation over the entire reactor volume would be a computationally enormous task. Approximations are used to render this equation more manageable. Static or time-independent approximations assume a steady state neutron behavior. Diffusion approximations assume that both directional independence of the neutron flux and isotropic scattering and point kinetics assume a constant spatial flux shape. Almost all codes that solve the neutron transport equations are static calculations.

Kinetic or dynamic codes model the temporal variation of the neutron flux. Space-Time kinetics takes into account the spatial effects that influence the dynamic behavior of a reactor. A distinction that is sometimes made is that a kinetic code models only the neutron behavior without temperature feedback while a dynamic code will include some form of temperature feedback. Using this distinction, *the code that is developed here is a space-time dynamic code.*

Nuclear Reactor Analysis[1] and, *Nuclear Reactor Analysis*[2] are two popular general reactor physics text books. *Nuclear Reactor Theory*[3] provides a thorough theoretical basis for solving the neutron transport equation. *Computational Methods of Neutron Transport*[4] gives a good overview of numerical techniques for neutron transport calculations and is quite readable.

2.1.1 General Physics

Transport codes give a more accurate picture of the flux behavior within a reactor but they are expensive to run. Therefore, it would be preferable to simulate only a portion of the reactor. Reactors are usually comprised of an array or lattice of fuel cells. In a large reactor, it would be reasonable to assume that the neutron behavior in one of these lattice cells will give a good representation of neutron behavior in the reactor as a whole. Lattice codes simulate one cell assuming that the reactor is comprised of an infinite array of these cells. A super lattice code will simulate larger sections of the reactor than a lattice code, this is to allow the inclusion of reactivity and control devices in the calculation. Reactor design calculations are performed in a series of steps. The generation of multigroup coefficients, which are parameters in the equations that describe the neutron behavior at various energies in various regions of the reactor, is an important first step in this process. To generate the multigroup cross sections, the flux energy profile must be known. Lattice and super lattice codes are usually used for this purpose.

Certain transport codes may be used to generate cross sections however they are not necessarily limited to this application. *ANISN* and *DOT* are examples of general purpose transport codes that fall into this class. They both use the discrete ordinates method[5] of solution of the transport equation. *ANISN* is a one dimensional spherical code, and *DOT* (discrete ordinates transport) is a two dimensional code. DOT is of course more computationally demanding that ANISN but can handle greater geometric detail. These codes are useful in the areas where diffusion codes are inadequate such as at an interface where there is a large change in cross sections (i.e. close to control rods). The output of these codes can be used to generate parameters for the boundary conditions of diffusion codes when, for example, control rod worth is being determined.

Lattice Codes

The *Winfrith Improved Multigroup Scheme* (WIMS)[6] is a general code for lattice cell calculations. Its ability to accept as wide range of geometries and fuel arrangements for either fast or thermal reactors makes this quite a general code. The basic cross section library comes with 14 fast groups, 13 resonance groups

---

1 Duderstadt and Hamilton (1976)
2 Henry (1975)
3 Bell and Glasstone (1970)
4 Lewis & Miller (1984)
5 see Section 3.2.2.
6 Askew *et al.* (1964)

and 42 thermal groups. Temperature dependent thermal scattering matrices for a variety of scattering laws are included in the library for the principal moderators. There is an alternative given between the integral and differential solutions of the transport equation. The differential solution is discrete ordinates usually using a S_4 approximation. The integral solution is achieved by the generation of collision probabilities with the resulting equations being integrated by a method that is dependent on the geometry. A number of methods are available to modify the infinite lattice results obtained from the main transport calculation to include leakage effects in a finite reactor. The focus of this code is detailed lattice cell calculations for use in designing reactors. The effects of different lattice geometries and fuel compositions can be studied and by using WIMS in the supercell mode, the effects of the cell environment can be included, which makes this code ideal for the generation of group parameters for use in a diffusion simulation.

WIMS has been modified at Chalk River[7] to become what is known as *WIMS-AECL*[8]. This is an important workhorse code for the generation of multigroup parameters.

POWDERPUFS (restricted) code is a lattice code based upon empirical data for natural uranium. This code is in common usage for fuel studies in CANDU reactors. The integral transport equation is used to solve for the neutron flux.

Superlattice Codes

SHETAN[9] is a three dimensional neutron transport code. It is based on the block method of solving the integral transport equation. The code uses mixed rectangular and cylindrical coordinates, cylindrical fuel channels and reactivity devices can be accurately modeled within a rectangular cell. The block method is a combination of the collision probability (CP) method and the interface current (IC) method.[10] The system is divided into blocks, each block is subdivided into regions and surface subdivisions. All blocks are linked by interblock boundary currents. The main advantage of this method is that the collision probabilities are calculated for only a few block types since blocks of identical geometry and composition have the same collision probability matrix. The numerical precalculations of the coupling matrices (the elements of which are the collision probabilities) require expensive multidimensional quadrature. For an increase in the number of regions, there is a rapid increase in storage requirements and computational effort. Fewer large regions can be used but, there is an associated loss of accuracy.

MULTICELL[11] is a supercell code that combines transport and diffusion theory. Its purpose is the precalculation of cross-sectional properties of fuel and reactivity devices for use in reactor simulations. The output consists of flux weighted cross sections for the supercell and changes in these cross sections due to the movement of the reactivity devices. In materials where the absorption rate is small with respect to the scat-

---

7 J.V. Donnelly (1985)
8 J.V. Donnelly (1986)
9 Chow (1980)
10 see section 2.2.2, Directional Dependence - Transport Methods
11 Dastur A.R. & D.B. Buss (1983)

tering rate (D_2O, Zircaloy) diffusion theory is used. Inside regions where the absorption rate is high (fuel channel, reactivity device) integral transport theory is used. The result of the integral transport calculation is converted into boundary conditions for the diffusion calculation. The finite difference approximation is used in the diffusion calculation, with the difference equations being solved using a successive over relaxed Gauss-Seidel iteration.

2.1.2 Reactor Codes

CHEBY (restricted) is a system of three dimensional diffusion theory programs for heavy water reactors. It calculates the critical flux and power distribution by solving the two group diffusion equation in three dimensional Cartesian geometry. The finite difference approximation is used to solve the neutron diffusion equation. All fission neutrons are born into the fast group. Some effects that are treated explicitly are: the presence of reactivity devices, changes in fuel composition due to fuel burnup and fission products and, reactivity feedback due to changes in temperature and density of lattice components. Neutron leakage at the reactor surface is accounted for by using precalculated current to flux ratios. A Gauss-Seidel iteration using point-wise overrelaxation is used to solve the set of algebraic equations. An unusual feature of the code is that it is tailored to the neutronics of natural uranium, heavy water reactors (CANDU).

Some related and auxiliary programs are: XEMAX, CHEBYXEMAX, CHEBYDISCRETE, FMDP, MATMAP, POWDERPUFS, MULTICELL and, CERBERUS.

CHEBYXEMAX is used to simulate xenon transients. The solution over time of the spatially dependent xenon and iodine equations are expressed analytically, assuming a constant neutron flux over the time interval. Xenon effects in the diffusion equation are represented by a spatially dependent increment in the thermal neutron absorption cross section. FMDP is used to simulate fuel management. The diffusion equations are coupled with the equations that give the variation of cross sections and diffusion coefficients with irradiation. CERBERUS[12], a transient diffusion code, uses the steady state solution of the equations as an initial condition. POWDERPUFS and LATREP[13] are used to calculate the two group material cross sections for the lattice and the reflector. Supercell codes such as MULTICELL[14], GETRANS[15] and, SHETAN[16] are used to obtain cross sections for regions containing reactivity devices and reactor structures. CHEBY has a historical relationship with the various codes. It was the starting point for their development.

12 Kugler & Dastur (1976)
13 Phillips & Griffons (1971)
14 Dastur & Buss (1983)
15 Roshd & Chow (1978)
16 Chow & Roshd (1980)

Static Calculation - Reactor Criticality

3DDT[17] is a three dimensional (Cartesian and cylindrical coordinates) multigroup diffusion theory code with no upscattering. The code features calculation of the effective neutron multiplication factor, K-effective, and criticality searches on reactor composition, or time absorption (α) by means of either the normal or adjoint flux equations.

The criticality equation is[18]

$$\vec{\nabla} \cdot D_g \vec{\nabla} \Phi_g - \Sigma_{r_g} \Phi_g + \frac{\chi_g}{k_e} S_g = 0$$

where Φ_g is the group flux, D_g is the group diffusion coefficient, Σ_{r_g} is the removal cross section, χ_g is the fission neutron production term and k is the neutron multiplication factor. The source terms is

$$S_g = \sum_{g'=1}^{G} [\nu \Sigma_{f_{g'}} + \Sigma_{g g'}] \Phi_{g'}$$

Σ_{f_g} is the fission cross section, ν is the number of neutrons produced per fission and $\Sigma_{g g'}$ is the scattering cross section. The standard time dependent multigroup diffusion equation is:

$$\frac{1}{v_g} \frac{\partial \Phi_g(\vec{r}, t)}{\partial t} = \vec{\nabla} \cdot D_g \vec{\nabla} \Phi_g - \Sigma_{r_g} \Phi_g + \chi_g S_g(\vec{r}, t)$$

The assumption for the alpha search is that $\Phi_g(\vec{r}, t) = \Phi_g(\vec{r}) \exp(\alpha t)$. The time dependent multigroup diffusion equation now becomes:

$$\vec{\nabla} \cdot D_g \vec{\nabla} \Phi_g - \left(\Sigma_{r_g} + \frac{\alpha}{v_g} \right) \Phi_g + \chi_g S_g = 0$$

The spatial difference equations are obtained by integrating the group diffusion equations over the volume associated with each mesh point. The surface leakage terms are obtained by integrating the flux gradients at the boundaries. The flux gradients are approximated by the difference between the two adjacent flux values. Reflective, vacuum and, periodic boundary conditions are available. Convergence of the difference equations is accelerated by group rebalancing and successive overrelaxation.

Citation[19] is a multigroup diffusion theory code that was developed at Oak Ridge Laboratories. The problem may be solved using a number of different geometries and dimensions. The difference equation is generated using the explicit finite differencing approximation in space and time. The neutron flux eigenvalue problem is solved by direct iteration to determine the multiplication factor (K-effective) or the nuclide densities required for a critical system. The code is designed for depletion (burn up) and reactivity worth calculations. Its flexible nature makes this code quite useful and hence its popularity. The limits of the problem that may be run using this code are determined by machine memory capacity and execution time.

---

17 Vigil (1970)

18 See Appendix B for the definition of the multigroup parameters.

19 Fowler *et al.* (1971)

2.1.3 Reactor Kinetics

The object of reactor dynamics is the accurate prediction of the time dependence of the neutron density. Stewart (1973) states that reactor dynamic analysis must be performed for three reasons: evaluation of reactor stability during normal operation, evaluation of the adequacy of operable safety systems and, the analysis of reactor accidents. Reactor transients induced by localized perturbations can produce both spectral and spatial changes in the neutron distribution. Point kinetics is not adequate for the analysis of such perturbations and a coupled space-energy-time dependent model is required.

The numerical methods of solution of the time dependent neutron diffusion equation can be placed in two general categories, direct methods and approximate methods. Analytic solutions have been achieved but these are restricted to simple models that are used primarily for comparison to numerical solutions. Lee and Rotter (1986) present an analytic solution of the multigroup space-time kinetics equation for one dimensional multiregion slab and spherical geometries. Yasinsky and Henry (1965) compare solutions of the two group space-time diffusion equation with point kinetics, the adiabatic approximation and, the space-time synthesis method for two slab reactors. The numerical methods are briefly described in the following sections.

Methods of Solution of the Time-Dependent Neutron Diffusion Equation

A. Direct finite differencing in space and time

1) GAKIN method of implicit integration
2) TWIGL - cyclic Chebychev polynomial method
3) Alternating Direction Implicit (ADI)
4) Semi-Implicit Runge-Kutta: the method employed here

B. Approximation methods

1) Point kinetics
2) Space-Time synthesis
3) Modal Methods
4) Nodal methods
5) Factorization approximations
   i) Adiabatic method
   ii) Quasistatic method
   iii) Improved Quasistatic method

Stacey (1969) gives an overview of the methods employed in space-time kinetics. The most popular methods in present day usage are the Improved Quasistatic (IQS) method, Modal methods and Nodal methods. A review of the literature reveals numerous recent papers in the area of Nodal methods, applied to both multigroup diffusion and transport calculations.

2.1.3.1 Direct Methods

Direct finite difference methods are the most straightforward approach to the solution of space-time problems and usually give the most accurate results, and are useful for providing a benchmark. They are also the most computationally demanding of the methods of solution, although increasing computational power is reducing this liability. Direct finite difference methods all use very similar spatial differencing. The time dependent multigroup diffusion equations are finite differenced in space, in whatever dimension or coordinate system that is desired, to arrive at the time dependent problem,

$$\frac{d}{dt}\overline{\Psi}(t) = \overline{A}\,\overline{\Psi}(t)$$

where $\Psi(t)$ is the data vector that contains all the time dependent grid points, and A is a matrix, sometimes referred to as the Jacobian matrix, that is commonly written as a matrix of block matrices.

The method of solution of the time dependent problem is the main area where the simulations that use direct finite difference vary. The solution of this linear problem is a major area of effort. One of the main problems that is associated with the solution of this matrix equation is the stiffness of the A matrix. It has been shown by numerical studies[20] that the smallest eigenvalues can be of the order of $-\nu_g\Sigma_{ag}$, which are approximately -10^{-4} for thermal neutrons and approximately -10^{-8} for fast neutrons. An explicit solution of the time dependent problem would require very small time steps, $\Delta t < 10^{-8}$ seconds, to remain numerically stable. If it is assumed that the time derivative of the epithermal flux is the same as the thermal flux (equal derivatives), larger time steps, of the order of 10^{-4}, may be used. This stability problem can be avoided by the use of implicit techniques. These methods allow larger time steps, but require the inversion of a matrix and hence are computationally expensive. It is not uncommon to solve for only the asymptotic behavior of the flux and ignore the very short time behavior.

Considerable work has been done on direct methods: Nahavandi & Von Hollen (1964) and, Agresta & Borst (1960). Saphier (1972) and Kelber *et al.* (1961) have used coupled analog computers to solve the time dependent portion of the multigroup diffusion equation while using the finite difference approximation for the spatial dependence. Alcoufe & Albrecht (1970) use a set of trial functions to reduce the number of discrete mesh points required and hence the computational effort. Andrews & Hansen (1968) and Buckner & Stewart (1976) use a transformation that assumes exponential time behavior of the flux and precursors over each time step. This method is shown to reduce truncation error and enhance numerical stability.

20 L.R. Foulke (1966)

GAKIN

Stacey (1969) discusses the *GAKIN* method for implicit integration of the time dependent problem. The GAKIN code itself[21] is a one dimensional kinetics code. The matrix A is factored into the form $A = L + U + \Gamma + D$ where L contains all the submatrices below the block diagonal, U contains all the submatrices above the block diagonal and both Γ and D are block diagonal matrices. The block diagonal matrices are factored into two matrices, $A_{ii} = \Gamma_{ii} + \nu_i D_i$, where D_i represents the coupling among the mesh points due to the diffusion term. All the block matrices are diagonal except D_i. The time dependent equation now has the form

$$\frac{d}{dt}\Psi(t) - \Gamma\Psi = (L + U)\Psi + D\Psi$$

This equation is then integrated over the interval $(0, \Delta t)$

$$\Psi(t_j + 1) = \exp(\Gamma\Delta t)\Psi(t_j) + \int_0^{\Delta t} e^{\Gamma(\Delta t - t')}(L + U)\Psi(t_j + t')dt'$$

$$+ \int_0^{\Delta t} e^{\Gamma(\Delta t - t')}D\Psi(t_j + t')dt'$$

The assumption is made $\Psi(t_j + t') = \exp(\omega_i t')\Psi(t_j)$. With some manipulation, the difference equation in time will appear as, $F_1\Psi(t_j + 1) = F_2\Psi(t_j)$, where F_1 and F_2 are functions of L, D, U and Γ. It can be shown that the algorithm is numerically stable for all real ω_i and Δt. If ω_i is the largest eigenvalue of A, then $\exp(\omega_i \Delta t)$ is an eigenvalue of $F_1^{-1}F_2$. For constant reactivity, the method yields the exact eigen-solution and asymptotic behavior.

TWIGL

TWIGL[22] is a two dimension (rectangular or cylindrical geometry), two group space-time neutron diffusion simulation with temperature feedback. The two group diffusion and delayed precursor equations are differenced in both space and time. The precursor terms are centrally differenced in time, $C_i \approx (C_i^{j+1} + C_i^j)$ and the flux terms are variably differenced in time[23], $\sigma\phi \approx \theta\sigma^{j+1}\phi^{j+1} + (1 - \theta)\sigma^j\phi^j$, $(0 < \theta \leq 1)$, over the time interval (t_{j+1}, t_j), where σ represents all the cross sections and diffusion terms, and j indicates time step. The resultant matrix difference equation is divided into two couple matrix equations, one for each group

$$A_1 = S_1 + B_2\Phi_2$$

$$A_2 = S_2 + B_1\Phi_1$$

The cyclic Chebychev polynomial method[24] is then used to solve the above matrix equations.

---

21 K.F. Hansen & S.R. Johnson, (1967)

22 Yasinsky, Natelson & Hageman, (1968)

23 see section 5.4, The Solution of ODEs.

24 *Matrix Iterative Analysis*, Varga, p.150

ADI - Alternating Direction Implicit

The solution of the vector equation given at the beginning of the section usually proceeds in a sequential manner, always moving in the same direction (*i.e.* the solution starts at the beginning of the first row, proceeding to the beginning of the second row, etc.). Convergence is often improved by following the first sequence in the row direction with a second in the column direction. The complete iteration consists of a first half in the row direction followed by a second in the column direction. Such methods are designated *alternating direction implicit* (ADI). The first of these was developed by Peaceman and Rachford (1955).

Hageman & Yasinsky (1969) develop an *ADI*[25] method and compare it to the implicit difference approached used in the TWIGL program in order to analyze the method for accuracy and stability. In model (simple) problems, the ADI method was as accurate as the TWIGL method and much faster computationally. However, numerical comparisons showed that the ADI method is asymptotically unstable for more realistic problems unless extremely small time steps are used. In the comparison, ADI methods were found to be inferior to the TWIGL method for practical problems.

Wight *et al.* (1971) present an algorithm that is based on the *ADI* method that uses an exponential (frequency) transformation as a method of reducing truncation error so that the method becomes usable for practical computations. They emphasis that this is a non-iterative method but requires some matrix inversion. The matrices that are inverted are banded though, and this structure is exploited. Several codes that were in use or were under development were compared : MITKIN, SADI, LUMAC and TWIGL. A related paper presents the development of the code LUMAC and includes a more detailed numerical analysis.[26]

Semi-Implicit Runge-Kutta

The method presented in this work is a direct finite difference method. The difficulty in solving the neutronics and thermal conduction equations is the stiffness of the system. The difficulty in solving the thermalhydraulic equations is their nonlinearity. The time integration of the variables that have the shortest period,[27] the flux and the fuel temperature, is solved using a semi-implicit Runge-Kutta ODE solver. The use of a semi-implicit solver is more computationally expensive than explicit solvers but, it is useful in the solution of stiff systems of variables. The delayed neutron precursors have a much longer period and consequently a less expensive explicit Runge-Kutta solver may be used. A Crank-Nicolson solver is used for the time integration of the thermalhydraulic equations.

---

25 *ibidem*, p.209.

26 Reed and Hansen (1970)

27 The period of a variable is a measure of the exponential characteristics in time of that variable. A short period implies a rapidly changing variable and conversely for a long period.

2.1.3.2 Approximation Methods

Computational effort can be greatly reduced if some assumptions are made about the spatial portion of the space-time kinetics equation. The approximation methods differ in the manner in which the space and time dependent portions of the problem are separated and what assumptions are made about the spatial solution of the problem. Some methods assume a fixed spatial shape and other assume a spatial shape that varies slowly in time.

Point Kinetics

The most popular method obtains only the total power in the reactor based on the implicit assumption of a fixed spatial distribution. The point kinetics equations are derived in any nuclear engineering text. Their standard form is

$$\frac{dn(t)}{dt} = \frac{\rho(t)-\beta}{\Lambda} n(t) + \sum_i \lambda_i C_i(t)$$

$$\frac{dC_i(t)}{dt} = \frac{\beta_i}{\Lambda} n(t) - \lambda_i C_i(t)$$

where $\rho(t)$ is the reactivity, λ is the mean generation time, $C_i(t)$ is the i th neutron precursor density and λ_i is the associated decay constant, and β_i is the fission yield fraction for the i th precursor ($\beta = \sum \beta_i$). The maximum number of delayed (neutron precursor) groups used is six and some times only one effective delayed group is used. If photoneutrons are included in the equation, they appear as more delayed neutrons. Delayed neutrons play an important role in the temporal behavior of the neutronics therefore their inclusion is essential.

Space-Time Synthesis

Space-Time synthesis expands the neutron flux as a sum of products of known spatial functions and unknown time dependent coefficients. The group flux is approximated by

$$\Phi_g(\vec{r},t) = \sum_{k=1}^{K} \Psi_k^g(\vec{r}) T_k(t)$$

where $\Psi_k^g(\vec{r})$ are the known spatial trial functions which are found solving the static or dynamic group diffusion equations. The time-dependent coefficients can be found by weighted residual or variational techniques. Yasinsky (1967) employs a variational principal that is modified to allow the specification of different trial functions for time different intervals during a transient. Fuller and Meneley (1970) apply the weighted residual method to provide several methods that differ in the manner in which the trial (spatial) functions are generated. The methods used to generate the trial functions are: modal expansion, synthesis method, nodal method and, quasistatic method. Their purpose was to show that these methods are all applications of the weighted residual method.

Modal Methods

Modal methods expand the time-dependent neutron flux into a series of time-independent spatial modes, $\Psi_n(\vec{r},E)$, with time-dependent coefficients, $a_n(t)$

$$\Phi(\vec{r},E,t)=\sum_n \Psi_n(\vec{r},E)a_n(t)$$

It can be seen that these methods are closely related to space-time synthesis. "The modal approach applied to purely space-dependent dynamics problems is often called the *space-time synthesis* method".[28] In an early work, Kaplan (1961) presents a two group modal method. Hishino and Wakabayashi (1965) develop a method that uses the Laplace transform for time to express the solution in the form of a source transfer function.

Foulke and Gyftopoulos (1967) express the multigroup diffusion equation in the form

$$\frac{\partial\Psi(\vec{r},t)}{\partial t}=[H(\vec{r},t)]\Psi(\vec{r},t)+S(\vec{r},t)$$

where $\Psi(r,t)$ is a data vector that contains the neutron and the precursor concentrations, $[H(r,t)]$ is a $K\times K$ matrix operator that consists of all the production and destruction operators and, $K=G+I$ where G is the number of energy groups and I is the number of precursors. The eigenvalue equation that is used to determine the eigenvalues ω_{mk} and the eigenvectors $\Psi_{mk}(r)$ is

$$[H_0(\vec{r})]\Psi_{mk}(\vec{r})=\omega_{mk}\Psi_{mk}(\vec{r})$$

where $[H_0(r)]$ is a reference steady state condition. The adjoint eigenvalue equation is:

$$[H_0^*(\vec{r})]\Psi_{mk}^*(\vec{r})=\omega_{mk}^*\Psi_{mk}^*(\vec{r})$$

The solution vector $\Psi(r,t)$ is expanded in a finite series,

$$\Psi(\vec{r},t)=\sum_{k=1}^{K}\sum_{m=1}^{M}A_{mk}(t)\Psi_{mk}(\vec{r})$$

This series is substituted into the multigroup equation and the equation is multiplied by the adjoint eigenvector. When this expression is integrated over the entire reactor, the orthogonality of the eigenvectors gives rise to a set of coupled ODEs,

$$\frac{d\overline{A}}{dt}=diag[\omega]\overline{A}+\overline{PA}+\overline{S}$$

where A is a vector that contains all the time dependent coefficients. The source term is expanded in a series of time-dependent functions,

$$S_{mk}(t)=\frac{<\Psi_{mk}^*(\vec{r}),S(\vec{r},t)>}{<\Psi_{mk}^*(\vec{r}),\Psi_{mk}(\vec{r})>}$$

and the elements of the perturbation matrix P are

$$P_{ij}(t)=\frac{<\Psi_{ij}^*(\vec{r}),[H(\vec{r},t)-H_0(\vec{r})]\Psi_{mk}(\vec{r})>}{<\Psi_{ij}^*(\vec{r}),\Psi_{ij}(\vec{r})>}$$

---

28 Ott & Neuhold p.292 (1985)

where $\mu = (n-1)K + j$ and $\gamma = (m-1)K + k$. The inner product symbol $<\bullet,\bullet>$ denotes spatial integration over the entire reactor. The implied orthogonality relationship is

$$< \Psi^{*}_{nj}(\vec{r}), \Psi_{mk}(\vec{r}) > = 0 \qquad \text{for nj} \neq \text{mk}$$

$$\neq 0 \qquad \text{for nj} = \text{mk}$$

Foulke and Gyftopoulos go on to say that the natural modes come in clusters of K, that have components of similar shape. The modes become more oscillatory in space as the index m increases. The results of some numerical experiments are given where the method is compared against a direct finite difference method.

SMOKIN[29] (Spatial Modal Kinetics) is an example of a modal code that is in common usage in the Canadian Nuclear Industry. SMOKIN is intended for use on CANDU pressurized heavy-water reactor neutronics problems and being a predominantly thermal reactor, a one neutron energy group assumption is made. Coupled neutron kinetics equations incorporating the effects of delayed neutrons, feedback and xenon are solved.

Nodal Methods

Nodal methods visualize the reactor as a relatively small number of coupled regions. The derivation of the nodal equations begins with the neutron transport equation. This equation is integrated over all directions and over discrete energy groups (multigroup approximation) to yield

$$\frac{1}{v_g}\frac{\partial}{\partial t}\Phi_g(\vec{r},t) + \vec{\nabla}\cdot\vec{J}_g(\vec{r},t) + \Sigma_{tg}(\vec{r})\Phi_g(\vec{r},t) = \sum_{g'=1}^{G}[M_{gg'}(\vec{r}) + \Sigma_{gg'}(\vec{r})]\Phi_{g'}(\vec{r},t)$$

where g denotes the energy group, $M_{gg'}$ is the fission neutron production term, $\Sigma_{gg'}$ is the scattering term, Σ_{tg} is the total cross section, v_g is the group velocity, and $J_g(r,t)$ is the neutron current. Subtract $\Sigma_{gg}\Phi_g$ from both sides and integrate over the nodal volume to yield the *nodal balance equation*,

$$\frac{1}{v_g}\frac{\partial}{\partial t}\Phi_g^{ijk}(t) + \frac{1}{V_{ijk}}\sum_{n=1}^{6}\int_{S_n}\vec{J}_g(\vec{r},t)\cdot\vec{n}\,ds + \Sigma_g^{ijk}\Phi_g^{ijk}(t) = \sum_{g'=1}^{G}M_{gg'}^{ijk}\Phi_{g'}^{ijk}(t) + \sum_{g'\neq g}\Sigma_{gg'}^{ijk}\Phi_{g'}^{ijk}(t)$$

where

$$V_{ijk}\Phi_g^{ijk}(t) = \int_{V_{ijk}}\Phi_g(\vec{r})\,dV$$

and V_{ijk} is the nodal volume. All nodal methods use the above equation but particular methods differ as to how the face integrated currents are related to the volume averaged fluxes.

Henry (1986) presents the three dimensional, two group, Cartesian geometry nodal code *QUANDRY*. He describes the structure of the matrices involved in a time-dependent nodal simulation and derives the point

29 Gold (1990)

kinetics equations, as the governing equation for time behavior, from the time dependent nodal equations. The results of a simulation using this method are compared to the results of a simulation based on the Improved Quasistatic method.

Lawrence (1986), in a review paper, examines the more recent nodal diffusion methods, which are characterized by the systematic derivation of spatial coupling relationships that are entirely consistent with the multigroup diffusion equation. These ideas are similarly applied to the neutron transport equation. These more recent methods, referred to as *consistently formulated*, can be viewed as true coarse mesh approximations of the neutron diffusion equation and thus can be expected to converge to the exact solution in the limit of zero mesh spacing.

The spatial detail increases with the number of nodes used but, there is an associated increase in computational effort. Nodal methods which include both diffusion and transport methodologies, are quite flexible and can be applied to a variety of problems thus assuring their continued popularity.

Factorization Methods

Ott and Meneley (1969) review the factorization methods for approximating the time dependent multigroup diffusion equation. This equation can be written using operator notation

$$[-M + F_p]\Phi(\vec{r}, E, t') + S_d[\Psi(\vec{r}, E, t')] = \frac{1}{v}\frac{\partial}{\partial t}\Psi(\vec{r}, E, t)$$

where M is the removal and scattering operator and F_p as the prompt fission operator. The delayed neutron source $S_d[\Phi(\vec{r}, E, t')]$ is a convolution integral over the flux history. The flux is factored into an amplitude function, $\phi(t)$, and a shape function, $\psi(\vec{r}, E, t)$

$$\Phi(\vec{r}, E, t) = \phi(t)\psi(\vec{r}, E, t)$$

$\phi(t)$ contains the main time dependence and the time dependence of $\psi(\vec{r}, E, t)$ has to account for only the relatively slow space-energy variations. According to Henry (1958)

$$\int\int \frac{\psi^*(\vec{r}, E, 0)\psi(\vec{r}, E, t)}{v} d\vec{r}dE = \text{constant}$$

fulfills the requirements for the constraint conditions and facilitates the transition to the point kinetics equation. The time-dependent multigroup diffusion equation is split into two equations, one for the shape function and one for the amplitude function. The $\phi(t)$ equation becomes the point kinetics equation

$$\frac{d\phi(t)}{dt} = \frac{\rho(t) - \bar{\beta}(t)}{\Lambda(t)}\phi(t) + \sum_k \lambda_k C_k(t)$$

The shape equation becomes

$$[-M + F_p]\psi(\vec{r}, E, t') + \frac{S_d[\phi(t') \cdot \Psi(\vec{r}, E, t')]}{\phi(t)} = \frac{1}{v}\left[\frac{d\phi}{dt} \cdot \frac{\psi(\vec{r}, E, t)}{\phi(t)} + \frac{\partial}{\partial t}\psi(\vec{r}, E, t)\right]$$

The *Improved Quasistatic Method* solves the complete shape equation after the time derivative of the shape function is replaced by a first order backward difference,

$$\frac{\partial}{\partial t}\psi(\vec{r},E,t) \approx \frac{\psi(\vec{r},E,t)-\psi(\vec{r},E,t-\Delta t)}{\Delta t}$$

where $t-\Delta t$ is the time of the last calculation. Δt can be quite large as the shape function is slowly varying.

The *Quasistatic Method* takes advantage of the fact that the time variation of the shape function is of lesser importance than the time variation of the amplitude function. The time derivative of the shape function is therefore neglected.

The *Adiabatic Approximation* makes two additional simplifications. The shape of the delayed neutron source is not distinguished from the shape of the prompt source and it neglects both time derivative terms in the shape equation. This method has been shown to describe the major part of the spatial effects in kinetics.

Some further work has been done on the quasistatic method by Devooght & Mund (1980) where they present the *Generalized Quasistatic Method* and look at the mathematical basis of the quasistatic method. There have been other factorization methods presented, such as Chao (1982), but with the success of the IQS method there is little need.

The Quasistatic methods have proven to be adequately accurate and relatively inexpensive for routine analysis of multidimensional problems. There has been an emphasis on the use of the improved quasistatic (IQS) method for the analysis of CANDU reactors.[30] The code *CERBERUS*, and the closely related code *CERKIN*, solve the two-group time dependent neutron diffusion equation in one, two, or three dimensions using the IQS method.

2.2 Thermalhydraulics

The conservation equations for mass, momentum and energy, along with the equation of state are used to calculate the variables of interest: density, velocity, enthalpy (or temperature)[31], and pressure. The manner in which these equations are used depends upon the problem that is being solved. A problem in which density variation due to thermal expansion is the forcing term in the momentum equation is a natural convection problem. The pressure field may be implicitly assumed in such a problem and need not be explicitly calculated. Forced convection assumes that an external force (pressure) is the forcing term in the

30 McDonnel *et al.* (1977)

31 Temperature and enthalpy are interchangeable in the case of a single phase fluid. One may easily be calculated if the other is known. Their relationship is more complex in the case of two phase flow.

momentum equation. The pressure field is explicitly calculated, if it is not given, in such a problem[32]. Natural convection problems may be treated as forced convection problems (the pressure is explicitly calculated) but the converse is not true. A standard benchmark problem, free convection in a unit cell, was used to check the steady state accuracy of the thermalhydraulic simulation. The benchmark problem is a free convection problem but the method used here explicitly calculates the pressure.

The calculation of the pressure field in the solution of incompressible flow problems is numerically difficult. A method of calculating the pressure that uses the rate form of the equation of state is presented here. This form of the equation of state is well suited for use with the dynamic (unsteady or time-dependent) form of the conservation equations (mass, momentum and energy).

Solutions to the incompressible Navier-Stokes equations can been placed in four general categories: the vorticity/stream-function method, the projection method, methods which involve coupling between the momentum and continuity equations and the artificial compressibility method. Of these methods, the method of artificial compressibility is most closely related to the method presented here.

The vorticity stream/function method[33] solves the vorticity transport equation which is constructed by taking the curl of the momentum equation. The terms containing pressure may be eliminated by using the components of the equation of motion. The method requires the use of vorticity boundary conditions, which are difficult to implement, and an additional calculation is required if the pressure is desired.

The projection method[34] is a fractional step method in which an intermediate velocity and pressure are calculated. The intermediate velocity and pressure are then corrected sequentially by the pressure gradient and the divergence of the intermediate velocity (continuity equation) respectively. New values for the pressure and velocity are obtained until the divergence of the velocity vanishes. The SIMPLE method[35] and related methods are of this variety. The pressure correction equation is the basis of these methods, the derivation of which requires the use of approximate forms of the momentum equations and the continuity equation.

In one method that couples the continuity and momentum equations, the divergence operator is applied to the momentum equation resulting in a Poisson equation for pressure[36]. The pressure field is calculated using this equation so as to assure that the rate of change of the velocity divergence vanishes everywhere. This is an explicit method in which the momentum and Poison equation for pressure are solved separately.

---

32 The vorticity/stream-function method is the exception, where the pressure has been eliminated from the governing equations.
33 Gosman A.D., W.M. Pun, A.K. Runchal, D.B. Spalding & M. Wolfstein (1969)
34 Chorin A.H. (1968)
35 Patankar S.V. & D.B. Sterling (1972)
36 Harlow F.H. & J.E. Welch (1965)

A semi-implicit scheme in which the viscous terms of the Navier-Stokes equation and the pressure are solved implicitly is presented by Moin and Kim (1980). The continuity equation is used to solve for pressure. A pseudo-spectral formulation for a particular problem is given for this scheme.

An artificial equation of state $P = \delta\rho$, is the basis of the method of artificial compressibility[37]. This method differs from the above method in that the continuity equation is not satisfied until a steady state solution is reached. The *artificial compressibility*, δ, defines an artificial speed of sound, $c=1/\delta^{1/2}$. The value of δ used is that which will cause the method to reach steady state in the least amount of time. The equation for the rate of change of pressure,

$$\frac{\partial P}{\partial t} = -\delta\vec{\nabla}\cdot\vec{V}$$

is arrived at by differencing the artificial equation of state with respect to time and substituting the continuity equation for the rate of change of density. Chorin uses central differences in both space and time in an implementation designed for steady solutions. Soh (1987) presents a method using an ADI finite-difference scheme for steady state flows. A method for unsteady solutions is presented by Soh and Goodrich (1988). Their choice of the compressibility parameter, δ, is based on an eigenvalue calculation for a one dimensional problem. The compressibility parameter is chosen such that the eigenvalues are closest to the wave propagation velocity. They employ the Crank-Nicolson method for time integration.

A verification of the steady state convergence of the rate form of the equation of state is presented here. A well known comparison problem[38] is used for this verification. The choice of the compressibility parameter for optimal steady state convergence is explored.

The time dependent conservation of mass, momentum and energy equations are (respectively)

$$\frac{\partial\rho}{\partial t} + \vec{\nabla}\cdot\rho\vec{V} = 0$$

$$\frac{\partial(\rho\vec{V})}{\partial t} + \vec{\nabla}\cdot\rho\vec{V}\vec{V} = \rho\vec{g} - \vec{\nabla}P + \vec{\nabla}\cdot\bar{\bar{\tau}}$$

$$\frac{\partial}{\partial t}(\rho h) + \vec{\nabla}\cdot(\rho h\vec{V}) = \vec{\nabla}\cdot\vec{q} + S(\vec{r},t) + \bar{\bar{\tau}}:\vec{\nabla}\vec{V} + \frac{\partial P}{\partial t} + \vec{V}\cdot\vec{\nabla}P$$

where $\{\rho, \vec{V}, h, P\}$ is the density, velocity, enthalpy, and pressure, respectively and $\{\vec{g}, \vec{q}, \bar{\bar{\tau}}\}$ is gravity, the heat flux, and the shear stress tensor, respectively, and $S(\vec{r},t)$ is the internal heat source term (see Chapter 4 and Appendix A for a more detailed explanation). The equation of state can simply be written in a form where pressure is a function of density and enthalpy, $P = \pi(\rho, h)$.

---

37 Chorin A.H. (1967)

38 De Vahl Davis G. & I.P. Jones (1983), and De Vahl Davis (1983)

If the above conservation equations are integrated over an arbitrary volume, the result is a system of equations that can be represented by a system of nodes and links. This is the nodal form of the conservation equations. The time dependent conservation of mass, momentum and energy equations, written in nodal form are, respectively,

$$\frac{d}{dt}M = \sum_{\text{input}} W - \sum_{\text{output}} W$$

$$\frac{d}{dt}W = f(P, W, t)$$

$$\frac{d}{dt}H = \sum_{\text{input}} hW - \sum_{\text{output}} hW + Q$$

where $\{W, H, P, Q\}$ is the mass flow rate, total enthalpy, pressure, and total heat flux respectively. The conservation of momentum equation is a complicated nonlinear equation and is given in a simplified form here. The equation of state is now a function of the total energy and the total mass, $P = \pi(U, M)$. A total mass, total internal energy, and press re is defined at each node, while a mass flow rate is defined for each link connecting the nodes.

A popular method of solving the nodal equations is that developed by Porsching.[39] This is the basis for the thermalhydraulic code *FIREBIRD*.[40] The equation of state is solved using an iterative method, such as Newton-Raphson. All the variables for all the nodes and links, with the exception of the pressure, are placed into a data vector. A Jacobian for this system of equation is derived and used in the implicit integration of the equations. This implicit solution gives the method much better numerical stability than any explicit method, hence large time steps are possible. Implicit methods are more computationally expensive though and any real savings in computer time will be made only if large time steps are used. FIREBIRD is a code that was designed to model a complete thermalhydraulic system, such as the primary coolant loop in a reactor and, as such it has modules for such components as pumps.

A good introduction into the basic conservation laws that are involved in thermalhydraulics is given in *Basic Equations for Thermalhydraulic Analysis*.[41] This covers the basic conservation equation and its application to the conservation of mass, momentum and energy. The primary focus of this paper is the macroscopic form of the conservation equations such as is normally used in the simulation of reactor thermalhydraulics. *Transport Phenomena*[42] gives a very extensive account of the equations involved in heat and mass transport. This book gives a good account of all the different forms of the conservation equations. *Computational Fluid Mechanics and Heat Transfer*[43] is focused on the problems of numerical

39 T.A. Porsching, J.H. Murphy & J.A. Redfield (1971)
40 M.R. Lin *et al.* (1979)
41 Garland (Engineering Physics internal document)
42 Bird *et al.* (1960)
43 Anderson, Tannehill and Pletcher (1984)

solution of the thermalhydraulic problem. This is quite a general reference that covers PDEs, their classification, the application of finite difference methods to their solution and, some of the problems that are more specific to heat transfer and fluid flow.

2.2.1 The Rate Form of the Equation of State

The equation of state is a function of density and enthalpy, $P = \Pi(\rho, h)$. If the equation of state was differentiated with respect to time the result would be

$$\frac{\partial P}{\partial t} = \left(\frac{\partial P}{\partial \rho}\right)\frac{\partial \rho}{\partial t} + \left(\frac{\partial P}{\partial h}\right)\frac{\partial h}{\partial t}$$

This is the rate form of the equation of state. The terms in brackets in the above equation are replaced by constants which are periodically calculated

$$\frac{\partial P}{\partial t} = G_1\frac{\partial \rho}{\partial t} + G_2\frac{\partial h}{\partial t}$$

Generally, and especially for incompressible fluids, the parameter G_1 is larger than the parameter G_2. This expression is integrated over an arbitrary volume (see section 6.2) for use with the nodal form of the conservation equations. A method of pressure adjustment is required with the nodal form to alleviate pressure drift.

Such a method of pressure adjustment to prevent drifting away from the true values is given in *The Rate Form of the Equation of State for Thermalhydraulic Systems: Numerical Considerations.*[44] This method reduces error in the steady state pressure. This paper also contains a more involved derivation of the rate form of the equation of state. *Generalized Rate Form of the Equation of State for Thermalhydraulic Systems*[45] as well as the previous paper discuss and compare the two methods of calculation of the pressure, the other method being the more common iterative method mentioned in the previous section. In a system of nodes and links, the data vector will now contain the pressure as well as the other variables mentioned, the total mass, total enthalpy (or internal energy), and the mass flow rate. A good systems perspective is given where the Jacobian matrices that arise when using the nodal form are presented and examined. The system of equations is integrated implicitly in time giving the method good numerical stability. Some numerical results are given demonstrating the robust behavior of the algorithm. The use of this algorithm coupled with the approximation functions as given in *Approximate Functions for the Fast Calculation of Light-Water Properties at Saturation*[46] and *Simple Functions for the Fast Approximation of Light Water Thermo-*

44 Garland and Sollychin (1987)
45 Garland and Sollychin
46 Garland and Hoskins (1987)

dynamic[47] makes for a viable thermalhydraulic simulation code. These approximation functions will approximate various thermodynamic coefficients including G_1 and G_2 that are used in the rate form of the equation of state.

Missing from these papers is a simulation that uses the rate form of the equation of state and is not based on the nodal form of the thermalhydraulic equations. Such a simulation is developed and presented here. The method of adjustment of the pressure, such as that employed in the nodal form to prevent pressure drift, is not required. Pressure is a boundary condition in the simulation and all pressures will be calculated relative to that boundary pressure.

As mentioned above, the G_1 parameter is much larger than the G_2 parameter for incompressible fluids. The thermalhydraulic benchmark simulation ignored the G_2 parameter in the rate form of the equation of state. Substituting for the rate of change of density, the pressure rate equation now appears as:

$$\frac{\partial P}{\partial t} = -G_1 \overline{\nabla} \cdot \rho \overline{V}$$

The rate form of the equation of state now appears very similar to the artificial equation of state used in the method of artificial compressibility. However, the manner in which the equations are derived and the choice of the compressibility parameter is quite different for the two methods.

The derivation of the rate form of the equation of state involves only the equation of state. The continuity equation and the conservation of energy equation are naturally part of this expression, which is more intuitive and involves physical parameters, such as compressibility. The rate form of the equation of state is designed for use in transient simulations where the true compressibility would be used, although other values of compressibility could be used if rapid convergence to a steady state solution is required. If the enthalpy term is ignored, the rate form of the equation of state is very similar to the artificial equation of state. The inclusion of the enthalpy term in the rate form of the equation of state may be of greater use for compressible or two-phase problems, but this is subject to verification.

2.3 Combined Neutronics and Thermalhydraulics

The dynamic behavior of reactors is effected by coolant dynamics, especially in accident situations where the coolant voiding takes place. Combined neutronic - thermalhydraulic simulations to study such scenarios is an emerging area of interest. The point kinetics approximation is frequently used to model the neutron behavior and a macroscopic (lumped parameter) simulation, such as FIREBIRD, for the thermalhydraulic model. *MXSIM* (MAPLE-X simulation) is an example of such an approach. Coolant

47 Garland and Hand (1988)

dynamics will effect the energy and spatial distribution of the neutron flux. A rigorous solution of this problem would require the solution of the time dependent multigroup equations while solving the thermal-hydraulic problem.

The recent trend in industry (Canadian Nuclear Industry - CANDU reactors) is to use just such an approach for accident analysis. While the exact details of these simulations is proprietary, the general details can be described. Typically the reactor core is divided into several representative regions, regions that have similar fuel burn-up and are part of the same coolant flow loop (*i.e.* the channels in these regions are connected to the same inlet and outlet headers). The thermalhydraulic simulation, a nodal approximation, calculates the behavior of the coolant and passes coolant & fuel temperature, and coolant density information to the reactor kinetics simulation. The reactor kinetics simulation, a space-time simulation, uses this information to calculate material properties and passes the regional powers back to the thermalhydraulics. The frequency of this interaction is implementation dependent.

March-Leuba *et al.* (1984) create a dynamic simulation that is based on a point kinetic model of the neutronics and a two equation lumped parameter model of the coolant dynamics of a boiling water reactor (BWR). The primary topic is the nonlinear dynamics of the reactor model they have devised as the heat transfer coefficient in the equations is varied. As the heat transfer coefficient is increased, the phase plot of the excess neutron density versus the excess fuel temperature undergoes bifurcations; initially one limit cycle is seen, then two, then four, etc. Adebiyi (1988), in a critical review of March-Leuba's work, points out errors in the assumptions made while deriving the kinetic equations. March-Leuba *et al.* (1986) provide a more detailed derivation of the model used by March-Leuba but the model effectively remains uncharged.

Younis *et al.* presented a paper at the 1988 CNS Simulation Symposium that discussed a simulation that was the combination of the neutron dynamic code, CERBERUS, and the thermalhydraulic code, FIREBIRD-III MOD1-77. This was a true space-time kinetic simulation with reactivity feedback effects due to the thermalhydraulic behavior.

SPORTS - Special Predictions of Reactor Transients and Stability

SPORTS[48] is primarily considered a thermalhydraulic code but, it contains special features, such as neutron kinetics, that are specific to nuclear reactors and hence is included in this section. SPORTS was developed for two phase stability studies. Incorporated into the simulation are some special effects such as subcooled boiling, a fuel model and, neutron kinetics. A simple finite difference scheme that does not use a staggered mesh is used. One dimensional equations for homogeneous two phase flow are used.

48 Chatoorgoon (1986)

The difference equations are arrived at by integrating the PDEs (the conservation equations) over the spatial intervals of interest. For example, the conservation of mass equation is

$$\frac{\partial \rho}{\partial t}+\frac{\partial(\rho u)}{\partial x}=0$$

where ρ is the fluid density and u is the flow velocity. This equation is integrated from spatial points i to $i+1$,

$$\frac{\Delta x}{2}\frac{d}{dt}[\rho_i+\rho_{i+1}]+(\rho u)_{i+1}^{n+1}-(\rho u)_i^{n+1}=0$$

where the superscript n denotes time step. Forward differences are used to approximate the time derivative. Once this substitution is made the resultant difference equation in space and time is

$$\frac{\Delta x}{2}[(\rho_i^{n+1}+\rho_{i+1}^{n+1})-(\rho_i^n+\rho_{i+1}^n)]+(\rho u)_{i+1}^{n+1}-(\rho u)_i^{n+1}=0$$

The effect of integrating across a spatial interval is analogous to averaging properties in that interval. "This scheme was adopted because of its simplicity and because it was found to be useful".[49]

The solution algorithm for the nodal convergence requires iteration. The three conservation equations and the equation of state for closure are used to solve for the four unknowns, density, velocity, pressure and enthalpy.

SPORTS is equipped with a dynamic fuel model that calculates the center, surface and average temperatures. A radially parabolic temperature distribution is assumed. The parabolic profile is considered valid only in the fuel and not in the sheath where the response time is considered instantaneous. The neutron kinetic model is a point kinetic model with six delayed groups that allows for changes in reactivity due to coolant void, coolant temperature and fuel temperature. The neutron kinetic equations along with the fuel temperature equations are solved in an implicit form to determine the amount of heat transferred to fluid. This value is used to determine mean values of void, coolant temperature and fuel temperature and, their resultant effect on reactivity. The neutron kinetic and fuel temperature equations are solved using this value of reactivity to find a new value of heat production. If the difference between this value of heat production and the previous estimate are not within tolerance, another iteration is undertaken.

SPORTS has been used to simulate small reactors such as the SLOWPOKE and MAPLE reactor. The inclusion of reactor kinetics, a fuel model and, subcooled boiling makes it useful for such tasks.

---

49 *ibidem*, p.55

TANK - Transient Analysis Neutron Kinetics

TANK[50] is a two dimensional, two group space-time reactor kinetics code that was developed at Whiteshell Nuclear Research Establishment. It was designed as a tool to simulate transients in the MAPLE class of research reactors. TANK can be used to analyze a variety of reactivity insertion transients in both UO_2 and metallic fuelled MAPLE reactors. The designated neutron energy groups are fast, $E_n > 0.625$, and thermal neutrons, $E_n < 0.625$. The lattice code WIMS is used to generate the multigroup coefficients. The diffusion code 3DDT is used to calculated the axial power distribution. The kinetics parameters for the control rods and shutdown rods as a function of axial position are determined based on the 3DDT axial power distributions. The improved quasistatic (IQS) method is used to solve the two group space-time diffusion equation. TANK has the capability to handle up to 15 delayed neutron precursors or photoneutron groups but, in general six delayed neutron groups are used.

The effects of coolant temperature and density and fuel temperature on reactivity are simulated. The MAPLE class of light water reactors are undermoderated and as such have a negative coolant density reactivity effect. The fuel temperature reactivity effect is dominated by Doppler broadening of the U^{238} resonance cross sections. This is a negative fuel temperature reactivity effect as is most evident in low enriched fuels. A nodal model is used to determine the average fuel, sheath and coolant temperatures. The cladding-coolant heat transfer coefficient used for steady state is the Dittus-Boelter correlation for single phase turbulent flow. After the start of a transient, the heat transfer package of the SPORT-M[51] thermalhydraulic code is used to determine the cladding-coolant heat transfer coefficients for subcooled and saturated boiling. The production of void near the cladding surface is modelled using the heat transfer package of SPORTS-M. The coolant flow through the core is assumed a constant during transients at a velocity determined from the SPORTS-M simulation.[52] TANK keeps track of the coolant circuit time in order to determine changes in coolant inlet temperature due to changes in coolant outlet temperature. The removal of heat by the heat exchanger is used to determine the coolant inlet temperature at the next circuit. A decaying exponential flow velocity is used to model a pump run-down situation. Fluid velocity dynamics are not modelled but, fluid temperature dynamics are modelled by TANK.

TANK is a true space-time neutron dynamic simulation with fuel temperature and thermalhydraulic reactivity feedback effects. Due to the use of the quasistatic approximation for the neutron kinetics and nodal models for the fuel and coolant temperatures, TANK will not provide as much detail as the model under consideration here. However, TANK has the capacity to simulate whole reactors of the research variety.

---

50 Ellis R.J. (1988)

51 Mills & Shim (1989). Better references for SPORTS-M do exist but, they are restricted. SPORTS-M is only loosely related to SPORTS and is not a derivative of it (Mills).

52 R.J. Ellis personal communication, April 1990.

2.4 Numerical Methods

There is a great deal of literature available on numerical methods. The numerical methods that are specifically applied to either neutronic or thermalhydraulic simulations have been discussed above. Some of the general references that were used are: *Numerical Methods*[53] is a good general reference, *Applied Numerical Methods*[54] has an engineering lean to it and contains some FORTRAN code for some simple sample problems and, *Numerical Recipes in C*[55] which is a recipe book as the name implies.

2.4.1 Solution of Partial Differential Equations

Partial Differential Equations (PDEs) may be solved by a variety of methods. The two basic numerical techniques used are finite difference and finite elements. The finite element technique approximates the solution of an operator equation by a set of local basis functions containing adjustable constants. The region of interest is divided into finite elements, usually triangular, interconnected at nodal points on their boundaries. The basis functions are linearly independent and are known apriori. A linear system problem is setup to solve for the adjustable constants. The last chapter of *Numerical Methods for Partial Differential Equations*[56] is useful as an introduction to the subject. The finite element method is primary designed for use on boundary value problems and is cumbersome for initial value (time dependent) problems and hence, will be discussed no further.

The use of the finite difference method for the solution of PDEs is well documented in the literature. *Numerical Methods for Partial Differential Equations* looks at the solution of the three categories of PDEs (parabolic, elliptic & hyperbolic), methods of evaluating the stability of PDEs and methods of accelerating the convergence of a solution. *Computational Fluid Mechanics and Heat Transfer*, as mentioned above, can be used as a reference for the solution of PDEs in general but has an emphasis on thermalhydraulics.

There are two methods that may be used to transform a PDE into a difference equation. Representing the partial derivatives directly as finite differences is one method. These finite differences are derived using Taylor expansions of the partial derivatives. This form of the thermalhydraulic equations is referred to as the microscopic model and, as the direct finite difference method for the neutronic equations. If the PDE is integrated over a number of finite volumes, the spatial derivatives become differences between the quantities averaged over the volume. This is referred to as the macroscopic model for the thermalhydraulic equations and as a nodal method for the neutronic equations.

---

53 Dahlquist and Björck (1974)
54 Carnahan (1969)
55 Press *et al.* (1988)
56 Ames (1977)

The finite differencing of PDEs will give rise to a sparse linear system problem. Either *natural* or *red/black* ordering of the point variables is used[57]. Natural ordering, where the spatial indices are varied in sequence,

$$\overline{\Psi}(t) = \left[\Psi_{1,1}(t), \Psi_{1,2}(t) \ldots \Psi_{1,n_y}(t) \ldots \Psi_{n_x,n_y}\right]$$

is used in the simulation here. The linear system solver that is used must make use of the resultant sparsity of the system in order to be reasonably efficient. Iterative techniques are more commonly used, due to the properties of the matrices, in the solution of the resultant linear system problem as will be discussed in the next section.

The Method of Lines

The Method of Lines is a numerical method where all but one of the independent variables of a partial differential equation (PDE) are discretized. This is a method of effectively transforming a PDE into a system of coupled ODEs. The independent variable that is not discretized is usually time. This approach is advantageous for two reasons. The solution of the problem in time may be handled in whatever manner is desired: implicit, explicit or semi-implicit. It also permits the derivation of the characteristic time constants of the system, sometimes called the Lyapunov coefficients. These coefficients are dependent on the point of evaluation in the system phase space due to the nonlinear nature of the system. Many variables including nuclide concentrations and temperature determine phase space position. Spatial discretization, which may be done in different coordinate systems and dimensions, is used to elucidate the spatial dependence of the dynamics. Carver[58] has done an extensive exploration of the use of the Method of Lines, and as he points out, it is not a method but a collection of methods.

2.4.2 Solution of Ordinary Differential Equations

Algorithms for the solution of ODEs can be found in any general numerical methods book. Methods of solution of ODEs can be categorized as: single step methods, multistep methods, and predictor-corrector methods. Runge-Kutta methods are a popular class of single step methods for the solution of ODEs and within this class the classical forms are most commonly used[59]. The simulation here uses R-K methods because of their good truncation error properties and numerical efficiency. This is covered more extensively in the chapter on numerical methods.

2.4.3 Numerical Linear Algebra

The solution of a linear system is probably the most common numerical problem and the available literature on the subject is vast. Both finite difference and finite element methods of numerical simulation

---

57 L.A. Hageman & D.M. Young, (1981) p.13

58 M.B. Carver & H.W. Hinds (1978), Carver (1979), Carver (1981).

59 Lapiudus & Seinfeld p.46

require the use of linear system solvers. There are three types of methods of solution of linear system: direct, iterative, and semi-iterative. The method that you choose to use will depend on the properties of the matrix involved. Iterative methods are preferred for the solution of large sparse linear systems, such as those that arise from the finite differencing of PDEs, because they can take advantage of the sparsity of the matrix and tend to be self correcting. If the properties of the matrix involved are unknown or, are not suitable to iterative methods, direct methods may be preferable. Semi-iterative methods may be used for sparse matrices that are not well suited to iterative methods (i.e. matrices that are symmetric positive definite but not diagonally dominant).

Some references on the subject of numerical linear algebra are: *A Handbook of Numerical Matrix Inversion and Solution of Linear Systems*[60] contains a discussion and comparison of the different methods and which method is best suited to which problem, *Matrix Computations*[61] discusses quite a number of special systems and gives pseudocode for the algorithms and *Handbook for Automatic Computation: Linear Algebra*[62] is considered the "bible" on this subject. The last book is in two parts the first being the solution of linear systems, least squares and, linear programming and the second part on the algebraic eigenvalue problem. The last two books discuss the algebraic eigenvalue problem but, *The Algebraic Eigenvalue Problem*[63] is a more complete reference on the subject.

Direct Methods

Direct methods, such as LU (lower-upper) decomposition, solve the system using a fixed number of operations. These algorithms usually use Gauss transformations to triangularize the system but other transformations, such as the Householder transformation, may be used. Pivoting is used to reduce round-off error and prevent any unnecessary terminations of the decomposition due to zeros along the diagonal. Partial pivoting, where only the rows are interchanged, is usually used to reduce the number of interchanges and hence the computational effort.

The bandwidth of the matrix is preserved under decomposition[64]. Versions of these algorithms for banded matrices are available that exploit this fact. If the bandwidth of the matrix is quite small, significant savings in computational time may be made. If the matrix to be inverted is in a block tridiagonal form, algorithms exist that can take advantage of this matrix structure. The Block Tridiagonal method[65] can perform an LU decomposition and solve in a block fashion. Significant savings in storage space and computation effort can be realized with this method.

---

60 Westlake (1968)
61 Golub and Van Loan (1983)
62 Wilkinson and Reinsch (1971)
63 Wilkinson (1965)
64 G. Dahlquist & Å. Björck, (1974) p.165
65 Goulub & Van Loan (1983), p.110

Iterative Methods

Iterative methods, such as the Jacobi method or the Gauss-Seidel method, use a variable number of operations depending upon the rate of convergence. They are particularly good for diagonally dominant systems where convergence is rapid. The disadvantage of these methods is the possibility of slow or irregular convergence. This may be remedied by acceleration techniques such as *successive overrelaxation* (SOR).

The Gauss-Seidel (GS) method splits the matrix A into a lower triangular, a diagonal, and a upper triangular matrix, $A = D(L + I + U)$, $D = diag[a_{ii}]$. The successive approximation for the solution is described by

$$x^{k+1} = -Lx^{k+1} - Ux^{k} + D^{-1}b$$

where k indicates the approximation number. Unlike the Jacobi method, the Gauss-Seidel method uses information from recently calculated elements, hence the x^{k+1} term on the right hand side. The GS method convergences at twice the rate of the Jacobi method which doesn't use recent information.

Successive overrelaxation (SOR) can be used to accelerate the convergence of the GS method. This accelerated method is often referred to as the SOR method and has been used quite successfully in many applications. Hageman & Young (1981) give the pseudocode for this method. The current GS method can be written as

$$x_i^{k+1} = x_i^{k} + r_i^{k}$$

where

$$r_i^{k} = \frac{-\sum_{j=1}^{i-1} a_{ij}x_j^{k+1} - \sum_{j=i}^{n} a_{ij}x_j^{k} + b_i}{a_{ii}}$$

The SOR method that is used to accelerate convergence can be written as

$$x_i^{k+1} = (1-\omega)x_i^{k} + \omega r_i^{k}$$

where $(1 < \omega < 2)$ is the relaxation parameter that is chosen to maximize convergence. Methods of calculating the relaxation parameter for maximum rate of convergence can be found in Wachspress (1966), and Hegeman & Young (1981).

Semi-Iterative Methods

The *Conjugate Gradient Method* is a semi-iterative method that could fall into either category but, is usually used as an iterative method because it is iterated until the desired accuracy is obtained. This method may also be used as a method of solution of large sparse matrices. The CG method and the GCG method may be accelerated using Chebychev acceleration[66].

" The conjugate-gradient method combines the features of the conjugate direction method and gradient methods. ... A conjugate-direction method in which the residual vectors are mutually orthogonal is essentially a conjugate-gradient method."[67] Hestenes and Stiefel (1952) produced what is considered the seminal work on conjugate gradient methods. There has been considerable interest in the use of this method, Bunch (1976), Eisenstat (1981), Saad & Schultz (1986), and Vatsya (1989) to name a few.

The original method was designed for use on positive definite, symmetric linear systems. This method can be generalized to an arbitrary nonsingular matrix with a simple transformation. If the matrix A is nonsingular, then the matrix A^TA is symmetric positive definite. The linear system that will be solved now is: $A^TA\,x = A^Tb$. The linear system solver that is based on this method is call the *Generalized Conjugate Gradient* (GCG) method[68]. The use of this transformation degrades the performance of the algorithm but it has a wider area of applicability.

---

66 Hageman & Young (1981), p.138 and p.339.
67 J.R. Westlake, (1968) p.46
68 *ibidem*, p.51

3 Neutronics

The term neutronics refers to the equation(s) that govern neutron density (or more commonly neutron flux), delayed neutrons and delayed neutron precursors. The general form of the neutronics equation is a function of energy, position, direction and time. This is referred to as the neutron transport equation.

3.1 Neutron Transport

The neutron transport equation is a function of direction, energy, space and time. This equation assumes that the number of neutrons is large and therefore describes their average behavior. The solution of this equation yields the angular neutron density, $N(\vec{r},\vec{\Omega},E,t)$. It is more common to use the neutron flux instead of the neutron density as the dependent variable. The velocity of the neutrons can be written as a product of the speed and the direction, $\vec{V}=v\vec{\Omega}$. The directional flux is defined as the product of the neutron speed and the neutron density

$$\Psi(\vec{r},\vec{\Omega},E,t)=vN(\vec{r},\vec{\Omega},E,t)=\text{directional flux}$$

The neutron transport equation can be written using the directional flux

$$\frac{1}{v(E)}\frac{\partial\Psi}{\partial t}+\vec{\Omega}\cdot\vec{\nabla}\Psi(\vec{r},\vec{\Omega},E,t)+\Sigma_t(\vec{r},E)\Psi=\int_{E'}dE'\int_{\Omega'}d\vec{\Omega}'\Sigma_s(\vec{r};\vec{\Omega}',E'\rightarrow\vec{\Omega},E)\Psi(\vec{r},\vec{\Omega}',E',t)$$

$$+\frac{\chi(E)}{4\pi}\int_{E'}dE'\Sigma_f(\vec{r},E')\Psi(\vec{r},\vec{\Omega}',E',t)$$

The terms in the above equation are as follows: the time rate of change, the streaming term, the total cross section, the scattering production term, and the fission production term.

Infinite Plane Geometry

If infinite plane geometry is assumed, then radial symmetry can be assumed. The above equation can be integrated over $(0<\theta<2\pi)$. The direction differential becomes $d\vec{\Omega}=2\pi d\mu$ where $\mu=\vec{z}\cdot\vec{\Omega}$ is the cosine of the scattering angle

$$\frac{1}{v}\frac{\partial\Psi}{\partial t}+\mu\frac{\partial\Psi}{\partial z}+\Sigma_t\Psi=\int_{E'}dE'\int_{\mu'}d\mu'\Sigma_s(z;\mu',E'\rightarrow\mu,E)\Psi(z,\mu',E',t)$$

$$+\frac{\chi(E)}{2}\int_{E'}dE'\Sigma_f(z,E')\Psi(z,\mu',E',t)$$

This the common form of the time-dependent neutron transport equation in plane geometry that will be of use later.

If the product of the direction vector and the flux are integrated over direction, the result will be the neutron current vector which represents the magnitude and direction of the average neutron flow

$$\vec{J}(\vec{r},E,t)=\int_{\Omega} d\vec{\Omega}'\vec{\Omega}\Psi(\vec{r},\vec{\Omega},E,t)$$

If the directional flux is integrated over all directions the result will be what is called the scalar flux. The scalar flux is used in the neutron diffusion equation

$$\Phi(\vec{r},E,t)=\int_{\Omega} d\vec{\Omega}\Psi(\vec{r},\vec{\Omega},E,t)$$

3.2 Solution of the Neutron Transport Equation (NTE)

These methods of solution of the neutron transport equation can be classified into the two broad areas of deterministic solutions and stochastic solutions. The two most popular methods of solution of the neutron transport equation, the Spherical Harmonic Expansion method, also known as the Pn approximation, and the Discrete Ordinates method, also known as the Sn method, are deterministic methods. The Monte Carlo method is the best known stochastic method. In general, deterministic methods, usually discrete ordinates or diffusion, are used in reactor simulations. These methods are not too computationally demanding and provide sufficiently accurate results for most purposes.

The spatial dependence of the NTE, when solved in a deterministic manner, is usually handled through finite differencing. This is a common method of handling the spatial dependence of PDEs and is not used just for the solution of the NTE. This subject will be covered in the chapter for numerical methods.

The first problem to be addressed in the deterministic solution of the neutron transport equation is the directional dependence of the neutron flux.

3.2.1 Monte Carlo

The Monte Carlo Method (MCM) of solution of the neutron transport equation is a statistical method of solution. The MCM is based on a series of particle histories. The flight of a particle is tracked and statistics are gather during its flight. The greater the number of histories that are acquired, the greater the accuracy of the results (the lower the variance).

The Monte Carlo method has the ability to handle complexity whether it be due to geometry or composition. It is useful for problems that can not be easily represented in less that three dimensions or have a fine structure. It can be used to determine the flux in the resonance region and hence the group constants. Its weakness lies in its stochastic nature. Its not well suited to the predicting of accurate values of flux over

extended ranges of space, direction or energy. A very large number of histories are required for accurate results over any range of variables, hence computational expense. The method can be used to determine criticality but is very poorly suited to kinetic problems due to computational expense. This method is not used here and will not be discussed further.

3.2.2 Directional Dependence - Transport Methods

The discrete ordinates method (Sn) is the most popular method when directional dependence of the flux must be considered. The equations from the spherical harmonic expansion method (Pn) are extensively coupled which makes them impractical except for relatively simple geometry. The Sn method circumvents this difficulty by finding the flux in only a discrete number of directions. The accuracy of the simulation can be increased by increasing the number of directions.

The multigroup equations arising from both the Pn and Sn methods are differential equations that are converted into a system of algebraic equations by spatial discretization. These equations are most useful for determining the overall transport for problems of fairly simple geometry. This yields results of high precision, precision that is limited mainly by the uncertainties of the cross sections.

The main concerns are the values of the group constants, the degree of detail required in the neutron angular expansion, energy spacing (number of groups) and the space mesh. The group constants are flux weighted averages. The choice of the appropriate weighting functions is a central problem.

Simulations using these methods are done in one or two dimensions. If the geometry or composition of the object to be simulated is sufficiently complex (two dimensions is not enough or it's difficult to calculate the group constants), the Monte Carlo method may need to be employed.

Anisotropic scattering is usually assumed when doing a transport calculation, and a convenient method of representation of the differential cross sections is required. It is customary to assume that the scattering cross sections are symmetric in the azimuthal direction ϕ using spherical coordinates (r, θ, ϕ), hence they are a function of the polar angle θ only. It is also customary to let $(0 \le \mu \le 1)$ be the cosine of the polar angle, $\mu = \cos(\theta)$. It is now possible to expand the differential scattering cross sections in Legendre polynomials, $P_l(\mu_0)$, where $\mu_0 = \Omega \times \Omega''$ is the cosine of the scattering angle in the laboratory system. The scattering cross sections will appear as

$$\Sigma_s(\vec{r}, E' \rightarrow E, \mu_0) = \sum_{l=0}^{\infty} (2l+1)\Sigma_{sl}(\vec{r}, E' \rightarrow E)P_l(\mu_0)$$

For isotropic scattering only the first Legendre moment of the scattering cross section, $\Sigma_{s0}(\vec{r}, E' \rightarrow E)$, will be nonzero.

Spherical Harmonics Method

In the spherical harmonics method the directional dependence on the NTE is expanded in Legendre Polynomials. Legendre Polynomials are orthogonal polynomials which can be used to derive a set of orthogonal basis functions for describing the directional dependence of the flux

$$\Psi(\vec{r},\mu_0,E)=\sum_{l=0}^{\infty}(2l+1)\Psi_l(\vec{r},E)P_l(\mu_0)$$

$$\Psi_l(\vec{r},E)=\frac{1}{2}\int_{-1}^{1}P_l(\mu)\Psi(\vec{r},\mu,E)d\mu$$

where $P_l(\mu)$ is the l th order Legendre Polynomial.

The Spherical Harmonic expansion fo: an infinite plane can be derived by multiplying the infinite plane NTE by the appropriate Legendre polynomial, integrating over direction, and using the identity,

$$(2n+1)\mu P_n(\mu)=(n+1)P_{n+1}(\mu)+nP_{n-1}(\mu)$$

The fission source term is usually considered to be isotropic, therefore it only appears in the zero moment (Ψ_0) equation

$$(2n+1)\frac{1}{v}\frac{\partial}{\partial t}\Psi_n(z,t)+\frac{\partial}{\partial z}[(n+1)\Psi_{n+1}+n\Psi_{n-1}]+(2n+1)\Sigma_t\Psi_n$$

$$=(2n+1)\int_{E'}dE'\Sigma_{sn}(z;E'\rightarrow E)\Psi_n(z,E',t)+\delta_{0,n}(2n+1)\frac{\chi(E)}{2}\int_{E'}dE'\Sigma_f(z,E')\Psi_n(z,E',t)$$

Discrete Ordinates Method

In the discrete ordinates method, the directional integral is evaluated using numerical quadrature. Gaussian quadrature is most commonly used. The net effect of this numerical integration is that the flux is evaluated in a number of discrete directions (hence the name)

$$\int_{-1}^{1}\Psi(\vec{r},\mu',E)d\mu'=\frac{1}{2}\sum_{i=1}^{N}\omega_i\Psi(\vec{r},\mu_i,E)$$

where $\{\mu_i\}$ is the set of discrete directions and $\{\omega_i\}$ are the quadrature weights. A more thorough treatment of this subject can be found in *Nuclear Reactor Theory*.[69]

The Integral Transport Equation and the Collision Probability Method

Integral transport methods are very different from the discrete ordinates method. Integral transport methods are based on integrating out the angular dependence from the transport equations leaving only the scalar flux and partial currents across cell boundaries. The angular variable can be treated with whatever accuracy is required, which depends on the numerical integration of the expression for the angular

---

69 Bell and Glasstone (1980), p.214

integration. High accuracy multidimensional quadrature used for the angular integration can be quite expensive but needs only to be done once. "Since the neutron transport equation is a linear first order partial differential-integral equation, it can be converted to an integral equation by a standard procedure known as the method of characteristics".[70] "Integral transport methods are most frequently applied to situations where the effects of isotropic scattering are small, ...".[71] Integral transport methods are used for lattice codes, where highly absorbing regions but small spatial domains are characteristic, and in shielding calculations.

The collision probability method is the most commonly used technique for the solution of the integral transport equation. The collision probability equation solves for the scalar flux ϕ,

$$\Sigma_{T_j}^g \phi_j^g = \sum_{i=1}^{N} P_{ij}^g \left[\sum_{g'=1}^{G} (\Sigma_{ii}^{g' \to g} \phi_i^g + \lambda \chi^g \nu \Sigma_{fi}^{g'} \phi_i^{g'}) + S_i^g \right]$$

where the subscripts j and i denote region, the superscript g denotes energy group, Σ_T is the total cross section for that energy group and region, Σ_s is the scattering cross section, χ^g is the fraction of neutrons appearing in group g, λ is the eigenvalue, $\nu\Sigma_f$ is the number of fission neutrons produced, S_i is the source term and, P_{ij} can be interpreted as the first flight collision probability that neutrons produced in region i will make their first collision in region j. The interface current method relates the flux in a given region to sources within that region and the currents at the boundary surfaces. The lowest order of approximation of this method is[72]

$$\Sigma_{T_j}^g \phi_j^g = P_{jj}^g \left[\sum_{g'=1}^{G} (\Sigma_{ij}^{g' \to g} \phi_j^g + \lambda \chi^g \nu \Sigma_{fj}^{g'} \phi_j^{g'}) + S_j^g \right] + \sum_{m=1}^{6} P_{mj}^g J_{i,m}^g$$

$$J_{o,m}^g = P_{jm}^g \left[\sum_{g'=1}^{G} (\Sigma_{ij}^{g' \to g} \phi_j^g + \lambda \chi^g \nu \Sigma_{fj}^{g'} \phi_j^{g'}) + S_j^g \right] + \sum_{m' \neq m}^{6} P_{m'm}^g J_{i,m'}^g$$

where $J_{i,m}^g$ is the total inward current of group g at surface m and, $J_{o,m}^g$ is the total outward current of group g at surface m. The interface current method eliminates cross coupling terms between regions and hence reduces storage requirements.

The collision probability method is given as background information for some of the lattice codes.

---

70 Bell & Glasstone, (1970) p.22

71 Lewis & Miller (1984) p.211

72 Mueller A. & M.R. Warner (1972) p.280

3.2.3 Energy Dependence

The multigroup formulation is the most common method of handling the problem of energy dependence of the neutron diffusion equations. The energy spectrum is discretized into several levels referred to as groups. The group flux for the g'th group is defined as

$$\Phi_g(\vec{r},t)=\int_{E_g}^{E_{g-1}}\Phi(\vec{r},E,t)dE$$

The neutronics equations are integrated with respect to energy over the energy intervals

$$\frac{1}{v_g}\frac{\partial}{\partial t}\Phi_g(\vec{r},t)=\vec{\nabla}\cdot D_g(\vec{r})\vec{\nabla}\Phi_g(\vec{r},t)-\Sigma_{tg}(\vec{r})\Phi_g(\vec{r},t)+\sum_{g'=1}^{G}\Sigma_{sg'g}(\vec{r})\Phi_{g'}(\vec{r},t)$$

$$+\chi_{pg}(1-\beta)\sum_{g'=1}^{G}\nu_{g'}\Sigma_{fg'}(\vec{r})\Phi_{g'}(\vec{r},t)+\sum_i\chi_{dg}\lambda_i C_i(\vec{r},t)$$

$$\frac{\partial C_i(\vec{r},t)}{\partial t}=\beta_i\sum_{g'=1}^{G}\nu_{g'}\Sigma_{fg'}(\vec{r})\Phi_{g'}(\vec{r},t)-\lambda_i C_i(\vec{r},t)$$

The multigroup coefficients that are used in the above equations are defined in Appendix B.

The neutron energy spectrum can be roughly divided into three groups: thermal interval ($E_n < 1\ eV$), resonance interval ($1\ eV < E_n < 100\ keV$), and the fission interval ($100\ keV < E_n$). The thermal interval is characterized by the thermal motion of the neutrons and the neutron energy spectrum will be a Maxwellian distribution[73] for thermal reactors (the spectrum is more complicated for fast reactors). The neutron energy spectrum in the resonance interval is strongly influenced by the moderator that is used in the reactor. If a thermal reactor is moderated by hydrogen only (water), the spectrum in this interval will have an overall $1/E$ characteristic[74], but there will localized dips in the energy spectrum due to resonance absorption in the fuel. The spectrum in the fission interval will be dominated by fission neutrons, as the name suggests.

The multigroup formulation is used to handle the energy dependence of the time dependent diffusion approximation used here.

3.2.4 Time Dependence

The method of lines is used to transform the PDEs into a system of coupled ODEs. The linear system approximation $d\vec{x}/dt = A\vec{x}+\vec{b}$ is eventually discretized in time, giving rise to a matrix difference equation, $d\vec{x}/dt \approx (\vec{x}^{k+1}-\vec{x}^{k})/\Delta t$, where k denotes time step, and Δt is the step size. The range in sizes of the time step

73 Henry (1975) p.96
74 ibidem, p.85

that are allowed, in terms of stability, will depend on how the difference equation is arrived at. The two general methods that are used are: *explicit* methods, where the vector on the right hand side is evaluated at time step k, and *implicit* methods where the vector on the right hand side is evaluated at time step $k+1$. In general implicit methods have a greater range of stability but are more computationally demanding than explicit methods.

The time independent solution of the NTE usually involves the calculation of an eigenvalue, or k value, which is an indication of the exponential time behavior of the neutron density. This is referred to as a criticality calculation.

3.3 Neutron Diffusion: the P_1 Approximation

The P_1 approximation is considered to be the theoretical basis for the diffusion approximation.[75] The flux is expanded in the two Legendre moments, $P_0 = 1$ and $P_1 = \mu$ with the result becoming the scalar flux and the neutron current,

$$\Psi_0(\vec{r},E) = 2\pi \int_{-1}^{1} \Psi(\vec{r},\mu,E)d\mu = \Phi(\vec{r},E)$$

$$\Psi_1(\vec{r},E) = 2\pi \int_{-1}^{1} \mu\Psi(\vec{r},\mu,E)d\mu = \vec{J}(\vec{r},E)$$

The equations for the neutron current and the scalar flux can be found by substituting $n=1$ and 0 respectively into the P_n approximation equation,

$$\frac{3}{v}\frac{\partial}{\partial t}J(z,E,t)+\frac{\partial}{\partial z}\Phi(z,E,t)+3\Sigma_t J = 3\int_{E'} dE'\Sigma_{s1}(z;E'\rightarrow E)J(z,E',t)$$

$$\frac{1}{v}\frac{\partial}{\partial t}\Phi(z,E,t)+\frac{\partial}{\partial z}J(z,E,t)+\Sigma_t\Phi = \int_{E'} dE'\Sigma_{s0}(z;E'\rightarrow E)\Phi(z,E',t)$$

$$+\frac{1}{2}\chi(E)\int_{E'} dE'\Sigma_f(z,E')\Phi(z,E',t)$$

A fundamental postulate of diffusion theory is the validity of Fick's Law. This states that the neutron current is equal to the flux gradient multiplied by a diffusion constant, which must be defined,

$$\vec{J}(z,E,t) = D(z,E)\vec{\nabla}\Phi(z,E,t)$$

The equation for the first moment is used to define the diffusion coefficient,

$$D(z,E) = \frac{1}{3}\left(\Sigma_t + \frac{\frac{1}{v}\frac{\partial}{\partial t}J - \int dE'\Sigma_{s1}J'}{J(z,E,t)}\right)^{-1}$$

75 Henry (1975), p.386.

The scattering integral is usually approximated by

$$\int dE' \Sigma_{s1}(z, E' \rightarrow E) J(z, E', t) \approx \Sigma_{s0}(z, E) \bar{\mu}_0(z, E) J(z, E, t)$$

where $\bar{\mu}_0(z, E) = \Sigma_{s1}/\Sigma_{s0}$ represents that average scattering angle and the Σ_{sn} is defined as:

$$\Sigma_{sn}(z, E) = \int dE' \Sigma_{sn}(z, E' \rightarrow E)$$

The transport cross section is defined as,

$$\Sigma_{tr}(z, E) = \Sigma_t(z, E) + \bar{\mu}_0 \Sigma_{s1}(z, E)$$

The definition of the diffusion coefficient that is most commonly used, and is used here,

$$D(z, E) = \frac{1}{3\Sigma_{tr}(z, E)}$$

is that for the time independent problem.

The zero moment equation becomes the neutron diffusion equation. Delayed neutrons could have been added to the neutron transport, but they are added to the diffusion equation now for simplicity. Diffusion simulations can be done in three spatial dimensions but two or less is more common. The two energy group approximation, fast and thermal, is the main stay of the industry for thermal reactor simulation. Sometimes the assumptions associated with diffusion, isotropic scattering and no angular dependence of the flux, are not adequate, especially in the region of a strong absorber. When angular dependence of the flux must be taken into account, a neutron transport simulation must be done.

The time rate of change of the neutron density can be written in terms of the neutron flux. This equation is referred to as the time dependent neutron diffusion equation

$$\frac{\partial}{\partial t} N(\vec{r}, E, t) = \frac{\partial}{\partial t}\left(\frac{1}{v(E)} \Phi(\vec{r}, E, t)\right)$$

$$\frac{\partial}{\partial t}\left(\frac{1}{v(E)} \Phi(\vec{r}, E, t)\right) = \vec{\nabla} \cdot D(\vec{r}, E) \vec{\nabla} \Phi(\vec{r}, E, t) - \Sigma_t(\vec{r}, E) \Phi(\vec{r}, E, t)$$

$$+ \int_0^\infty \Sigma_s(\vec{r}, E' \rightarrow E) \Phi(\vec{r}, E', t) dE' + \sum_j \chi_p^j(E)(1 - \beta^j) \int_0^\infty \nu^j(E') \Sigma_f^j(\vec{r}, E') \Phi(\vec{r}, E', t) dE'$$

$$+ \sum_i \chi_i(E) \lambda_i C_i(\vec{r}, t)$$

The terms on the right hand side are itemized as follows: leakage (diffusion), total removal cross section, inscattering from other energies, fission neutron production (prompt neutrons), delayed neutron production. The super script j denotes the fissile isotope, and this may be dropped if there is only one.

The associated precursor equation is

$$\frac{\partial C_i(\vec{r}, t)}{\partial t} = \sum_j \beta_i^j \int_0^\infty \nu^j(E') \Sigma_f^j(\vec{r}, E') \Phi(\vec{r}, E', t) dE' - \lambda_i C_i(\vec{r}, t)$$

where $\beta = \sum_i \beta_i$.

The number of delayed neutron precursors is usually a maximum of six. When only one delayed group is used, this is referred to as one effective delayed group. The average decay constant

$$\lambda \equiv \left[\frac{1}{\beta} \sum_i \frac{\beta_i}{\lambda_i} \right]^{-1}$$

is used in this case.

3.3.1 Boundary Conditions

There are some conservation relationships that must be satisfied at the interfaces and boundaries. The two quantities that are conserved at an interface in neutron diffusion are the neutron flux and the neutron current.

The conservation of flux simply means that the flux must be continuous at an interface. If there are two regions labeled 'a' and 'b' then this relation is

$$[\Phi_a(\vec{r}, E, t)]_{\vec{r}=surface} = [\Phi_b(\vec{r}, E, t)]_{\vec{r}=surface}$$

The other conservation condition that must be satisfied is the conservation of current. In neutron diffusion the neutron current is equal to the gradient of the neutron flux multiplied by the diffusion constant. This current must be continuous at an interface,

$$D_a[\vec{n} \cdot \vec{\nabla}\Phi_a(\vec{r}, E, t)]_{\vec{r}=surface} = D_b[\vec{n} \cdot \vec{\nabla}\Phi_b(\vec{r}, E, t)]_{\vec{r}=surface}$$

where the surface normal is $\vec{n}$.

3.4 Thermal Feedback: Doppler Broadening of Cross Sections

Of primary importance in this analysis are the feedback mechanisms at play. Doppler broadening of the absorption cross sections is the predominant temperature feedback mechanism in the fuel. Voiding and thermal expansion are the thermal feedback mechanisms in the coolant and moderator.

The multigroup coefficients (the cross sections) used in the multigroup equations are flux weighted (see appendix B). They are the product of the energy dependent cross sections and neutron flux integrated over the appropriate energy interval, and divided by the total flux within that energy interval, the group flux. The group flux is

$$\Phi_g(\vec{r}, t) = \int_{E_g}^{E_{g-1}} \Phi(\vec{r}, E) dE$$

and the flux weighted microscopic cross section is:

$$\sigma_{x_g}(\vec{r}) = \frac{1}{\Phi_g}\int_{E_g}^{E_{g-1}} \sigma_x(\vec{r}, E)\Phi(\vec{r}, E)dE$$

Doppler broadening of the flux weighted cross sections occurs in the resonance interval ($1\ eV < E_n < 100\ keV$). The heavy nuclides, and especially U^{238}, have resonance absorption peaks in this interval. An absorption peak is when the absorption cross section of the isotope, when plotted as a function of the incident neutron energy, has a high narrow peak. Some isotopes have many such resonance peaks, and some of the resonance peaks are so closely spaced as to be indistinguishable from one another. Increased thermal vibration of the atoms causes a perceived broadening of the absorption peaks due to the increased uncertainty of the neutron incident energy. If the microscopic cross section is integrated over an energy interval that contained a resonance peak, the value of this integral would not change significantly as a function of temperature, that is to say the area under the peak will remain constant. Therefore there is an associated reduction in the height of the resonance peak. The product of the flux and the microscopic cross section does increase though with increasing temperature. How the flux weighted absorption cross sections vary as a function of temperature depends on the energy spectrum of the neutron flux. Different energy profiles will produce different temperature behaviors of the flux weighted cross sections. The energy profile of the flux is strongly influenced by the material composition of the reactor.

4 Heat Transport

Heat transport and its effects play a significant role in the dynamics of a reactor. Heat transport take two primary forms: conductive heat transfer within the fuel and convective heat transfer within the coolant. The flow of thermal energy begins with the production of heat through nuclear fission in the fuel pins. This heat is conducted to the outer surface of the fuel pins where it is transferred to the primary coolant. The primary coolant carries the heat to the primary heat exchanger usually known as a steam generator.

4.1 Thermalhydraulics

The thermal hydraulic equations update the variables $(\rho, h, \vec{V}, P)$ density, enthalpy, velocity and pressure respectively. It is assumed here that these four variables are functions of both space and time. The conservation equations for momentum, mass and energy are used to calculate velocity, density and enthalpy respectively. A fourth equation, the equation of state, is required for closure. It is usually used to calculate the pressure.

The general form of the conservation equation for some variable of interest, $\Psi(\vec{r}, t)$, written in the macroscopic form is

$$\int_{vol} \frac{\partial \Psi}{\partial t} dv + \int_{s} \Psi \vec{V} \cdot \vec{n} ds = \int_{vol} \Gamma(\vec{r}, t) dv$$

where 'vol' is some volume of interest and $\Gamma(\vec{r}, t)$ is the source term concentration. The second term on the left hand side represents the flow of this variable through the surface of the volume of interest. Gauss's theorem can be used to convert the surface integral to a volume integral,

$$\int_{vol} \left[\frac{\partial \Psi}{\partial t} + \vec{\nabla} \cdot \Psi \vec{V} - \Gamma \right] dv = 0$$

If it is assumed that the volume of integration is arbitrary, then the expression within the brackets must be zero

$$\frac{\partial \Psi}{\partial t} + \vec{\nabla} \cdot \Psi \vec{V} - \Gamma = 0$$

This is the general conservation equation. This equation will be used to derive the various conservation equations.

4.1.1 Conservation Equations

The conservation equations for momentum, mass, and energy can be derived by replacing the variable of interest in the general equation by the appropriate quantity.

Mass Equation

The variable that is conserved is mass which is represented by the density which is the mass per volume. The result when this variable is substituted into the general conservation equation is

$$\frac{\partial \rho}{\partial t} + \vec{\nabla} \cdot \rho \vec{V} = 0$$

There are no source terms for the conservation of mass equation. This equation may written in a more expanded form

$$\frac{\partial \rho}{\partial t} + \rho \vec{\nabla} \cdot \vec{V} + \vec{V} \cdot \vec{\nabla} \rho = 0$$

If the fluid is incompressible[76], then it is assumed that density variations are very slight. This gives rise to the continuity equation for incompressible fluids[77],

$$\vec{\nabla} \cdot \vec{V} = 0$$

Momentum Equation

The momentum of the fluid per unit volume is the product of the density of the fluid and the velocity of the fluid. Therefore the variable of interest in the conservation of momentum equation is

$$\Psi(\vec{r}, t) = \rho(\vec{r}, t)\vec{V}(\vec{r}, t)$$

Substituting into the general conservation equation and adding source terms

$$\frac{\partial (\rho \vec{V})}{\partial t} + \vec{\nabla} \cdot \rho \vec{V}\vec{V} = \rho \vec{g} + \vec{\nabla} \cdot \vec{\vec{\sigma}}$$

The terms on the right hand side are (respectively), body forces (gravity) and surface effects. Surface effects include pressure and viscous drag. These effects are represented in the stress tensor in the far right

76 No fluid is truly incompressible. It is common to refer to nearly incompressible fluids as incompressible fluid (such as most liquids).

77 Bird, Stewart & Lightfoot, p.75

hand term. This tensor can be subdivided into its normal and tangential components, which are pressure and shear stress respectively, $\bar{\bar{\sigma}} = -P\bar{\bar{I}} + \bar{\bar{\tau}}$. The conservation of momentum equation can be simplified to a form that is more convenient for the calculation of the velocity of the fluid (see appendix A)

$$\rho\frac{\partial \vec{V}}{\partial t} + \rho(\vec{V}\cdot\vec{\nabla})\vec{V} = \rho\vec{g} - \vec{\nabla}P + \vec{\nabla}\cdot\bar{\bar{\tau}}$$

This equation can be simplified further if the sources of shear stress are examined. The form that shear stress will take is dependent on the flow regime that the fluid is in.

Viscosity and Flow Regime

The two flow regimes of fluid mechanics are laminar flow and turbulent flow, with possibly a transitional region in between. The flow regime will have an effect primarily in the areas of viscosity and heat flux. It is assumed that the fluids are Newtonian and hence the shear stress tensor is a linear function of the velocity gradient

$$\bar{\bar{\tau}} = \lambda(\vec{\nabla}\cdot\vec{V})\bar{\bar{I}} + 2\mu\bar{\bar{s}}$$

There are two coefficients for viscosity in the above equation $\{\mu,\lambda\}$. The rate of strain tensor, $\bar{\bar{s}}$, is defined as

$$\bar{\bar{s}} = \frac{1}{2}(\vec{\nabla}\vec{V} + (\vec{\nabla}\vec{V})^T)$$

If the above equations are substituted into the conservation of momentum equation, the result would be (see Appendix A)

$$\rho\frac{\partial \vec{V}}{\partial t} + \rho(\vec{V}\cdot\vec{\nabla})\vec{V} = \rho\vec{g} - \vec{\nabla}P + \mu\nabla^2\vec{V} + (\lambda+\mu)\vec{\nabla}(\vec{\nabla}\cdot\vec{V}) + (\vec{\nabla}\cdot\vec{V})\vec{\nabla}\lambda + 2\bar{\bar{s}}\cdot\vec{\nabla}\mu$$

If the fluid is assumed incompressible ($\vec{\nabla}\cdot\vec{V} = 0$) and the viscosity constant ($\vec{\nabla}\mu = 0$), the conservation of momentum equation becomes the famous *Navier-Stokes equation*[78]

$$\rho\frac{\partial \vec{V}}{\partial t} + \rho(\vec{V}\cdot\vec{\nabla})\vec{V} = \rho\vec{g} - \vec{\nabla}P + \mu\nabla^2\vec{V}$$

In the laminar flow regime the fluid has Newtonian behavior and the viscosity is defined accordingly. In the turbulent flow regime the eddy viscosity is used.

Energy Equation

78 Bird, Stewart & Lightfoot, P.80

Energy is the variable that is conserved here. The total energy per volume is the sum of the internal energy per volume, energy that results form being in a particular state, and the kinetic energy per volume that arises from motion.

$$\Psi = \rho\left(e + \frac{1}{2}V^2\right)$$

Substituting this variable into the general conservation equation and adding source terms

$$\frac{\partial}{\partial t}\left[\rho\left(e + \frac{1}{2}V^2\right)\right] + \vec{\nabla}\cdot\rho\left(e + \frac{1}{2}V^2\right)\vec{V} = -\vec{\nabla}\cdot\vec{q} + S(\vec{r},t) + \rho\vec{g}\cdot\vec{V} + \vec{\nabla}\cdot(\bar{\sigma}\cdot\vec{V})$$

The terms on the right hand side are (respectively): surface heat flux, internal sources, work due to body forces (gravity) and work due to surface effects. This equation will eventually be used to calculate the enthalpy of the fluid. The enthalpy (per volume) of the fluid is defined as $h = e + P/\rho$.

The conservation of energy equation can be simplified to a form that is more convenient for the calculation of the rate of change of the enthalpy (see Appendix A)

$$\rho\frac{\partial h}{\partial t} + \rho\vec{V}\cdot\vec{\nabla}h = \vec{\nabla}\cdot\vec{q} + S(\vec{r},t) + \bar{\tau}:\vec{\nabla}\vec{V} + \frac{\partial P}{\partial t} + \vec{V}\cdot\vec{\nabla}P$$

This equation can be modified to represent temperature if the fluid is single phase and the identity

$$h = C_p T + constant$$

is used[79].

It is common to assume that all the terms on the RHS of the equation are negligible with respect to the heat flux term (most often the source term is zero). Let us also assume that heat flux is governed by Fick's law,

$$\vec{q}(\vec{r},t) = -k(\vec{r})\vec{\nabla}T(\vec{r},t)$$

where k is the coefficient of thermal conductivity. Making these substitutions we arrive at

$$\frac{\partial T}{\partial t} + \vec{V}\cdot\vec{\nabla}T(\vec{r},t) = 1/(\rho C_p)\vec{\nabla}\cdot k\vec{\nabla}T(\vec{r},t)$$

4.1.2 Free and Forced Convection

In summary, the time dependent conservation equations are mass, momentum and energy (respectively),

$$\frac{\partial\rho}{\partial t} + \vec{\nabla}\cdot\rho\vec{V} = 0$$

$$\rho\frac{\partial\vec{V}}{\partial t} + \rho\vec{V}\cdot\vec{\nabla}\vec{V} = \mu\vec{\nabla}^2\vec{V} + \rho\vec{g} - \vec{\nabla}P$$

79 The relationship between enthalpy and temperature is more complicated in the case of two phase flow.

$$\frac{\partial T}{\partial t} + \vec{V} \cdot \vec{\nabla} T = \alpha \nabla^2 T$$

where $\{\rho, \vec{V}, T, P\}$ are the density, velocity, temperature, and pressure respectively and $\{\vec{g}, \alpha, \mu\}$ are gravity, thermal diffusivity[80], and viscosity respectively. These are the equations that would normally be used for solving a forced convection problem. The usual form of the equation of state can simply be written in a form where pressure is a function of density and enthalpy, $P = \pi(\rho, h)$.

The density of the fluid is temperature dependent, and is calculated using the expression,[81]

$$\rho = \bar{\rho} - \bar{\rho}\bar{\beta}(T - \bar{T})$$

where $\bar{\rho}$ is the density at some reference temperature, $\bar{T}$, and $\bar{\beta}$ is the coefficient of volumetric expansion at that temperature.

The assumption made in free convection is that the pressure gradient is approximately the hydrostatic pressure gradient $\vec{\nabla} P \equiv \bar{\rho}\vec{g}$. This allows the pressure and gravity terms to be combined into one expression

$$\rho\vec{g} - \vec{\nabla} P = -\vec{g}\bar{\beta}\bar{\rho}(T - \bar{T})$$

The momentum equation for free convection becomes

$$\rho\frac{\partial \vec{V}}{\partial t} + \rho\vec{V} \cdot \vec{\nabla}\vec{V} = \mu\vec{\nabla}^2\vec{V} - \vec{g}\bar{\rho}\bar{\beta}(T - \bar{T})$$

The density and pressure need not be explicitly calculated in free convection.

4.1.3 The Benchmark Equations

The simulation was benchmarked for the steady state case using a convective cavity, as suggested by De Vahl Davis & Jones (1983). It is assumed that the cavity has unit dimensions. The top and bottom boundaries are thermally insulated. The left wall is at a temperature of one degree, while the right wall is at a temperature of zero, therefore the temperature differential is unity.

The variables used in the conservation equations are normalized: $\vec{V} = \vec{V}/\alpha$, $t = \alpha t$, $\rho = \rho/\bar{\rho}$, $P = P/\bar{\rho}$, and $\vec{g} = \vec{g}/\alpha^2$. The reference temperature, $\bar{T}$, is zero. The normalized momentum equation (for forced and free convection respectively) is

$$\frac{\partial \vec{V}}{\partial t} + \vec{V} \cdot \vec{\nabla}\vec{V} = Pr\vec{\nabla}^2\vec{V} + \vec{g} - \frac{1}{\rho}\vec{\nabla} P$$

$$= Pr\vec{\nabla}^2\vec{V} - RaPrT\frac{\vec{g}}{|g|}$$

---

80 Bird, Stewart & Lightfoot, (1960) pp 246

81 Bird, Stewart & Lightfoot, (1960) p 299

and the normalized energy equation is

$$\frac{\partial T}{\partial t} + \vec{V} \cdot \vec{\nabla} T = \nabla^2 T$$

where *Ra & Pr* are the Rayleigh and Prandtl numbers ($Ra = \overline{\beta} \mid g \mid /\alpha\nu$, $Pr = \nu/\alpha$), and ($\nu = \mu/\rho$) is the kinematic viscosity. De Vahl Davis & Jones use *Ra & Pr* as the input parameters, therefore the actual values of β and g are not explicitly known. The model used here is a forced convection model, therefore β and g must be known (or calculated).

The model used by De Vahl Davis & Jones assumes that the thermal diffusivity, α, is constant and the Prandtl number is fixed ($Pr = \mu/(\rho\alpha)$). This necessarily implies that density variations and hence β are quite small. Density is explicitly calculated using,

$$\rho = 1 - \overline{\beta} T$$

the normalized thermal expansion equation. The density must be greater than zero for a physically realizable system. This requirement places constraints on the choice of the volumetric expansion coefficient. If the temperature differential is one, then the temperature of the fluid will be less than one. Therefore for this set of normalized equations, β must be less than or equal to one for physically realizable results. The Rayleigh and Prandtl numbers are input parameters, and they are used to determine gravity and β. Either gravity or β will be fixed and the other calculated. The above argument shows that β has a maximum value which will place constraints on the value gravity must have if it is to remain fixed. The thermal expansion coefficient, β, is calculated using

$$\overline{\beta} = RaPr / \mid g \mid$$

given *Ra, Pr* and some chosen value of gravity. For the largest value of the Rayleigh number, β must remain less than one, which requires that gravity be sufficiently large to assure this.

The rate form of the equation of state has little dependence on temperature for incompressible fluids, therefore for simplicity, G_2 is set to zero, ignoring the enthalpy term. The conservation of mass equation is substituted into the rate equation for pressure,

$$\frac{\partial P}{\partial t} = -G_1 \vec{\nabla} \cdot \rho \vec{V}$$

These equations make up the benchmark simulation. The input parameters are: gravity, the Rayleigh number, the Prandtl number and G_1.

4.1.4 Conservation Equations - Nodal Form

The nodal form of all the conservation equations can be derived by integrating the conservation equation over a volume of interest, in which the quantities are assumed constant. Thermalhydraulic simulations based on this form of the equations take the form of a system of nodes and links. Total enthalpy, mass and, pressure are defined at each node and a mass flow rate is defined at each link.

Momentum Equation

The conservation of momentum equation is integrated over a volume of interest to derive the nodal form of the momentum equation,

$$\frac{\partial}{\partial t}\int_{vol}(\rho\overline{V})dv+\int_{s}\rho\overline{V}\overline{V}\cdot\vec{n}ds=\int_{vol}\rho\vec{g}dv+\int_{s}\overline{\sigma}\cdot\vec{n}ds$$

The total momentum, $\vec{p}$, and the total mass, M, are defined as

$$\vec{p}=\int_{v}\rho\overline{V}dv\equiv\text{total momentum}$$

$$M=\int_{v}\rho dv\equiv\text{total mass}$$

Defining the average of a power of velocity (such as the average squared velocity) as the velocity to the appropriate power of velocity integrated over some surface of interest. It is assumed that the surface normal is parallel to the direction of the velocity pointing either in the same direction or in opposite directions

$$<\overline{V}^{n}>=\frac{1}{S}\int_{s}\overline{V}^{n}ds$$

where S is the total surface area.

The total momentum is not a function of spatial position therefore a total time derivative is used. It is assumed that there is an inlet surface and outlet surface. The stress tensor is divided into its normal and shear components. The macroscopic momentum equation can now be written as

$$\frac{d}{dt}\vec{p}+\rho S_{out}<\overline{V}^{2}>_{out}-\rho S_{in}<\overline{V}^{2}>_{in}=M\vec{g}+P_{in}S_{in}-P_{out}S_{out}-\int_{s}\overline{\tau}\cdot\vec{n}ds$$

The last term on the right hand side is usually replaced with an empirical correlation function.

The term *mass flow rate* is introduced $W=\rho<\overline{V}>S$

Using this term we can write

$$\rho S<\overline{V}^{2}>=W\frac{<\overline{V}^{2}>}{<\overline{V}>}$$

If the velocity profiles are known the above ratio can be calculated. If the flow is turbulent the velocity profile is assumed to be flat and the approximation can be made,

$$\frac{<\overline{V}^{2}>}{<\overline{V}>}\equiv<\overline{V}>$$

The momentum equation in nodal form can now be written as

$$\frac{d}{dt}\vec{p} + W_{out}\frac{<\vec{V}^2>_{out}}{<\vec{V}>_{out}} - W_{in}\frac{<\vec{V}^2>_{in}}{<\vec{V}>_{in}} = M\vec{g} + P_{in}S_{in} - P_{out}S_{out} - \int_s \vec{\tau}\cdot\vec{n}ds$$

Mass Equation

The conservation of mass equation is integrated over the volume of interest,

$$\frac{d}{dt}M = W_{in} - W_{out}$$

where some quantities have been defined in the previous section.

Energy Equation

The conservation of energy equation can be written as (see Appendix A)

$$\frac{\partial}{\partial t}(\rho h) + \vec{\nabla}\cdot(\rho h\vec{V}) = \vec{\nabla}\cdot\vec{q} + S(\vec{r},t) + \vec{\tau}:\vec{\nabla}\vec{V} + \frac{\partial P}{\partial t} + \vec{V}\cdot\vec{\nabla}P$$

This equation is integrated over the volume and some of the volume integrals are converted into surface integrals,

$$\int_v \frac{\partial}{\partial t}(\rho h)dv + \int_s \rho h\vec{V}\cdot\vec{n}ds = \int_s \vec{q}\cdot\vec{n}ds + \int_v S(\vec{r},t)dv + \int_v \vec{\tau}:\vec{\nabla}\vec{V}dv + \int_v\left[\frac{\partial P}{\partial t} + \vec{V}\cdot\vec{\nabla}P\right]dv$$

It is assumed that the enthalpy is constant over the inlet and outlet surfaces

$$\Rightarrow \int_s \rho h\vec{V}\cdot\vec{n}ds = W_{out}h_{out} - W_{in}h_{in}$$

The total heat flux is defined as

$$Q \equiv \int_s \vec{q}\cdot\vec{n}ds \equiv \text{total heat flux}$$

The total enthalpy for the volume is

$$H(t) \equiv \int_v \rho h dv \equiv \text{total enthalpy}$$

The macroscopic form of the conservation of energy is now

$$\frac{d}{dt}H(t) = W_{in}h_{in} - W_{out}h_{out} + \int_v \vec{\tau}:\vec{\nabla}\vec{V}dv + \int_v\left[\frac{\partial P}{\partial t} + \vec{V}\cdot\vec{\nabla}P\right]dv + Q$$

The turbulent heating term is usually very small and therefore neglected,

$$\int_v \vec{\tau}:\vec{\nabla}\vec{V}dv \equiv 0$$

The integral term involving pressure is also often neglected as it is small

$$\int_v\left[\frac{\partial P}{\partial t} + \vec{V}\cdot\vec{\nabla}P\right]dv \equiv 0$$

Eliminating these two gives rise to the form of the macroscopic energy equation that is most commonly used

$$\frac{d}{dt}H(t) = W_{in}h_{in} - W_{out}h_{out} + Q$$

4.1.5 Equation of State

A fourth equation is required for closure, and is usually used to calculate the pressure. This fourth equation is called the equation of state as it is used to determine the state of the system and whether all things are in balance. The equation of state has the form, $P = \Pi(\rho, h)$. This equation can be used in a rate form. The use of the rate form has several advantages, the most relevant here being the ability to incorporate the equation of state into the overall matrix rate equation for the system thereby generating a more intuitive feel for overall system behavior[82]. The partial derivative with respect to time of the equation of state is

$$\frac{\partial P}{\partial t} = \left(\frac{\partial P}{\partial \rho}\right)_h \frac{\partial \rho}{\partial t} + \left(\frac{\partial P}{\partial h}\right)_\rho \frac{\partial h}{\partial t}$$

The two partial derivatives of pressure on the right hand side are replaced by constant coefficients. The result,

$$\frac{\partial}{\partial t}P(\vec{r},t) = G_1\frac{\partial}{\partial t}\rho(\vec{r},t) + G_2\frac{\partial}{\partial t}h(\vec{r},t)$$

is the time rate of change of the pressure described as a linear combination of the rate of change the mass and enthalpy. This is the form of the equation of state that will be used here.

If a two phase system is assumed in a more detailed discussion, the enthalpy and specific volume are,

$$h = xh_g + (1-x)h_f$$

$$v = xv_g + (1-x)v_f$$

respectively, where x is the quality, and $v \equiv 1/\rho$ is the specific volume.

If it is assumed that the volume and total mass within a node are relatively constant, the rate of change of the specific volume (and hence the density) with respect to the pressure will be zero (i.e. there is no explicit dependence of specific volume on pressure). Using that assumption the partial derivative of the enthalpy with respect to the pressure can be used to calculate the rate of change of the pressure:

$$\frac{\partial h}{\partial P} = (1-x)\frac{\partial h_f}{\partial P} + x\frac{\partial h_g}{\partial P} + \frac{\partial x}{\partial P}(h_g - h_f)$$

The rate of change of the quality with respect to the pressure is required to complete this equation. The quality and the rate of change of the quality with respect to the pressure can be written as:

---

82 Garland, *Basic Equations for Thermalhydraulic System Analysis*

$$x = \frac{v - v_f}{v_g - v_f}$$

$$\frac{\partial x}{\partial P} = \frac{-1}{(v_g - v_f)}\left[\frac{\partial v_f}{\partial P} + x\left(\frac{\partial v_g}{\partial P} - \frac{\partial v_f}{\partial P}\right)\right]$$

4.2 Thermal Conduction

The equation that governs the transfer of heat in solid material, such as the fuel pencils, is the thermal conduction equation. The variable of interest here is temperature, $T(\vec{r},t)$. The thermal conduction equation is dependent on time and spatial position,

$$\rho C_p \frac{\partial T(\vec{r},t)}{\partial t} + \vec{\nabla}\cdot\vec{q} = S(\vec{r},t)$$

where $S(\vec{r},t)$ is the internal heat source term, $\vec{q}$ is the heat flux term, and C_p is the specific heat. The assumption is that Fick's Law

$$\vec{q} = -k(\vec{r})\vec{\nabla}T(\vec{r},t)$$

accurately describes the heat flux.

The internal heat source term from nuclear fission is

$$S(\vec{r},t) = \int_0^{\infty} \omega_f(E')\Sigma_f(\vec{r},E')\Phi(\vec{r},E',t)dE'$$

The thermal conduction equation can now be written as

$$\frac{\partial T(\vec{r},t)}{\partial t} = \frac{1}{\rho C_p}[S(\vec{r},t) + \vec{\nabla}\cdot k(\vec{r})\vec{\nabla}T(\vec{r},t)]$$

This is the equation that will be used to describe the heat flow in the fuel. There is a convective interface between the fuel and the coolant. This is described in the section on interface conditions.

4.3 Boundary Conditions

Thermal Boundary Conditions

The conservation conditions that must be satisfied at a thermal interface are the conservation of temperature and thermal flux. If the interface is a convective interface, such as between the fuel and the coolant, a convective boundary condition might be used. The conservation of thermal flux boundary condition is written as

$$[\vec{n} \cdot \vec{q}]_{\vec{r}=surface} = k_{fuel}[\vec{n} \cdot \vec{\nabla} T_{fuel}(\vec{r},t)]_{\vec{r}=surface} = \bar{h}(T_{fuel}(\vec{r},t) - T_{\infty})_{\vec{r}=surface}$$

where T_{∞} is the bulk temperature of the coolant, and h is the heat transfer coefficient. This is referred to as *Newton's law of cooling*[83].

If the interface is a conductive interface, such as between two conductive materials, the conservation of thermal flux boundary condition is given by Fick's law

$$k_a[\vec{n} \cdot \vec{\nabla} T_a(\vec{r},t)]_{\vec{r}=surface} = k_b[\vec{n} \cdot \vec{\nabla} T_b(\vec{r},t)]_{\vec{r}=surface}$$

The conservation of temperature boundary condition is all cases would be

$$[T_a(\vec{r},t)]_{\vec{r}=surface} = [T_b(\vec{r},t)]_{\vec{r}=surface}$$

Fluid Boundary Conditions

The boundary conditions for fluid flow are the tangential and normal flow at the boundary. In all cases where the boundary is a wall, the tangential and normal flows will be assumed to be zero. If the boundary is not a wall, either the pressure or the flow normal to the boundary must be specified. It is common to specify the inlet flow and the outlet pressure.

---

83 Bird, Stewart & Lightfoot, p.267

5 Dynamics

The system in question, a nuclear fission reactor, has both a time and space dependent nature. The term dynamics is used to describe the time behavior of the system. There are two primary methods that are used to analyze the time behavior of systems, linear and nonlinear system theory.

5.1 Continuous Functions: Flows

Functions that are continuous (in time usually) will be referred to as flows. For some n dimensional vector a flow is defined as

$$f: u \rightarrow \Re^n \text{ where } u \subseteq \Re^n$$

where f is a smooth vector valued function that defined on some subset of n space, u. The vector field f generates a flow $\Psi(x,t)$. The flow may be considered the solution of an ODE defined by f

$$\psi_t: u \rightarrow \Re^n$$

$$x \in u \text{ and } \psi_t(s) \in \Re^n$$

where

$$\frac{d}{dt}(\psi(x,t))\Big|_{t=\tau} = f(\psi(x,\tau))$$

The initial conditions that must be satisfied are

$$\psi(x_0, 0) = x(0) = x_0$$

This flow defines a family of solution curve trajectories,

$$\psi(x_0, \cdot): I \rightarrow \Re^n$$

where $I = (a,b)$ *(interval).* For each initial condition x_0, the flow generates a trajectory $\Psi(x_0, t)$.

A special form of the function f is the linear function. The far more general case is the nonlinear function. An examination of linear functions, or linear systems, is required before nonlinear functions can be approached.

5.2 Linear Systems

The topic here is linear flows. These are encountered in the study of state space theory. The vector valued function being examined is a linear function

$$\dot{x} = f(x) = Ax$$

$$x \in \Re^n$$

where x is some vector in n space. A is therefore a 'n' square matrix.

In state space theory the definitions

$$A \equiv \text{state feedback matrix}$$

$$x \equiv \text{state vector}$$

are usually used.

This is an initial value problem. The time dependent behavior of the state vector can be calculated given its initial value. The time dependent matrix that governs the time behavior of the state vector is called the state transition matrix. The solution the rate equation given an initial condition x_o can be written as

$$x(x_0, t) = \exp(At)x_0$$

$$\exp(At) \equiv \text{state transition matrix}$$

A flow from n space to n space is defined by the matrix

$$\exp(At):\Re^n \rightarrow \Re^n$$

5.2.1 Algebraic Eigenproblem

The Algebraic Eigenproblem is how to decompose a matrix into its eigenvalues and eigenvectors, sometimes referred to as an eigensystem decomposition. An eigensystem decomposition can be performed on the state feedback matrix. The state feedback matrix has n eigenvalues and n eigenvectors. A matrix is said to be diagonalizable if it can be put in a diagonal form by using a similarity transformation. Some matrices can not be put in a diagonal form. In this case some of the eigenvectors will be *generalized* eigenvectors. All matrices that have distinct eigenvalues and some that don't can be diagonalized. For simplicity the following examples have distinct eigenvalues. The eigenvalues and eigenvectors can be determined (analytically) using the equations

$$det(\lambda_j I_n - A) = 0$$

$$(\lambda_j I_n - A)v_j = 0$$

where

$$\lambda_j \equiv \text{j th eigenvalue}$$

$$v_j \equiv \text{j th eigenvector}$$

Let T be the eigenvector matrix.

$$T = [v_1, v_2, \ldots, v_n]$$

The eigenvector matrix can be used to do an eigensystem decomposition of the state feedback matrix,

$$A = TJT^{-1}$$

The matrix J is the Jordan matrix. The state transition matrix can also be decomposed,

$$\exp(At) = T\exp(Jt)T^{-1}$$

If all the eigenvalues of the state feedback matrix are distinct, then the Jordan matrix has the eigenvalues along its diagonal,

$$J = \begin{bmatrix} \lambda_1 & 0 & . & . & 0 \\ 0 & . & . & . & . \\ . & . & \lambda_j & . & . \\ . & . & . & . & 0 \\ 0 & . & . & 0 & \lambda_n \end{bmatrix}$$

$$\Rightarrow \exp(Jt) = \begin{bmatrix} \exp(\lambda_1 t) & 0 & . & . & 0 \\ 0 & . & . & . & . \\ . & . & \exp(\lambda_j t) & . & . \\ . & . & . & . & 0 \\ 0 & . & . & 0 & \exp(\lambda_n t) \end{bmatrix}$$

If a matrix is not diagonalizable, then the Jordan matrix has ones along its superdiagonal,

$$J = \begin{bmatrix} \lambda_1 & 1 & . & . & 0 \\ 0 & . & . & . & . \\ . & . & \lambda_j & 1 & . \\ . & . & . & . & 1 \\ 0 & . & . & 0 & \lambda_n \end{bmatrix}$$

This problem is most often solved by a method that is referred to as "The QR Algorithm". This algorithm is used in the well known subroutine package, *EISPACK*.[84] A further discussion of this algorithm and its implementation can be found in *Linear Algebra*,[85] *The Algebraic Eigenvalue Problem*.[86]

5.2.2 Invariant Subspaces

The state transition matrix contains global information on the set of solutions of the matrix equation

$$\dot{x} = f(x) = Ax$$

The solutions of this equation lie in the linear subspaces spanned by the eigenvectors of the state feedback matrix. These subspaces are *invariant* under the state transition matrix.

84 J. Dongarra et al (1978)
85 J.H. Wilkinson & C. Reinsch (1971)
86 Wilkinson (1965)

For example, say that the initial state vector happens to be equal to an eigenvector, multiplied by a constant, of the state feedback matrix. The solution of this problem remains in the subspace spanned by this eigenvector for all time,

$$x_o = cv_j$$

$$Av_j = \lambda_j v_j$$

where 'c' is a constant.

$$x(cv_j, t) = \exp(At)cv_j$$

$$= \exp(\lambda_j)cv_j$$

$$x(cv_j, t) \in \text{Span}\{v_j\}$$

5.2.3 Stable, Unstable and Center Subspaces

The eigenvectors of the state feedback matrix span n space. These eigenvectors can be subdivided into subspaces according to the stability of the eigenvalues associated with them. In a continuous time systems stability is determined by whether the real part of an eigenvalue is negative, positive or zero. This corresponds to stable, unstable and center eigenvalues, respectively, of the state feedback matrix. The stable subspace is the subspace spanned by the eigenvectors associated with the stable eigenvalues, similarly for the unstable and center subspaces:

$$E^s = span\{v_1, \ldots, v_{n_s}\} \equiv \text{stable subspace}$$

$$E^u = span\{u_1, \ldots, u_{n_u}\} \equiv \text{unstable subspace}$$

$$E^c = span\{w_1, \ldots, w_{n_c}\} \equiv \text{center subspace}$$

5.3 Nonlinear systems

The vector valued function of interest here is a nonlinear function. The function is a smooth function with a solution to the initial value problem defined in some interval,

$$\dot{x} = f(x)$$

Fixed points are defined as the points in phase space where the function is zero,

$$f(\hat{x}) = 0$$

The linear behavior of the function near the fixed point can be characterized by the Jacobian evaluated at the fixed point,

$$\dot{\bar{x}} = A\bar{x}$$

$$\bar{x} = x - \hat{x}$$

$$A = \frac{\partial f(\hat{x})}{\partial x}$$

The Jacobian can be used to characterize the behavior of the function in some neighborhood of the fixed point. The Jacobian can be analyzed like the state feedback matrix in the linear system to determine the stability of the function near the fixed point. If the Jacobian has no zero or purely imaginary eigenvalues (the center subspace is empty), at the fixed point, then the stability of the fixed point can be determined.

5.3.1 Local Stable and Unstable Manifolds

This is analogous to the stable and unstable subspaces of the linear system. The definitions of the stable and unstable manifolds in a neighborhood of a fixed point are:

$$W^s_{loc}(\hat{x}) = \{x \in U \mid \psi_t(x) \to \hat{x} \text{ as } t \to \infty \text{ and } \psi_t(x) \in U, \ \forall t \geq 0\}$$

$$W^u_{loc}(\hat{x}) = \{x \in U \mid \psi_t(x) \to \hat{x} \text{ as } t \to -\infty \text{ and } \psi_t(x) \in U, \ \forall t \leq 0\}$$

If the stable and unstable subspaces of the fixed point are defined as the subspaces of the Jacobian at the fixed point then the stable and unstable manifolds are tangential to the stable and unstable subspaces, respectively, at the fixed point. This can easily be deduced as the Jacobian is a linear approximation of the function at the fixed point.

5.4 Discrete Functions: Maps

Functions that are discrete in time are referred to as maps. Difference equations in discrete time are analogous to differential equations in continuous time systems. As in continuous time systems, there can be linear and nonlinear maps. An example for some vector x is

$$x_{k+1} = Bx_k \quad \text{or} \quad x \mapsto Bx \ \text{ linear map}$$

$$x_{k+1} = G(x_k) \quad \text{or} \quad x \mapsto G(x) \ \text{ nonlinear map}$$

where G is a vector valued function.

All computer simulations are maps. In order to simulate a set of equations on a computer, the equations must first be discretized. The stability of discrete time systems has to be handled a little differently from continuous time systems. The stability of a linear map can be determined from the eigenvalues of the state feedback matrix, B in the above equation, in a fashion similar to a linear flow. The primary difference is that stable eigenvalues are now the eigenvalues who's absolute values are less than one. Unstable eigenvalues have absolute value greater than one. Stable, unstable and center subspaces can be defined in a manner analogous to linear vector fields.

The linear map defined by the difference equation

$$x_{k+1} = Ax_k$$

has solution

$$x_k = A^k x_0$$

This state feedback matrix can be decomposed just as in the continuous time case,

$$A = TJT^{-1}$$

$$\Rightarrow A^k = TJ^kT^{-1}$$

$$J^k = \begin{bmatrix} \lambda_1^k & 0 & . & . & 0 \\ 0 & . & . & . & . \\ . & . & \lambda_j^k & . & . \\ . & . & . & . & 0 \\ 0 & . & . & 0 & \lambda_n^k \end{bmatrix}$$

For stability, the state vector must remain bounded,

$$\lim_{k \to \infty} x_k < \infty$$

This implies that the eigenvalues of the state feedback must have absolute value less than or equal to one

$$|\lambda_j| \leq 1 \Rightarrow \text{stability} \qquad \forall j$$

in order for the solution to be stable.

6 Numerical Methods

The theory that governs the processes involved is of a physical nature and the theory that governs the solution of the equations and analysis is of a mathematical nature. The mathematical methods employed when solving these equations on digital computers are called numerical methods. The method used to solve an equation can have dramatic effects on the solution and the amount of computational effort required. Approximations must always be made when solving equations numerically. Which methods are best suited to which problems is the subject of this chapter.

6.1 Finite Differencing

A continuous function is represented by a series of discrete points that represent the value of the functions at those points in space and time. Partial and ordinary differential equations are approximated by difference equations. Derivatives are approximated by differences. The finite differences that are used to approximate the derivatives can be derived using combinations of Taylor series expansions. For example, let us assume that we wish to find the finite difference approximation to $\partial\psi(x)/\partial x$ at the mesh points $(x = i\Delta x)$. There are infinitely many approximations that may be used. We need to decide as to what spatial grid points we wish to use and the desired accuracy of the approximation. Assume that we wish to use the two spatial grid points ψ_i and ψ_{i+1}. The Taylor series expansion of ψ_{i+1} about the spatial grid point $(x = i\Delta x)$ is

$$\Psi_{i+1} = \Psi_i + \frac{\partial \Psi_i}{\partial x}(\Delta x) + \frac{1}{2!}\frac{\partial^2 \Psi_i}{\partial x^2}(\Delta x)^2 + O[(\Delta x)^3]$$

The *backward difference* approximation of the derivative can now easily be derived using the above equation,

$$\frac{\partial \Psi_i}{\partial x} = \frac{(\Psi_{i+1} - \Psi_i)}{\Delta x} + O[(\Delta x)]$$

The $O[(\Delta x)^n]$ symbol is used to signify the accuracy of the expansion, it indicates the order of the truncation error. It means that the remaining terms are at most a constant multiple of the expression in brackets as $(\Delta x \to 0)$. The error of the backward difference approximation is proportional to Δx. The *central difference* approximation, with an error that is proportional to $(\Delta x)^2$, can be derived using the above expansion and an expansion of Ψ_{i-1} about Ψ_i,

$$\frac{\partial \Psi_i}{\partial x} = \frac{(\Psi_{i+1} - \Psi_{i-1})}{2\Delta x} + O[(\Delta x)^2]$$

Higher order differencing may be used if greater accuracy is required. In theory any order of differencing is possible but in practice usually quite low order of differencing is used due to the increased computational burden associated with high order differencing. Higher order derivatives can be approximated in the same

manner. The finite difference approximation of the second order derivative $\partial^2\Psi_i/\partial x^2$ that is most commonly used is the central difference approximation,

$$\frac{\partial^2\Psi_i}{\partial x^2} = \frac{(\Psi_{i-1} - 2\Psi_i + \Psi_{i-1})}{(\Delta x)^2} + O[(\Delta x)^2]$$

This approximation will be used extensively in this simulation.

6.2 The Solution of Partial Differential Equations

In all but a few very simple cases, Partial Differential Equations (PDEs) are impossible to solve analytically. Some numerical techniques will be required for the solution of the equation. Deterministic methods of solution of PDEs are common but sometimes stochastic methods, such as Monte-Carlo, are used. The Method of Lines used here is a deterministic method.

PDE Classification

The equations governing the physical processes have quite distinctive features thus the numerical methods used to solve the different PDEs will have some fundamental differences. PDEs are classified according to certain characteristics. To elucidate these characteristics, an example is used.

Consider the second order equation

$$a u_{xx} + b u_{xy} + c u_{yy} = f$$

where a,b,c and f are functions of x,y,u,u_x and u_y (the subscript denotes partial differentiation). This equation can be reduced to a system of first order equations[87]:

$$d(u_x) = u_{xx}dx + u_{xy}dy$$

$$d(u_y) = u_{xy}dx + u_{yy}dy$$

Together with the original equation, these equations can be written in a matrix form

$$\begin{bmatrix} a & b & c \\ dx & dy & 0 \\ 0 & dx & dy \end{bmatrix} \begin{bmatrix} u_{xx} \\ u_{xy} \\ u_{yy} \end{bmatrix} = \begin{bmatrix} f \\ d(u_x) \\ d(u_y) \end{bmatrix}$$

If the determinant of the matrix is nonzero, then a unique solution exists, $a(dy)^2 + b\,dy\,dx + c(dx)^2 = 0$

The characteristic equation for this system is then $b^2 - 4ac$. The original equation is hyperbolic, parabolic and elliptic if the characteristic equation is greater than, equal to and less than zero respectively. An example of each of these types of PDE is: parabolic, the diffusion equation, $u_t = \alpha u_{xx}$; elliptic, Laplace's equation, $u_{xx} + u_{yy} = 0$; and hyperbolic, the wave equation, $u_{tt} = c^2 u_{xx}$.

87 Ames (1977) p.7

The Method of Lines

The Method of Lines is a numerical technique in which all but one of the independent variables of a partial differential equation (PDE) are discretized. The variable that is not discretized here initially is time. Spatial discretization using a finite difference approximation is used. This method, which can be used with a variety of coordinate systems and dimensions, gives rise to a system of coupled ordinary differential equations (ODEs). This system of equations can be written as a single time dependent vector equation in which all the dependent variables are placed in a state vector. After the spatial discretization, the time dependent equation is solved using whatever ODE solver is best suited to the problem. The Method of Lines allows flexibility in the choice of the method used to solve the time dependent portion of the problem.

The derivative of the rate of change of the state vector with respect to the state vector itself gives rise to what is called the Jacobian matrix. The Jacobian matrix of the vector equation has a number of nonzero bands parallel to the diagonal. This type of Jacobian matrix is said to be banded. The structure of the Jacobian matrix is of great importance in the solution of the problem since this structure can be exploited in order to achieve a more numerically efficient solution. The properties of this vector form of the equation is the subject of investigation here.

6.3 Jacobian Matrix Properties

The properties of the Jacobian matrix arising in the solution of PDEs is dependent on several factors. The number and location of the bands is a function of the spatial differencing and the spatial dimension. The higher the order of differencing and the higher the spatial dimension, the more bands will appear in the Jacobian. Other properties of the matrix, such as whether it is symmetric or not, depends on the PDE itself as well as the differencing.

If the PDE is linear and Dirichlet boundary conditions are used, the matrix will be symmetric and negative semi definite. If Neumann or Cauchy boundary conditions are used, the matrix may be asymmetric even if the PDE is linear. All but very few nonlinear PDEs will produce asymmetric matrices.

Example Equations: Conductive heat transfer and convective heat transfer.

The conductive heat transfer equation,

$$\frac{\partial T(\vec{r},t)}{\partial t} = \frac{k}{\rho C_p} \nabla^2 T(\vec{r},t)$$

is a linear PDE. The convective heat transfer equation,

$$\frac{\partial T(\vec{r},t)}{\partial t} + \vec{V}(\vec{r},t) \cdot \vec{\nabla} T(\vec{r},t) = \frac{k}{\rho C_p} \nabla^2 T(\vec{r},t)$$

is a nonlinear PDE due to the convective term, which a product of velocity and temperature. It is assumed that the spatial grid is constant and that the coefficient of heat conduction is not dependent on temperature. The convective heat transfer equation is for a single phase fluid of constant density. For the sake of illustration, these equations are simplified versions of those used in the simulation.

Spatial Geometry: Two dimensional X Y geometry.

The spatial geometry used in these examples is two-dimensional X-Y. Second order central differencing is used for the second order derivatives in the Laplacian term and first order backward differencing for the convective term in the convective heat transfer equation. The temperature is approximated by the temperature at the spatial grid points,

$$T(x,y,t) \approx T(t)_{i,j} \qquad \text{for } 1 \le i \le n_x \text{ and } 1 \le j \le n_y$$

The temperature variables for all the spatial grid points are placed into one vector. This is the data vector mentioned above. The second subscript is advanced first in the ordering of the grid points used here,

$$\vec{T}(t) = \left[T_{1,1}(t), T_{1,2}(t), \ldots, T_{1,n_y}(t), T_{2,1}(t), \ldots, T_{2,n_y}(t), \ldots, T_{n_x,n_y}(t)\right]$$

The order in which the subscripts are advanced effects the locations of the bands in the Jacobian matrix. A consistent ordering scheme is required for recognizable patterns in the Jacobian.

The General Matrix Form

The vector equation to be solved now is an n dimensional ($n = n_x n_y$) linear system,

$$\frac{d}{dt}\vec{T}(t) = \overline{A}(t)\vec{T}(t)$$

The Jacobian matrix as defined above now appears as

$$\overline{A}(t) = \frac{\partial\left(\frac{d}{dt}\vec{T}(t)\right)}{\partial(\vec{T}(t))}$$

The Jacobian for a linear PDE will be a constant and hence has no time dependence. For a nonlinear PDE, the Jacobian is not a constant and will need to be updated periodically hence, the time dependence above. The frequency of updating the Jacobian depends on its rate of change.

The Conductive Heat Transfer Equation

The conductive heat transfer equation is now spatially discretized. This differencing scheme is the common central differencing scheme,

$$\frac{d}{dt}T_{i,j}(t)=\frac{\alpha}{(\Delta x)^2}(T_{(i+1),j}(t)-2T_{i,j}(t)+T_{(i-1),j})+\frac{\alpha}{(\Delta y)^2}(T_{i,(j+1)}(t)-2T_{i,j}(t)+T_{i,(j-1)})$$

The constant $\alpha = k/\rho C_p$ is called the thermal diffusivity. This is called "five point differencing", as five spatial points are involved. The resulting Jacobian has five bands. The first band, the one furthest below the diagonal, is at a distance n_x below the diagonal. The second band is directly below the diagonal (a sub-diagonal) while the third band is on the diagonal. Bands four and five are the same distance above the diagonal as two and one are below the diagonal respectively. The elements of bands one and five are equal to $k'/(\Delta y)^2$. The elements of bands two and four are equal to $k'/(\Delta x)^2$. The central band is the inverse sum of the four other bands $-2k'/(1/(\Delta x)^2 + 1/(\Delta y)^2)$. It can be shown that the matrix is symmetric, diagonally dominant and negative semidefinite. Diagonal dominance is defined by

$$|a_{ii}| \geq \sum_{j=1, j\neq i}^{n} |a_{ij}|$$

$$\geq \sum_{j=1, j\neq i}^{n} |a_{ji}|$$

The properties of the Jacobian as described here can be exploited for efficient numerical solutions of the equations.

The Convective Heat Transfer Equation

With the exception of the convection term, the convective heat transfer equation is identical to the conductive equation. This exception is significant. Due to the nonlinearity of the PDE the Jacobian will not be symmetric, diagonally dominant or in any definitive state due to the fact that the state of the matrix depends on the magnitude and direction of the velocity term. Obviously, the equation reduces to the conductive heat transfer equation for zero velocity.

The Jacobian has five bands in the same locations as before, assuming the same differencing scheme and first order differencing is applied to the convective term. Backward differencing is assumed for the first order derivatives,

$$\frac{\partial T}{\partial x}=\frac{T_{i,j}-T_{i-1,j}}{\Delta x}$$

$$\frac{\partial T}{\partial y}=\frac{T_{i,j}-T_{i,j-1}}{\Delta y}$$

The elements in band one will be $k'/(\Delta y)^2 + Vy_{i,j}/\Delta y$, the elements in band two will be $k'/(\Delta x)^2 + Vx_{i,j}/\Delta x$ and the elements in bands four and five will be unchanged. The diagonal band will again be the inverse sum of the four other bands $-2k'(1/(\Delta x)^2 + 1/(\Delta y)^2) - Vx_{i,j}/\Delta x - Vy_{i,j}/\Delta y$.

6.4 The Solution of Ordinary Differential Equations

The numerical solution of ordinary differential equations (ODEs) is a broad topic about which much has been written. The methods of integration of ODEs can be place into three main categories: single-step methods, multi-step methods and predictor-corrector methods[85]. All the methods used here fall into the single-step category, therefore the discussion will limited to single-step methods.

The system of ODEs to be solved has the form

$$\frac{d\bar{y}}{dx} = \bar{y}'(x) = \bar{f}(x, y)$$

where y is a vector, and x is the independent variable. Let the solution of this equation at $x=x_n$ be given by $y(x_n)=y_n$ and let the step size be $h=x_{n+1}-x_n$ The Taylor series expansion of $y(x_{n+1})$ is

$$y(x_{n+1}) = y(x_n) + h\frac{dy(x_n)}{dx} + \frac{h^2}{2!}\frac{dy^2(x_n)}{dx^2} + \frac{h^3}{3!}\frac{dy^3(x_n)}{dx^3} + \ldots$$

This expansion will be used to determine the coefficients of some of the methods used.

Single Step Methods

The simplest method of ODE integration is *Euler's method*,

$$y_{n+1} = y_n + hy'_n + O(h^2)$$

This is derived by truncating the Taylor expansion at two terms.
This is an explicit method which uses only information that is available at step x_n on the RHS.

Iterpolation may be used to represent the derivative on the RHS. A popular method of this type is the *modified Euler formula* or the *Crank-Nicolson method*,

$$y_{n+1} = y_n + (h/2)(y'_n + y'_{n+1}) + O(h^3)$$

This is a semi-implicit method which uses information that is not available at step x_n, and hence matrix inversion will be required.

This may be generalized even further to some general semi-implicit method with a variable coefficient,

$$y_{n+1} = y_n + (h/2)((1-\alpha)y'_n + \alpha y'_{n+1})$$

The truncation error for this method has is a minimum when $\alpha = 1/2$, which is the Crank-Nicolson method.

---

88 Lapidus & Seinfeld (1971)

Implicit and semi-implicit methods have a larger radius of convergence, that is they permit a larger step size to be used. The price to be paid is that they are more computationally demanding. The trade-off is between using larger step sizes and the amount of computational effort required per step. Which method is best depends on the problem and how efficiently matrix inversion may be implemented.

Runge-Kutta Methods

The most popular of the single step methods are the Runge-Kutta methods. Higher order derivatives need not be evaluated when using R-K methods and there is a large variety of methods of this type available. The particular method used can be customized to the problem being solved.

The general form of Runge-Kutta methods is

$$y_{n+1} = y_n + \sum_{i=1}^{v} w_i k_i$$

where w_i are the weighting coefficients to be determined, v is the order of the method, and k_i satisfy the sequence,

$$k_i = hf\left(x_n + c_i h, y_n + \sum_{j=1}^{m} a_{ij} k_j\right)$$

where m is the upper limit of the summation. Some identities that characterize Runge-Kutta methods are

$$\sum_{i=1}^{v} w_i = 1$$

$$c_i = \sum_{j=1}^{m} a_{ij} \qquad i \neq 1, i = 2, \ldots, v$$

A more condensed notation may achieved by placing all the $\{a_{ij}\}$ into a matrix, call it A_{RK}, all the $\{c_i\}$ into a vector, C_{RK}, and all the $\{w_i\}$ into a vector, W_{RK}.
In the design of a R-K method, the $\{c_i\}$ are taken to be free parameters and the remaining parameters are evaluated using the method of undetermined coefficients[89] or other methods such as Gaussian quadrature[90]. Truncation error analysis of R-K methods is quite involved and beyond the scope this introduction[91].

These methods may subdivided into three arrangements according to the value of m[92].

89 ibidem p.17
90 ibedem p.43
91 ibidem p.56
92 ibidem p. 60 and Butcher (1964)

Explicit	Total of $v(v+1)/2$ parameters to choose. Upper limit on summation is $i-1$, in any k_i use only k_j to k_{i-1}.
Semi-Implicit	Total of $v(v+3)/2$ parameters to choose. Upper limit on summation is i, in any k_i use only k_j to k_i.
Implicit	Total of $v(v+1)$ parameters to choose. Upper limit on summation is v, in any k_i use k_j to k_v.

The coefficients of the matrix A_{RK} are nonzero only below the diagonal for explicit methods, are nonzero only on and below the diagonal for semi-implicit methods, and are unrestricted for implicit methods. Fully implicit methods will require iteration to solve for the various k_i, and semi-implicit methods will require matrix inversion.

The most popular Runge-Kutta methods are the explicit methods. Iteration or matrix inversion is not required due to their explicit nature. The accuracy of the method can be traded off against computational effort, increasing the order of the method increases both accuracy and computational effort. Like any explicit method, the stability of the method will limit the maximum step that can be used. Accuracy of the solution will also depend on the step size and adaptive step size control is usually used to control the accuracy[93].

The explicit fourth order R-K solver that is used here to solve some of the ODEs is a fairly typical fourth order method:

$$k_1 = hf(x_n, y_n)$$

$$k_2 = hf(x_n + h/2, y_n + k_1/2)$$

$$k_3 = hf(x_n + h/2, y_n + k_2/2)$$

$$k_4 = hf(x_n + h, y_n + k_3)$$

$$y_{n+1} = y_n + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4)$$

The motivation for semi-implicit methods lies in obtaining stable Runge-Kutta processed as well as maintaining computational efficiency. A second order semi-implicit method has the form

$$y_{n+1} = w_1 k_1 + w_2 k_2$$

$$k_1 = h[I - a_1 \vec{A}(y_n)]^{-1} f(y_n)$$

$$k_2 = h[I - a_2 \vec{A}(y_n + c_1 k_1)]^{-1} f(y_n + b_1 k_1)$$

where $\vec{A}$ is the Jacobian matrix.

93 Press, Flannery, Teukolsky & Vetterling (1988) p.574

The semi-implicit R-K method used here is that suggested by Calahan (1968).

$a_1 = a_2 = 0.788675$ $\quad c_1 = 0$

$b_1 = -1.15470054$ $\quad w_1 = 0.75$ $\quad w_2 = 0.25$

Both the Gauss-Seidel and LU Decomposition methods of matrix inversion are used in the implementation of this R-K method. If the system to be solved is linear, then the matrices to be inverted are invariant. This implies that the matrix need only be decomposed once if LU Decomposition is used for matrix inversion. This represents a significant savings of computation effort and is one of the advantages of the method.

6.4.1 System of Linear Equations

The ODE problem that will be most commonly solved is a system of first order linear ODEs that arises from the spatial differencing of the PDEs.

The system to be solved has the form

$$\frac{d}{dt}X(t) = A(t)X(t) + B(t)$$

where $X(t)$ is a time dependent vector, $A(t)$ is the Jacobian matrix and, $B(t)$ is a vector. The time dependence of the matrix A will depend on the problem being solved. For linear equations, such as the thermal conduction equation and the neutron multigroup diffusion equation, the matrix A will be a constant. The matrix A is not a constant for nonlinear equations, such as any of the thermalhydraulic equations, where it is assumed constant only for a certain interval.

This system of equations can be solved by a variety of methods such as suggested by the previous section. Euler's method and the general semi-implicit method for example would respectively be:

$$X_{n+1} = (I + hA)X_n$$

$$X_{n+1} = [I - h\alpha A]^{-1}[(I + h(1-\alpha)A)\overline{X}_n + h\overline{B}]$$

When the parameter α is set equal to one half, the result is the Crank-Nicolson method, which has the lowest truncation error.

The matrix A will be banded. How these bands arise is discussed in the section on Jacobian matrix properties. Efficient methods of inverting the matrix must take the sparsity of this matrix into account.

Assume that we are dealing with the conductive heat transfer equation in two dimensional X-Y geometry, as outlined in the previous section, and we solve the resulting linear system of equations in an explicit man-

ner. The properties of A will be changed. Previously A was symmetric, diagonally dominant and negative semidefinite. The matrix that we will need to invert, call it A', will be symmetric, diagonally dominant and positive definite. These properties are ideal for iterative solution techniques such as Gauss-Seidel.

6.5 Matrix Inversion

The techniques of numerical linear algebra for matrix inversion can be classified in three categories: direct methods, such as the LU decomposition method; iterative methods, such as Gauss-Seidel method; and Semi-iterative methods, such as the conjugate gradient method (see section 1.4.2). Both the Gauss-Seidel method and the Conjugate Gradient method can be implemented on matrices that are stored using the banded storage method introduced here. If the matrix to be inverted is in a block tridiagonal form, algorithms that exploit this structure may be employed.

Matrix partitioning will help reduce that storage requirements for the solution of the linear problem, but will not reduce the computational requirements.[94] It is a useful method where storage space is limited. This method is used in conjunction with matrix partitioning in the thermalhydraulic simulation to reduce both storage and computational requirements.

Direct Methods

When using the *LU decomposition* method, a matrix is factored into lower and upper tridiagonal matrices, $A = LU$, hence the name. The Gauss transformation is the most common method of accomplishing this. The banded matrix structure is partially preserved under an LU decomposition[95]. The bandwidth is preserved but any bands of zeros between nonzero bands, such as in the two dimensional case, will be filled. If the bands of a matrix are widely spaced, (the matrix has a wide bandwidth) little increase in performance will be possible by taking into account the banded structure of the matrix. The matrix will have a tendency to fill up when using the LU decomposition method. If the matrix has a narrow bandwidth, a significant increase in performance can be realized by using algorithms for LU decomposition of banded systems.

After the matrix A is decomposed into an upper triangular matrix and a lower triangular matrix, $A = LU$, the solution of the linear problem, $Ax = LUx = b$, is handled in two steps. The first step is the solution of $Ly = b$, the lower triangular matrix L is inverted to arrive at the intermediate solution y. The second step is the solution of $Ux = y$, the matrix U is inverted to arrive at the desired solution x.

When the Laplacian term in a PDE is finite differenced, certain structures arise in the Jacobian matrix (depending on the variable ordering - natural ordering is assumed). In one dimension, the Jacobian matrix will be a tridiagonal matrix, and a tridiagonal matrix inversion algorithm can be used[96]. In two or more

94 Westlake (1968), p.26
95 Dahlquist & Anderson, (1974) p.165
96 Dahlquist & Anderson (1974) p.166

dimensions, the Jacobian will be a block tridiagonal matrix, and an algorithm that takes advantage of this matrix structure is *block tridiagonal LU decomposition*[97]. These matrices are frequently diagonally dominant and iterative methods are sometimes more efficient.

Iterative Methods

The *Gauss-Seidel* method of solution of a linear system is an iterative method in which A is unaltered and thereby preserving the banded matrix structure. By taking into account this sparsity, a substantial increase in performance is possible even if the bands are widely spaced. This is the main advantage of the Gauss-Seidel method. This method requires that A be diagonally dominant. While this is not a problem for linear PDEs, such as the thermal conduction equation, this is a problem for nonlinear PDEs, such as the thermalhydraulic equations. The time step size may need to be reduced to assure that

$$A' = \left(\frac{1}{\Delta t} I - A \right)$$

is diagonally dominant.

For the conductive heat transfer equation, the diagonal elements are equal to

$$a'_{ii} = \frac{1}{\Delta t} + 2k' \left(\frac{1}{(\Delta x)^2} + \frac{1}{(\Delta y)^2} \right)$$

and it follows that

$$|a'_{ii}| > \sum_{j=1, j \neq i}^{n} |a'_{ij}|$$

hence A' is diagonally dominant. It can be shown that A' is positive definite as well.

For the convective heat transfer equation, the diagonal elements are

$$a'_{ii} = \frac{1}{\Delta t} + 2k' \left(\frac{1}{(\Delta x)^2} + \frac{1}{(\Delta y)^2} \right) + \frac{Vx}{\Delta x} + \frac{Vy}{\Delta y}$$

This does not allow you to draw any conclusions about the state of A'. It can be seen, however, that if the time step is sufficiently small, the matrix can be made diagonally dominant. If you wish to see all the transient effects when running a simulation, this is not a problem. However, if you wish to advance the simulation rapidly in time with large time steps another method of matrix inversion may be required.

---

97 Golub & Van Loan (1983), p.110

The Gauss-Seidel (GS) method splits the matrix A into a lower triangular, a diagonal, and a upper triangular matrix, $A = D(L + I + U)$, $D = diag(a_{ii})$. The successive approximation for the solution can be described by

$$x^{k+1} = -Lx^{k+1} - Ux^{k} + D^{-1}b$$

where k indicates the approximation number. The successive overrelaxation (SOR) method may be used to accelerate convergence of the GS method. The current GS method can be written as

$$x_i^{k+1} = x_i^k + r_i^k$$

where

$$r_i^k = \frac{-\sum_{j=1}^{i-1} a_{ij}x_j^{k+1} - \sum_{j=i}^{n} a_{ij}x_j^k + b_i}{a_{ii}}$$

The SOR method that is used to accelerate convergence can be written as

$$x_i^{k+1} = (1-\omega)x_i^k + \omega r_i^k$$

where ω is the relaxation parameter that is chosen to maximize convergence. A version of this method in which only the bands of A are stored is used. This is simply called the *Banded Gauss-Seidel* (BGS) routine.

Semi-Iterative Methods

The semi-iterative method that is used here is the *Conjugate Gradient* (CG) method. The Conjugate Gradient method is not a single method but a whole family of methods. The original conjugate gradient method was designed for use with symmetric positive definite (SPD) matrices. This method can be generalized to an arbitrary nonsingular matrix with a simple transformation to give rise to the *Generalize Conjugate Gradient* (GCG) method. If the matrix A is nonsingular, then the matrix A^TA is symmetric positive definite. The linear system that will be solved now is: $A^TA\,x = A^Tb$. This method has poor convergence properties as a consequence of the matrix A^TA having a *spectral condition number*[98] that is in general much greater than that of A. This method can use the same banded storage method as is used in the BGS routine, and as a consequence savings can be made on storage requirements.

The GCG method uses three vectors: x^i is the i th approximation to the solution, r^i is the i th residue where the residue is defined as $r = b - Ax$, and p^i is the i th gradient direction. The vectors $\{p^i\}$ are linearly independent and Gram-Schmidt orthogonalization may be used to ensure that the new direction vectors are indeed orthogonal to the previous direction vectors (this may not be the case due to cumulative truncation error). The algorithm can be written as follows.

98 Hageman & Young (1981), p.9 & 331

Calculate the initial values r^0 and p^0 given the initial value x^0,

$$r^0 = b - Ax^0 \qquad p^0 = A^T r^0$$

Begin the iteration. The first step in the iteration is to normalize p,

$$p^i \leftarrow \frac{p^i}{|Ap^i|}$$

The coefficient α_i is calculated and used to update the values of x and r, $\alpha_i = (r^i, Ap^i)$

$$x^{i+1} = x^i + \alpha_i p^i$$

$$r^{i+1} = r^i - \alpha_i A p^i$$

where the expression $(\cdot,\cdot)$ is used to indicate inner (scalar) product.

The coefficient

$$\beta_i = -(Ap^i, AA^T r^{i+1})$$

is calculated and used to update the gradient direction p,

$$p^{i+1} = A^T r^{i+1} + \beta_i p^i$$

The iteration is continued for a maximum of n iterations where n is the dimension of the system.

Matrix Partitioning

Matrix partitioning subdivides the linear system problem into two lower order problems. The standard linear system problem is: $Ax = b$ and $dim(A) = n$ *by* n.
This can be written as two lower order problems,

$$\begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} b_1 \\ b_1 \end{bmatrix}$$

where A_{11} and A_{22} are square matrices. The first step is substitute for one of the data vectors,

$$x_1 = A_{11}^{-1}(b_1 - A_{12}x_2)$$

This expression is substituted into the second equation and that equation is then solved. Any method may be used to invert the matrix A_{11}, preferably some method that takes advantage of any structure it might have.
The resultant matrix,

$$(A_{22} - A_{21}A_{11}^{-1}A_{12})x_2 = b_2 - A_{21}A_{11}^{-1}b_1$$

is usually quite full and some direct method will probably be required to invert it. The result X_2 is back substituted to solve for the first data vector, X_1. This method of two stage matrix inversion will reduce memory requirements, but may not reduce computational effort although in some circumstances it might, depending on structure of the component matrices.

6.6 Computational Effort and Storage Requirements

The usual method of measuring how much computational effort that an algorithm requires is to measure the number of floating-point operations (flops) that are required. Obviously an algorithm that requires fewer flops than another that accomplishes the equivalent task is the better algorithm. It is always desirable to minimize the amount of storage that a simulation requires. The more efficient a simulation is with data storage, the less page swapping (of memory) that will be required and hence the faster the simulation will run. Above all an algorithm must be robust, for if the simulation fails (crashes) little is gained. These are the criteria that will be used to evaluate a few algorithms for the solution of an n dimensional linear system.

The full storage method is where the entire matrix is stored and any sparsity of the matrix is ignored. This requires n^2 storage locations. Using Gaussian elimination with pivoting and ignoring the banded structure will require $n^3/3$ flops for the solution of such a system. This is the least efficient method, both in terms of computational effort and storage.

If the matrix to be inverted is asymmetric and banded, a certain amount of computation time may be saved by taking into account the banded structure. The subrouuine package LINPACK stores a banded matrix in such way that the bands are on the bottom rows of the matrix. The same number of storage locations as the full storage method are required but some may not be used. The band widths, upper and lower, are preserved under LU decomposition. If the upper bandwidth, say p, is equal to the lower bandwidth, then $np^2-(2/3)p^3+np$ flops will be required for LU decomposition. An additional $np-p^2/2$ flops will be required for back substitution.

The Gauss-Seidel method of solution of a linear system is an iterative solution that may be carried on to any desired accuracy. In order to use this method that matrix that is to be inverted must be diagonally dominant, otherwise it will not converge. Usually successive overrelaxation (SOR) methods are used with the Gauss-Seidel iteration to speed convergence. The number of flops that are required will depend on the number of iterations. Therefore quick convergence of the iteration is desirable. If the banded structure of the matrix is ignored n^2 flops per iteration are required. The number of flops that are required per iteration is equal to the number of nonzero elements in the matrix. If we have a five band matrix, of any bandwidth, less that $5n$ flops per iteration will be required. If only the bands are stored, $5n$ storage locations are required.

A Conjugate Gradient method for symmetric positive definite matrices requires $5n$ flops and a vector matrix multiplication per iteration. Theoretically, a maximum of n iterations are required for convergence.

The matrix restrictions can be relaxed if the matrix is preconditioned. This will require some operations but will significantly accelerate convergence. Other methods of acceleration convergence are also available.[99]

There are other less common methods that are available. The Householder transformation, requiring $2n^2$ flops, can be used to convert a banded symmetric matrix to a symmetric tridiagonal matrix and a banded asymmetric matrix to an upper Heisenberg matrix. The solution of a tridiagonal symmetric system requires $5n$ flops. The LU decomposition of an upper Heisenberg matrix, of bandwidth p, requires approximately $np/2$ flops and a few more flops will be required for back substitution.

6.7 Boundary Conditions

There are three general types of boundary conditions: Dirichlet, Neumann and Cauchy. Dirichlet boundary conditions specify the value of the function at the boundary. Neumann boundary conditions specify the gradient of the function at the boundary and Cauchy boundary conditions specify a linear combination of the function and its gradient at the boundary.

6.7.1 Dirichlet Boundary Conditions

The value of the function is fixed at the boundary using these boundary conditions,

$$\Psi(\vec{r},t)\big|_{\vec{r}=interface} = constant$$

Say that the temperature is fixed at the left boundary, $T_{0,j}(t) = constant = T_0$.

A constant will arise in the rate equation and will determine some elements in the boundary condition vector B,

$$\frac{dT_{1,j}(t)}{dt} = \frac{k}{\rho C_p}\left[\frac{(T_{2,j}(t) - 2T_{1,j}(t) + T_{0,j}(t))}{(\Delta x)^2} + \frac{(T_{1,j+1}(t) - 2T_{1,j}(t) + T_{1,j-1}(t))}{(\Delta y)^2}\right]$$

99 G.H. Golub & C.F. Van Loan (1983), p.375.

$$\Rightarrow \vec{B}_T = \begin{bmatrix} \frac{kT_0}{\rho C_p(\Delta x)^2} \\ \cdot \\ \cdot \\ \frac{kT_0}{\rho C_p(\Delta x)^2} \\ 0 \\ \cdot \\ \cdot \\ 0 \end{bmatrix}$$

Dirichlet boundary conditions can be used at the outside boundary in the neutronics equations but, with what is referred to as extrapolated boundary conditions. This means that the flux goes to zero some small distance beyond the outside boundary $\Phi_d^g = 0$. This may be used to determine the boundary condition vector in a way that is very similar to that in the temperature equation. The boundary condition vector will be zero using these boundary conditions $\vec{B}_\Phi = [0, \ldots, 0]^T$. The other boundary condition that may be used at the outside boundary of the flux rate equations is the Marshak boundary condition which is a Cauchy type boundary condition.

6.7.2 Neumann Boundary Conditions

The gradient of the function is a constant at the boundary when Neumann boundary conditions are used. When the gradient of the function is set to zero at the boundary this is referred to as reflective or insulated boundary conditions. Reflective boundary conditions assume that there is no net thermal flux or neutron current cross the boundary,

$$\vec{n} \cdot k\vec{\nabla}T(\vec{r},t)\big|_{\vec{r}=interface} = 0 \qquad \vec{n} \cdot D_g\vec{\nabla}\Phi_g(\vec{r},t)\big|_{\vec{r}=interface} = 0$$

In the one dimensional problem this would specify the derivative $\partial\Phi/\partial x = 0$ at the boundary. Assume that the left (starting) boundary is a reflective boundary. The Taylor expansion of Ψ_1 about Ψ_0 is

$$\Psi_1 = \Psi_0 + (\Delta x)\frac{\partial\Psi_0}{\partial x} + \frac{(\Delta x)^2}{2}\frac{\partial^2\Psi_0}{\partial x^2} + \frac{(\Delta x)^3}{3!}\frac{\partial^3\Psi_0}{\partial x^3} + \ldots$$

To implement the reflective boundary condition, set the first order derivative in the above expansion to zero. The resultant expression for the second derivative (usually the Laplacian) will be

$$\frac{\partial^2\Psi_0}{\partial x^2} = \frac{2}{(\Delta x)^2}(\Psi_1 - \Psi_0) + O[(\Delta x)]$$

This expression is used to approximated the second derivative at a reflective boundary.

6.7.3 Cauchy Boundary Conditions

Cauchy boundary conditions are when a linear combination of the function and its gradient is specified at the boundary. This type of boundary condition occurs at the interface between two media where neutron flux and the neutron current are continuous at an interface in the neutron diffusion equation and temperature and heat flux in the thermal conduction equations[100]. If grid points lie on the boundary, the continuity of temperature or flux is automatic and hence not a problem. The continuity of heat flux and neutron current is not quite so simple.

The conservation of heat flux at the boundary is expressed in the equation

$$k_A[\vec{n}\cdot\vec{\nabla}T_A(\vec{r},t)]_{\vec{r}=\text{boundary}} = k_B[\vec{n}\cdot\vec{\nabla}T_B(\vec{r},t)]_{\vec{r}=\text{boundary}}$$

The above expression can be written in a simplified fashion for the one dimensional heat conduction equation,

$$k_A\left(\frac{\partial T}{\partial x}\right)_{iA} = k_B\left(\frac{\partial T}{\partial x}\right)_{iB}$$

It is assumed that grid point T_i lies on the boundary and the point T_{i-1} is in region A and the point T_{i+1} is in region B. T_{i-1} and T_{i+1} are expanded in a Taylor series about the point T_i. These Taylor series are rearranged so the first order derivative is on the left hand side. Using the above equality, the two expansions are multiplied by the appropriate coefficient and equated,

$$\begin{aligned} k_A\left(\frac{\partial T}{\partial x}\right)_{iA} &= k_A\frac{T_i - T_{i-1}}{(\Delta x)_A} + k_A\frac{(\Delta x)_A}{2}\left(\frac{\partial^2 T}{\partial x^2}\right)_{iA} + O[(\Delta x)_A^2] \\ &= k_B\frac{T_{i+1} - T_i}{(\Delta x)_B} - k_B\frac{(\Delta x)_B}{2}\left(\frac{\partial^2 T}{\partial x^2}\right)_{iB} + O[(\Delta x)_B^2] \end{aligned}$$

The conduction equation for this problem has the form

$$\frac{\partial T_i}{\partial t} = \alpha\left(\frac{\partial^2 T_i}{\partial x^2}\right)$$

100 The approach used here is a generalization of the approach used by Carnahan, Luther & Wilkes (1969), p.462.

The above equation is used to substitute the second order derivative, in the expansion, with the rate of change of the temperature. With some rearrangement, the difference equation describing the conservation of heat flux at the boundary will be

$$\frac{\partial T_i}{\partial t} = \left[\frac{k_A(\Delta x)_A}{2\alpha_A} + \frac{k_B(\Delta x)_B}{2\alpha_B} \right]^{-1} \left[\left(\frac{k_A}{(\Delta x)_A} \right) T_{i-1} - \left(\frac{k_A}{(\Delta x)_A} + \frac{k_B}{(\Delta x)_B} \right) T_i + \left(\frac{k_B}{(\Delta x)_B} \right) T_{i+1} \right]$$

where $\alpha_A = k_A/(\rho C_p)$ in the thermal conduction equation, $\alpha = v_g D_g$ and $k = D_g$ in the neutron multigroup diffusion equation. The mesh is not assumed constant across the boundary (i.e. $(\Delta x)_A \neq (\Delta x)_B$).

Marshak Boundary Conditions

The Marshak boundary condition[101] is a Cauchy boundary condition for flux at a free surface boundary. The general boundary condition for diffusion theory is

$$\Phi + b\vec{n} \cdot \vec{\nabla}\Phi = 0$$

where b is some constant.

The Marshak free surface condition, zero incoming current, is obtained when the constant $b = 2D_g$. For the one dimensional case this may be written as

$$\Phi_i + 2D\frac{\partial \Phi_i}{\partial x} = 0$$

If central differencing is used to approximate the derivative, then this equation becomes

$$2D\frac{\partial \Phi_i}{\partial x} = \frac{D}{h}(\Phi_{i+1} - \Phi_{i-1}) + O(h^2)$$

$$= -\Phi_i$$

where $h = \Delta x$. We use this equation to find an expression for Φ_{i+1},

$$\Phi_{i-1} = \Phi_{i+1} + \frac{h}{D}\Phi_i$$

This is substituted into the a central difference approximation for the second derivative,

$$D\frac{\partial^2 \Phi_i}{\partial x^2} = \frac{D}{h^2}(\Phi_{i+1} - 2\Phi_i + \Phi_{i-1})$$

$$= \frac{2D}{h^2}(\Phi_{i+1} - \Phi_i) + \frac{1}{h}\Phi_i$$

This expression is used to approximate the second derivative (Laplacian) at the free surface boundaries in preference to the use of an extrapolated boundary.

---

101 Bell and Gladstone, p.134.

7 Numerical Thermalhydraulics

Special numerical methods are required when solving the thermalhydraulic equations due to the nonlinearity of the equations. The spatial differencing and the calculation of pressure must be handled in a special manner.

7.1 Spatial Differencing

If some straight forward differencing scheme is used, the resulting difference equations would be somewhat messy and the numerical simulation will probably be unstable. Special techniques are required when simulating fluid flow.

The conservation equations that are encountered in thermalhydraulics all have the general form

$$\frac{\partial}{\partial t}\rho\Psi+\vec{\nabla}\cdot\rho\Psi\vec{V}=\mu\nabla^2\Psi+\Gamma$$

where μ is a constant, Γ is a source term and $\Psi(\vec{r},t)$ is equal to $\{\vec{V},h,1\}$ for the conservation of momentum, enthalpy (or internal energy) and mass respectively. These conservation equations, other than mass, can be simplified if the conservation of mass component of the LHS is removed. This may accomplished by expanding the LHS,

$$\Psi\frac{\partial\rho}{\partial t}+\rho\frac{\partial\Psi}{\partial t}+\rho\vec{\nabla}\cdot\Psi\vec{V}+\Psi\vec{\nabla}\cdot\rho\vec{V}=\mu\nabla^2+\Gamma$$

If the conservation of mass equation is used to eliminate terms and the result is divided by the density, the conservation will appear as

$$\frac{\partial\Psi}{\partial t}=-\vec{V}\cdot\Psi\vec{V}+\frac{\mu}{\rho}\nabla^2\Psi+\frac{1}{\rho}\Gamma$$

This may simplified further using the assumption that the fluid is incompressible ($\vec{\nabla}\cdot\vec{V}=0$). The form of the conservation equations that is used in the simulation is

$$\frac{\partial\Psi}{\partial t}=-\vec{V}\cdot\vec{\nabla}\Psi+\frac{\mu}{\rho}\nabla^2\Psi+\frac{1}{\rho}\Gamma$$

Therefore, the two dimensional (X-Y) conservation equation for some variable $\Psi(x,y,t)$ written in long form is

$$\frac{\partial}{\partial t}\Psi(x,y,t)=-V_x\frac{\partial\Psi}{\partial x}-V_y\frac{\partial\Psi}{\partial y}+\frac{\mu}{\rho(x,y)}\left(\frac{\partial^2\Psi}{\partial x^2}+\frac{\partial^2\Psi}{\partial y^2}\right)+\frac{1}{\rho}\Gamma$$

The velocity, $\vec{V}$, will be assumed constant over one time step and updated at the beginning of each time step. This causes the Jacobian to not have coupling terms with the velocity. Five point differencing is used and hence the Jacobian will have five bands. If the velocity is not assumed constant over the time step, then two more bands coupling Ψ with the velocity components, V_x, V_y, will appear in the Jacobian.

Central differencing is used for the second order derivatives and presents little problem. The first order derivative in the convective term is a problem if not handled correctly. *Upwinding*, backward differencing against the direction of flow, is the most stable differencing method for the first order derivative in the convective term. This means that the differencing, with respect to the coordinate system, is dependant on the direction of flow; backward differencing is used if the flow is in the positive direction and forward differencing is used if the flow is in the negative direction.

The discretized variable is $\Psi_{i,j}(t)$ and the five point difference equation will have the form (using symbols from Patankar)

$$\frac{\partial}{\partial t}\Psi_{i,j}(t) = a_P\Psi_{i,j} + a_E\Psi_{i+1,j} + a_W\Psi_{i-1,j} + a_S\Psi_{i,j-1} + a_N\Psi_{i,j+1}$$

The coefficients, a_P, a_N, a_S, a_E, a_W, will be functions of the fluid velocity. The scheme

$$a_E = \frac{\mu}{(\Delta x)^2\rho_{i,j}} + [[-V_{x\ i,j}, 0]]$$

$$a_W = \frac{\mu}{(\Delta x)^2\rho_{i,j}} + [[V_{x\ i,j}, 0]]$$

$$a_N = \frac{\mu}{(\Delta y)^2\rho_{i,j}} + [[-V_{y\ i,j}, 0]]$$

$$a_S = \frac{\mu}{(\Delta y)^2\rho_{i,j}} + [[V_{y\ i,j}, 0]]$$

$$a_P = -a_E - a_W - a_N - a_S$$

calculates the coefficients so as to implement upwinding, where $[[\cdot,\cdot]]$ is a function that selects the largest value like *DMAX1* in FORTRAN. Upwinding is used in the conservation of energy and momentum equations in the thermalhydraulic simulation.

A staggered mesh is used during the calculation of the various quantities. V_x and V_y are evaluated at the center of the vertical (left and right) and the horizontal (top and bottom) surfaces respectively of the pressure control cell. The remaining quantities, density, pressure and temperature (enthalpy) are evaluated at the center of the pressure control cell.

7.2 The Calculation of Pressure

The rate form of the equation of state,

$$\frac{\partial}{\partial t}P(\vec{r},t) = G_1\frac{\partial}{\partial t}\rho(\vec{r},t) + G_2\frac{\partial}{\partial t}h(\vec{r},t)$$

is used to update the pressure in a cell. The conservation of mass equation

$$\frac{\partial\rho}{\partial t} = -\vec{\nabla}\cdot\rho\vec{V}$$

was introduced in Section 4.1.2. Using the above diagram, the difference equation that is used to calculate the rate of change of the density

$$\frac{\partial \rho}{\partial t} = -\frac{1}{(\Delta x)}[V_x(i,j)\rho(i+1/2,j) - V_x(i-1,j)\rho(i-1/2,j)]$$

$$-\frac{1}{(\Delta y)}[V_y(i,j)\rho(i,j+1/2) - V_y(i,j-1)\rho(i,j-1/2)]$$

can be derived.

The difference equation for the rate of change of the enthalpy is basically that of convective heat transfer. These two difference equations, the rate of change of mass and enthalpy, are then substituted into the rate form of the equation of state to arrive at a difference equation for pressure. The fluids that are being dealt with are incompressible (or are very close to it), therefore the coefficient G_1 is much larger than the coefficient G_2 (i.e. the pressure is a weak function of the enthalpy in the region of interest). The rate of change of density is much more important than the rate of change of the enthalpy in the calculation of the pressure. The effects that the coefficient G_1 has on the solution of the thermalhydraulic problem are covered further in Section 10.2, the results section for thermalhydraulic simulations, and in the discussion.

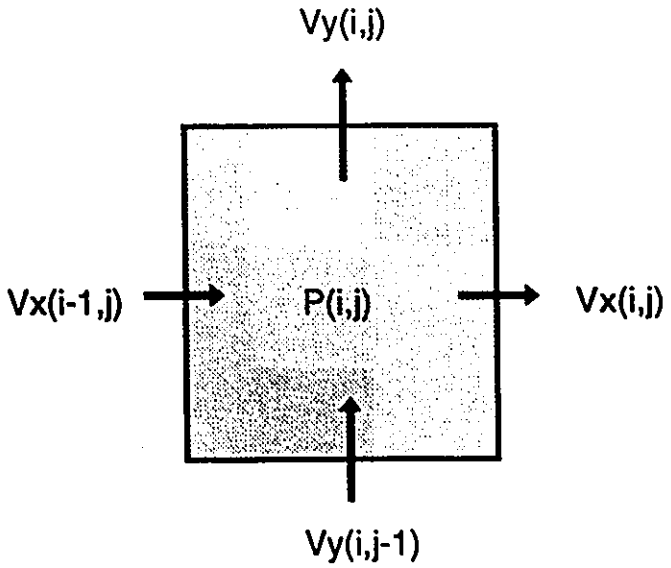

Figure 7.2-1: Pressure Control Cell

7.3 Boundary Conditions

The boundary conditions for fluid flow are the tangential and normal flow at the boundary. In all cases where the boundary is a wall, the tangential and normal flows will be assumed to be zero. If the boundary is not a wall, either the pressure or the flow normal to the boundary must be specified. It is common to specify the inlet flow and the outlet pressure.

The no slip boundary condition is used at all the wall boundaries. Either the temperature or its gradient is specified at each of the walls. The use of the staggered mesh creates some problems of its own at the boundary walls. The temperature is evaluated at the cell center, and all the surrounding temperatures are one grid spacing away, except when that cell borders the cavity. In that case, special differencing techniques are required for the evaluation of some of the derivatives because of the half grid spacing to the wall. The evaluation of derivatives for fluid velocities that are parallel and adjacent to a zero velocity boundary, such as V_x at the top and bottom of the cavity, also require that special differencing techniques be used.

For example, the second order derivative $\partial^2\Psi_i/\partial x^2$ can be approximated using the finite difference equation

$$\frac{\partial^2\Psi_i}{\partial x^2} = \frac{4}{3(\Delta x)^2}(2\Psi_{i-1/2} - 3\Psi_i + \Psi_{i+1}) + O[(\Delta x)]$$

where one point is $\Delta x/2$ away (half grid spacing), and another point is Δx away. Difference equations of this variety were required at the boundary walls.

8 The Complete Simulation

The complete simulation requires the solving of neutronic, thermal and thermalhydraulic equations. These equations will be coupled by fission heat production, temperature and density feedback. This temperature feedback in the fuel will be in the form of cross-section changes and in the form of changes in density (and hence scattering cross sections) in the coolant and moderator.

8.1 The Problem Outline

The problem is set up in three regions. The first region (from the left) is fuel, the second region is coolant and the third region is moderator. The simulation in its present form is in two dimensional Cartesian geometry (X-Y).

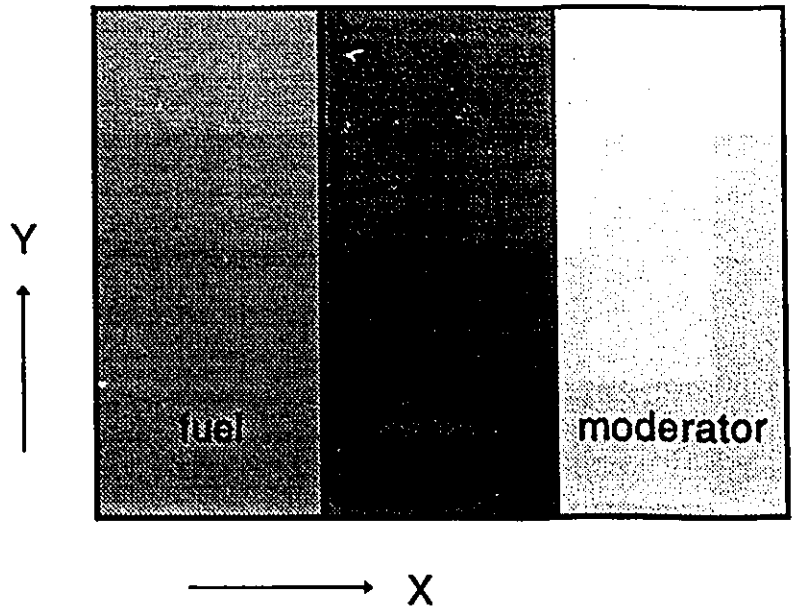

Figure 8.1-1: Problem Geometry

The time dependent multigroup diffusion approximation is used to model the flux behavior in all three regions. Thermal conduction is used to calculate the temperature of the fuel. The parabolic form of the

conservation of momentum, mass, and energy equations is used to calculate the fluid velocity, pressure and temperature (enthalpy) respectively. Density of the coolant is updated explicitly using the thermal expansion coefficient and the temperature.

The boundary conditions on the left and right for the neutronics is reflective while Marshak (free surface) boundary conditions or a water reflector (partial reflection) are used at the top and bottom boundaries. The continuity of neutron flux and current are the boundary conditions used at the two interfaces between the regions.

The left boundary condition, for thermal conduction in the fuel, is reflective as are the top and bottom boundary conditions. The thermal boundary between the coolant and moderator is insulated (reflective boundary) and the boundary between the fuel and the coolant has continuity of temperature and heat flux boundary conditions. The inlet (bottom) temperature for the coolant region is fixed and the outlet temperature is allowed to drift. Fission heat generated in the fuel is transferred by conduction to the coolant, which transports this heat away by natural convection.

The left and right boundary conditions for the fluid in the coolant region are fixed walls while the top and bottom boundaries are open, across which a fixed hydrostatic pressure gradient is maintained. A vertical extension upward may be made to the walls of the coolant region (not shown in figure 8.8-1). This feature was added to allow the flow to "settle" before encountering the constant pressure boundary at the top. These wall extensions are thermally insulated.

8.2 Spatial Differencing and Matrix Structure

The program has two main data vectors and associated matrices. The first data vector contains the information for calculation of the flux: group flux, temperature in the three regions, and delayed neutron precursor density. The second data vector contains the information for the thermalhydraulic simulation: the fluid velocity, pressure and temperature. The coolant density and temperature is the primary area of coupling between the two sections of the code. This information is shared between the two data vectors,

$$\vec{X}_1 = [\vec{\Phi}_1, \vec{\Phi}_2, \ldots \vec{\Phi}_G, \vec{T}, \vec{C}] \qquad \vec{X}_2 = [\vec{V}_x, \vec{V}_y, \vec{P}, \vec{T}_c]$$

where

$$\vec{T}(t) = [\vec{T}_{fuel}, \vec{T}_{coolant}, \vec{T}_{moderator}]$$

is the temperature is the three regions. If $\Psi_{i,j}(t)$ is a variable and $\vec{\Psi}(t)$ is the data vector that contains all the data points of that variable, then the ordering of the data vector will be

$$\vec{\Psi}(t) = \left[\Psi_{1,1}(t), \Psi_{1,2}(t) \ldots \Psi_{1,n_y}(t) \ldots \Psi_{n_x,n_y}\right]$$

8.2.1 The Jacobian Matrices

The Jacobian matrices that arise have a very definite structure. The problem is set up to make maximum usage of this structure. The Jacobian matrix is defined as

$$\overline{\overline{A}} = \frac{\partial\left(\frac{d}{dt}\overline{X}(t)\right)}{\partial\overline{X}(t)}$$

The continuous time solution of the problem is approximated by that of a linear system,

$$\frac{d}{dt}\overline{X}(t) = \overline{\overline{A}}\,\overline{X}(t) + \overline{B}$$

The neutron multigroup diffusion equation is a linear partial differential equations (PDE), hence the Jacobian matrix for this problem would be constant for constant properties. Some adjustments to this matrix are required for eigenvalue (k effective) adjustments and changes in material properties. The Jacobian matrix for the thermalhydraulic simulation is constantly changing (assuming that the simulation is not at a steady state) due to the nonlinear PDEs being solved.

The Laplacian term ($\nabla^2\Psi$) in the various equations will give rise to a block tridiagonal Jacobian matrix (assuming central second order differencing is used). Let us assume that the equation of interest has the form

$$\frac{\partial}{\partial t}\Psi(\vec{r},t) = \alpha\nabla^2\Psi$$

This equation is then spatially discretized using five point differencing,

$$\frac{d\Psi_{i,j}(t)}{dt} = \alpha\left[\frac{(\Psi_{i,j+1}(t) - 2\Psi_{i,j}(t) + \Psi_{i,j-1}(t))}{(\Delta y)^2} + \frac{(\Psi_{i+1,j}(t) - 2\Psi_{i,j}(t) + \Psi_{i-1,j}(t))}{(\Delta x)^2}\right]$$

The data vector is ordered as previously outlined and a vector equation representing the above difference equation is

$$\frac{\partial}{\partial t}\overline{\Psi}(t) = A_\Psi\overline{\Psi}(t) + \overline{B}$$

The self feedback matrix A_Ψ will have a block tridiagonal structure due to the Laplacian term in the equation. It can be expressed as, $A_\Psi = \alpha I_T$ where I_T is the general form of the block tridiagonal matrix of dimension ($n_x n_y \times n_x n_y$).

$$I_T = \begin{bmatrix} [T] & [D] & 0 & \dots & 0 \\ [D] & [T] & [D] & \dots & 0 \\ . & . & . & . & . \\ . & . & . & . & . \\ 0 & \dots & [D] & [T] & [D] \\ 0 & \dots & 0 & [D] & [T] \end{bmatrix}$$

The matrices T and D are tridiagonal and diagonal ($n_x \times n_y$) matrices that are defined as,

$$T = -2[1/(\Delta x)^2 + 1/(\Delta y)^2]I_n + [1/(\Delta y)^2]I'_n$$

$$D = [1/(\Delta x)^2]I_n$$

where I' is a square matrix defined as,

$$I_n' = \begin{bmatrix} 0 & 1 & 0 & \dots & 0 \\ 1 & 0 & 1 & \dots & 0 \\ . & . & . & . & . \\ 0 & \dots & 1 & 0 & 1 \\ 0 & \dots & 0 & 1 & 0 \end{bmatrix}$$

and I is the identity matrix (the subscript n denotes the dimension of the matrix which in this case is equal to n_y). It can be seen that the A_ψ matrix has a five banded structure with the outer two bands quite widely separated from the central three bands depending on the dimension n_y. The matrix I_T will be useful in describing the structure of any Jacobian matrix of a spatially differenced PDE that contains a Laplacian.

8.2.2 Neutron Diffusion and Thermal Conduction

The neutron multigroup equation in discretized form is

$$\frac{1}{v_g}\frac{d}{dt}\Phi^g_{i,j}(t) = D_g\left[\frac{(\Phi^g_{(i,j+1)}(t) - 2\Phi^g_{i,j}(t) + \Phi^g_{(i,j-1)}(t))}{(\Delta y)^2} + \frac{(\Phi^g_{(i+1,j)}(t) - 2\Phi^g_{i,j}(t) + \Phi^g_{(i-1,j)}(t))}{(\Delta x)^2}\right]$$

$$-\Sigma_{tg}\Phi^g_{i,j}(t) + \sum_{g'=1}^{G}\Sigma_{sg'g}\Phi^{g'}_{i,j}(t) + \chi_{pg}(1-\beta)\sum_{g'=1}^{G}\nu_{g'}\Sigma_{fg'}\Phi^{g'}_{i,j}(t) + \sum_{k=1}^{K}\chi_{dg}\lambda_k C^k_{i,j}(t)$$

$$\frac{d}{dt}C^k_{i,j}(t) = \beta_k\sum_{g'=1}^{G}\nu_{g'}\Sigma_{fg'}\Phi^{g'}_{i,j}(t) - \lambda_k C^k_{i,j}(t)$$

where the superscript g denotes the energy group for the cross sections and flux, and the superscript k denotes the neutron precursor, $C_k(\vec{r},t) \equiv C^k_{i,j}(t)$. The neutronics equations can be arranged into a matrix equation

$$\frac{d}{dt}\vec{\Phi}_g(t) = \sum_{g'=1}^{G}A_{\Phi(g,g')}\vec{\Phi}_{g'}(t) + \sum_{k=1}^{K}A_{\Phi,C(g,k)}\vec{C}_k(t) + \vec{B}_\Phi$$

$$\frac{d}{dt}\vec{C}_k(t) = \sum_{g'=1}^{G}A_{C\Phi(k,g')}\vec{\Phi}_{g'}(t) - \lambda_k\vec{C}_k(t)$$

by ordering the data into a vector as previously outlined. In practice, up to fourteen delayed groups (photo neutrons included) have been used, but (the more common value of) six delayed groups are used here (K = 6).

The matrices used in the neutronics matrix equations can now be defined as

$$A_{gg'} = v_g(\Sigma_{sg'g} + \chi_{pg}(1-\beta)\nu_{g'}\Sigma_{fg'})I_n + v_g[\Sigma_{tg}I_n + D_g I_T]\delta_{gg'} \qquad (n = n_x n_y)$$

$$\delta_{gg'} = 1 \quad for \quad g = g'$$

$$= 0 \quad for \quad g \neq g'$$

$$A_{\Phi C(g,k)} = \nu_g \chi_{dg} \lambda_k I_n \qquad A_{C\Phi(k,g')} = \beta_k \nu_{g'} \Sigma_{fg'} I_n$$

These group flux vectors may be brought into one large vector that contains all the group flux vectors. This will be called flux vector. Similarly the delayed neutron precursors can be brought into one large vector,

$$\vec{\Phi}(t) = \left[\vec{\Phi}_1^T(t), \vec{\Phi}_2^T(t), \ldots, \vec{\Phi}_G^T(t)\right]^T$$

$$\vec{C}(t) = \left[\vec{C}_1^T(t), \vec{C}_2^T(t), \ldots, \vec{C}_K^T(t)\right]^T$$

The matrices $A_{\Phi\Phi}$ and $A_{C\Phi}$ can be written as block matrices,

$$A_{\Phi\Phi} = \begin{bmatrix} A_{\Phi(1,1)} & \cdots & A_{\Phi(1,G)} \\ . & \cdots & . \\ A_{\Phi(G,1)} & \cdots & A_{\Phi(G,G)} \end{bmatrix} \qquad A_{\Phi C} = \begin{bmatrix} A_{\Phi C(1,1)} & \cdots & A_{\Phi C(1,K)} \\ . & \cdots & . \\ A_{\Phi C(G,1)} & \cdots & A_{\Phi C(G,K)} \end{bmatrix}$$

The thermal conductive equation for a homogeneous medium with a fission heat source term

$$\frac{\partial T(\vec{r},t)}{\partial t} = \frac{1}{\rho C_p}\left[\omega_f \sum_{g'=1}^{G} \Sigma_{fg'}(\vec{r})\Phi_{g'}(\vec{r},t) + k\nabla^2 T(\vec{r},t)\right]$$

is used in this problem. The difference equation is the conductive equation in Section 6.3, with addition of the fission source term,

$$\frac{dT_{i,j}(t)}{dt} = \frac{k}{\rho C_p}\left[\frac{(T_{i,j+1}(t) - 2T_{i,j}(t) + T_{i,j-1}(t))}{(\Delta y)^2} + \frac{(T_{i+1,j}(t) - 2T_{i,j}(t) + T_{i-1,j}(t))}{(\Delta x)^2}\right]$$

$$+ \frac{\omega_f}{\rho C_p} \sum_{g'=1}^{G} \Sigma_{fg'} \Phi_j^{g'}(t)$$

As with the neutronics equations, the temperature equation can be placed into a matrix form by ordering the data points into a vector,

$$\frac{d}{dt}\vec{T}(t) = A_{TT}\vec{T}(t) + A_{T\Phi}\vec{\Phi}(t)$$

where

$$A_{TT} = \frac{k}{\rho C_p} I_T \qquad A_{T\Phi(g')} = \frac{\omega_f}{\rho C_p} \Sigma_{fg'} I_n$$

$$A_{T\Phi} = [A_{T\Phi(1)}, A_{T\Phi(2)}, \ldots, A_{T\Phi(G)}]$$

For the neutronic and thermal conduction simulation, the matrix equation has the form

$$\frac{d}{dt}X_1(t)=\frac{d}{dt}\begin{bmatrix}\vec{\Phi}(t)\\ \vec{T}(t)\\ \vec{C}(t)\end{bmatrix}=\begin{bmatrix}A_{\Phi\Phi} & A_{\Phi T} & A_{\Phi C}\\ A_{T\Phi} & A_{TT} & 0\\ A_{C\Phi} & 0 & A_{CC}\end{bmatrix}\begin{bmatrix}\vec{\Phi}(t)\\ \vec{T}(t)\\ \vec{C}(t)\end{bmatrix}+\begin{bmatrix}\vec{B}_\Phi\\ \vec{B}_T\\ 0\end{bmatrix}$$

where $\vec{B}$ is a vector that contains boundary condition information (if required). The block diagonal matrices for the flux, $A_{\Phi(i,j)}$, and the self feedback matrix for the temperature, $A_{T,T}$, are block tridiagonal matrices due to the two dimensional Laplacian term in their rate equations. The off-diagonal block matrices for the flux, $A_{\Phi(i,j)}$ $(i \neq j)$, will be diagonal matrices that contain the group to group scattering information. Block matrices comprise the cross term matrices, such as $A_{T\Phi}$, and are all diagonal matrices. The self feedback matrix for the delayed neutron precursors, A_{CC}, will also be a diagonal matrix that contains the decay constant information.

8.2.3 Thermalhydraulic Equations

The data points for the thermalhydraulic simulation are arranged using the ordering scheme given in Section 8.2 but, the width in the X direction is that of the coolant region (i.e. $n_x = [n_x]_{coolant}$). The five point differencing scheme

$$\frac{\partial}{\partial t}\Psi_{i,j}(t)=a_P\Psi_{i,j}+a_E\Psi_{i+1,j}+a_W\Psi_{i-1,j}+a_S\Psi_{i,j-1}+a_N\Psi_{i,j+1}$$

used for the thermalhydraulic variables of velocity (X and Y directions) and the temperature (or enthalpy) is given in Section 7.1.

The data vectors $\{\vec{V}_x,\vec{V}_y,\vec{P},\vec{T}\}$ comprise the main time dependent data vector, $X_2(t)$, and contain all the spatial points for $\{V_x(i,j),V_y(i,j),P(i,j),T(i,j)\}$ respectively. The density is not included in this time dependent data vector as the density is not explicitly a function of time. Density as a function of temperature is calculated using the equation for thermal expansion[102],

$$\rho_{i,j}=\bar{\rho}-\bar{\rho}\bar{\beta}(T_{i,j}-\bar{T})$$

where $\bar{\rho}$ is the density at some reference temperature, $\bar{T}$, and $\bar{\beta}$ is the coefficient of volumetric expansion at that temperature.

102 Bird, Stewart & Lightfoot, (1960) p 299

For the thermalhydraulic simulation, the matrix equation has the form

$$\frac{d}{dt}X_2(t) = \frac{d}{dt}\begin{bmatrix} \vec{V}_x(t) \\ \vec{V}_y(t) \\ \vec{P}(t) \\ \vec{T}(t) \end{bmatrix} = \begin{bmatrix} A_{xx} & 0 & A_{xP} & 0 \\ 0 & A_{yy} & A_{yP} & 0 \\ A_{Px} & A_{Py} & 0 & A_{PT} \\ 0 & 0 & 0 & A_{TT} \end{bmatrix} \begin{bmatrix} \vec{V}_x(t) \\ \vec{V}_y(t) \\ \vec{P}(t) \\ \vec{T}(t) \end{bmatrix} + \begin{bmatrix} \vec{B}_x \\ \vec{B}_y \\ \vec{B}_P \\ \vec{B}_T \end{bmatrix}$$

The B vectors contain some boundary condition information. The block matrices along the diagonal for velocity and temperature A_{xx}, A_{yy}, A_{TT}, will have a block tridiagonal structure due to the five point differencing that is used, but their elements can not be as easily defined as in Section 8.2.1 because of the nonlinearity of their original PDEs. The elements of these matrices are velocity dependent as discussed in Section 7.1.

The four coupling matrices between the pressure and the velocities all have two bands of varying band width. The matrix A_{Py} has a band on the diagonal and one on the superdiagonal, A_{yP} has a band on the diagonal and one on the subdiagonal, A_{Px} has a band on the diagonal and one n_y above the diagonal, and A_{xP} has a band on the diagonal and one n_y below the diagonal.

The coupling matrix between the temperature and the pressure is equal to A_{TT} multiplied by G_2, $A_{PT} = G_2 A_{TT}$, and hence, will have five bands. The matrix A_{PT} will be zero if G_2 is zero, which is the case here. That effectively decouples the velocity/pressure calculation from the temperature calculation. This allows the pressure/velocity calculation to be solved using a partitioning scheme, for greater numerical efficiency. Either a block tridiagonal solver or a Gauss-Seidel Solver (using successive over relaxation) is used. It was found that an implicit solution in time had greater numerical stability.

9 Programming and Coding

The simulation is written using the C programming language for the UNIX operating system. The program in its present form will run on any machine running Unix/Xenix but can be easily ported to any machine with a C compiler. Some of the mathematical subroutines are available in assembly language for the Intel 386/387 and it is advantageous to use the assembly language routines (if you are using a 386/387 or 486 processor) for increased speed of execution. C is programming language of choice due to its flexibility and its suitability for use in a Unix environment.

An understanding of C is necessary in order to follow in detail the listings of the various routines. The main routine is the most complicated and a general explanation will be given for its sections. Explanations will not be given for the other routines due to their relative simplicity.

9.1 The Main Routine

The main routine sets up the problem, handles the data input and output, memory allocation and generally controls the solution of the problem. The filling of the arrays, with the exception of the pointer arrays, and the solution of the problem are all handled by subroutines. The main iteration loop calls several subroutines that handle filling the matrices for the linear system problem (Jacobian matrix), making any necessary adjustments before using the solver routine, solving the linear system problem and making updates to the accounting variables. The main routine controls the simulation but the actual work is carried out in the subroutines.

The following is a general description of the main routine. It explains, section by section, the operation of the code.

1. Include the necessary files. The include file *react.h* contains declarations for some of the custom routines in *react.c*. The include file *ma* *.h* contains the declarations for the custom mathematical routines.
2. Declare all the external (global) variables. This includes the array pointers, the arrays containing the delayed neutron data, and some format character strings.
3. Begin the main routine.
    1. Declare all the local variables. The routine for handling some interrupts is defined as *sig_handle*.
    2. Read the problem data file.
    3. Check the problem control input and set the condition of various parameters.
    4. Calculate the various constants for the finite difference mesh and calculations.

5 Request memory allocation for the arrays (data vectors, matrices and pointer arrays), and check that memory was allocated.

6 Fill the pointer arrays and subdivide the data vectors into their components.

7 Initialize the band arrays, the integer arrays that indicate matrix band position.

8 Allocate memory for the banded matrices and the block tridiagonal matrices.

9 Open and read the cross sectional data file (binary).

10 Initialize the data arrays using either a "cold start" condition or the restart binary file.

11 Open the plot file and the status file and write initial conditions to the plot file.

12 The routine *Flux* initializes the Jacobian matrix for the neutron diffusion problem. The routine *Thermal* initializes the Jacobian matrix for the thermal conduction problem.

13 The thermal feedback coefficients are calculated.

14 The main loop begins.

1 The routine *Fluid* calculates the coefficients of the Jacobian matrix for the thermalhydraulic problem.

2 Step size control for the neutronics problem, both static and transient, is implemented. The routines *RKSI_GS* and *RKSI_LUb* are used to solve the static neutronics problem and *RKSI_GS_N* and *RKSI_LU_N* are use to solve the transient problem. All these routines use the semi-implicit Runge-Kutta method of time integration.

3 Adjustments are made to the thermal feedback coefficients.

4 *K-effective* is calculated for static simulations using Chebyshev acceleration. The routine *AdjustK* adjusts the coefficients of the neutronics Jacobian due to the change in *K-effective*.

5 Step size control for the thermalhydraulic problem is implemented and an implicit Euler method is used to solve the time integration. The linear system problem is solved using matrix partitioning and block tridiagonal matrix inversion.

6 The routine *Density* is used to calculate the coolant density. The mean coolant density is calculated and is used in the routine *XSCoolant* to calculated new cross sections in the coolant and update the neutronics Jacobian.

7 Time is advanced and the plot file is updated as required. The results of this iteration are written either to the monitor or to the status file. The time step size is adjusted if required.

8 This is the end of the main loop. The decision is made whether to end the simulation or iterate again. If an error condition has occurred, the simulation may terminate before this point.

15 The conservation of mass error for the thermalhydraulic simulation is calculated using the routine *Mass*. The mean and the variance of the mass error is calculated.

16 The problem statistics: CPU time, total simulation time, mean flux and *K-effective* are displayed.

17 The state of the simulation is saved in a binary file if requested (restart file).

18 The problem results for both the neutronic simulation and the thermalhydraulic simulation is printed out in ASCII files. Flux maps, temperature maps and flow maps are printed if requested.

19 This is the end of main routine.

4 Several ancillary routines are defined.

5 End of file.

9.2 Jacobian Matrix Routines

There are several special routines mentioned above that calculate and store the coefficients of the two main Jacobian matrices. The two main routines are *Flux* and *Fluid* which calculate and store the coefficients for the neutron diffusion problem and the thermalhydraulics problem respectively. The routine *Thermal* calculates and stores the coefficients for the heat transfer problem. The neutron diffusion and heat transfer problems store coefficients in the first Jacobian, called the neutronics Jacobian, and the hydraulics problem (fluid velocity and pressure) store coefficients in the second Jacobian, called the hydraulics Jacobian.

The routines *AdjustK* and *XSCoolant* make adjustments to the neutronics Jacobian due to changes in the *K-effective* and the cross sections in the coolant respectively. The routine *Fluid* calculates the coefficients for convective heat transfer in the coolant which are stored in the neutronics Jacobian for use in solving the heat transfer problem. *Fluid* is the only routine which calculates coefficients for the hydraulics problem.

All these routines both calculate and store the coefficients for the two Jacobians. Some further manipulation of the Jacobians is usually required before the time integration can be solved.

9.3 Mathematical Routines

The most important mathematical routines are the time integration routines. Semi-implicit Runge-Kutta integration is used for the neutronics problem. This algorithm requires matrix inversion and two methods of matrix inversion, Gauss-Seidel iterative method and LU decomposition are used. Separate versions of the algorithm were required for the static and transient simulations, therefore four different versions of the algorithm are available.

There are many vector and matrix routines used, most are quite simple. Some special matrix handling routines are available for dealing with the banded matrices.

Matrix Inversion Routines

There are three types of linear system solver routines that are used: the Gauss-Seidel iterative, the LU Decomposition method (Gaussian elimination), and the Generalized Conjugate Gradient method. All of these types of routines can be used for the solution of banded and full storage problems. Each of these methods has its attributes and liabilities. A more detailed discussion of these methods may be found in chapter 5 - Numerical Methods.

The matrix to be inverted is not modified when the Gauss-Seidel method is used. The zero bands of a banded matrix can be ignored allowing the storage of only the nonzero bands. The banded Gauss-Seidel solver routine that is used here requires only the nonzero bands of the matrix. An integer array is used to tell the subroutine where these bands lie with respect to the central band.

Standard LU decomposition routines are very common and hence there is little need to discuss them here. Block tridiagonal matrix inversion is an LU decomposition type algorithm which is useful for the matrices that typically arise with the finite differencing of PDEs.

All these algorithms are discussed in more detail in the Numerical Methods chapter.

10 Simulation Results

The overall simulation has two distinct components, the neutronic simulation and the thermalhydraulic simulation. The thermalhydraulic simulation is a new method therefore it is extensively tested. The neutronic simulation by itself would not be new but the combination of the neutronics and thermalhydraulics with such direct coupling is new. Much effort has been placed on writing efficient code, and pivotal to efficient code are solution methods for the linear algebraic problem as this is where the majority of CPU time is spent. A comparison of the solver methods entertained is given in the next section.

CPU Time Reporting

CPU time is allocated to a process in *time-slice* increments when using the UNIX operating system. The reporting of the CPU time used by a process is given as the number of *time-slices* used, the size of which is dependent on the processor. The machines used are: an IBM Model 80-111 running Xenix/386 which has a time slice of 0.02 seconds and an IBM RS6000 running AIX which has a time slice of 0.01 seconds. All of the comparison runs for the matrix inversion problems were run on the IBM Model 80 therefore, the CPU times quoted should be accurate to within 0.01 seconds. However, occasionally the operating system must use the CPU to perform house keeping duties and the associated swapping of processes will add to the CPU time used by a process. The simulations for the thermalhydraulic benchmarking and all of the combined neutronics - thermalhydraulics simulations were run on the IBM RS6000. Not all machines are created equal and the CPU time for a given simulation will depend heavily on the machine being used.

10.1 Linear Algebraic Problem - Matrix Inversion

Different methods of matrix inversion[103] are compared on the basis of execution time for different matrices. The intention is to show which methods are best suited to which type of matrix. The problem to be solved has the form

$$\overline{AX} = \overline{B}$$

where A and B are known. There are three classes of methods of solving the linear system problem: direct methods, iterative methods, and semi-iterative methods. LU decomposition (LU), banded LU decomposition (LUb), tridiagonal LU decomposition (triLU), and block tridiagonal LU decomposition (bltriLU) are the direct methods that are tested. The Gauss-Seidel (GS) iteration, and the successive over relaxation method (SOR) are the iterative methods tested. The conjugate gradient (CG) method, the boosted conjugate gradient method (BCG), and the generalized conjugate gradient method (GCG) are the semi-iterative methods tested. All the iterative and semi-iterative methods can be implemented using the banded storage method presented here.

---

103 See section 5.5 for a description of each method.

Direct versus Iterative methods

A common matrix type found in the simulation is a block tridiagonal matrix (or five banded) where the block diagonal matrix is tridiagonal and the block superdiagonal and subdiagonal matrices are diagonal matrices (see section 7.2.1). Five coefficients can be used to represent the test matrix, one for each band in sequence. Coefficient a_1 is element in the first band (the diagonal in the block subdiagonal matrix), a_2 is for the second band, et cetera. In order for the matrix to diagonally dominant, the element in the central band, a_3, must be larger in magnitude than the sum of the magnitudes of all the other elements (see appendix E).

The methods compared are Gauss-Seidel (GS) for banded and full storage, Gauss-Seidel with Successive Over Relaxation (GSSOR) for banded and full storage, LU decomposition, banded LU decomposition[104], and block tridiagonal LU decomposition. The first dimension indicates block size (square), and the second dimension indicates the number of blocks along the block diagonal. The block size will determine the bandwidth of the matrix, and the overall dimension of the system is the product of the block size and the number of blocks. Both C and assembly language versions of the solvers were used.

The results given in Table 10.1-3b are shown in Figure 10.1-1.

Problem #1: Coefficients: 1.0, 1.0, -4.5, 1.0, 1.0

Dimensions:	10, 10		10, 20		20, 10	
Method	C	Asm	C	Asm	C	Asm
GS	15.52	7.20	69.89	31.82	69.99	31.96
GSSOR	6.27	4.43	28.62	12.86	28.89	12.93
GS - banded	1.10	0.65	2.52	1.45	2.50	1.45
GSSOR - banded	0.43	0.42	na[105]	0.62	na	0.62
LU	1.07	0.68	4.40	2.80	7.03	4.73
LU - banded	0.19	0.13	0.40	0.28	1.24	0.93
Block Tridiag	0.27	0.17	0.58	0.39	1.66	1.15

Table 10.1-1: Matrix Inversion, Problem 1

104 The method is a banded method but full storage is actually used (see section 5.5).

105 The method did not converge.

Problem #2: Coefficients: 1.1, 1.0, -5.0, 1.0, 0.8

Dimensions:	10, 20		20, 10		15, 20		20, 15	
Method	C	Asm	C	Asm	C	Asm	C	Asm
GS	35.96	16.75	36.90	17.09	84.39	38.96	85.28	39.50
GSSOR	20.88	9.64	21.89	10.07	50.52	23.07	52.79	23.65
GS band	1.34	0.78	1.36	0.80	2.11	1.24	2.12	1.25
GSSOR band	0.76	0.48	0.82	0.50	1.33	0.78	1.34	0.78
LU	4.41	2.82	6.96	4.71	13.20	8.70	16.12	10.91
LU band	0.39	0.28	1.22	0.91	1.17	0.87	1.95	1.46
Block Tri	0.58	0.38	1.66	1.16	1.67	1.15	2.64	1.84

Table 10.1-2: Matrix Inversion, Problem 2

Problem #3: Coefficients: 1.0, 1.0, -6.0, 1.0, 1.0

Dimensions:	10, 20		20, 10	
Method	C	Assembly	C	Assembly
GS	25.68	11.90	25.81	12.01
GSSOR	18.48	8.51	18.56	8.58
GS - banded	0.93	0.54	0.93	0.54
GSSOR - banded	0.76	0.42	0.75	0.41
LU	4.37	2.80	6.98	4.76
LU - banded	0.38	0.28	1.22	0.92
Block Tridiagonal	0.58	0.38	1.67	1.16

Table 10.1-3a: Matrix Inversion, Problem 3a

Dimensions:	10, 10	12, 12	15, 15	18, 18	20, 20	22, 22
Method	Assembly	Assembly	Assembly	Assembly	Assembly	Assembly
GS	2.82	6.07	15.45	32.49	49.44	75.33
GSSOR	1.90	4.42	10.79	23.51	35.76	53.93
GS - banded	0.25	0.38	0.63	0.91	1.14	1.41
GSSOR - banded	0.18	0.28	0.46	0.71	0.88	1.05
LU	0.67	1.63	4.84	11.74	19.68	31.79
LU - banded	0.13	0.26	0.63	1.29	1.92	2.84
Block Tridiag	0.18	0.36	0.84	1.67	2.54	3.67

Table 10.1-3b: Matrix Inversion, Problem 3b

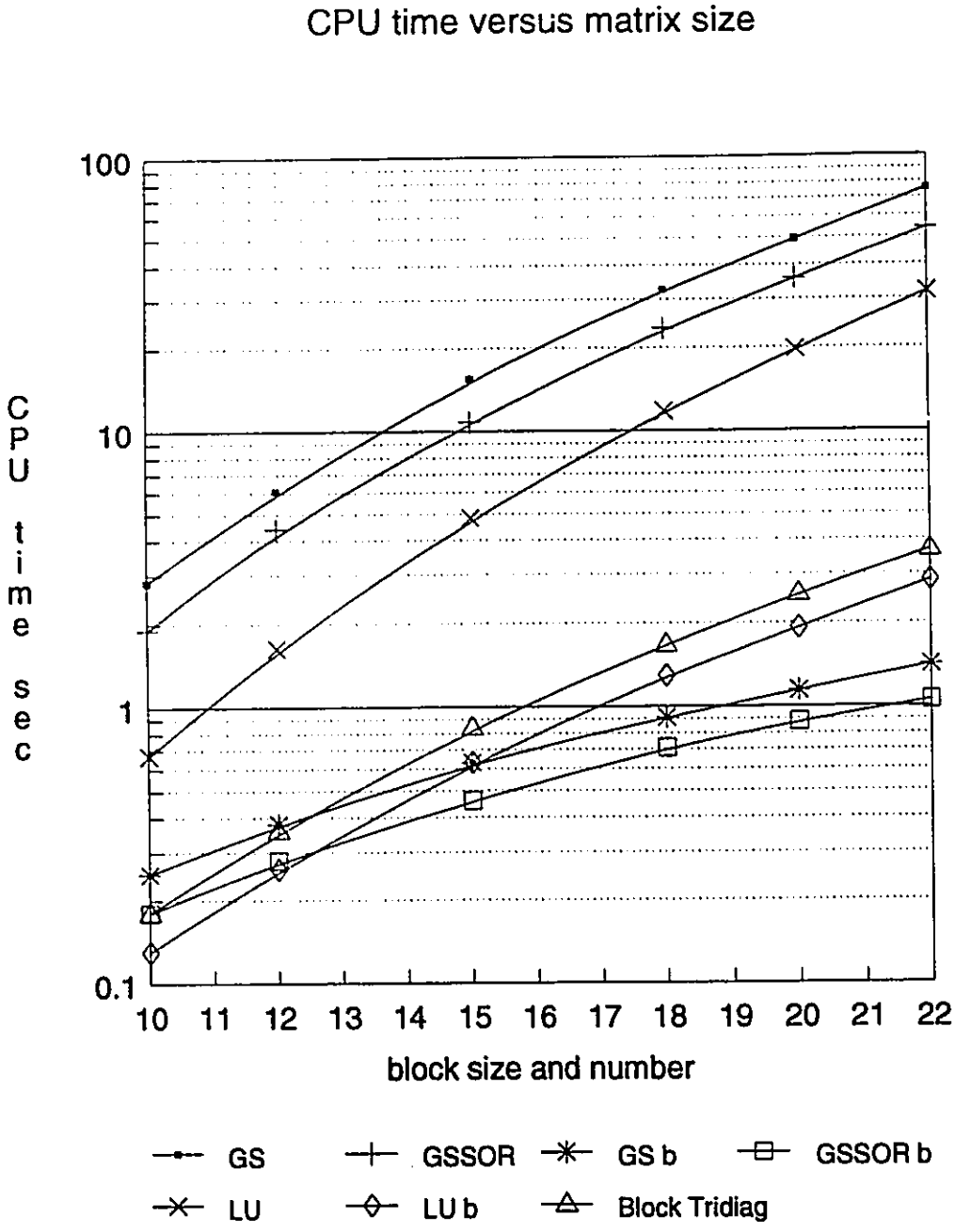

Figure 10.1-1: Matrix Inversion Methods

Coding: C Language versus Assembly Language

A comparison was made between the C language and the assembly language versions of the solver routines. From the above tables, it can be seen that the routines benefit by different amounts in the conversion to assembly language. A mean ratio of the CPU time for the C language version over CPU time for the assembly language version was calculated for the solver routines.

Method	t_c/t_{asm}
GS	2.17
GSSOR	2.19
GS - banded	1.72
GSSOR - banded	1.72
LU	1.50
LU - banded	1.35
Block Tridiagonal	1.44

Table 10.1-4: Matrix inversion, C versus Assembly

Conjugate Gradient Methods

There are several variations of the conjugate gradient method. The original conjugate gradient method (CG method) is designed for use on symmetric positive definite matrices. The generalized conjugate gradient method (GCG method) can be used for any nonsingular matrix, but the convergence properties of the method are not as good as the CG method. The boosted conjugate gradient[106] method (BCG method) is a modified version of the CG method that is suppose to be useful for a larger class of matrices. These three methods are compared for a number of test problems.

The test matrix was a five banded matrix

$$A = \begin{bmatrix} a_2 & a_3 & 0 & a_4 & . & . & . & . & 0 \\ a_1 & a_2 & a_3 & 0 & a_4 & . & . & . & 0 \\ 0 & a_1 & a_2 & a_3 & 0 & a_4 & . & . & 0 \\ a_0 & 0 & a_1 & a_2 & a_3 & 0 & a_4 & . & 0 \\ 0 & a_0 & 0 & a_1 & a_2 & a_3 & 0 & a_4 & 0 \\ 0 & . & a_0 & 0 & a_1 & a_2 & a_3 & 0 & a_4 \\ 0 & . & . & a_0 & 0 & a_1 & a_2 & a_3 & 0 \\ 0 & . & . & . & a_0 & 0 & a_1 & a_2 & a_3 \\ 0 & . & . & . & . & a_0 & 0 & a_1 & a_2 \end{bmatrix}$$

106 S.R. Vatsya (1989)

with a diagonal band, a band one above and below the diagonal, and a band three above and below the diagonal. The system dimension was variable and the coefficients, $\{a_i\}$ $(i=0,4)$, were varied so that the matrix could be symmetric or asymmetric, positive definite or not positive definite. The coefficient m denotes the number of memory vectors that are used. The more memory vectors that are used, the less is the problem of loss of orthogonality. The residue is defined as $r = b - Ax$. The magnitude of the residue vector of a solution is given as a measure of the convergence properties of the method, the smaller the residue vector, the better the solution. The magnitude of the residue vector for LU decomposition is given for comparison. The number n denotes the system dimension and the number of iterations of the algorithm.

Case 1: Symmetric matrix, n=8, $a_2 = -4$, $a_i = 1.0$ (i=0,1,3,4);
m=1,2,3

LU	CG	BCG	GCG
1.06×10^{-15}	1.19×10^{-21}	8.49×10^{-7}	7.65×10^{-17}
"	5.18×10^{-20}	1.48×10^{-12}	9.21×10^{-19}
"	2.23×10^{-32}	7.37×10^{-34}	1.33×10^{-30}

Table 10.1-5: Conjugate Gradient Methods 1

Case 2: Asymmetric matrix, n=8, $a_2 = -3.2$, $a_3 = 1.2$, $a_i = 1.0$ (i=0,1,4);
m=2,3,4,5,6,7

LU	CG	BCG	GCG
7.03e-15	0.0135	0.287	1.58e-14
"	0.0049	0.0352	1.46e-13
"	0.00122	0.0007	3.34e-14
"	0.00039	0.0238	1.46e-14
"	2.67e-6	0.0157	1.62e-14
"	7.92e-15	3.89e-15	1.44e-14

Table 10.1-6: Conjugate Gradient Methods 2

Case 3: Asymmetric matrix, n=20, $a_2 = -3.2$, $a_3 = 1.2$, $a_i = 1.0$ (i=0,1,4);
m=1,3,5,7

LU	CG	BCG	GCG
1.99e-15	0.0364	0.351	2.38e-5
"	0.0162	0.0971	1.51e-5
"	0.00605	0.0597	4.18e-6
"	0.00037	0.0112	5.04e-6

Table 10.1-7: Conjugate Gradient Methods 3

Case 4: Asymmetric matrix, n=20, a_2 = -3.2, a_1 = 2.0, a_i = 1.0 (i=0,1,4);
m=3,5,7

LU	CG	BCG	GCG
7.76e-15	1.41	1.316	8.2e-4
"	1.395	1.311	8.0e-4
"	1.287	1.284	1.54e-5

Table 10.1-8: Conjugate Gradient Methods 4

Comparison of The Banded Solvers

The three primary types of methods, direct, iterative and semi-iterative, can be written a form that exploits the sparsity of a matrix. The semi-iterative methods, conjugate gradient and generalized conjugate gradient, can be written in a form that uses the same banded storage scheme as the Gauss-Seidel solvers. The two banded direct methods, banded LU decomposition and block tridiagonal LU decomposition, use a storage scheme that is unique to them. These methods were compared for increasing matrix sizes using the first matrix (problem #1) that was used to compare the direct and iterative methods. The iterative and semi-iterative methods were run until the error in each element was within 10^{15}. This error is the same order of magnitude as the direct methods. These results are shown in Figure 10.1-2.

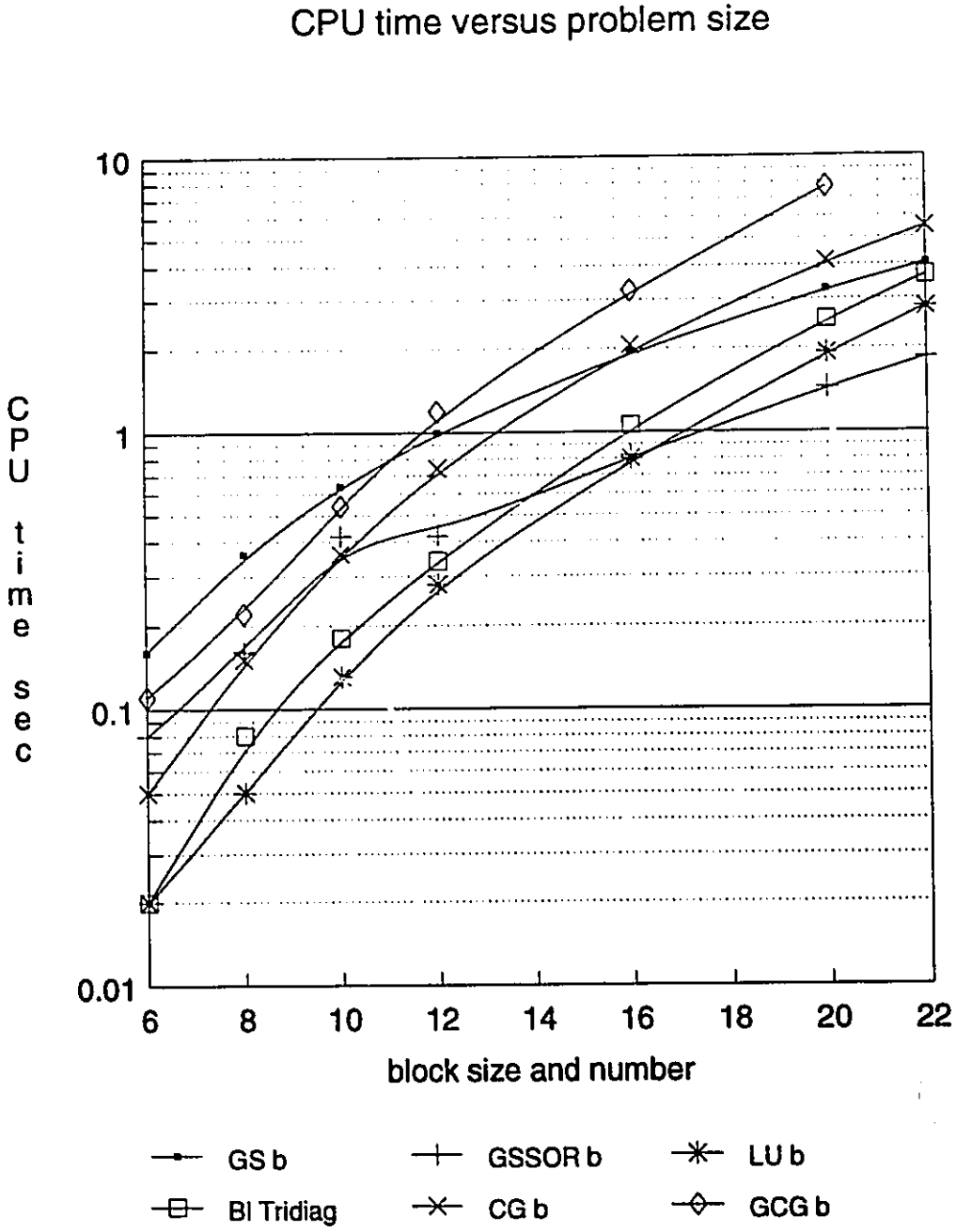

Figure 10.1-2: Banded Matrix Inversion Methods

10.2 Benchmarking the Thermalhydraulic Simulation

The equations used for benchmarking had three adjustable parameters, gravity, G_l and Ra (Section 4.1.3 gives a detail explanation of the benchmark equations). The range of Rayleigh numbers used is 10^3 to 10^6, and the Prandtl number was taken to be fixed at $Pr = 0.71$, as suggested by De Vahl Davis and Jones. The thermal expansion coefficient β was calculated using the given Rayleigh number and gravity. The true value of G_l for water at standard temperature and pressure is approximately $2x10^6$ (m^2 / s^2). The value of G_l used in the benchmark simulation will depend on the units used for pressure and density. The pressure unit used here is kPa, density is in g/cm^3 and hence the true value of G_l for the normalized pressure is $2x10^6$ (see Section for the definition of the normalized pressure). The effect of varying G_l from well below to well above its true value is reflected in the range of G_l used in the numerical simulations, 10^2 to 10^{12}.

Progressively finer mesh sizes were chosen for each Rayleigh number until the computation became too expensive or memory limits were reached. Spatial truncation error will be reduced as mesh spacing is reduced. De Vahl Davis explored the question of solution accuracy as a function of mesh size in the benchmark solution. He extrapolated the results of progressively finer mesh sizes to the theoretically correct result that would be obtained using an infinitely fine mesh. He concluded that the results for the course mesh sizes ($\Delta x > 0.05$) were unreliable for high Rayleigh numbers ($Ra > 10^5$).

The mass error is the sum of all the mass entering less all the mass leaving a control cell. Ideally, in a steady state situation, the mass entering and leaving a control cell will be equal. The variance of the mass error is the criterion that is used to determine the *goodness* of the solution. The variance will give an indication of the imbalance that exists between the individual control cells. The mass error can be calculated from the conservation of mass equation,

$$e_{i,j} = \frac{1}{(\Delta x)}[V_x(i,j)\rho(i+1/2,j) - V_x(i-1,j)\rho(i-1/2,j)]$$

$$+ \frac{1}{(\Delta y)}[V_y(i,j)\rho(i,j+1/2) - V_y(i,j-1)\rho(i,j-1/2)]$$

where $e_{i,j} \equiv$ mass error. The mass error variance is the computed variance.

$$\sigma^2 = \frac{1}{n_x n_y} \sum_{i=1}^{n_x} \sum_{j=1}^{n_y} (e_{i,j} - \bar{e})^2$$

where $\bar{e}$ is the mean of the mass error. The value of G_l will have a great effect on the mass error variance.

10.2.1 Transient Heat Flow

Conductive and convective heat transfer are the two mechanisms by which thermal energy moves from the hot wall to the cold wall. It can be shown (see Appendix D) that the total amount of thermal energy moving through a plane that is parallel to the hot and cold walls is given by

$$Q(x) = \int_0^1 \left(TV_x - \alpha \frac{\partial T}{\partial x} \right) dy$$

This expression is evaluated at several x locations to chart the flow of thermal energy through the fluid. Thermal energy will be stored within the fluid, which causes a time delay as the fluid temperature changes. Initially, all the heat transfer is from the hot wall to the fluid. Eventually the fluid begins to transfer this heat to the cold wall. Steady state is reached when the heat transfer at the two walls balances (and all intervening planes that are parallel to the walls).

The Nusselt number

$$\mathrm{Nu} = \int_0^1 \frac{\partial T}{\partial x} dy$$

as used by De Vahl Davis & Jones in their comparison exercise, gives the heat transfer through a plane parallel to the hot and cold walls by thermal conduction only. This expression is true at the hot and cold walls where the velocity in the x direction is zero. The more complete expression is $Q(x)$, which De Vahl Davis refers to as Nu_x in the benchmark solution.

The total heat flux in the x direction is charted as a function of position at times equal to *0.05s, 0.1s, 0.15s, 0.2s, 0.5s, & 1.0s* for a *20x20* cell array with G_i equal to 10^6 (approximately the compressibility of water). This is shown in Figure 10.2-1. The planes are located at the center of the pressure cells with the cell number indicating position.

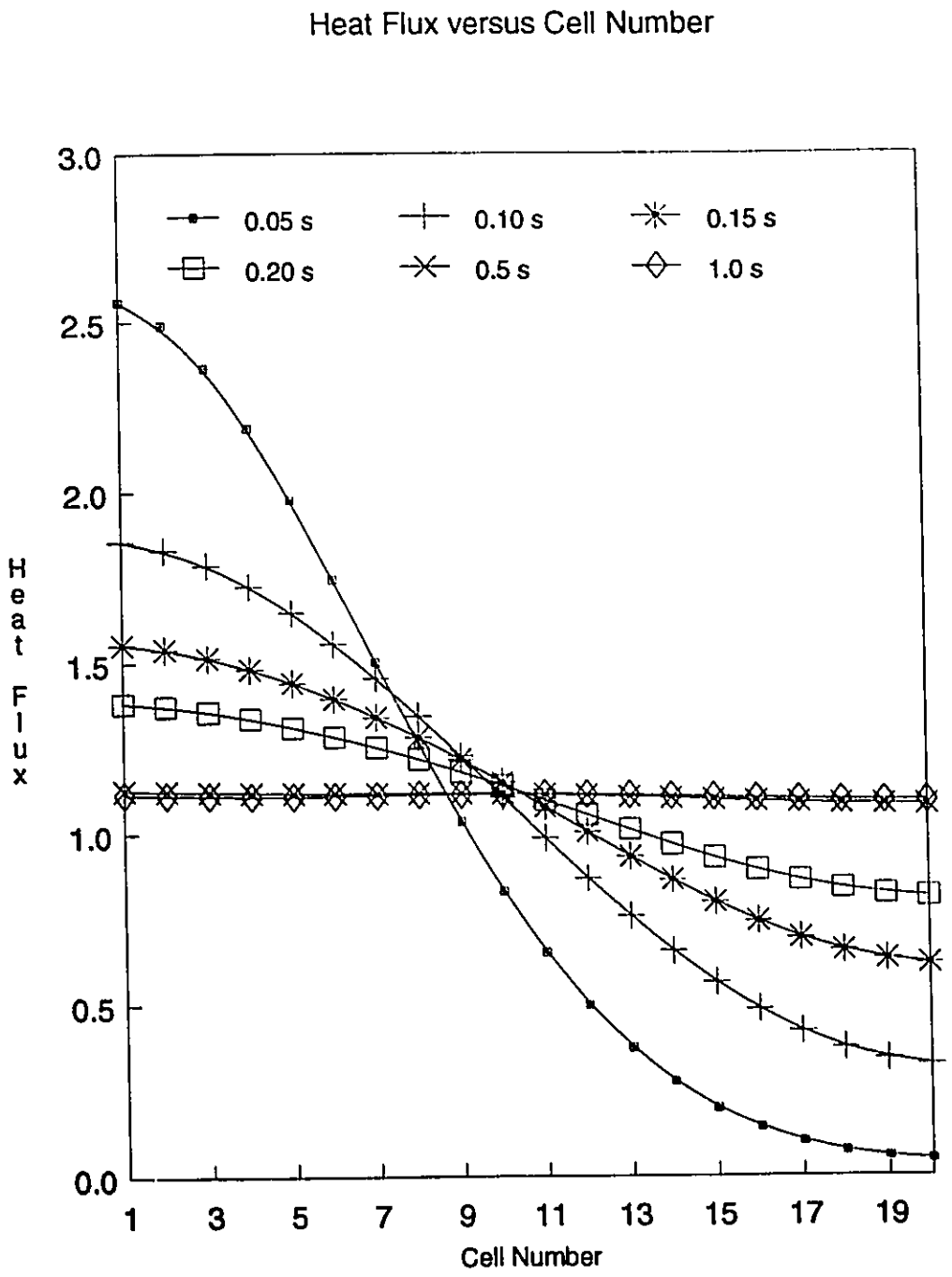

Figure 10.2-1: Transient Heat Flux

10.2.2 The Effect of G_1

In another set of experiments, the effect of the parameter G_1 is examined for Rayleigh numbers of 10^3 to 10^6 using a *16x16* cell array. The simulation was run for 100 iterations using a fixed time step of ($\Delta t = 0.01(s)$). The mass error variance is plotted in Figure 10.2-2 as a function of the parameter G_1, which is varied from 10^2 to 10^{12}. A smaller value of variance indicates a more converged solution. The results demonstrate that simulations using larger values of G_1 converge at a greater rate.

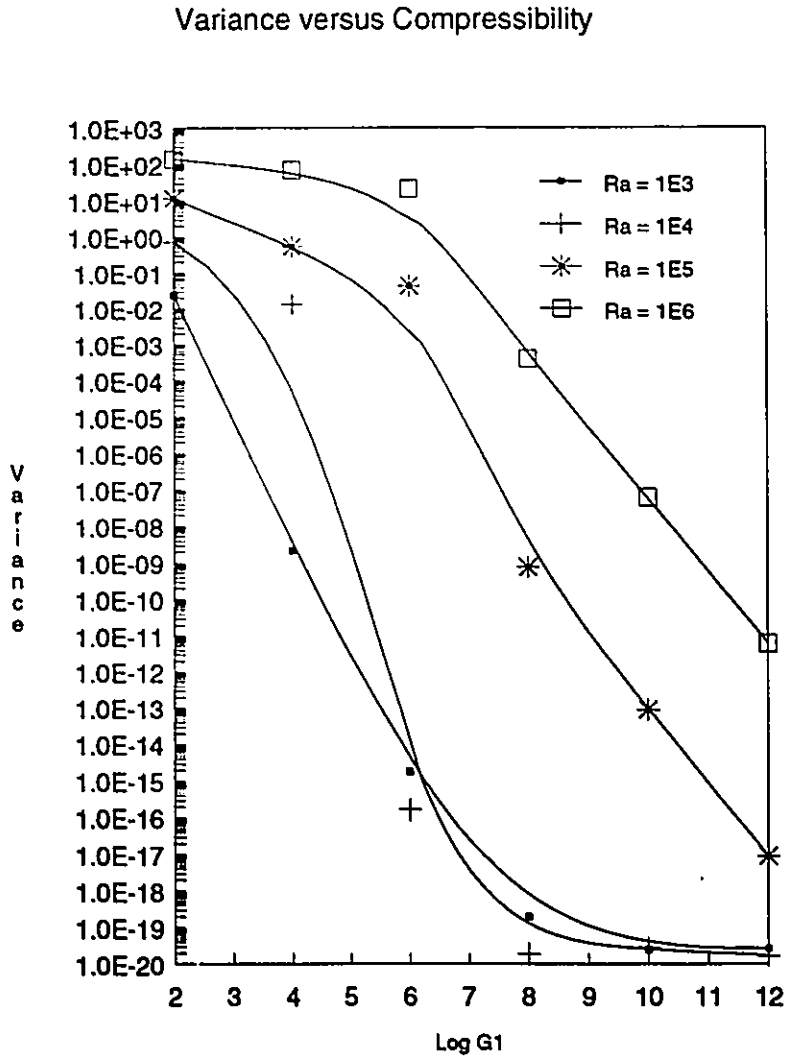

Figure 10.2-2: Variance versus Compressibility

10.2.3 Steady State Solutions

Converged solutions for simulations were sought using the various Rayleigh numbers. The G_l parameter used was whatever value had good convergence properties, as only the steady state solution was desired.

In Table 10.2-1, *Vx(max)* is the maximum horizontal velocity at the vertical mid-plane, and y is the location on that plane. *Vy(max)* is the maximum vertical velocity at the horizontal mid-plane and x is the location on that plane. The mean value of the heat flux in the *x* direction (*Q(x)* as described in section 6.1) is listed. The value given for the published results is the mean Nusselt number and not the total heat flux.

Figures 10.2.3-1 to 10.2.3-4 are the velocity vector plots for the *20x20* mesh using the various Rayleigh numbers. The velocities are normalized therefore the vector lengths indicate relative velocities. The velocities listed in Table 10.2-1 give a better indication of absolute velocities.

Array	Rayleigh	Vx(max)	y	Vy(max)	x	Q_x(mean)
16x16	10^3	3.502	0.844	3.557	0.156	1.107
20x20	10^3	3.569	0.825	3.608	0.175	1.108
30x30	10^3	3.602	0.817	3.638	0.183	1.111
36x36	10^3	3.610	0.819	3.652	0.181	1.112
Published	10^3	3.649	0.813	3.697	0.178	1.118
16x16	10^4	16.576	0.844	18.340	0.094	2.255
20x20	10^4	16.597	0.825	18.932	0.125	2.251
30x30	10^4	16.577	0.817	19.507	0.117	2.250
Published	10^4	16.178	0.823	19.617	0.119	2.243
16x16	10^5	40.170	0.156	66.308	0.906	5.282
20x20	10^5	38.888	0.875	70.518	0.075	4.859
30x30	10^5	38.422	0.150	69.376	0.917	4.845
Published	10^5	34.73	0.855	68.59	0.066	4.519
10x10	10^6	107.519	0.906	261.635	0.969	13.720
20x20	10^6	110.262	0.875	231.937	0.025	13.522
30x30	10^6	73.433	0.917	207.170	0.050	8.751
Published	10^6	64.63	0.850	219.36	0.0379	8.800

Table 10.2-1: Steady State Benchmark Simulation Results

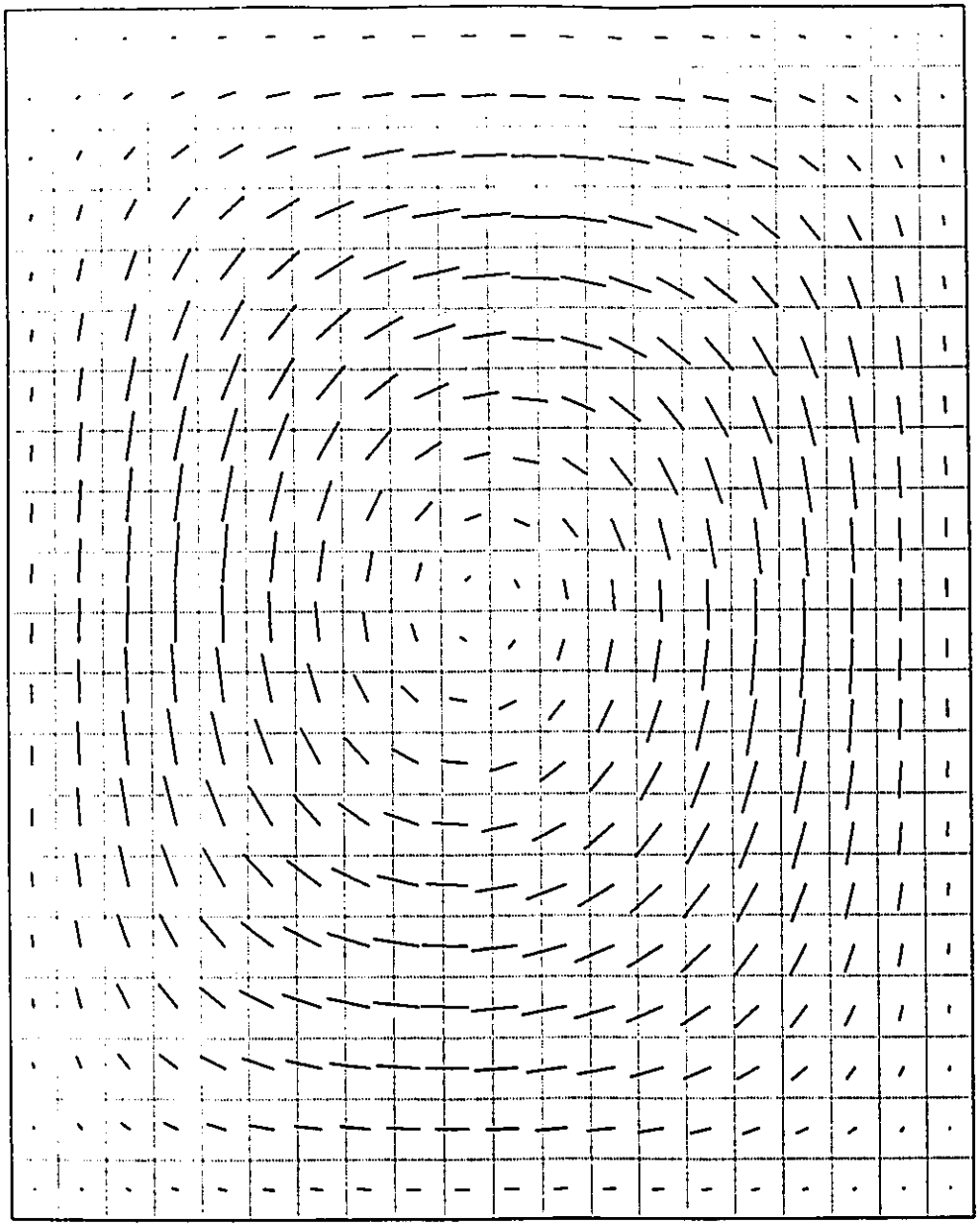

Figure 10.2.3-1: Vector Plot for Ra = 10^3

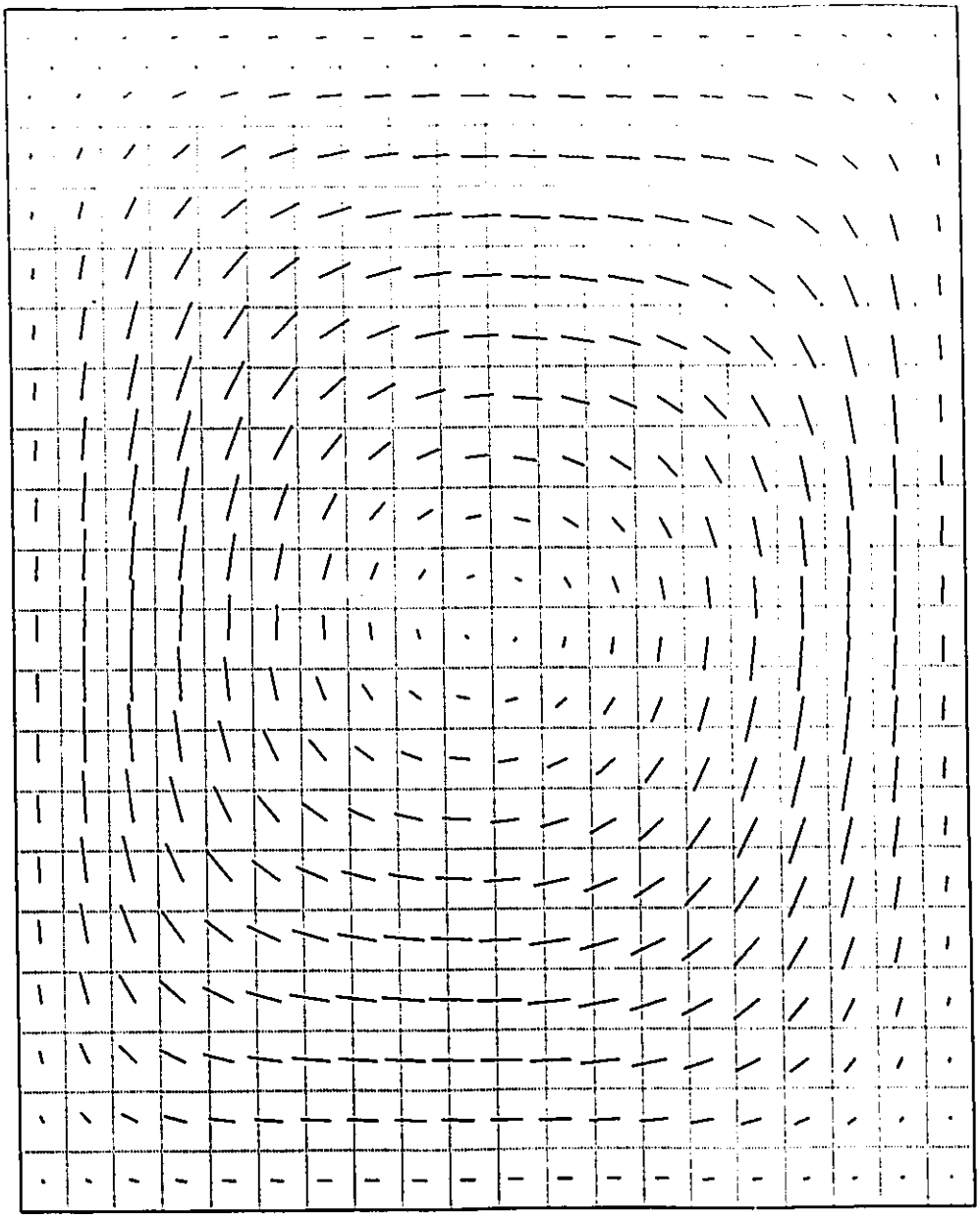

Figure 10.2.3-2: Vector Plot for Ra = 10^4

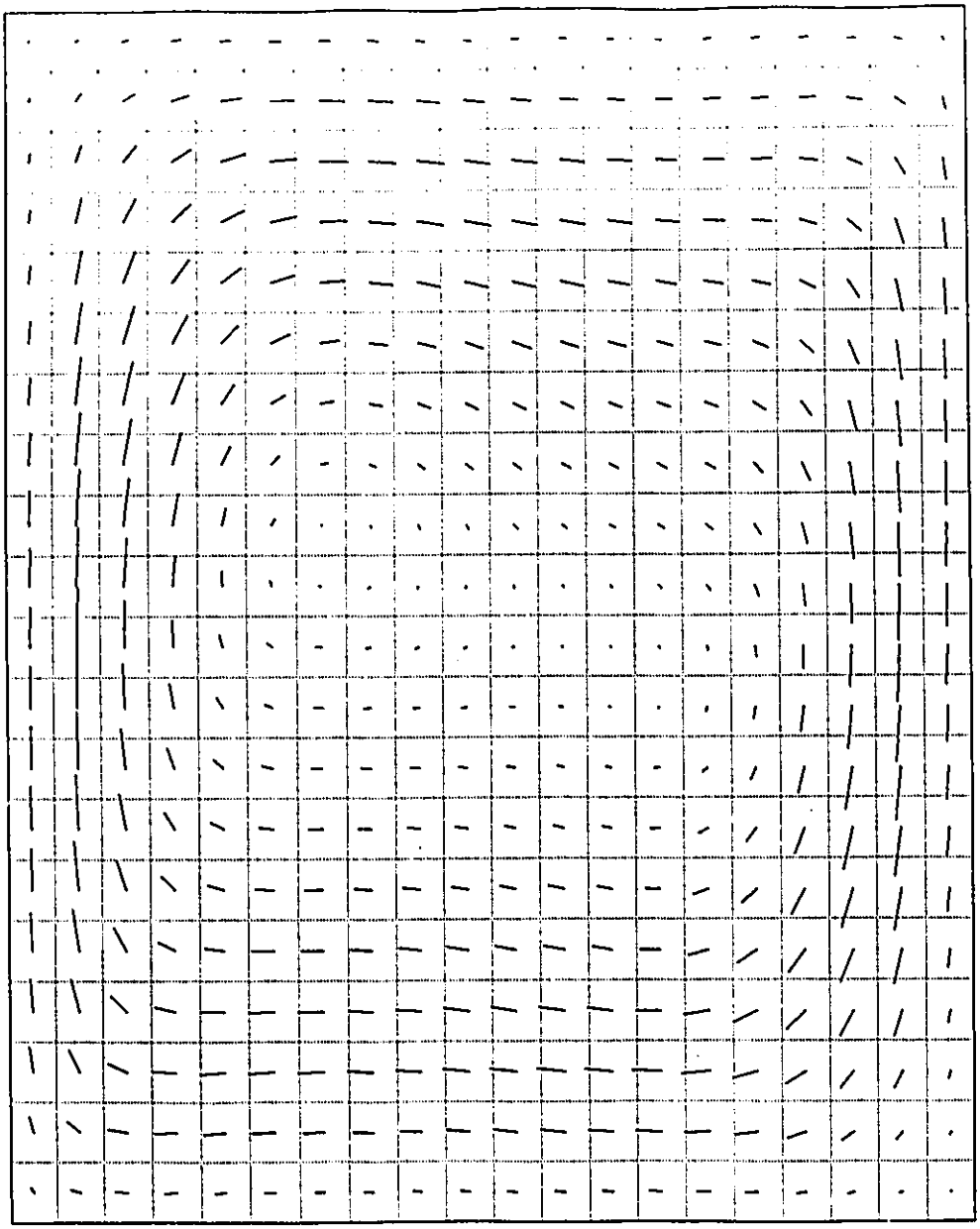

Figure 10.2.3-3: Vector Plot for $Ra = 10^5$

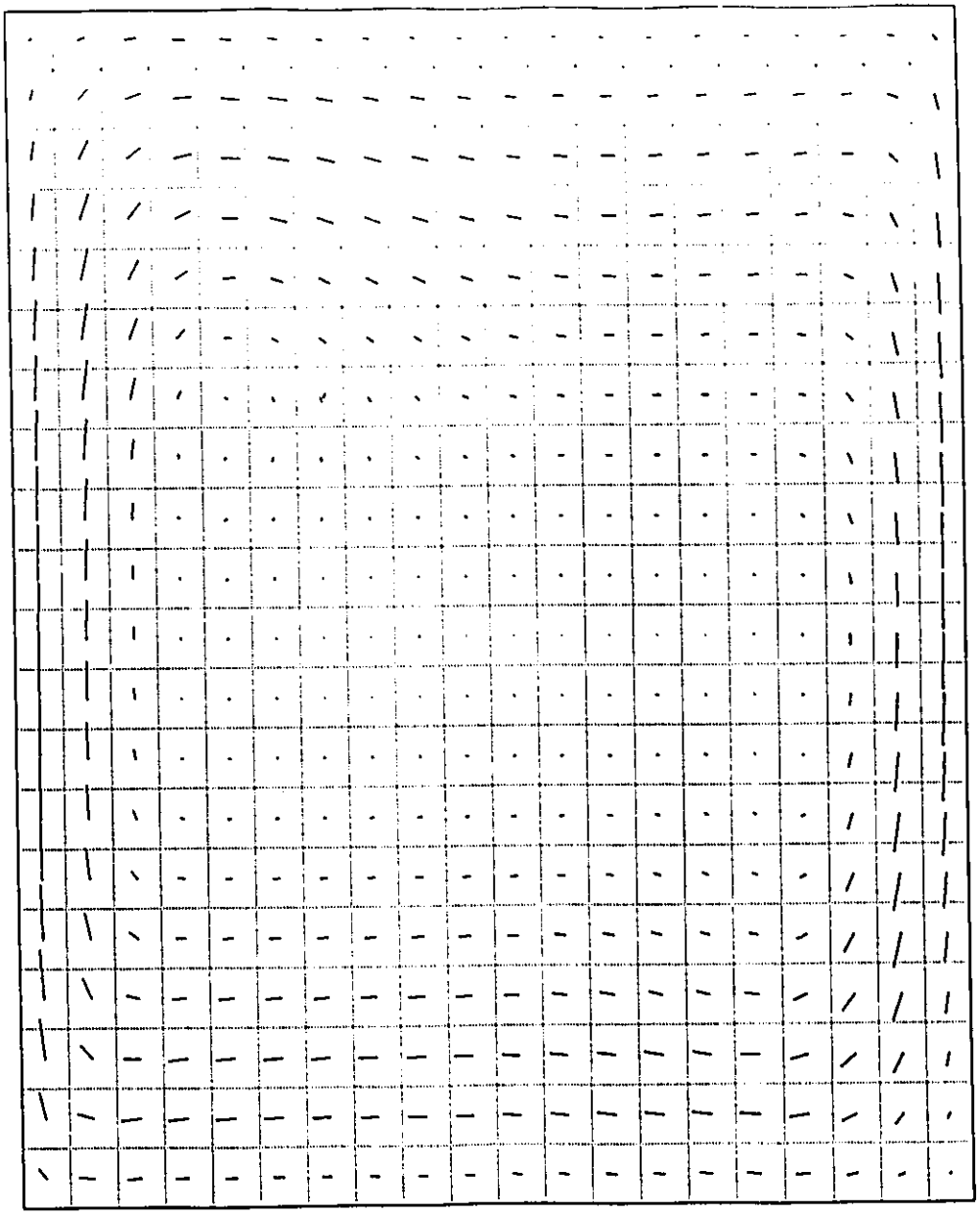

Figure 10.2.3-4: Vector Plot for Ra = 10^6

10.3 Convective Flow Through a Vertical Channel

The thermalhydraulic problem that is solved in the reactor simulation is convective flow through a vertical channel (see Figure 8.1-1). The properties of the fluid were that of 50 °C water, which was used in all subsequent flow simulations. The boundary conditions at the top and bottom are a constant hydrostatic pressure gradient. Cold fluid enters the bottom of the channel, is warmed by the left wall and exits through the top. The left wall is at a constant temperature for the flow simulation but varies in the combined simulation. The right wall is thermally insulated. It was found that a modification to the channel was required for less erratic behavior. A vertical extension was made to the top of the channel where both right and left walls were thermally insulated. This allowed a separation between the top of the hot wall on the left and the constant pressure boundary at the top.

The behavior displayed by the flow is chaotic in nature. The simulation showed little indication of converging to a steady state solution however its behavior was bounded. The geometry of the problem had a great effect on the behavior of the fluid as was shown in two series of simulations. Both simulations used 8 points in the *X* direction with a mesh spacing of 0.125 (cm). Both simulations used the same number of points in the *Y* direction, 8 point for heated wall section and 16 points for the riser section, but differed in the mesh spacing. When a course mesh (10.0 cm.) was used the flow showed general trends but did not develop into any regular patterns as shown in Figure 10.3-3. When a finer mesh was used (1.0 cm.) the flow would develop into a regular pattern as shown in Figure 10.3-1. A phase plot of the average vertical velocity versus the coolant temperature is given in Figure 10.3-2, which clearly shows limit cycle behavior. The geometry of the longer tube is closer to that used by the combined simulation therefore unruly behavior may be expected from the thermalhydraulics in the combined simulation.

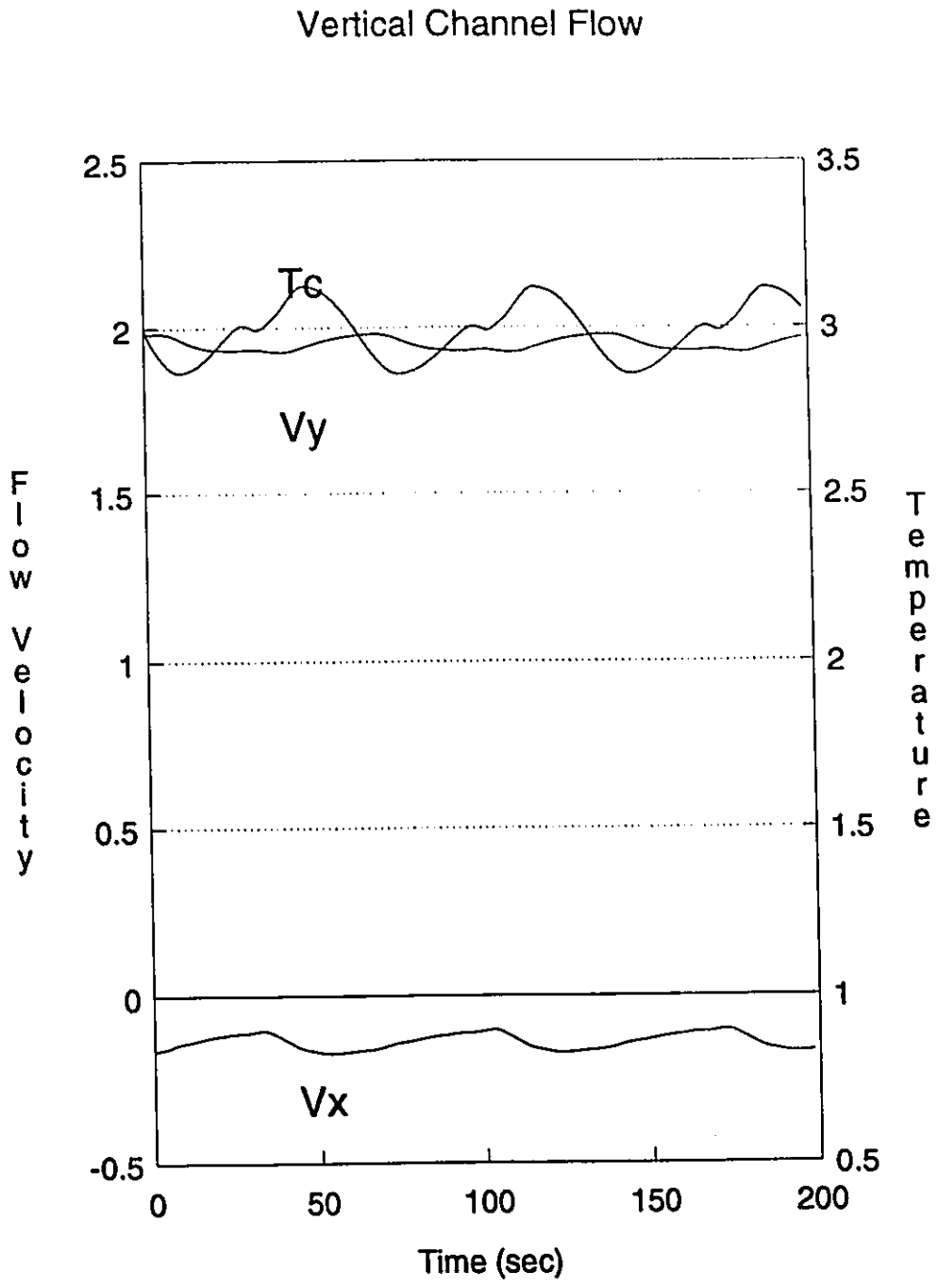

Figure 10.3-1: Vertical Channel Flow

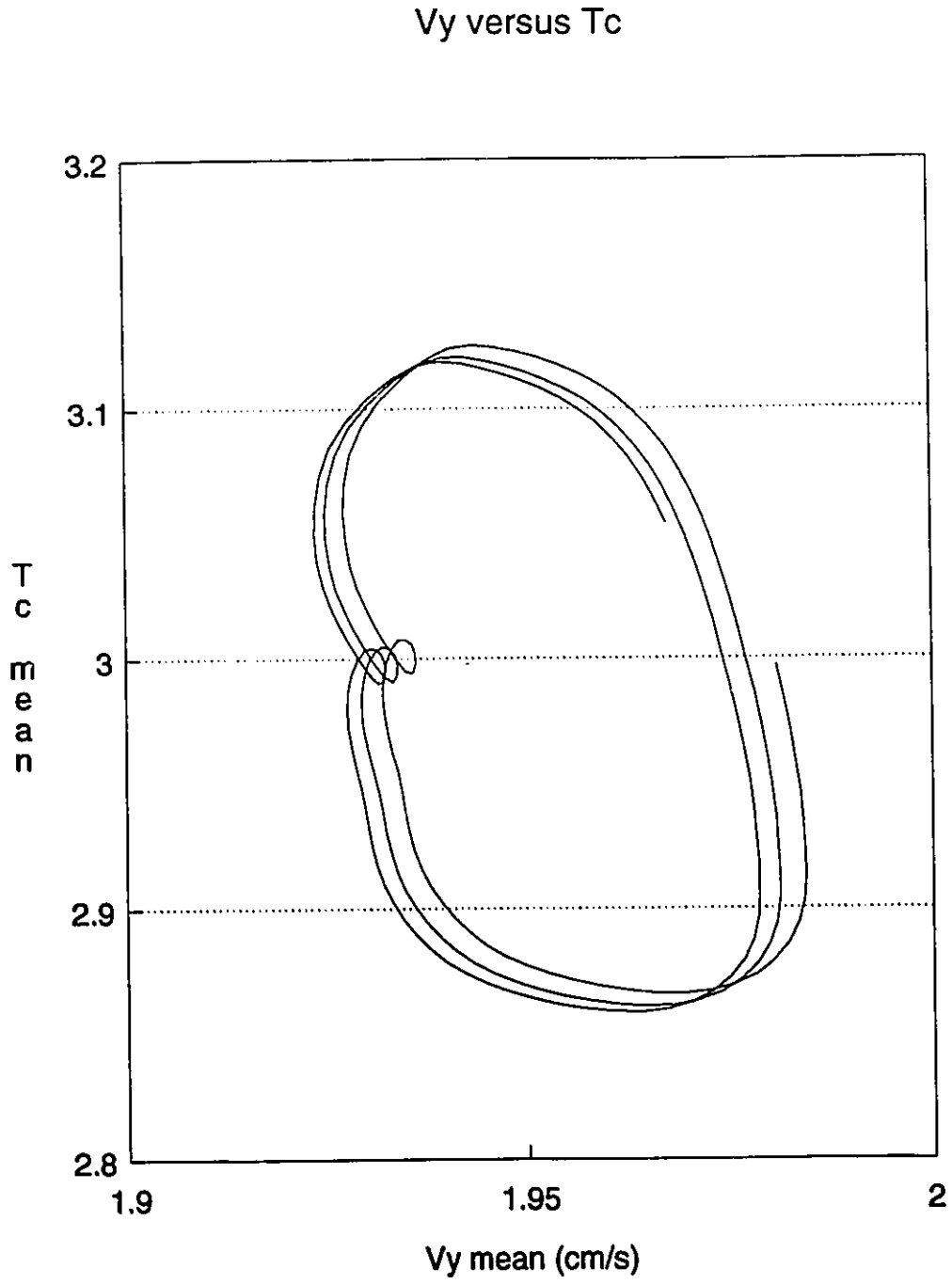

Figure 10.3-2: Vertical Channel Flow - Phase Plot

10.4 Neutronic Simulations

The neutronics simulation was checked both statically and dynamically. The check of the static portion neutronics simulation was a standard criticality calculation (*K-effective*). The results of the numerical criticality calculation are compared against analytical calculations of criticality (See appendix C). The transient calculation was checked by comparison against a point kinetics calculation. The results of these verification calculations are given below.

Nominal Data Values

The physical parameters used as the input data came from a variety of sources. The properties of the fluid are that of water at 20°C. The thermal properties of the fuel are that of U_3Si but, UO_2 properties are available. The cross-sectional information, while being realistic, is not particularly accurate. The main purpose here is to demonstrate a method and more accurate data would be a priority if real-world simulations were to be attempted.

Fluid Property:	**Value**	**Units**
Conductivity	6.44×10^{-3}	W /cm /°C
Prandtl number	3.55	(none)
Thermal expansion coefficient	6.1×10^{-4}	(none)
Density (0 °C)	1.0	g / cm^3
Viscosity	5.47×10^{-3}	poise
$\partial P/\partial \rho = G_1$ (see section 4.1.5)	2.0×10^{6}	kPa /(g /cm^3)
Fuel Property:		
Conductivity	1.5	W /cm /°C
Density	5.43	g / cm^3
Specific Heat	0.20	J /°C /g
Thermal Diffusivity	1.38	cm^2/s

Table 10.4-1: Nominal Data Values

10.4.1 Static Calculations

Analytical calculations of *K-effective* were made for some one neutron energy group problems. These calculations could then be compared to numerical calculations using a varying number of mesh points. The results for one and two dimensional are given below.

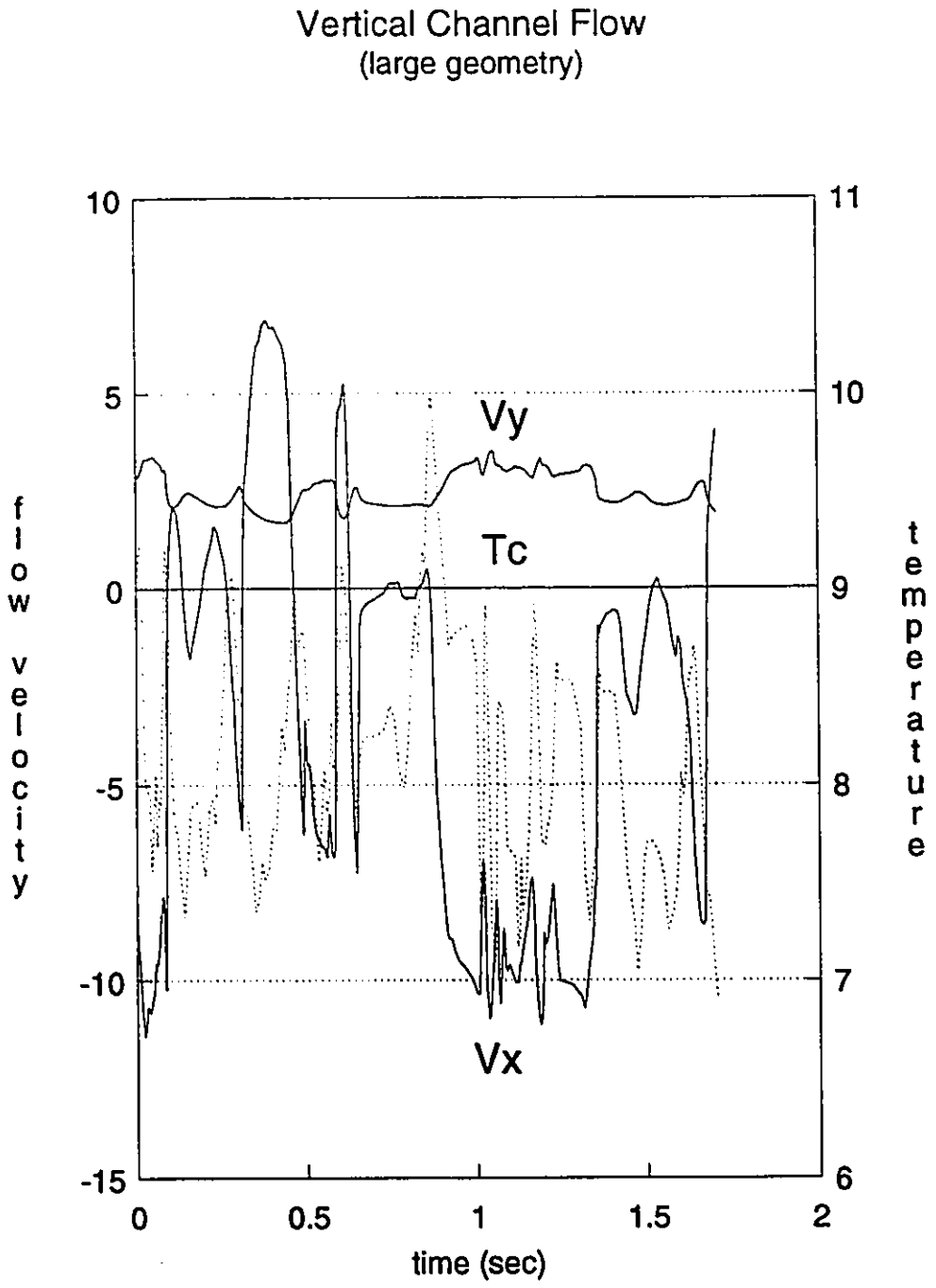

Figure 10.3-3: Vertical Channel Flow - Large Geometry

The default top and bottom boundary conditions for the neutronics in the two dimensional calculations is a water reflector. This is implemented using a Cauchy boundary condition (see Section 6.7.3) specifying a neutron current over flux parameter (J/Φ).

Numerical results are given for two neutron energy group simulations. Again, results are given for one and two dimensions. The two dimensional simulations explore the coolant temperature dependence of *K-effective* by holding the coolant temperature at several values (fixed and uniform).

Some results are given with fuel temperature feedback, some without. The fuel temperature (for fixed fuel properties) will depend on the coolant temperature and the neutron flux that is used. If the coolant temperature is not given, it is assumed to be zero. The flux level for the static calculations is 10^{12} (n/cm^2 /s). The fuel-coolant temperature differences is quite small in the static calculations at this flux level. The difference between the fuel temperature and the coolant temperature is proportional to the mean flux for the relatively low fluxes that are used.

In the following tables, K_n is the value for *K-effective* from a numerical calculation, K_a is the value from analytical calculation, x_1 is the distance from the left boundary to the fuel/coolant interface, x_2 is the distance to the right boundary, and *Tf* is the mean fuel temperature. Both the left and the right boundaries are reflective.

Static calculations were done for various geometries, either one or two neutron energy groups and either one or two spatial dimensions using a low mean flux level.

One Dimensional - One Group Static Calculations

A one dimensional, one neutron energy group, two region reactor in Cartesian coordinates is the model for the criticality calculation. Results are given with and without fuel temperature feedback.

X_1 (cm)	X_2 (cm)	K_a	K_n	Tf (°C)	# points
1.0	2.0	1.01495	1.01260	0.0	50
1.0	2.0	1.01495	1.01258	0.0880	50
1.0	2.0	1.01495	1.01349	0.0	80
1.0	2.0	1.01495	1.01348	0.0871	80
1.0	2.0	1.01495	1.01378	0.0	100
1.0	2.0	1.01495	1.01374	0.0868	100
1.0	2.0	1.01495	1.01440	0.0	200
1.0	2.0	1.01495	1.01438	0.0861	200
1.5	2.0	1.05975	1.05854	0.0	50
1.5	2.0	1.05975	1.05915	0.0	100
1.5	2.0	1.05975	1.05945	0.0	200

Table 10.4.1-1: One Dimensional - One Group Static Calculations

The values of *K-effective versus mesh spacing* for the cases with no fuel temperature feedback are plotted in Figure 10.4.1-1.

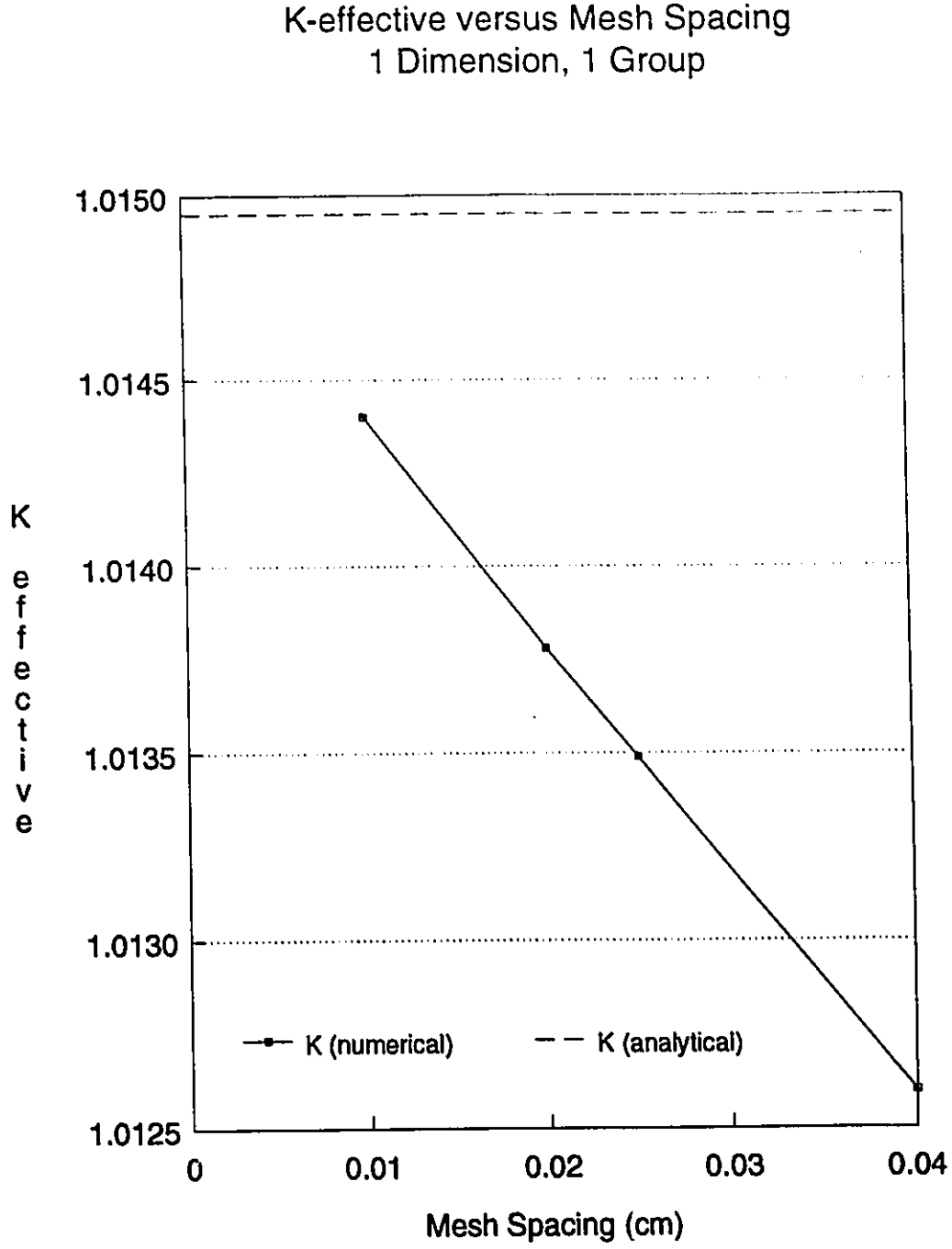

Figure 10.4.1-1: K-effective versus Mesh Spacing

Two Dimensional - One Group Static Calculations

Two dimensional, one neutron energy group, two region reactor in Cartesian coordinates is the model for the criticality calculation. In Table 10.4.1-2, $x_1=1.0$ is the distance from the left boundary to the fuel/coolant interface, $x_2=2.0$ is the distance to the right boundary, and *height* is the height of the reactor. Results are given with fuel temperature feedback.

height (cm)	K_s	K_n	*Tf* (°C)	# *X* points	# *Y* points
80.0	1.01250	1.01042	0.08678	100	8
80.0	1.01250	1.01103	0.08678	100	16
80.0	1.01250	1.01089	0.08679	100	32
50.0	1.01085	1.00853	0.08679	100	20

Table 10.4.1-2: Two Dimensional - One Group Static Calculations

One Dimensional - Two Group Static Calculations

One dimensional two group static calculations are included for comparison. All subsequent calculations use two neutron energy groups. No analytical calculations were done using two neutron energy groups therefore, that result is not given. These results can be compared to the two dimensional calculations using the same geometry in the *X* direction.

X_1 (cm)	X_2 (cm)	K_n	*Tf* (°C)	# points
1.0	2.0	1.08909	0.0	20
1.0	2.0	1.07341	0.0	50
1.0	2.0	1.07220	0.0	100

Table 10.4.1-3: One Dimensional - Two Group Static Calculations

Two Dimension - Two Group Calculations: K-effective versus Coolant Temperature

The dependency of K-effective on the coolant temperature is explored using a two dimensional, two group static calculation. The coolant temperature is fixed and uniform for these calculations while the fuel temperature is allowed to vary until coming to a steady state value. The right boundary condition for the fuel temperature is the fixed coolant temperature. These results, as shown in Figure 10.4.1-2, can be used to calculate a bulk coolant temperature feedback coefficient.

The mesh used in the *X* direction of the two dimensional calculations is slightly different from the 20 point case above. The fuel is 8x0.125 (1 cm) wide, the coolant is 8x0.125 (1 cm) and the moderator region is 4x0.01 (0.04 cm) wide. Eight mesh points are used in the *Y* direction with the total height being 8x10.0 (80 cm).

Coolant Temperature (°C)	*Fuel Temperature* (°C)	*K effective*
0.0	0.0939	1.07414
20.0	20.0935	1.07044
50.0	50.0930	1.06469
100.0	100.092	1.05584
200.0	200.090	1.03508

Table 10.4.1-4: K effective versus Coolant Temperature

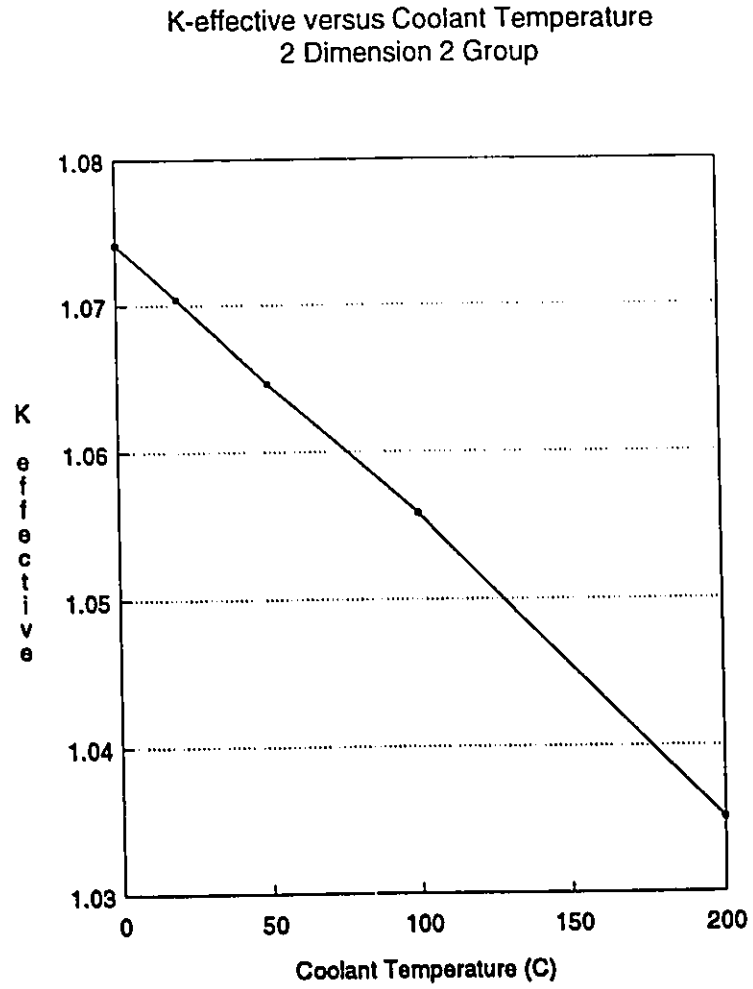

Figure 10.4.1-2: K-effective versus Coolant Temperature

10.4.2 Space-Time Kinetics versus Point Kinetics

The temporal behavior of the space-time kinetics simulation was verifyed by comparison with a point kinetics simulation. The problem chosen for the verification had to have sufficiently simple dynamics so that point kinetics would be adequate. The point kinetic model has only one neutron energy group therefore effects such as changes in moderation, which require at least a two neutron energy group model, were not considered. The problem chosen was a one-dimensional (slab) reactor with and without fuel temperature feedback. The coolant temperature is set at a uniform fixed value therefore no feedback effects are seen from the thermalhydraulics. A step reactivity insertion is used to perturb both the point kinetics and the space-time kinetics model.

A step reactivity insertion of 2.0 *mk* is used to perturb one dimensional transient simulations using one and two neutron energy groups and this output is compared to a point kinetics simulation. The geometry in the *X* direction is a three region model where $x_1 = 1.0$ (cm), $x_2 = 2.0$ and $x_3 = 2.5$ and a uniform mesh spacing of 0.1 (cm) was used. The coolant temperature is set at 20 (Celsius) which in these calculations fixes the temperature at the right fuel boundary at 20 degrees C. The converged results of static calculations is used as the starting point for the transient simulations. The one group static calculation had a *K-effective* of 0.97089 and a mean fuel temperature of 20.887(C). The two group static results were *K-effective* = 1.103229 and *Tf (mean)* = 21.024. The mean neutron flux used in the static calculations was 10^{13} (n/cm^{2}s).

The results of the transient simulations for t=20(ms) are given in Figures 10.4.2-1 to 10.4.2-3. Figures 10.4.2-4 and 10.4.2-5 give the results of the point kinetics transient and the two group transient for a longer time period, t=1.0(s) respectively. This is included as an indication of the longer time behavior of the simulation.

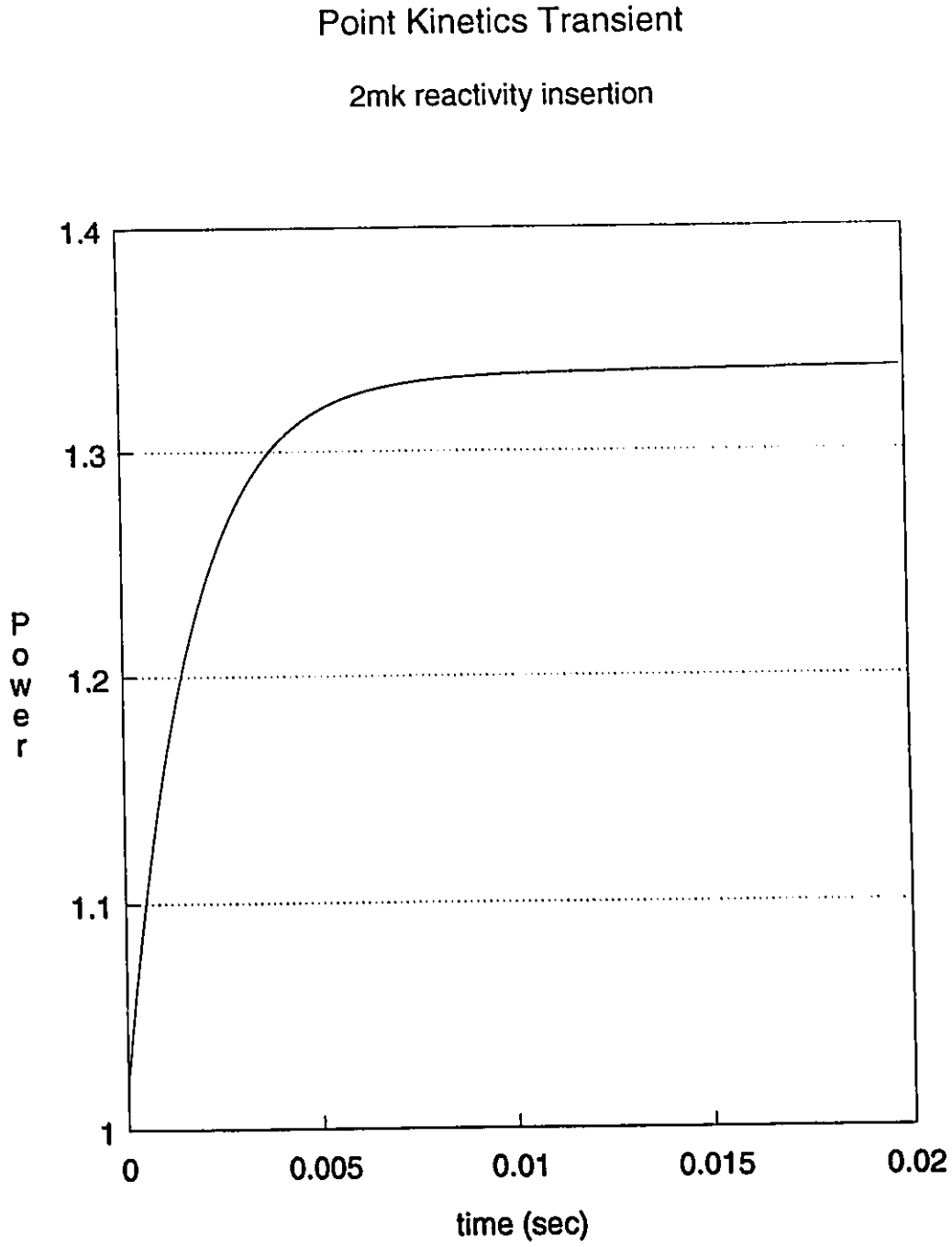

Figure 10.4.2-1: Point Kinetics 2 mk Transient

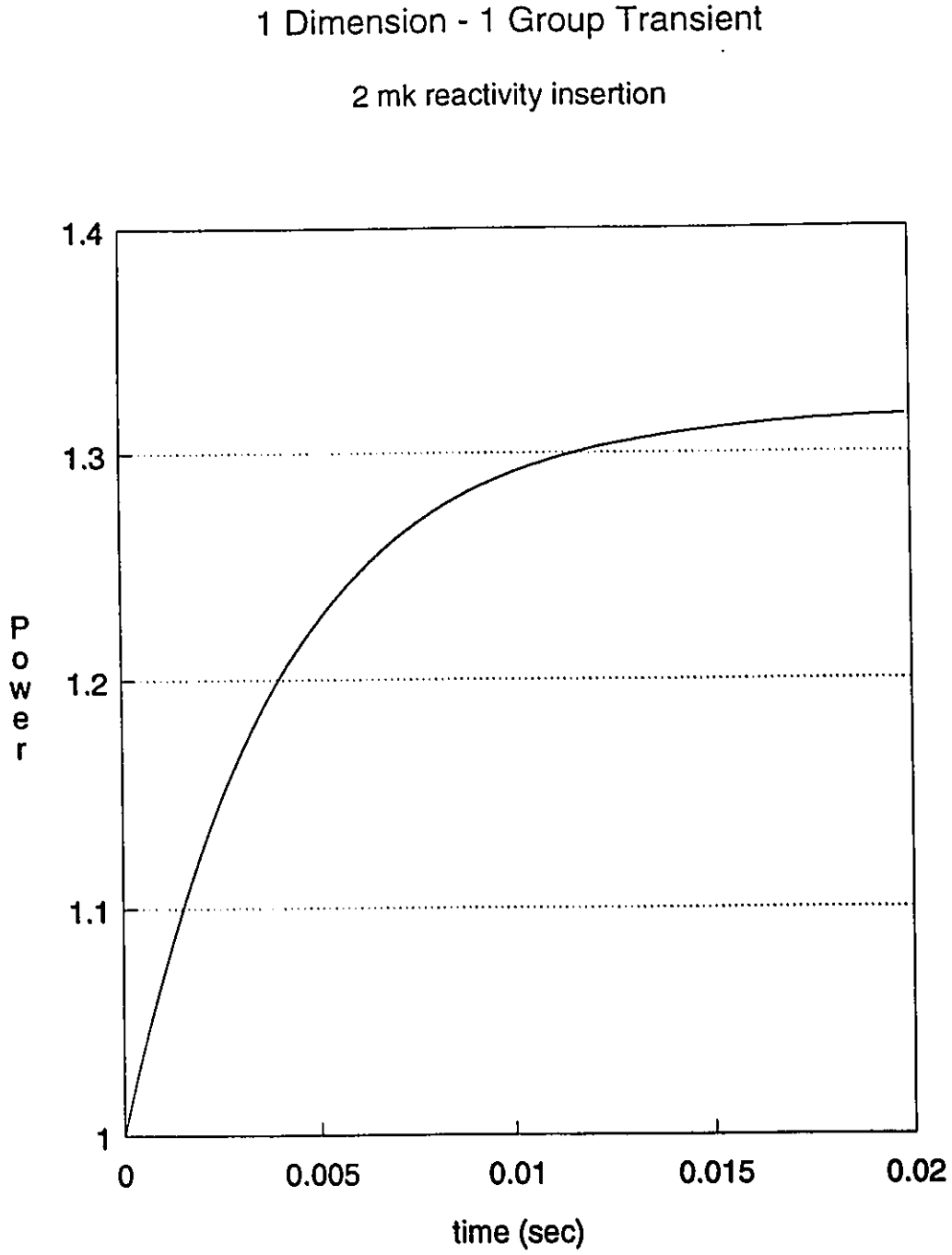

Figure 10.4.2-2: One Dimension - One Group 2 mk Transient

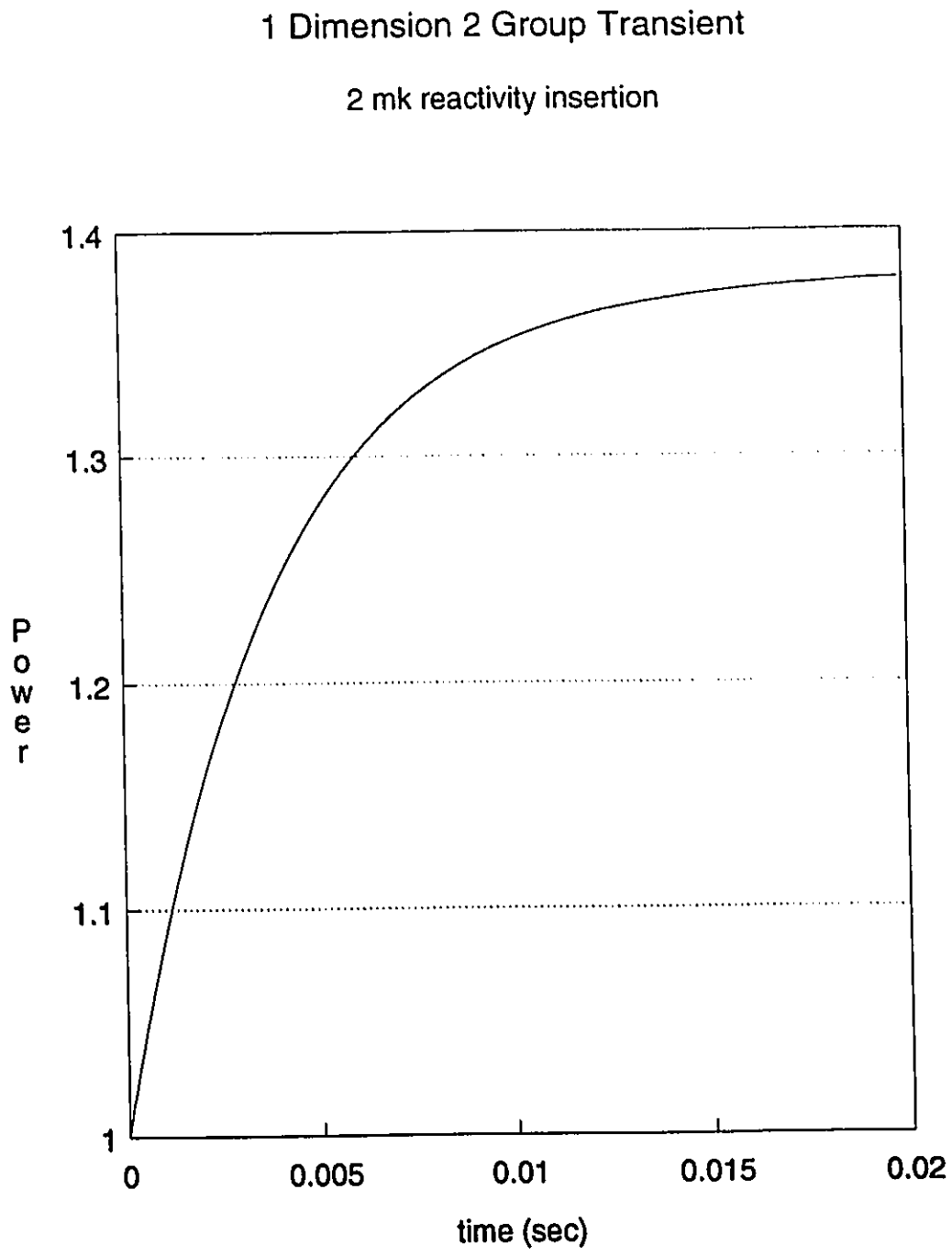

Figure 10.4.2-3: One Dimension - Two Group 2 mk Transient

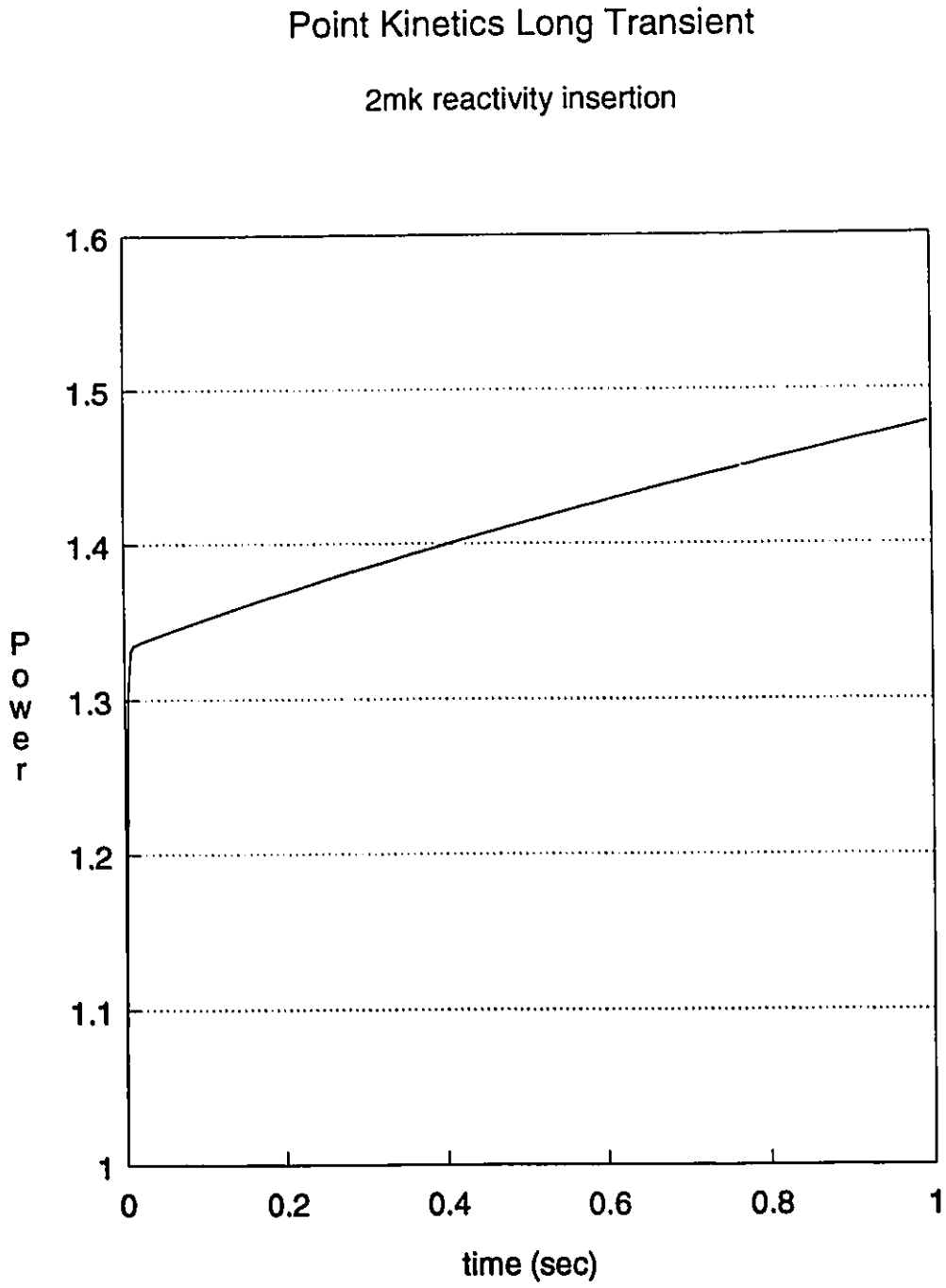

Figure 10.4.2-4: Point Kinetics Long Transient

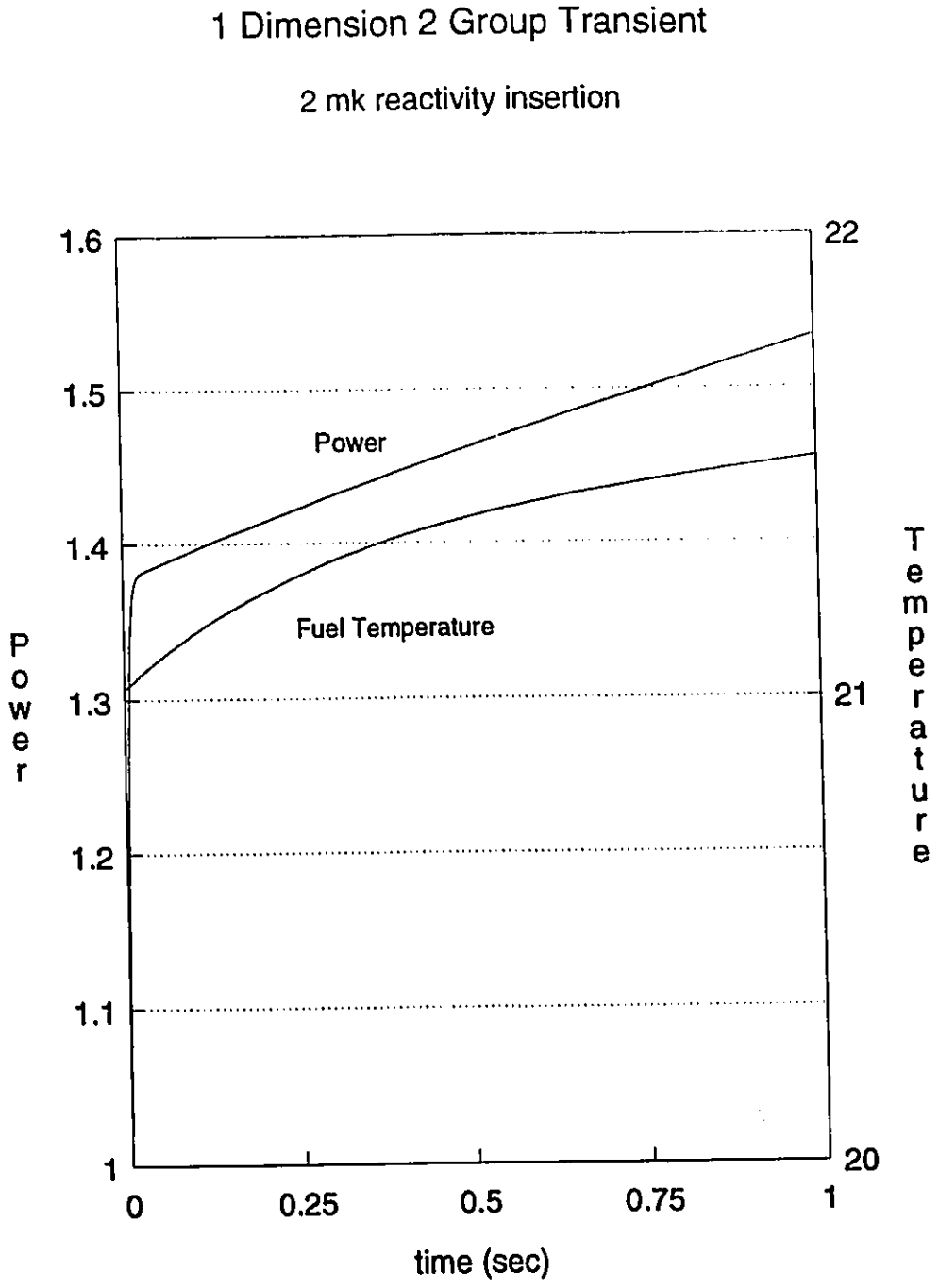

Figure 10.4.2-5: One Dimension - One Group Long Transient

10.5 The Complete Neutronic-Thermalhydraulic Simulation

The thermal behavior of the completed simulation is dominated by the coolant temperature. The fuel temperature will follow the coolant temperature even though the fuel was the source of heat for both the static and transient calculations. If a reactivity insertion is made at the beginning of a transient calculation, the reactor power would follow the same general trend as that for point kinetics initially but eventually the feedback effects of the fuel and coolant would predominate.

The unruly nature of the thermalhydraulic portion of the combined simulation created problems in the static calculation. The coolant density and temperature changed continuously during the static calculation. It was necessary to used a time-averaged value of the coolant density in order to achieve some sort of converged value for *K-effective*. Convergence was much less of a problem using the average density.

Combined neutronics-thermalhydraulics simulation results are given below. The power is held constant for the static simulations which are run until the fuel temperature achieves a relatively constant value. The final state of the system at the end of the static simulation is the starting point for the transient calculation. For the transient runs, 0, 2, 4 & 8 mk reactivity insertions are used to perturb the system. To investigate the possible effect that flow velocity might have on *K-effective*, some static calculations with constant flow velocity for the low flux case were run and the results are given below. The flux level of 10^{12} was initially used so a low value of fuel temperature would be obtained. The thermalhydraulic behavior was less violent with lower fuel temperatures.

One experiment used a fixed and uniform value for the vertical flow velocity in a static calculation. The fuel and coolant temperatures would rise, as expected, for lower values of *Vy*, but the value of *K-effective* would remain within the previously suggested bounds for low flux levels.

Tcold is the inlet coolant temperature, *Tf* is the mean fuel temperature and *Tc* is the mean coolant temperature in Table 10.5-1.

Vy (cm/s)	*Tcold*	*Tf* (°C)	*Tc* (°C)	$K_{effective}$
5.0	0.0	23.154	0.854	1.07331
10.0	0.0	17.668	0.438	1.07305
20.0	0.0	14.124	0.224	1.07331

Table 10.5-1: K effective versus Vertical Flow Velocity

Static Calculation

A static calculation was made where the thermalhydraulic simulation was unrestricted. The thermalhydraulics continued its unsettled behavior but the value of *K-effective* that was obtained, kn = 1.07022, was within the range that would be suggested by the *K-effective versus Coolant Temperature* calculations. A plot of the thermalhydraulic behavior including fuel temperature during the last run (the 13th) of the static calculation is given in Figure 10.5-1.

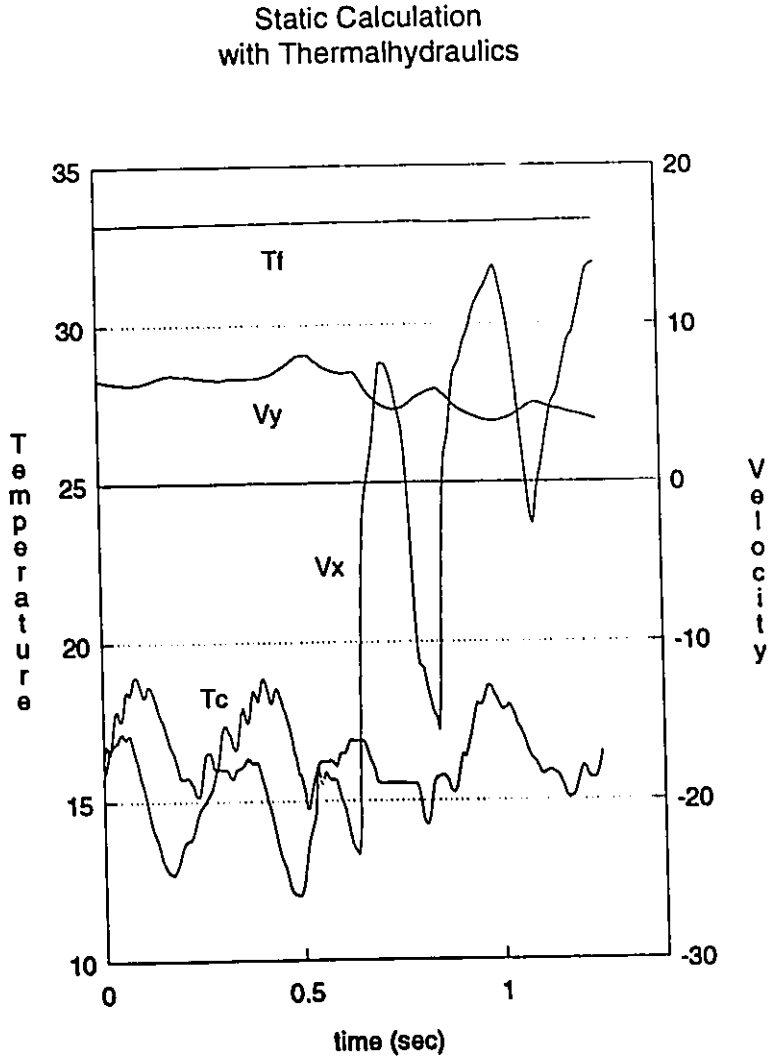

Figure 10.5-1: Static Calculation with Thermalhydraulics

Transient Calculation

The delayed neutron fraction for these simulations is *0.008* so that one dollar (1$) of reactivity, making the reactor prompt critical, would be 8mk. Transient simulations were run for reactivity insertions of 0$, 1/4$, 1/2$ & 1$. The shorter time scale (20 ms) is used to elucidate the prompt behavior of the simulations and the longer time scale (0.5s), relatively speaking, displays the longer time behavior. The results of all the transient simulation are given in Figures 10.5-2 to 10.5-11.

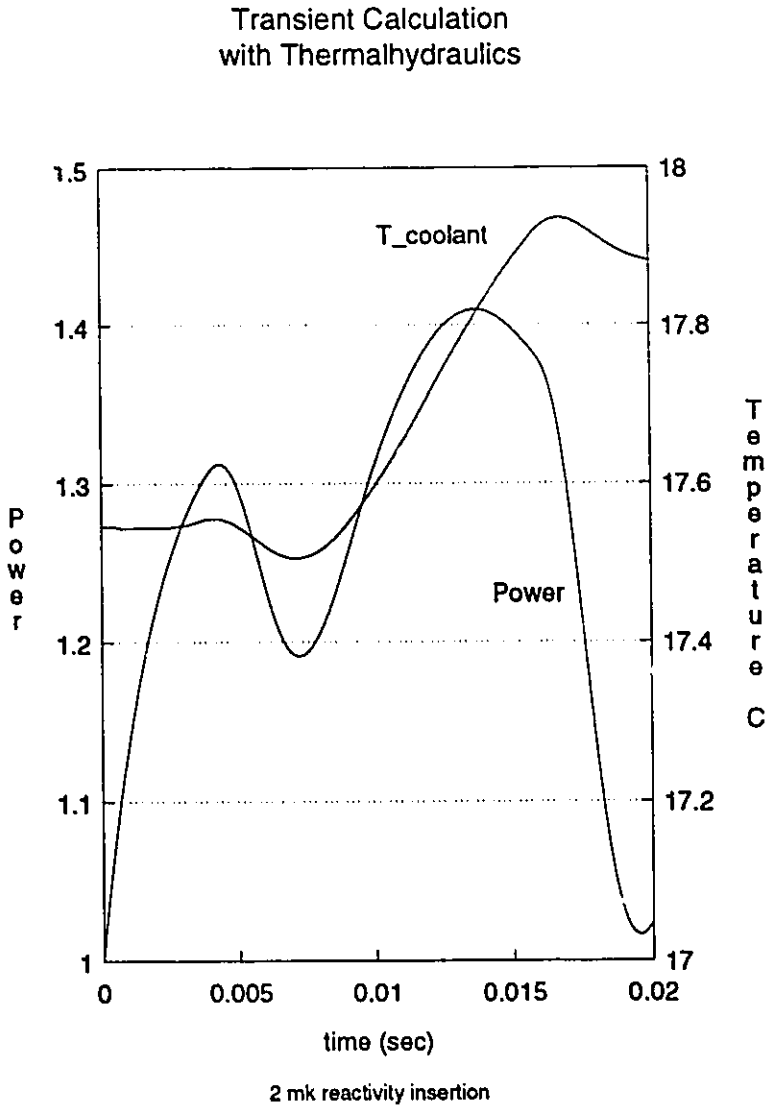

Figure 10.5-2: 2 mk Transient - Power versus Time

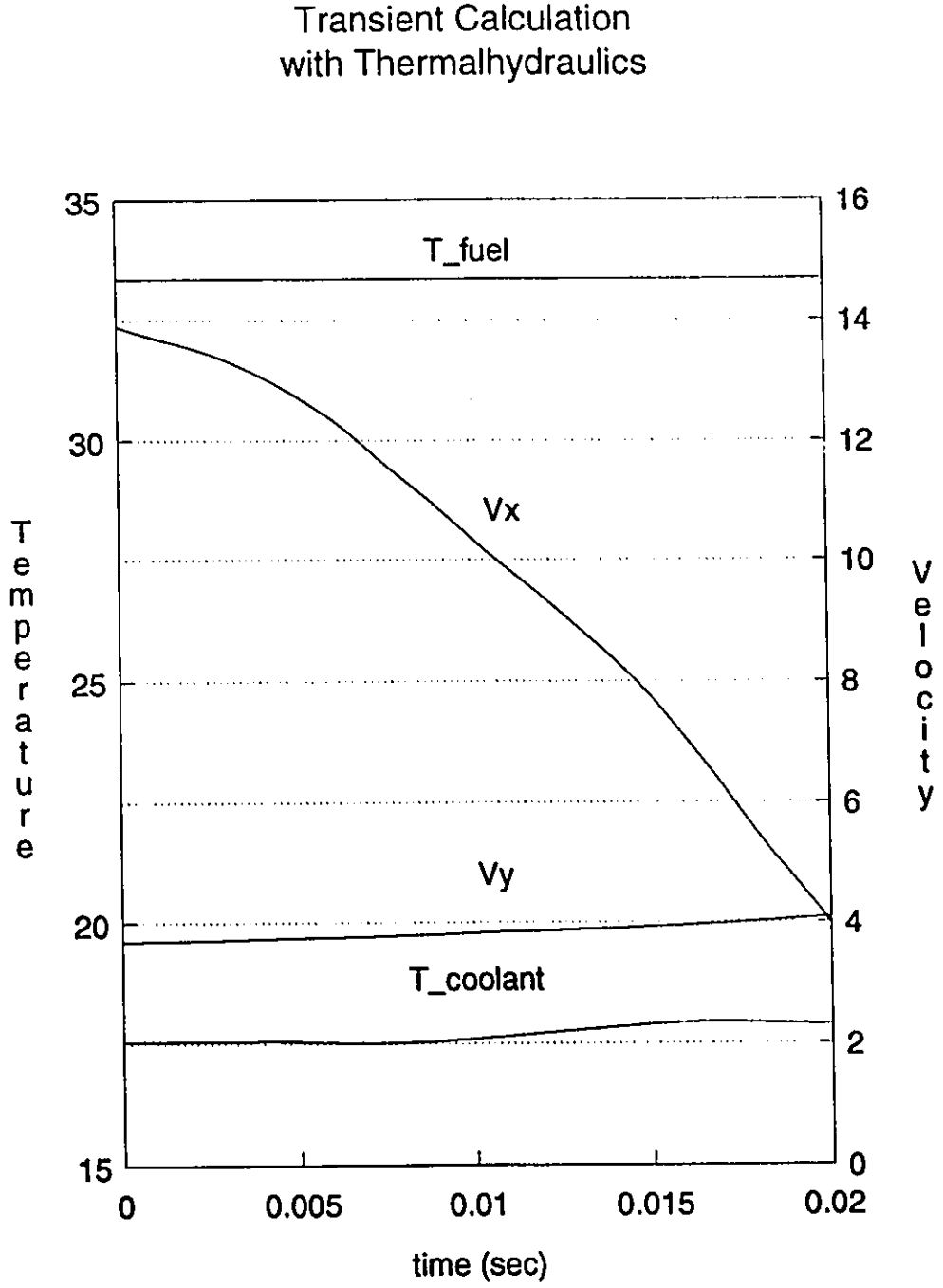

Figure 10.5-3: 2 mk Transient - Tf, Tc, Vx & Vy versus Time

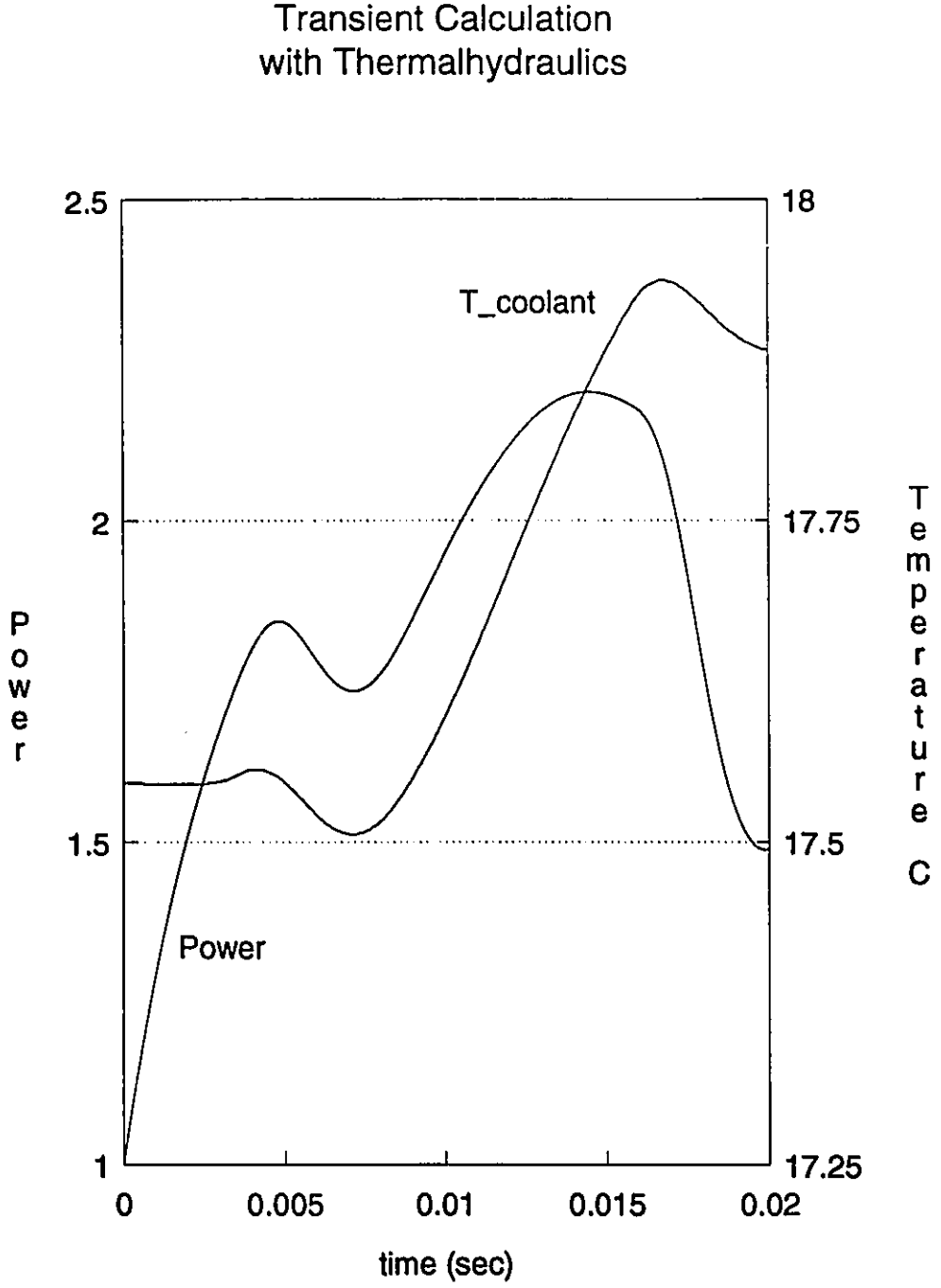

Figure 10.5-4: 4 mk Transient - Power & Tc versus Time

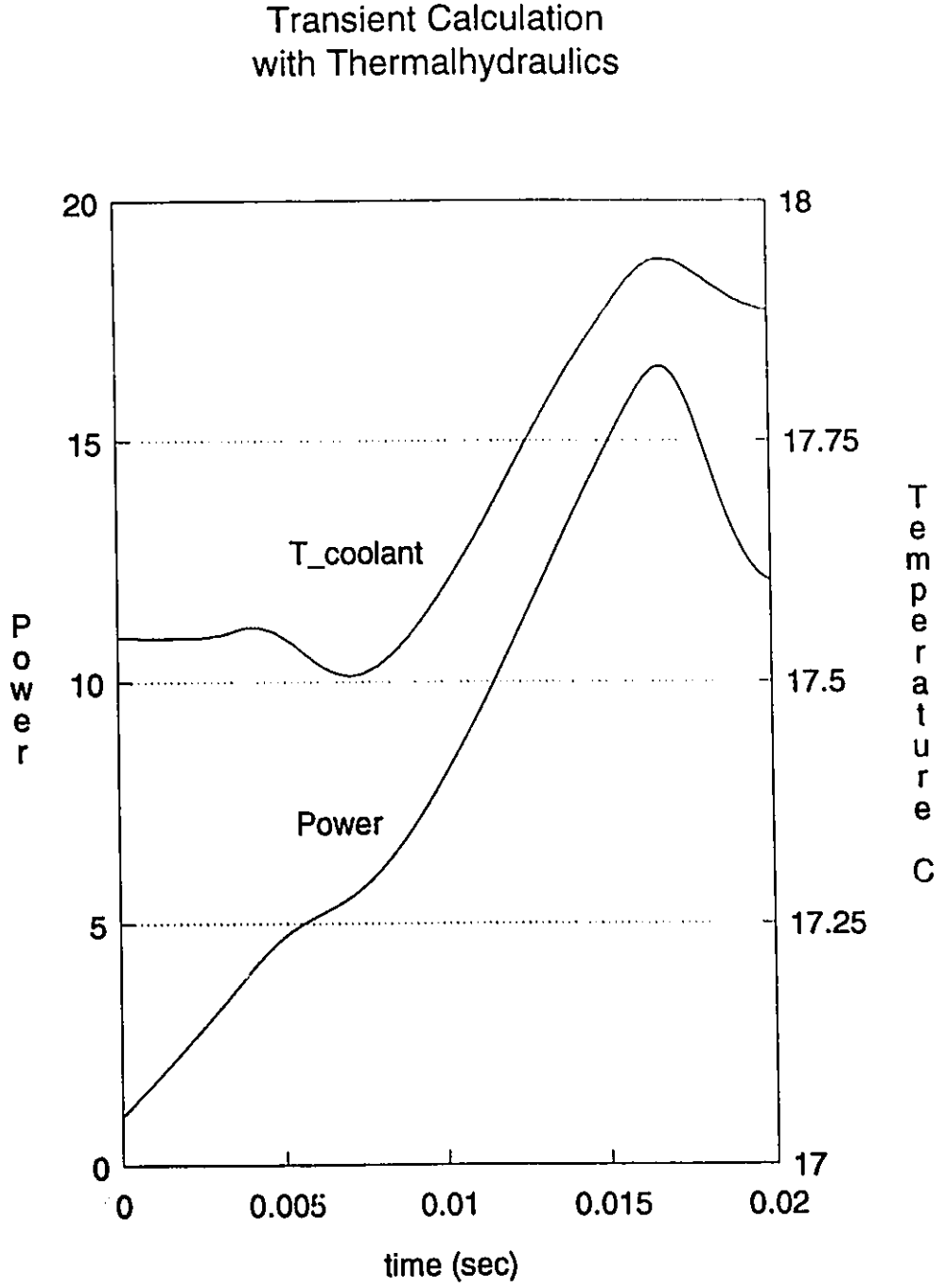

Figure 10.5-5: 8 mk Transient - Power & Tc versus Time

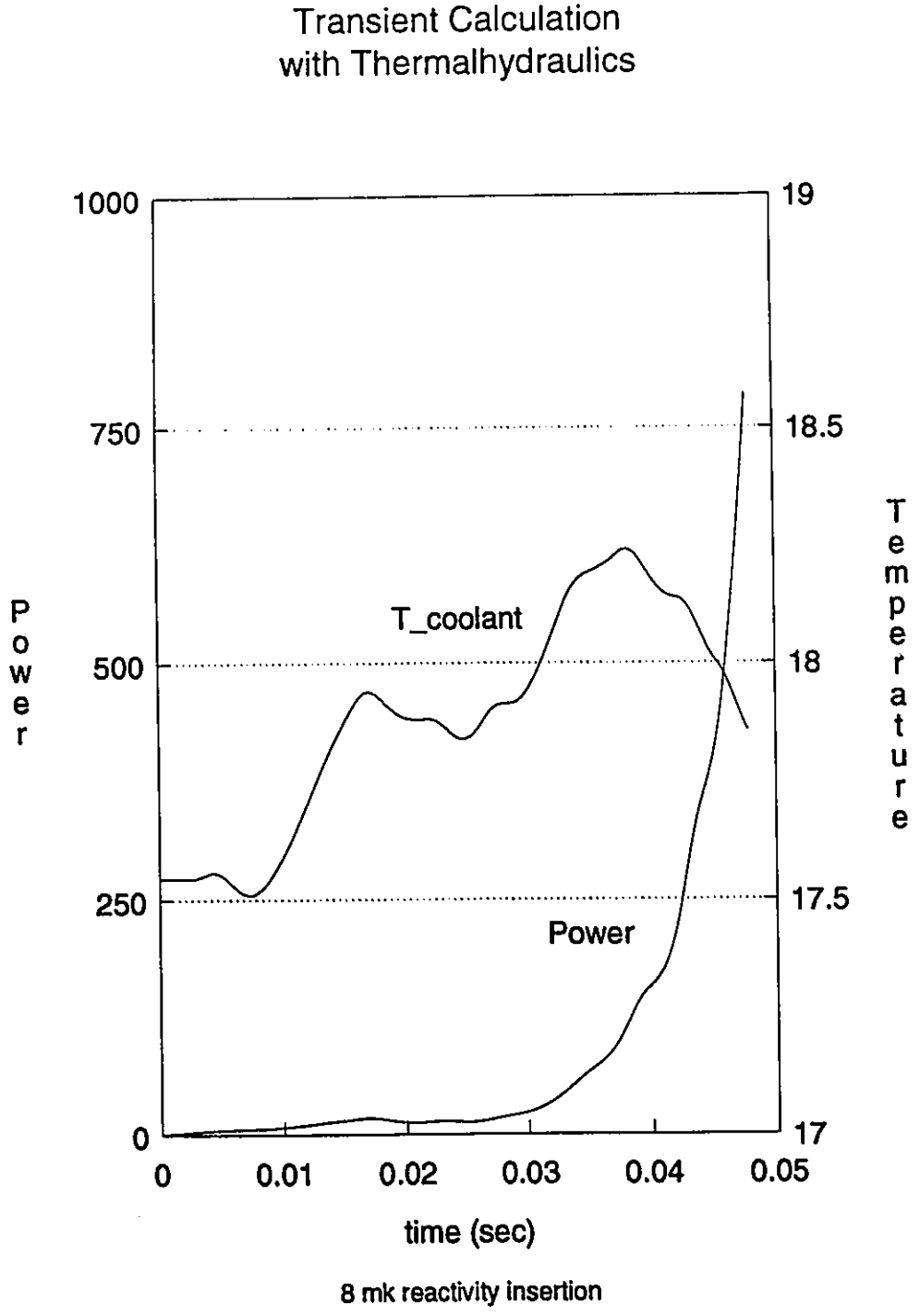

Figure 10.5-6: 8 mk Transient - Power & Tc versus Time (0.05s)

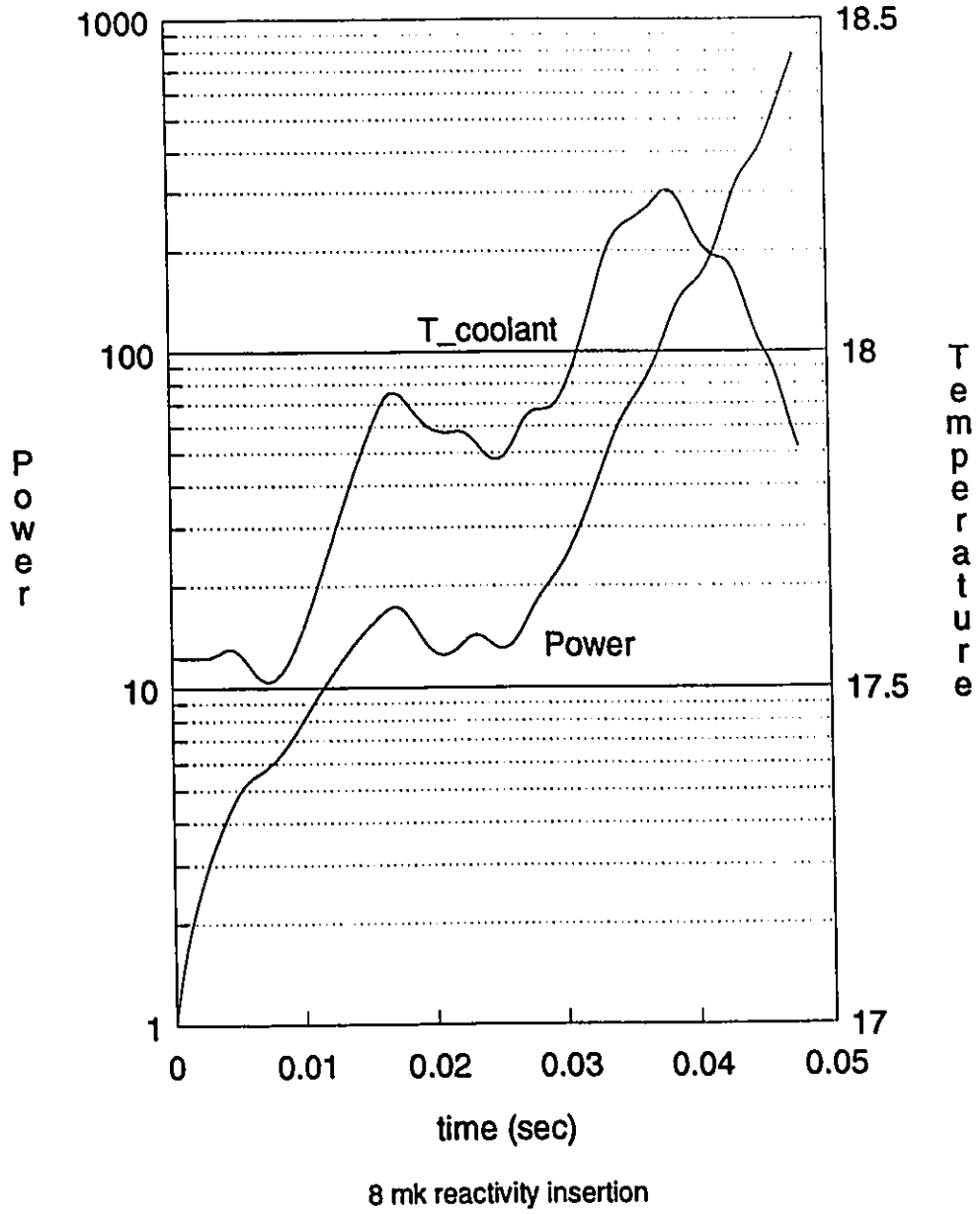

Figure 10.5-7: 8 mk Transient - Power(log) & Tc versus Time (0.05s)

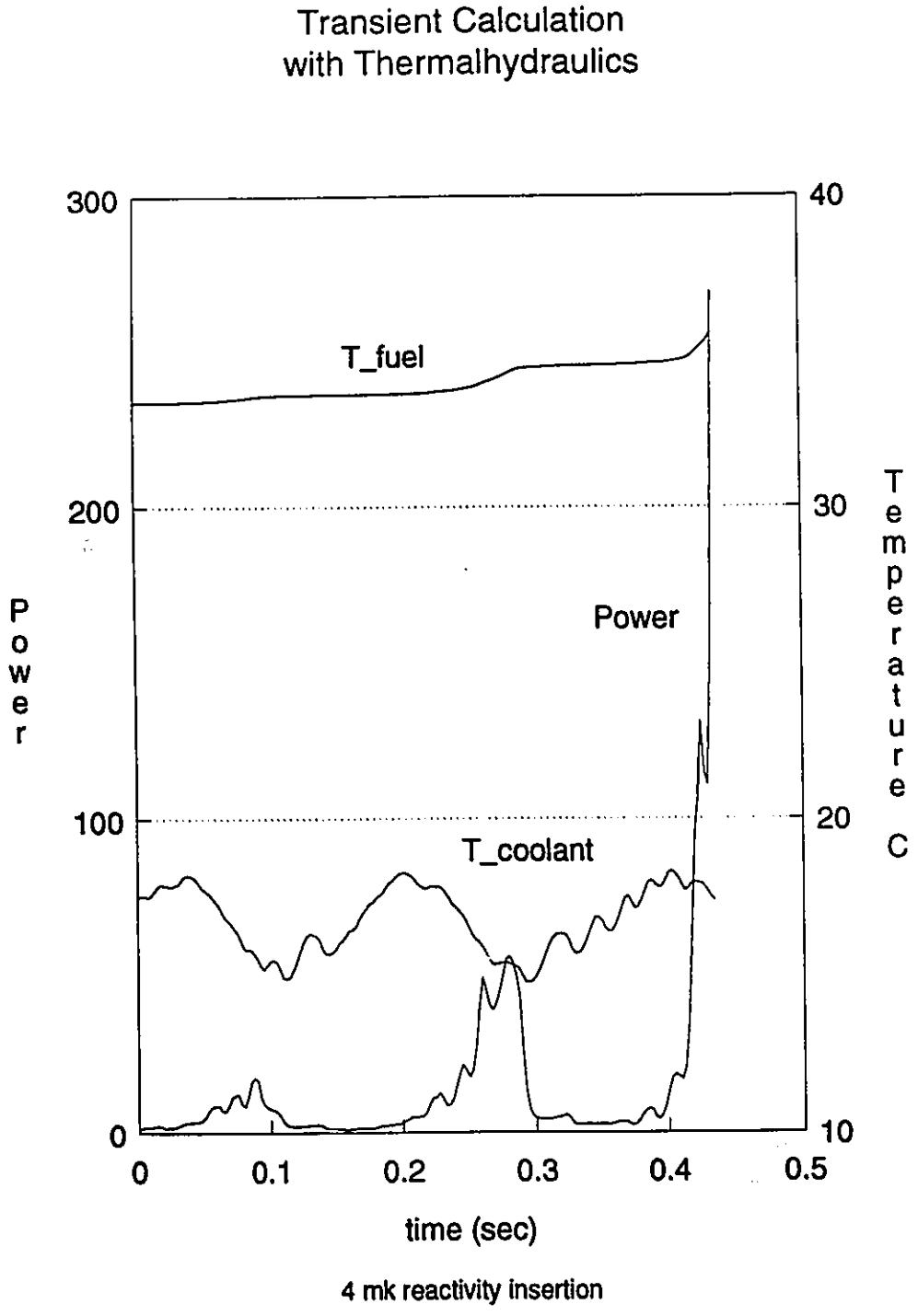

Figure 10.5-8: 4 mk Transient - Power & Tc versus Time (0.5s)

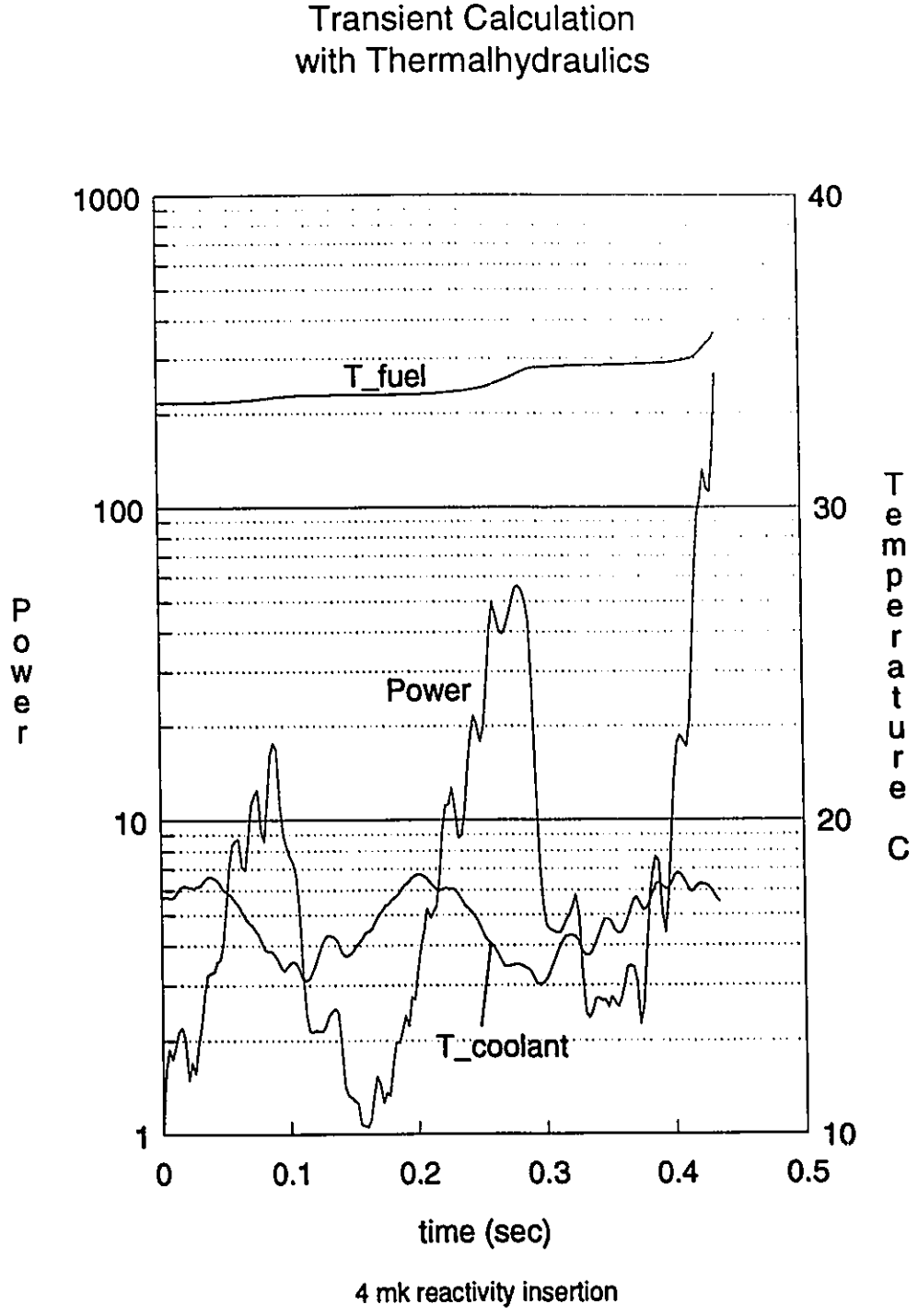

Figure 10.5-9: 4 mk Transient - Power(log) & Tc versus Time (0.5s)

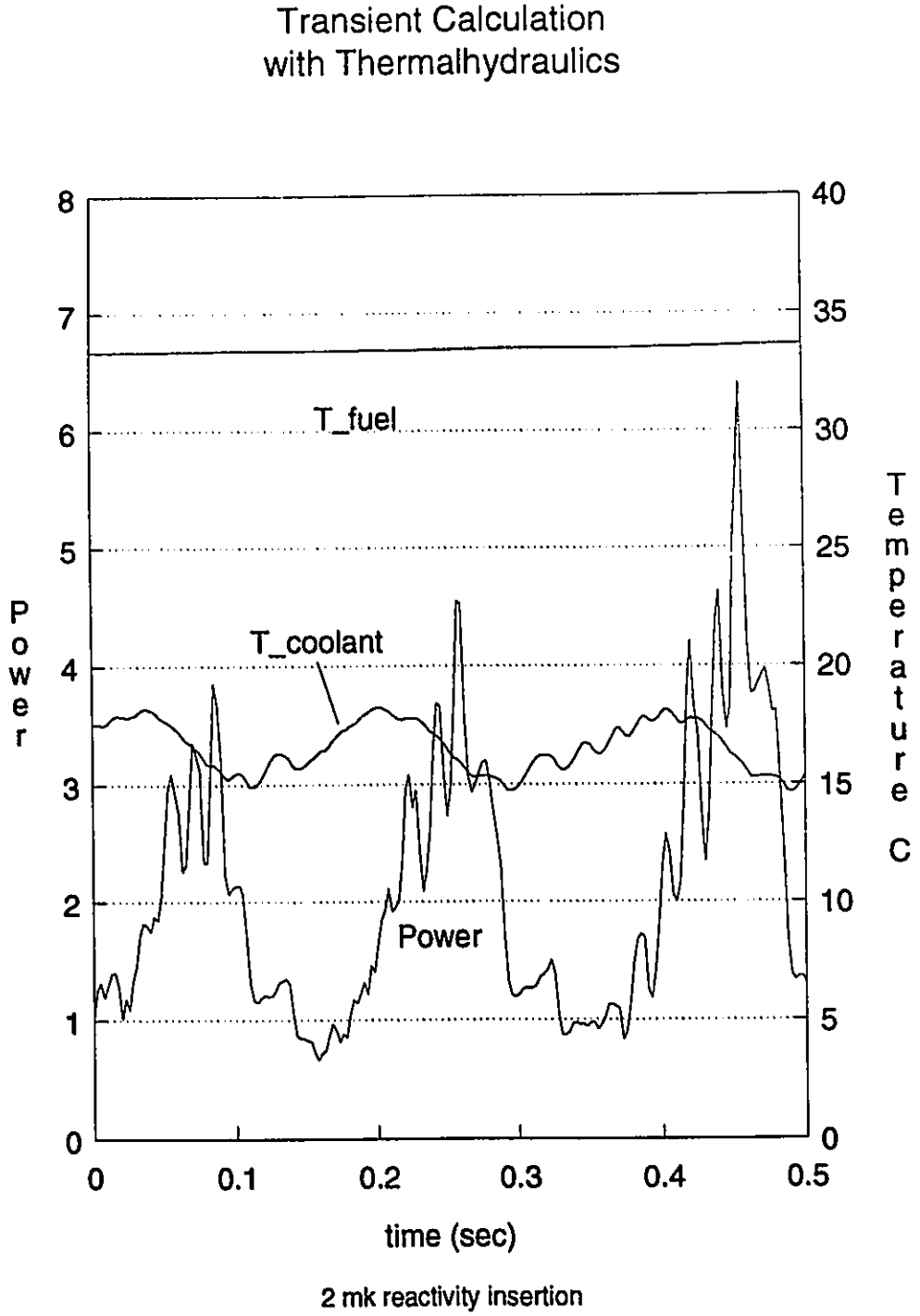

Figure 10.5-10: 2 mk Transient - Power & Tc versus Time (0.5s)

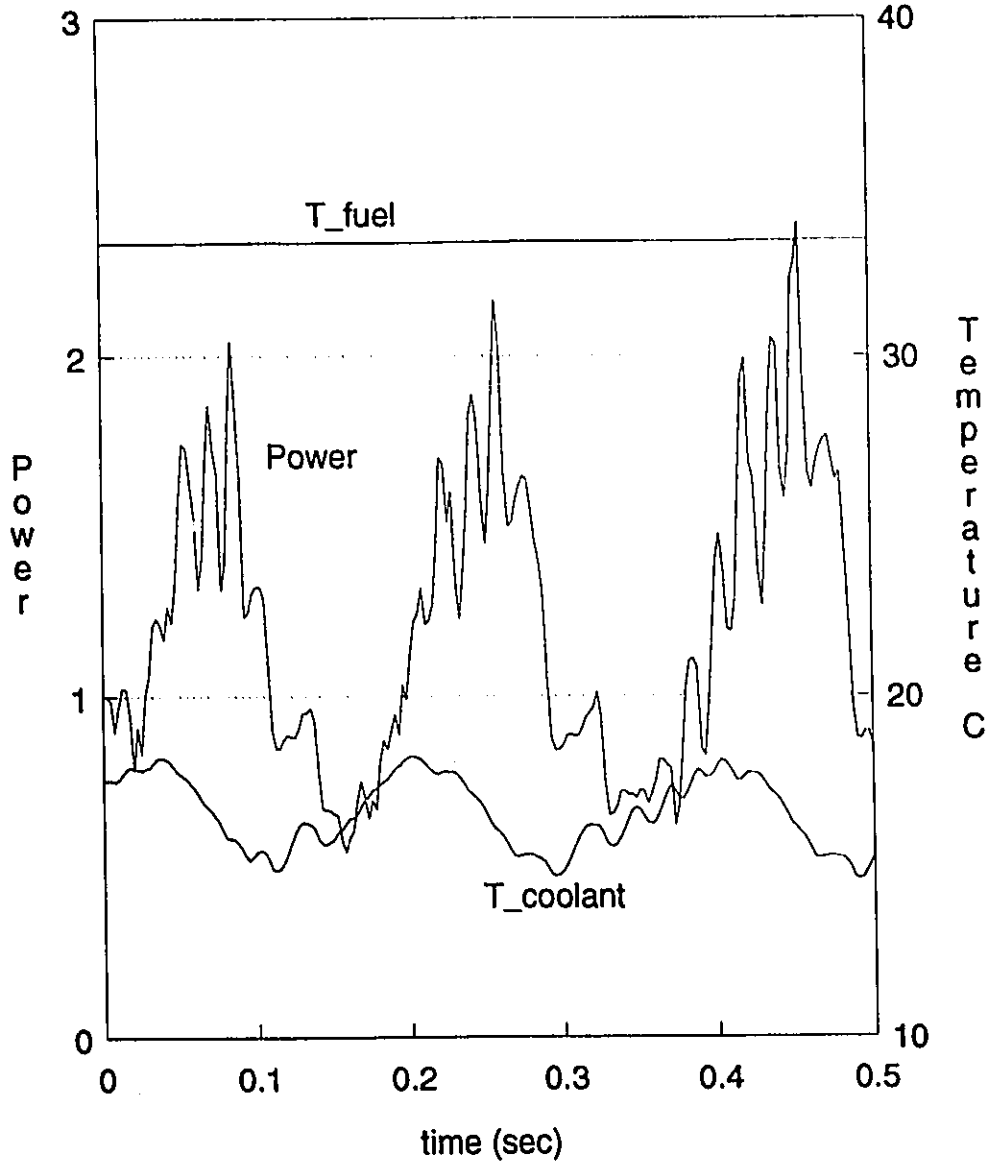

Figure 10.5-11: 0 mk Transient - Power & Tc versus Time (0.5s)

11 Discussion

The objective was to model as accurately as reasonably possible the behavior of a reactor with passive cooling. A pool type reactor with vertical flow tubes cooled by natural convection is the simplest implementation of passive cooling and it is not an uncommon design (SLOWPOKE for example). The dynamics of such a reactor are modeled here.

The simulation is modelling a reactor involving both temporal and spatial effects therefore it is classified as a space-time reactor kinetics simulation. However, this simulation has an integral thermalhydraulic module which is its primary distinguishing feature. The two interacting processes in the model are the space-time neutron behavior, modeled using the time dependent neutron multigroup diffusion equations, and the fuel-coolant temperature behavior, modeled using the conservation equations (mass, momentum & energy) and the rate form of the equation of state. The two physical processes communicate via thermal energy transfer and changes in cross-sections of the materials. The effect that these changes in cross-sections has upon the neutron kinetics is usually referred to in terms of reactivity feedback effects.

The only method that would properly model neutron kinetics - thermalhydraulics interaction in a reactor would be a full core simulation but such a simulation would be impractical and prohibitively expensive. The simpler approach of using a lattice cell calculation can be used to model the dynamics of the two processes and their interaction. Although the thermalhydraulic module could not model coolant voiding, which is responsible for the largest change in the density and hence the nuclear properties in the coolant, the geometry and material properties of the test simulation could be chosen so as to maximize the reactivity effects of the coolant. This included a large coolant to moderator volume ratio and a rather 'hard' neutron spectrum. A 'hard' neutron spectrum is when a relatively large percentage of the neutrons are epithermal (out of the thermal range - 0 to 1 eV). A change in coolant density affects the moderation and the production of thermal neutrons in turn. The thermal fission cross-section is much larger than the fast fission cross-section therefore changes in the thermal neutron population have a direct effect on the reactor power. This effect could be further increased by using larger thermal expansion coefficients for the coolant but only one thermal expansion coefficient was used.

The reactor power is measured by thermal energy production in the fuel. This thermal energy is transferred by conduction to the coolant at the fuel/coolant boundary which the coolant carries away by convective flow. Temperature dependent Doppler broadening of the multigroup cross-sections in the fuel is modeled using thermal feedback coefficients and the cross-sections of the coolant are directly dependent on the coolant density. Doppler broadening in the fuel is a rapid feedback effect and is therefore included as part of the neutronics portion of the simulation. Thermal expansion of the coolant and its effect on cross-sections proceeds comparatively slowly and therefore can be calculated separately. The two components of the simulation proceed with widely differing rates of change. An analysis of this is presented below.

11.1 System Dynamics

The dynamics of the overall system will be an interaction of the two components. An examination of the two systems will show how each behaves and responds to external stimuli. This and a look at the interaction of the two processes will help in the understanding of the behavior of the overall system.

The dynamics of the coolant are only mildly dependent on the dynamics of the neutronics within a normal range of reactor power. The coolant proceeds in its behavior patterns dictated primarily by the coolant properties and flow channel geometry only sluggishly responding to changes in reactor power. The high thermal conductivity of the metallic fuel used causes the fuel temperature to be greatly dependent on the coolant temperature. The net effect of this is that the reactor behavior is dominated by the dynamics of the coolant. The extent to which this is true will of course depend on the reactor geometry and composition including the coolant properties.

11.1.1 Neutron Kinetics

The time dependent neutron multigroup diffusion equations with delayed neutrons are

$$\frac{1}{v_g}\frac{\partial}{\partial t}\Phi_g(\vec{r},t) = D_g\nabla^2\Phi_g(\vec{r},t) - \Sigma_{tg}\Phi_g(\vec{r},t) + \sum_{g'=1}^{G}\Sigma_{sg'g}\Phi_{g'}(\vec{r},t)$$

$$+\chi_{pg}(1-\beta)\sum_{g'=1}^{G}\nu_{g'}\Sigma_{fg'}\Phi_{g'}(\vec{r},t) + \sum_{i=1}^{6}\chi_{ig}\lambda_i C_i(\vec{r},t)$$

$$\frac{\partial}{\partial t}C_i(\vec{r},t) = \beta_i\sum_{g'=1}^{G}\nu_{g'}\Sigma_{fg'}\Phi_{g'}(\vec{r},t) - \lambda_i C_i(\vec{r},t)$$

where Φ_g denotes the group flux and C_i is the *ith* delayed neutron precursor.

The R.H.S. of the rate equation for the group flux will be multiplied by the group velocity, as shown by the first equation above. Assuming that more than one neutron energy group is used and that the result of the R.H.S. of the rate equations does not vary greatly (within an order of magnitude), then the rate of change of the group flux will be highly dependent on the magnitude of the group velocity. This creates a very stiff problem; the rate of change of the fast flux is large with respect to the rate of change of the thermal flux. The solution of the coupled ODEs that arise from this problem require special time integration techniques as a consequence.

The point kinetics equations describe reactor time behavior using a point reactor model (fixed spatial flux distribution). If the flux spatial distribution is changing slowly with respect to the reactor power, these equations can be used to describe the short time behavior of the reactor and will give an indication of what

might be expected in longer time intervals. The point kinetics equations are derived in any nuclear engineering text. Their standard form is

$$\frac{dn(t)}{dt} = \frac{\rho(t) - \beta}{\Lambda} n(t) + \sum_i \lambda_i C_i(t)$$

$$\frac{dC_i(t)}{dt} = \frac{\beta_i}{\Lambda} n(t) - \lambda_i C_i(t) \qquad i = 1, .., 6$$

where $\rho(t)$ is the reactivity, Λ is the mean generation time, $C_i(t)$ is the i th neutron precursor density and λ_i is the associated decay constant, and β_i is the fission yield fraction for the i th precursor ($\beta = \sum \beta_i$). The number of delayed (neutron precursor) groups used here is six.

To examine the exponential behavior of this set of coupled equations, let us assume that the neutron density and the precursor density have an exponential characteristic, $n(t) \approx \exp(\omega t)$ and $C_i(t) \approx \exp(\omega t)$. With substitution and manipulation the *Inhour equation* is derived[107]. This equation may be written as

$$\rho = \omega\Lambda + \beta - \sum_i \frac{\beta_i}{(\omega + \lambda_i)}$$

For very short time periods (a few neutron generations) after a reactivity insertion, the prompt behavior of the neutrons will dominate. It can be shown that the rate of change of power for such a time period is given by: $dP/dt = \rho/\Lambda$. This may be used to estimate the neutron generation time if it is not known. The time constant for the longer time behavior, assuming the reactivity insertion is less than the delayed neutron fraction (not prompt critical), will be a function of the decay constants of the delayed neutron precursors. This can be easily seen in the point kinetics simulation but the influence of thermal and thermalhydraulic feedback effects will be felt by this time in the space-time simulation and therefore will not be as clearly seen.

The neutron generation time is calculated from the cross sections (and is therefore explicitly known) *a priori* for the point kinetics simulation but it is not explicitly known for the general space-time transient simulations. The very short initial time period of several transient simulations was used to calculate estimates of the neutron generation time which can then be compared to the value used in the simulation, if known. For the space-time simulations, *w/o TH* means without thermalhydraulics (the thermalhydraulic variables are constant) and *w TH* means with thermalhydraulics.

---

107 Duderstadt J.J. & L.J. Hamilton, *Nuclear Reactor Analysis*, p.243.

Simulation	ρ (mk)	dP/dt	Λ(μs) (calc)	Λ(μs) (known)
point kinetics	2	195.7	10.22	10.206
space-time (w/o TH)	1	88.65	11.28	unknown
space-time (w TH)	2	~165	12.12	unknown
space-time (w TH)	4	~320	12.5	unknown
space-time (w TH)	8	~640	12.5	unknown

Table 11.1.1-1: Neutron Generation Time

The neutron generation time for the point kinetics is calculated from the material properties and should not vary greatly from the value obtained from the space-time simulations, which it does not. The accuracy of the results for the space-time simulations is somewhat reduced due to the estimates of the rate of change but they are quite consistent, which is desirable.

The values for the neutron generation time as suggested by Henry[108] are from 10^{-3} for a very thermal reactor to 10^{-7} for a very fast reactor. The neutron generation time is a function of the moderation, the less the moderation the shorter the neutron generation time. The value of the neutron generation time that is calculated here is consistent with that for an undermoderated thermal reactor.

The response of the space-time simulation without feedback to a reactivity insertion will be similar to response of the point kinetics simulation. A point kinetics simulation for a 2 mk reactivity insertion is shown in Figure 10.4.2-1.

11.1.2 Fuel Temperature Time Constant

Thermal energy generated in the fuel by fission is transferred by conduction to the surface of the fuel where is it is carried away by the coolant. The temperature of the fuel will be determined by the rate at which heat produced in the fuel can be carried away by the coolant. Thermal conductivity is a measure of heat flow per unit area for a given temperature gradient. The specific heat of the fuel is a measure of the thermal energy (heat) capacity of the fuel. The time constant for the temperature of the fuel will decrease with increasing thermal conductivity and increase with increasing specific heat. Metallic fuel (U_xSi-Al) has a

108 Henry, A.F., *Nuclear Reactor Analysis*, p.306.

very high thermal conductivity when compared to ceramic fuel (UO_2). The time constant for the temperature of metallic fuel is much less than that for ceramic fuel. For a given reactor power, the temperature of metallic fuel will respond faster to a change in coolant temperature than ceramic fuel.

The thermal conduction equation governs fuel temperature behavior,

$$\frac{\partial T}{\partial t} = \alpha \nabla^2 T + \frac{1}{\rho C_p} S(\vec{r}, t)$$

where ($k, \rho, C_p, \alpha = k/\rho C_p$) are the thermal conductivity, density, specific heat and the thermal diffusivity respectively and $S(r,t)$ is the source term. The length of the fuel is much greater than the width, therefore it is assumed to be a conductive slab of infinite length. There is an insulated (reflective) boundary on the left and the coolant boundary on the right. For the purpose of calculating the time constant, it can be assumed that the source term is zero and the wall temperature at the coolant boundary is also zero. The conduction equation now appears as

$$\frac{\partial T}{\partial t} = \alpha \frac{\partial^2 T}{\partial x^2}$$

with boundary conditions

$$\left.\frac{\partial T(x,t)}{\partial t}\right|_{x=0} = 0 \qquad \text{and} \qquad T(x,t)|_{x=a} = 0$$

where a is the width of the slab. Using separation of variables $T(x,t)=G(t)F(x)$, the separated equations are

$$\frac{dG(t)}{dt} = -cG(t)$$

$$\frac{d^2F(x)}{dx^2} = -\frac{c}{\alpha}F(x)$$

where c is the constant of separation. The general solution of the spatial equation is ($m^2 = c/\alpha$),

$$F(x) = A\cos(mx) + B\sin(mx)$$

where the parameters A, B & m are solved for using the boundary conditions,

$$\frac{dF(x=0)}{dx} = 0 \qquad F(x=a) = 0$$

The solution is a modal expansion,

$$F(x) = \sum_{n=0}^{\infty} b_n \cos\left(\frac{(2n+1)\pi}{2a}x\right)$$

and the constant of separation is now $c_n = [(2n+1)\pi/2a]^2\alpha$. The complete solution is a series of decaying exponential functions,

$$T(x,t) = \sum_{n=0}^{\infty} b_n \exp(-c_n t)\cos\left(\frac{(2n+1)\pi}{2a}x\right)$$

The initial conditions can be used to determine the expansion coefficients (b_n) if required.

Of primary interest is the rate of decay of the fundamental mode as this will determine the time constant of the fuel. The decay constant of the fundamental mode is $c_0 = (\pi/2a)^2\alpha$ and the time constant is the reciprocal of the decay constant. The thermal diffusivity of the fuel is *1.388* & *0.01* (cm^2/s) for metallic and ceramic fuel respectively. For fuel of width *1 cm.*, the decay constants are *3.405* & *0.025* (s^{-1}). The time constant for the two types of fuel are therefore *0.2937 (s)* for the metallic fuel and *40.53 (s)* for the ceramic fuel.

The use of metallic fuel is common in small light water reactors, such as SLOWPOKE and MAPLE, and ceramic fuel is used in CANDU reactors which are large heavy water reactors. The reactor simulated here bears greater resemblance to the small light water reactors hence metallic fuel is used. It can be seen that the metallic fuel will respond much faster to changes in coolant temperature than the ceramic fuel. Metallic fuel properties used in the simulation gives the fuel temperature a fast response characteristic with respect to the thermalhydraulics. The fuel temperature has a great effect on the neutronics due to Doppler broadening of the cross sections. The fast response of the fuel temperature to changes in coolant temperature would mean that the dynamics of temperature feedback would be almost totally dominated by the dynamics of the thermalhydraulics and its sluggish time behavior.

11.1.3 Point Kinetics with Fuel Temperature Feedback

The point kinetics model of the reactor can be extended to include several feedback effects. The dynamics of the fuel temperature have been calculated above and are now added to the standard point kinetics model. The resulting model should reflect the behavior of the space-time kinetics simulation with one neutron energy group and no thermalhydraulics. This is similar to the verification done in section 10.4.2.

The point kinetics model is now

$$\frac{dn(t)}{dt} = \frac{\rho(t)-\beta}{\Lambda} n(t) + \sum_i \lambda_i C_i(t) + c_f T_f(t)$$

$$\frac{dC_i(t)}{dt} = \frac{\beta_i}{\Lambda} n(t) - \lambda_i C_i(t) \qquad i = 1,..,6$$

$$\frac{dT_f}{dt} = \omega_f n(t) - \lambda_f T_f(t)$$

where T_f is the normalized fuel temperature, $(\omega_f, \lambda_f, c_f)$ are the fission heat production coefficient, the fuel temperature decay constant and the fuel temperature feedback coefficient respectively. The fuel temperature was normalized to the neutron velocity, $T' = T/v$, in order eliminate the neutron velocity from the above equations. The fuel temperature feedback coefficient is derived from the one group multigroup diffusion equations. The temperature dependence of the fission and absorption cross-sections in the fuel are reflected in the fuel temperature feedback coefficients,

$$c_f = \frac{\partial}{\partial T_f}\left(\frac{\partial \Phi}{\partial t}\right) = v_g\left(\frac{\partial \Sigma_a}{\partial T_f} - \frac{\partial(\nu\Sigma_f)}{\partial T_f}\right)$$

The fission heat production coefficient will be a function of the mean flux levels used in the simulation. The nominal parameters used in the simulation are:
$\beta = 0.008, \Lambda = 12(\mu s), \omega_f = 0.354(\overline{\Phi} = 10^{12}), \lambda_f = 3.405, c_f = -14.98.$

The eigenvalues of the Jacobian for the system were calculated for increasing levels of initial flux, with flux levels of 10^{12}, 10^{13}, and 10^{14} $(n/cm^2\ s)$ corresponding to cases 1, 2 & 3 respectively. These are given in Table 11.1.3-1. The increasing flux level has the effect of increasing the heat production coefficient and consequently the temperature and temperature reactivity by a factor of ten and one hundred for cases 2 & 3 respectively. The initial fuel temperature in case 1 is *0.1039 (°C)* with the reactivity effect calculated to be *-0.0187 mk*. In all cases a reactivity insertion of *4 mk* was applied. As the temperature increases, the temperature reactivity increases causing a leftward shift of all the eigenvalues except the eigenvalue for the prompt behavior (the most negative). The inclusion of fuel temperature feedback increases system stability as would be expected.

Case 1: $\overline{\Phi} = 10^{12}$	Case 2: $\overline{\Phi} = 10^{13}$	Case 3: $\overline{\Phi} = 10^{14}$
-392.802	-392.679	-391.452
-3.701	$-3.609 \pm i0.089$	$-4.121 \pm i0.136$
-3.422	-1.134	-1.208
-1.124	-0.163	-0.176
-0.162	-0.048	-0.055
-0.047	-0.013	-0.014
-0.013	0.165	0.059
0.182		

Table 11.1.3-1: Eigenvalues for Point Kinetics (s^{-1})

There is one positive eigenvalue which is a result of the positive reactivity insertion. For the lowest temperature case, case 1, there is one eigenvalue that is close to the decay constant of the fuel temperature. Interaction between the fuel temperature and the delayed neutron precursors has produced a pair of complex eigenvalues in cases 2 & 3. This would imply that the reactor power would initially increase due to the reactivity insertion and then decrease slightly due to the increase in fuel temperature feedback. The response of the reactor power to a reactivity insertion will be slightly oscillatory as a result, but this effects will be short lived due to its rapid decay.

This is a point reactor model that is useful in elucidating the general effects that fuel temperature reactivity feedback has upon the reactor dynamics. Its accuracy is limited to cases where spatial effects are not that significant. The inclusion of thermalhydraulics in such a model would require that two neutron energy groups be used so that changes in moderation could be modeled. Spatial effects become more significant when thermalhydraulics are involved which would adversely effect the accuracy of the model.

11.1.4 Thermalhydraulics

The fluid used for the coolant is assumed to be single phase and incompressible. The time dependent conservation equations for the thermalhydraulics are mass, momentum and energy (respectively),

$$\frac{\partial \rho}{\partial t} + \vec{\nabla} \cdot \rho \vec{V} = 0$$

$$\rho \frac{\partial \vec{V}}{\partial t} + \rho \vec{V} \cdot \vec{\nabla} \vec{V} = \mu \vec{\nabla}^2 \vec{V} + \rho \vec{g} - \vec{\nabla} P$$

$$\frac{\partial T}{\partial t} + \vec{V} \cdot \vec{\nabla} T = \alpha \nabla^2 T$$

where $\{\rho, \vec{V}, T, P\}$ are the density, velocity, temperature, and pressure respectively and $\{\vec{g}, \alpha, \mu\}$ are gravity, thermal diffusivity[109], and viscosity respectively.

The usual form of the equation of state can simply be written in a form where pressure is a function of density and enthalpy, $P = \pi(\rho, h)$. The rate form of the equation of state

$$\frac{\partial P}{\partial t} = \left(\frac{\partial P}{\partial \rho} \right)_h \frac{\partial \rho}{\partial t} + \left(\frac{\partial P}{\partial h} \right)_\rho \frac{\partial h}{\partial t}$$

can be derived by taking the partial derivative with respect to time of the equation of state. The two partial derivatives of pressure on the right hand side are replaced by constant coefficients. The result,

$$\frac{\partial}{\partial t} P(\vec{r}, t) = G_1 \frac{\partial}{\partial t} \rho(\vec{r}, t) + G_2 \frac{\partial}{\partial t} h(\vec{r}, t)$$

is the time rate of change of the pressure described as a linear combination of the rate of change the mass and enthalpy. For incompressible fluids, the coefficient G_1 is much larger than G_2 ($G_1 >> G_2$), hence the approximation ($G_2 \approx 0$) is made. The conservation of mass equation is used to substitute the rate of change of density so that the equation of state now appears as

$$\frac{\partial}{\partial t} P(\vec{r}, t) = -G_1 \vec{\nabla} \cdot \rho \vec{V}$$

The density of the fluid is temperature dependent, and is calculated using the equation[110]

$$\rho = \bar{\rho} - \bar{\rho} \beta (T - \bar{T})$$

where $\bar{\rho}$ is the density at some reference temperature, $\bar{T}$, and β is the coefficient of volumetric expansion.

11.1.4.1 Vertical Flow Channel

The problem at hand is a vertical flow channel of length *"b"* and width *"a"*, where the coolant enters the bottom and is heated by the walls as it flows upward driven by natural convection. Several assumptions

---

109 Bird, Stewart & Lightfoot, (1960) pp 246

110 Bird, Stewart & Lightfoot, (1960) p 299

were made when analyzing the dynamics of such a system. The fluid is assumed incompressible ($\vec{\nabla} \cdot \vec{V} = 0$) and the flow velocity in the horizontal (X) direction is negligible with respect to the vertical (Y) direction. This implies that the flow velocity is a function of time and X only ($V_x = 0, \quad \therefore \quad \partial V_y/dy = 0$). Natural convection driven by thermal expansion of the fluid is assumed to be the only forcing term in the momentum equation (see section 4.1.2) therefore,

$$\rho\vec{g} - \vec{\nabla}P = -\vec{g}\beta\bar{\rho}(T - \bar{T})$$

The reference temperature can be assumed zero without loss of generality ($\bar{T} = 0$) and the reference density is assumed equal to one ($\bar{\rho} = 1$). The Navier-Stokes (momentum) equation now becomes

$$\frac{\partial v_y}{\partial t} = g\beta T + \mu\frac{\partial^2}{\partial x^2}v_y(x,t)$$

(Gravity is in the opposite direction to the direction of flow, hence the apparent sign change.)

Flow Profile in the X direction

Further assumptions are that the velocity profile in the X direction is a constant and temperature is a function of Y only. In reality the velocity profile must develop at the intake and the fluid is warmer near the wall but for a long thin tube these assumptions are reasonable. The steady state velocity profile is found by solving the equation

$$\frac{d^2v_y(x)}{dx^2} = -\frac{g\beta T}{\mu}$$

Integrating over the width of the channel ($0 \le x \le a$) and using zero flow boundary conditions at the walls, the flow velocity in the Y direction is

$$v_y(x) = \frac{g\beta T}{2\mu}(a - x)x$$

If this profile is valid throughout the length of the channel, the mean velocity (with respect to X) may be used in the time dependent equation

$$\bar{v}_y = \frac{1}{a}\int_0^a v_y(x)dx$$

$$= \left(\frac{g\beta T}{12\mu}\right)a^2$$

This effectively assumes a separated solution (t and X) where the solution in the X direction is a parabola. The time dependent velocity equation is now integrated over the width of the channel. The viscous drag term ($\mu\partial^2v_y/\partial x^2$) is integrated and replaced by a term that is a linear function of the mean flow velocity ($-\mu_e\bar{v}_y$) where $\mu_e = 12\mu/a^2$ is the effective viscosity coefficient for a given flow profile.

Heat Transfer

Convective heat transfer is assumed to be the dominant method of thermal energy transfer so that thermal conduction can be ignored ($k\nabla^2T \approx 0$). An additional heat source is convective heat transfer from the heated walls ($q = h(T_w - T)$), where T_w is the wall temperature.

Solution of the Time Dependent Equations

The conservation of momentum and energy equations now appear as

$$\frac{\partial V}{\partial t} = -\mu_e V + g\beta T$$

$$\frac{\partial T}{\partial t} + V\frac{\partial T}{\partial y} = \frac{h}{C_p}(T_w - T)$$

where V is the mean velocity $\bar{v}_y$. The momentum equation may be easily integrated in the vertical direction and divided by the length of the channel. The temperature in the first equation above is replaced by the mean fluid temperature ($\bar{T}$) over the length of the channel resulting in the equations

$$\frac{dV}{dt} = -\mu_e V + g\beta\bar{T}$$

$$\bar{T}(t) = \frac{1}{b}\int_0^b T(y,t)dy$$

A change of variables ($T'=T_w-T$) is used so that the wall temperature is now zero, making the fluid temperature equation homogeneous. The separation of variables (in y and t) is used for this equation ($T(y,t)=F(y)G(t)$). The separated equation is

$$\frac{1}{V(t)G(t)}\frac{dG(t)}{dt} + \frac{h}{C_p V(t)} = -\frac{1}{F(y)}\frac{dF(y)}{dy} = k$$

where k is the constant of separation.

The equation for the spatial (Y) component is

$$\frac{dF(y)}{dy} = -kF(y)$$

The solution of the spatial component is relatively simple,

$$F(y) = e^{-ky}F_0$$

where F_0 is the inlet temperature. The mean temperature over the length of the channel can be calculated,

$$\bar{T}(t) = G(t)\bar{F}$$

$$\bar{F} = \frac{1}{b}\int_0^b F(y)dy = \frac{F_0}{bk}(1 - e^{-kb})$$

The rate equations for the time dependent component of temperature and velocity are now

$$\frac{dG(t)}{dt} = \left(kV(t) - \frac{h}{C_p}\right)G(t)$$

$$\frac{dV}{dt} = -\mu_e V + g\beta[G(t)\bar{F} + T_w]$$

The Jacobian for this system is

$$\frac{\partial \dot{V}}{\partial V} = \mu_\epsilon \qquad \frac{\partial \dot{V}}{\partial G} = g\beta\tilde{F}$$

$$\frac{\partial \dot{G}}{\partial V} = kG(t) \qquad \frac{\partial \dot{G}}{\partial G} = kV - \frac{h}{C_p}$$

or

$$J = \begin{bmatrix} -\mu_\epsilon & g\beta\tilde{F} \\ kG(t) & kV - \dfrac{h}{C_p} \end{bmatrix}$$

Stability at the Fixed Point

A fixed point is the point in phase space (V^*,T^*) where the two rate equations are equal to zero. The stability of the fixed point is determined by the eigenvalues of the Jacobian matrix at the fixed point. (The stability of fixed points is discussed in more detail in chapter 5). The time dependent component of temperature is assumed equal to one $(G(t)=1)$ therefore $T^*(y)=F^*(y)$ is the temperature at the fixed point and V^* and velocity at the fixed point. The equation for the time rate of change of temperature is equal to zero.

$$\frac{dG(t)}{dt} = \left(kV^* - \frac{h}{C_p}\right)G(t) = 0$$

$$\Rightarrow k = \frac{h}{C_p V^*}$$

The constant of separation at the fixed point is therefore $k = h/(C_p V^*)$.

The Jacobian at the fixed point is now

$$J = \begin{bmatrix} -\mu_\epsilon & g\beta\tilde{T}^* \\ k & 0 \end{bmatrix}$$

The eigenvalues of the this matrix (the roots of the characteristic equation)

$$a(s) = det(sI - J) = s^2 + \mu_\epsilon s - kg\beta\tilde{T}^* \quad = 0$$

determine the stability of the fixed points. The roots of the characteristic equation must less than zero for stability,

$$s_{1,2} = \frac{-\mu_\epsilon \pm \sqrt{\mu_\epsilon^2 + 4kg\beta\tilde{T}^*}}{2} \quad < 0$$

The important root is the most positive root,

$$s_1 = \frac{-\mu_\epsilon + \sqrt{\mu_\epsilon^2 + 4kg\beta\tilde{T}^*}}{2}$$

Due to the change of variable, the temperature is always negative but may approach zero. It can be seen that this root is always negative and possibly complex which suggests that the system is stable. The roots become purely imaginary as the viscosity approaches zero and become more negative (and possibly real) as the viscosity increases.

Calculation of Flow Stability at the Fixed Points

From the flow simulation results, the flow velocity and fluid temperature at the fixed points are estimated to be *2.0 (cm/s)* and *3°C* with a wall temperature of *10°C*. The other data that are used in the simulation: thermal expansion coefficient $\beta = 6.1x10^{-4}$ *(1/°C)*, viscosity $\mu = 5.47x10^{-3}$ (poise or *g/cm.s*), specific heat C_p=*4.2 (J/g°C)*, channel width *a=1.0 (cm)*, and the total channel length is *24 (cm)*. The channel is in two sections, the lower section (b_1=*8 cm.*) has a heated wall and the upper section (b_2=*16 cm.*) has insulated walls. The heat transfer coefficient, *h*, is unknown but can be calculated using the mean temperature at the fixed point. The mean temperature at the fixed point for a channel in two sections is described by

$$\tilde{T}^* = \frac{T_0}{b_1 + b_2}\left[\frac{1}{k}\left(1 - e^{-kb_1}\right) + b_2 e^{-kb_1}\right]$$

where the temperatures after the change of variables are: $\tilde{T}^* = -7°C$ and $T_0 = -10°C$. The mean temperature is known but the constant of separation is not, therefore this equation is used to solve for the constant of separation, *k=0.0548 (1/cm.)* which gives a heat transfer coefficient of *h=0.46 (W/°C cm²)*. The effective viscosity is $\mu_e = 12\mu/a^2 = 0.0656(g/cm^3 s)$.

The Jacobian is now

$$J = \begin{bmatrix} -0.06564 & -4.1846 \\ 0.0548 & 0 \end{bmatrix}$$

The roots of the characteristic equation are $(-0.0328 \pm i0.4777)$ which implies a stable fixed point. This sort of fixed point is a referred to as a spiral sink where the system will spiral into the fixed point in phase space. The behavior of such a system would be convergent oscillations in time.

This analysis would indicate that the flow is stable at the fixed point and as such the flow would be expected to settle to some fixed point in phase space. The simulation exhibits generally stable behavior with a small limit cycle about a fixed point. This analysis, while being adequate for the overall system behavior, is probably not adequate for detailed system behavior due to the number of assumptions that were made. Only one wall is heated and it is very unlikely that the flow profile is a parabolic shape. The small real component of the roots would indicate slow convergence and the large imaginary component of the roots would indicate oscillatory behavior. A reasonable conclusion is that the system is stable in the large but has oscillatory tendencies.

11.1.5 Coupling of the Two Components

The two components of the simulation, neutron kinetics and thermalhydraulics, proceed at widely differing rates of change in time, the neutron kinetics proceeding at a much greater rate than the thermalhydraulics. They do influence each other by different methods. The production of thermal energy into the system is the method by which the neutronics drives the thermalhydraulics. Temperature variation of the cross sections in the fuel and coolant is the method by which the thermalhydraulics influences the neutron kinetics. Changes in temperature as a result of variations in power are rather sluggish due to the heat capacity of the system. Changes in the power due to variations in the cross sections will be almost immediate in the time frame of the thermalhydraulics. The dynamics of the combined system will have properties of both systems, but the relative insensitivity of the thermalhydraulics and the large influence it has on the neutron kinetics will cause it to dominate, assuming the reactor is not prompt critical.

Neutron moderation is the process by which fast neutrons lose energy through scattering and become thermalized. The rate at which this occurs is determined by the scattering cross-section of the moderator. Thermal fission (fission by thermal neutrons) is the dominant type of fission (as opposed to fast fission) in thermal reactors, as their name implies. The fuel has typically a much larger thermal fission cross section than fast fission cross section. Therefore changes in the thermal neutron population and hence the reactor power will be greatly influenced by changes in the moderator cross sections.

When the coolant is also (at least partially) a moderator, changes in coolant density will effect the scattering cross section of the coolant and hence neutron thermalization. The effect that this will have on the reactor power is dependent on the degree to which the coolant is responsible for neutron moderation. Reactors that have a large moderator to coolant volume ratio, such as a CANDU reactor, are referred to as over-moderated. This will reduce the effect that changes in coolant density will have on the thermal neutron population. In under-moderated reactors, the converse is true. The geometry chosen for the test reactor simulation used here is that of an under-moderated reactor so as to maximize the effect that changes in the coolant density have upon the thermal neutron population and hence the reactor power. This serves to accentuate the influence of the thermalhydraulics upon the neutronics.

11.2 Software Design

The philosophy used in designing the software was to structure the program in such a way that the coupling of the two components, neutron kinetics and thermalhydraulics, could be done with the minimum of effort and trouble. An integrated approach was required from the outset if this was to be achieved. The two competing practical constraints in the software design is the minimization of storage and computational requirements. Much attention had to be paid to the solution algorithms used to solve the resultant system of ODEs and their efficient implementation in software as this portion of the simulation would be the most demanding on computer resources.

11.2.1 General Program Structure

The problems encountered in the solution of the neutron kinetics problem are significantly different from those encountered in the solution of the thermalhydraulics problem. The stiffness of the Jacobian matrix (the wide range of eigenvalues) is the main difficulty encountered in the solution of the neutron kinetics problem. The group velocities were set equal to one in order to reduce the stiffness of the Jacobian for static calculations but such methods could not be used for the transient calculations as it would effect the dynamics of the neutronics. The nonlinearity of the governing equations was the main consideration in the solution of the thermalhydraulics problem. These differences can be stated mathematically by saying that the structure of the two Jacobians is significantly different hence requiring different methods to solve the resultant ODEs.

The parts of the simulations are solved sequentially, first the neutronic simulation and then the thermalhydraulic simulation, but this is arbitrary. The two simulations could be solved simultaneously, but this would add excessive complexity and reduce the efficiency of the simulation. The band structure of the resultant (combined) Jacobian matrix would increase significantly in complexity thereby reducing the possibility of finding efficient methods for the time integration. The coupling between the two simulations, as they are solved sequentially now, essentially occurs at every time step. There will be a small loss of accuracy associated with this approach, but its significance will depend on the time steps that are used. For practical purposes, the loss of accuracy is considered to be negligible.

The unsteady behavior of the thermalhydraulic simulation created convergence problems in the static calculations for the combined simulation. The Chebychev acceleration routine had difficulty due to the constantly changing cross sectional properties which would interfere with its calculation of eigenvalues for the purpose of acceleration and cause it to not accelerate properly. The *K-effective* calculation would be still useful as an indication of criticality but the accuracy would not be very good if the simulation did not accelerate properly. Manual methods would then be required for a more accurate determination of the *K-effective* which was mostly just an inconvenience. The manual method of criticality determination would mean manually adjusting *K-effective* until the power would remain relatively constant over a time interval in a transient calculation.

The coupling between the neutronics and the thermalhydraulics was an important aspect of the simulation. Both the thermalhydraulic and neutronic equations are represented using finite differences in space, as opposed to a nodal representation. The thermal feedforward between the neutron-thermal conduction simulation and the thermalhydraulic simulation comes in the form of boundary conditions, continuity of temperature and thermal flux, instead of lumped parameter feedback. This has the effect that heat transfer between the fuel and the coolant occurs at every spatial point along the mutual boundary. This is much more direct and will more closely reflect reality but at the price of complexity and computational time.

11.2.2 Storage Requirements

Assume, for example, that the simulation we had in mind had 10 spatial points in Y direction, 10 spatial points in the X direction in all three regions (see Section 7.1) and three neutron energy groups. This would mean that one energy group of the neutron flux data vector would contain 300 data points. The entire first data vector would contain 1500 data points. The associated Jacobian matrix would have 13 bands (this is dependent on the number of neutron energy groups). The thermalhydraulic data vector would contain 400 data points and the associated Jacobian would have 16 (a constant) bands.

In order to minimize storage requirements, only the nonzero bands of the matrices are stored. Both conjugate gradient methods and the Gauss-Seidel iterative method of solution of the linear problem can be performed while maintaining the matrix to be inverted in a banded storage format. LU decomposition methods require that the matrix be stored using full storage. Due to the wide bandwidth of the matrices involved, the storage requirements of the banded LU decomposition method are not much less that of the full storage.

If full storage was used for the matrices in the above example instead of banded storage, at least 2.5 million storage locations would be required (if double precision variables were used at 8 bytes per variable, this would mean 20 megabytes of storage). If banded storage, as outlined here, is used less than 8 thousand storage locations (64 kilobytes) would be required. If LINPACK style banded storage was used instead of full storage, only 0.1 million storage locations would be saved due to the wide band structure of the matrices involved. Gauss-Seidel iterative matrix inversion does not cause any of the zero bands between nonzero bands to fill up as do LU decomposition methods. That is to say, if the matrix to be inverted has five bands with the outer two bands quite widely separated from the central three bands, as would arise with a Laplacian in two dimensions, the zero bands between the outer bands and the central bands would fill up during decomposition. If only the nonzero bands of the Jacobian matrices are stored, many fewer storage locations are required if the appropriate solver routine, modified to handle matrices stored in this fashion, is used to solve the final linear system problem.

11.2.3 Linear Algebraic Problem

The solution of the linear system problem is a very important part of this and most simulation programs. This point can not be over emphasized. The time required to solve the linear problem is the majority of the program execution time, therefore a great deal of effort must be applied to solving this problem efficiently. Several methods of solution of the linear algebraic problem were presented: LU decomposition, block tri-diagonal LU decomposition, the generalized conjugate gradient method, the Gauss-Seidel iterative method, and the successive over-relaxation method. All methods have their merits and their liabilities.

LU decomposition will always give good results, assuming the matrix is nonsingular, but it requires a great deal of storage space and computational effort. If little is known about the matrix to be inverted and/or it does not have a regular structure such as a banded structure that can be exploited, LU decomposition is a good choice. If the sparsity of the matrix is such that all the nonzero elements are clustered close to the diagonal, the matrix is said to have a small bandwidth. A banded LU decomposition routine would be a good choice for such a matrix. The spatial finite differencing of PDEs generates some matrix structures for which special LU decomposition algorithms do exist. There are algorithms for tridiagonal matrices, common in one dimensional problems, and block tridiagonal matrices, which arise from Laplacian terms in two or more dimensions. If the matrix is diagonally dominant though, an iterative method might be a better choice, but otherwise these special algorithms are useful for such problems.

The Gauss-Seidel iterative method can use the method of banded storage discussed in the previous section. The limitations of the method are of course that the matrix must be diagonally dominant. In nonlinear problems, such as the thermalhydraulic problem, this is not always the case. The diagonal dominance of the matrix to be inverted is affected by the time step size that is used in the simulation, this places an upper bound on the step size. If relatively small time steps are used, the GS method will produce the fastest execution times by a significant margin. However, if large time steps are desired, a block tridiagonal solver will be a better choice. In a linear problem, such as thermal conduction, the matrix will almost always be diagonally dominant and hence the GS method is the best method.

The SOR method is a variation of the GS method and performed appreciably better that the GS method under test situations. It converged up to 3 time faster than the GS method for some problems. However, the GS method was found to perform better under a wider variety of conditions. When few iterations by either method were required, little advantage is gained using the SOR method. At the other end of the spectrum, when a very large number of iterations were required (the iteration was just able converge), the SOR method displayed erratic behavior. The SOR method can be said to have a narrow tuning curve when compared to the GS method. The SOR method was compared against the GS method in solving the thermalhydraulic simulation problem. The GS method was found to perform better under this more realistic situation, it had fewer problems converging and would have the lower execution times due to its simplicity. The inversion of matrices that were more consistently diagonally dominant, the SOR method was faster.

The generalized conjugate gradient method can use the banded storage method as well. It does not have the limitations on matrix properties that the GS method has, but it has a limit on system size. The convergence properties of the method degrades as the system size increases. It would be very useful for large stiff sparse systems if the convergence properties could be improved.

Some numerical tests were run to compare three methods that based on the conjugate gradient method (Tables 10.1-5 to 10.1-8), the (classical) conjugate gradient (CG) method, the boosted conjugate gradient method (BCG), and the generalized conjugate gradient (GCG) method. These tests (as well as others) indi-

cate that the CG and the BCG methods perform well on symmetric positive definite (SPD) matrices. Any loss of symmetry, however, will seriously degrade the performance of these algorithms. The loss of symmetry did have a significantly negative effect on the GCG method. As the system dimension increased, the performance of all the algorithms diminished. This is the major drawback to the use of these methods; even though the number of iterations was equal to the system dimension, the performance would decrease with increasing system dimension. The best candidate of these three methods for asymmetric matrices was the GCG method, but it will require some acceleration to become a viable method.

Comparison of Solution Methods

Figure 10.1-1 compares the direct and iterative matrix inversion methods for increasing matrix sizes. The matrix (problem #3) used for the comparison is characteristic of the matrices that would be solved during a simulation. Error control in the time integration and not the matrix inversion method is the limiting factor for the time step size in a simulation. The smaller the time step size, the more diagonally dominant is the matrix and, hence the more favorable are the iterative methods. The product of the block size and the number of blocks gives the total dimension of the system. As expected, the banded methods performed much better than the methods that ignored matrix structure. It can be seen that the banded iterative methods perform better, with respect to the banded direct methods, as the problem becomes larger with the Gauss-Seidel method with successive over relaxation (GSSOR) emerging as the clear favorite.

Figure 10.1-2 compares all the banded methods. The matrix that is inverted (problem #1) is not very favorable to the iterative methods. The intention was to handicap the iterative methods in the comparison. In spite of the handicap, the banded GSSOR method posted the best time as the problem became sufficiently large but, not by a significant margin. The iterative and semi-iterative methods require less storage space than the direct methods. If the matrix to be inverted is not diagonally dominant, is symmetric positive definite (SPD), the conjugate gradient method becomes a viable alternative to the direct methods, especially if storage requirements are at a premium. The generalized conjugate gradient method is not very competitive in Figure 10.1-2 but the conjugate gradient was reasonable.

Machine and Compiler Considerations

Table 10.1-4 compares the CPU times for the C language versus assembly language versions of several solver methods. The execution time of a high level routine will depend on the compiler that is used. The execution time of the assembly language routines could be considered the theoretical minimum execution time of that routine. These results would imply that the iterative methods would benefit to a greater extent than the direct methods if a better compiler was used.

The best routine to use may also be machine dependent, especially if the computer is an array processor. Routines that are more *vectorizable*[111] are better suited for use in array processors, such as the Intel 80860 and all super computers. Gauss-Seidel iterative methods use immediately previous results (i.e. X_{i-1}) when

111 Vectorizable routines can be expressed as a series of matrix and vector operations.

computing a result (i.e. X_i). This would cause significant pipeline penalties if this method were to be used in an array processor. Other iterative methods, Gauss-Jordan for example, fair somewhat better. The direct methods would benefit significantly if properly vectorized but, the semi-iterative methods would benefit the most. The more recent microprocessors (Intel 486, Motorola 68040) are highly pipelined processors and these considerations become more significant.

11.2.4 Coding of the Simulation

The simulation was written in C for the UNIX operating system. The machines used for code development and execution are: an IBM Model 80-111 running Xenix/386 (i386 - 20 MHz), a Multiflow Trace running (Multiflow) UNIX, an IBM RS6000 running AIX and an Intergraph workstation running UNIX to a small extent. The ease with which the code was transferred to the various machines attests to the portability of the code and is one of the major strengths of the UNIX operating system. The simulation was written in C because of the flexibility of this programming language and its suitability for use in a UNIX environment.

Assembly language versions of some of the important mathematical subroutines are available for 386/387 machines. A notable assembly language subroutine is the Gauss-Seidel banded linear system solver. The assembly language version of this subroutine was found to run 1.8 times faster that the equivalent subroutine in C. The modular nature of the code allows for any routine, in particular the mathematical software, to be replaced a faster version should such a thing be available.

One of the advantages of the C language is dynamic memory allocation. The simulation program reads the input data file and then calculates the amount of memory required. This memory is then requested using dynamic memory allocation. If the memory requested is too large, (the problem is too large for the machine) the simulation will simply terminate with an error message. This allows the size of the executable file to be independent of the size of the simulation. ANSI FORTRAN (77) has no provisions for dynamic memory allocation but it has been implemented on certain custom machines.

Another of the advantages of the C language is the ability to manipulate pointers with relative ease. Pointers, as the name implies, point to locations in memory. When a subroutine is called, a pointer that points to the beginning of an array is passed as a parameter to the subroutine (the same thing occurs in every programming language). A subroutine written using the FORTRAN language needs to know all but the last dimension of a multidimensional array. If for example the array is two dimensional, the FORTRAN subroutine must know the first dimension. When that address of an array element is calculated, the first subscript (minus one) is multiplied by the first dimension and then added to the second dimension to find the location in memory. This requires the use of integer multiplication. The C language allows the use of arrays of pointers. Assume again that the array is two dimensional. We would have an array of pointers that point to the beginning of each row of the array. To find the memory location of an array element, we use the first subscript to locate the appropriate pointer, in the array of pointers, for that row. The second subscript, along with that pointer, are used to find the location in memory. Integer multiplication is not

required, but some extra storage is required for the array of pointers, which is quite small. It was found that this method of addressing two dimensional arrays is about 10% faster that a method that requires integer multiplication.

Another advantage of pointer manipulation is the subdivision of arrays. After the memory allocation section of the main routine, many of the arrays are subdivided into the component parts with relative ease.

11.3 Thermalhydraulics

The thermalhydraulic simulation software had to be benchmarked before it could be used with reasonable confidence. A review paper by De Vahl Davis & Jones (1983) provided a good steady state benchmark. The problem is a unit convective cell with a hot wall on the left, a cold wall on the right and insulated top and bottom walls. Comparison of the simulation with the published results demonstrated good agreement for a sufficiently fine mesh. Vertical flow tubes, such as those used in the reactor simulation, are an appreciably different problem and this needed to be examined separately before it could be used in the combined simulation. These two problems are discussed below.

11.3.1 Benchmarking the Thermalhydraulic Simulation

In general, the algorithm had all the desired properties: stability, convergence and consistency for a wide range of parameters. The range of the parameter G_t for which the simulation would converge to the desired solution proved to be unbounded, although it was sensitive to the time step depending on the Rayleigh number. The higher the Rayleigh number, the more violent the flow will be and more violent flows require smaller time steps. The set of equations used here was more general than that used by De Vahl Davis & Jones. The parameters of the model were adjusted to coincide with the free convection problem of interest. The results concurred with De Vahl Davis' conclusion that the use of a course mesh with high Rayleigh numbers gives unreliable results. Better answers are possible with the use of finer mesh sizes, but at increasing computational expense.

In the section on the benchmark equations, it was stated that an upper limit of one exists on the coefficient of thermal expansion, $\beta < 1$, but it was found that 0.01 was a better upper limit. Gravity was chosen to be sufficiently large so that the calculated value of β, which is proportional to the Rayleigh number (Section 4.1.3), was in the desired range. The numerical results given in Table 10.2-1 show good agreement with the published results for a sufficiently fine mesh.

The simulation was intended for use as a transient simulation, but obviously it can be used to determine a steady state solution if allowed to run until convergence, as it was used here. One assumption made by De Vahl Davis & Jones is that one correct steady state solution exists. It is well known that the Navier Stokes

equations are highly nonlinear. Linear systems have one correct steady state solution but, nonlinear systems may have several or they may have none. The system may establish itself in a limit cycle where the simulation never converges but remains bounded. "It is well known that nonlinear systems possess limit sets other than fixed points; for example closed or periodic orbits frequently occur"[112]. While convergence was achieved for the various values of the Rayleigh number and it did appear that only one solution did exist for this problem, all results must be treated as suspect, especially for the high Rayleigh numbers.

The pressure field is adjusted, using the rate form of the equation of state, to reduce the mass error. If the pressure field is properly adjusted, the steady state solution will have a very small mass error. The variance of the mass error is the criterion that was used to determine the *goodness* of the solution. If the mean of the mass error is not zero, then there is an imbalance between what is entering the cavity and what is leaving. All fluids have a certain compressibility, therefore in a transient situation some mass error in any one cell is expected. The reciprocal of the parameter $G_1 = dP/d\rho$ is a measure of a fluid's compressibility.

The more incompressible a fluid is, the larger the value of G_1 and the quicker the pressure response to changes in mass flow rate, in other words, the greater the acoustic velocity. The parameter G_1 is approximately the square of the speed of sound. As the acoustic velocity increases, smaller time steps are required to follow the changes in pressure. If the simulation is solved implicitly in time, larger time steps may be possible without encountering numerical instabilities, although the detailed changes in pressure and flow rates will be lost. The simulation dynamics become more violent as the Rayleigh number increases, requiring smaller time steps for stable convergence.

Figure 10.2-1 shows how heat transfer within the fluid changes with time. Initially all the heat transfer is from the hot wall to the fluid. The temperature of the fluid rises and heat transfer from the fluid to the cold wall increases until eventually heat transfer at the end walls balances. The time required for heat transfer in the x direction to balance will be a function of G_1. The larger the value of G_1, the quicker the fluid velocity will approach its steady state value (as driven by pressure). It can be seen that the heat transfer in the x direction balances quite quickly for the value of G_1 used. De Vahl Davis & Jones use the Nusselt number Nu as an indication of the heat transfer in their comparison exercise. The more complete expression $Q(x)$ will (theoretically) achieve a balance across all planes parallel to the hot and cold walls and was therefore used here. De Vahl Davis does however, use this expression, referred to as Nu_x, in the benchmark solution.

Figure 10.2-2 shows that G_1 has a great affect on the mass error variance, as would be expected. One can draw the conclusion that using a large value of G_1 is desirable for variance reduction when a steady state result is desired; larger corrections are made to the pressure field for errors in mass conservation, hence the pressure field will converge more rapidly. There seems to be no optimal value for G_1 which minimizes the

112 Guckenheimer & Holmes, pp 15 (1983)

variance, instead the variance monotonically decreases as G_l is increased. The variance could be made arbitrarily small by using increasing values of G_l. Transient simulations will, of course, use the true value of G_l and the pressure field will converge at the rate determined by the fluid properties.

The true value of G_l is quite large for water, which would force rapid convergence for steady state problems. If a transient simulation is run for an incompressible fluid, the variance will be significant for only the fastest of transients. The simulation should be benchmarked under transient conditions, although this is more difficult. Future work should determine whether the simulation does indeed converge to a steady state condition or just establishes itself in a limit cycle.

The velocity vector plots for the converged solutions for increasing Rayleigh numbers (10^3 to 10^6) are given in figures 10.2.3-1 to 10.2.3-4. It can be seen that the area of high flow progressively moves toward the outer walls as the Rayleigh number increases. The maximum velocity increases quite dramatically with increasing Rayleigh number as shown in Table 10.2-1. This demonstrates that higher Rayleigh numbers, which are indicative of higher thermal expansion coefficients, induce more violent flow behavior.

The advantage of this algorithm is its simplicity which affords a more intuitive grasp of the fluid behavior. The derivation and execution are quite straight-forward. The use of the compressibility parameter, G_l, allows the simulation of fluids with different compressibilities, and should make for more physically realistic pressures under transient situations.

A number of matrix inversion routines were tried. The best solver combination proved to be the use of the Block Tridiagonal method for the velocity/pressure problem, and either the Gauss-Seidel method or the Block Tridiagonal method for the temperature problem. The Gauss-Seidel method had convergence problems for larger time steps when used on the velocity/pressure portion of the problem due to its nonlinear nature. The Block Tridiagonal method had no such convergence problems and could exploit the matrix structure of the velocity problem therefore making for a robust and efficient algorithm. The nonlinear nature of the temperature problem was less pronounced therefore allowing the use of the Gauss-Seidel solver if desired, although the differences in execution time between the two methods was negligible. This choice of solver routines provided good numerical efficiency and stability of the overall simulation.

11.3.2 Convective Flow Through a Vertical Channel

The thermalhydraulic problem in the complete simulation is convective flow through a vertical channel. Separate thermalhydraulic simulations were run in order to explore the behavior of convective flow using this geometry and to determine the best boundary conditions to use. The initial boundary conditions used were a hot wall on the left, a cold wall on the right and constant hydrostatic pressure boundaries at the top and bottom. Examination of the time behavior and the velocity vector plots showed complex patterns of flow without much structure. It was reasoned that an insulated extension to the top of the channel was

required in order for the flow to "settle down" before encountering the constant pressure boundary at the outlet. Without the vertical extension, the flow would show the expected general behavior (bulk movement upwards) but no noticeably regular behavior patterns would develop. With the extension and a sufficiently fine mesh, regular behavior patterns, sometimes in the form of a limit cycle, did develop.

The flow behavior was chaotic in nature. The simulation showed little indication of converging to a steady state solution however it was bounded. The geometry of the problem had great effect on the behavior of the fluid as was shown in two series of simulations. Both simulations used 8 points in the *X* direction with a mesh spacing of 0.125 (cm). Both simulations used the same number of points in the *Y* direction, 8 point for heated wall section and 16 points for the riser section, but differed in the mesh spacing. The variables plotted are the mean velocity in the *X* & *Y* directions and the mean coolant temperature versus time. When a finer mesh (1.0 cm.) was used, the flow would develop into a regular pattern as shown in Figure 10.3-1. A phase plot of the average vertical velocity versus the coolant temperature is given in Figure 10.3-2, which clearly shows limit cycle behavior. When a course mesh (10.0 cm.) was used, the flow showed general trends but did not develop into any regular patterns as shown in Figure 10.3-3. The course mesh (long tube) simulation is an indication of the type of behavior that may be expected in the combined simulation.

11.4 The Neutronic Simulation

The neutronics section of the simulation (without thermalhydraulics) is in two configurations: static and transient. The results of a one-dimensional one-group analytical calculation were used to verify the results of the static calculation and comparison with the results of a point kinetics simulation was used to verify the transient calculation. The Jacobian for the neutronic simulation becomes very "stiff", ie. there is a wide variation in the eigenvalues of the matrix, when the group velocities are applied in the transient simulation. Therefore, all the group velocities are set equal to some constant, usually 1.0, when a static calculation is desired. The stiffness problem for an eigenvalue (*K-effective*) calculation is thus eliminated allowing the use of the Gauss-Seidel method for the solution of the linear algebraic problem. This method significantly reduces computational cost. For static calculations, Chebychev polynomial acceleration is also used to accelerate the convergence of *K-effective*. The matrix inversion problem encountered in transient calculations usually requires the use of a direct solver method, such as LU decomposition, which significantly increases the computational effort for one iteration, but it allows the use of larger time steps reducing the overall computational effort. Two forms of the time integration routine, each using a different matrix inversion method, are available as a consequence. This allows some flexibility in methods used to solve the neutronics problem.

The results with and without fuel temperature feedback are given. The thermal properties of U_3SiAl are used for the fuel. This material has a high thermal conductivity therefore, the mean fuel temperature is only slightly above the coolant temperature.

Static Calculations

One dimensional static calculations using one neutron energy group were compared to analytical calculations of *K-effective* in section 10.4.1. The numerical results approach the result of the analytical calculation for progressively finer meshes as shown in Table 10.4.1-1 and Figure 10.4.1-1. This would indicate that the diffusion equation and boundary conditions are implemented correctly in the *X* direction. This comparison effectively verified the one group static calculation of the flux.

Two dimensional static calculation using one group were also compared to analytical calculations in Table 10.4.1-2. The results, while reasonably close to the analytical result, did not display the progressive approach with smaller mesh size to the analytical solution as displayed by the one dimensional simulation. More problems with convergence were encountered with the two dimensional simulations but these seem related to the large number of data points that are used. Accurate values of *K-effective* could be obtained but several attempts were required.

Analytical calculations for a larger number of neutron energy groups was considered too complicated and unnecessary for the purpose of code verification. Variations in the results of the static calculations were found when the number of neutron energy groups was changed, but this was considered to be an inaccuracy associated with the collapsing of the cross sections. Changes in moderation due to coolant density changes could not be effectively illuminated with one neutron energy group calculations. All calculations involving changes of coolant temperature were therefore done using two neutron energy groups. The results of one dimensional calculations using two neutron energy groups for progressively finer meshes are given in Table 10.4.1-3.

Static Calculations with Temperature Feedback

Two neutron energy group static calculations for various fixed values of coolant temperature were used to check the reactivity effect of the coolant temperature. As fuel temperature is slightly above the coolant temperature, both the fuel temperature feedback and the coolant temperature feedback would change as a result of the coolant temperature variations. The coolant temperature reactivity, as shown in Table 11.5-1 & Figure 10.4.1-2, is not quite linear and quite strongly negative. The geometry and material composition were chosen so as to produce an under-moderated reactor with strong negative coolant temperature feedback effects. This calculation reveals that this is indeed the case.

Temperature Range	Temperature Reactivity (mk/°C)
0 to 100	-0.1614
100 to 200	-0.1900
0 to 200	-0.1756

Table 11.5-1: Coolant Temperature Reactivity Worth

Transient Calculations

One dimensional transient simulations using one and two neutron energy groups were compared to a point kinetics simulation with the same (2 mk) reactivity insertion. The results (Figures 10.4.2-1 to 10.4.2-5) indicate that the response of the space-time simulation is similar to that of the point kinetics. Figures 10.4.2-1 to 10.4.2-3 indicate that the point kinetics simulation has a quicker response than the space-time simulation for the short term (2 ms). The short term response of the space-time simulation shows little dependence on the number of neutron energy groups, as shown in Figures 10.4.2-2 & 10.4.2-3. The longer time (1 s) response is a manifestation of the delayed neutron effect in the kinetics simulations. The point kinetics simulation is virtually identical to that of the space time simulation in that regard, as indicated in Figures 10.4.2-4 and 10.4.2-5. The temporal behavior of the space-time kinetics simulation was sufficiently similar to that of the point kinetics problem so as to consider it adequately verified.

This completes the verification of the neutronics (only) portion of the simulation. Simulations combining both neutronic and thermalhydraulic effects can now be discussed.

11.5 The Complete Simulation

The two component parts, neutronics and thermalhydraulics, vary simultaneously in the complete simulation. While the thermalhydraulics can only be run as a transient calculation (i.e. only the rate form is available), the neutronics may be run either as a static calculation, where the reactor power is held fixed and the eigenvalue (K-effective) is varied, or as a transient calculation where the reactor power is allowed to vary. Thermal feedback from the thermalhydraulic calculation comes in the form of fuel temperature feedback and coolant density changes, both of which cause a change in the multigroup cross sections. An increase in fuel temperature will increase absorption due to Doppler broadening of the resonance absorption peaks and a decrease in coolant density will reduce the scattering cross section (as well as other cross sections) in the coolant.

During a static calculation, the thermalhydraulic variables, the coolar temperature in particular, are varying and hence cause the cross sections within the coolant to vary. This variation of material properties created some convergence problems in the eigenvalue calculation, especially for the Chebychev acceleration. It was necessary to used a time-averaged value of the coolant density to achieve a converged value for *K-effective*. Convergence was less of a problem, but still only satisfactory, using the average density. This time-averaged value of criticality, while giving an indication of the average state of the reactor, does not indicate the state of the reactor at any specific instant in time. For a transient simulation, it is necessary to have an accurate determination of the eigenvalue at the time the reactor is perturbed. It was found that some manual variation of the eigenvalue was required to attain sufficient accuracy. While the need for such intervention is undesirable, no simpler solutions could be found. The thermalhydraulic behavior and fuel temperature response during a static calculation is given in Figure 10.5-1. It can be seen that significant coolant temperature variation does occur even though the fuel temperature remains relatively constant.

The initial state of the reactor, the state of both the neutronics and the thermalhydraulics, is the same for all the transient simulations. The delayed neutron fraction for these simulations is *0.008* therefore, one dollar (1$) of reactivity would be 8 mk. Transient simulations were performed using 0$, 1/4$, 1/2$ & 1$ reactivity insertions, corresponding to 0mk, 2mk, 4mk & 8mk, respectively. Thermalhydraulic behavior had an obvious effect on the reactor power as clearly indicated in the simulation results plotted in Figures 10.5-2 to 10.5-11. The prompt behavior ($t < 0.005s$), shown in Figures 10.5-2 to 10.5-5, is roughly what would be expected with or without the thermalhydraulic contribution for the various reactivity insertions, a prompt jump followed by delayed neutron effects. The feedback effects of the coolant temperature is quickly felt after that initial time period though.

The *mean* fuel temperature did not change significantly during the transients. However, the temperature at the surface of the fuel (fuel/coolant boundary), where the concentration of thermal neutrons in the fuel is the highest and hence the fission rate at a maximum, follows the coolant temperature. It can be seen in Figure 10.5-8 that the mean fuel temperature does change but rather slowly and is more a function of reactor power than coolant temperature for shorter time periods. All the longer transients, especially the 0 mk transient shown in Figure 10.5-11, demonstrate how the dynamic behavior of the coolant properties significantly affects reactor power.

The behavior of the coolant temperature for all but the largest reactivity insertion, displays the same dynamics in all cases, as illustrated in Figures 10.5-8, 10.5-10 & 10.5-11. This behavior bears strong similarity to that shown in the static calculation as given in Figure 10.5-1. This all indicates that the dynamics of the coolant is a dominant effect in the behavior of the reactor as designed. An objective in the reactor design was a reactor where the coolant dynamics would be felt through feedback effects. These results indicate that the design was more than successful in meeting that objective. The thermalhydraulics had an undesirably large influence on the reactor power. While this clearly illustrates the effect that the thermalhydraulics can have on the reactor behavior, the result is not a practical reactor design. A design using greater moderation less influenced by the coolant dynamics would be more practical. The choice of materials used in the design was hampered a lack of cross-sectional information, in particular, heavy water.

11.6 Conclusions and Recommendations

The objective of this work was to develop and test a reactor simulation that integrated neutron kinetics and thermalhydraulics to form a detailed dynamic simulation of a simplified reactor. This was successfully done within the limited context of the simplified problem. The various components of the code were tested as well as practically possible. The reactor design was chosen to maximize the effect that the thermalhydraulics had upon the reactor behavior, especially the power. The simulation code produced during this project brought to light complex behavior that was a consequence of the interaction of the neutronics and thermalhydraulics. The simulation warrants further investigation and possibly further development to overcome some the problems that are unique to the problem being solved.

The steady state behavior of the thermalhydraulic simulation was quite thoroughly tested using a benchmark problem. For completeness, the transient behavior should be tested but the lack of a benchmark problem is a hindrance. Benchmark problems exist for testing the behavior of the hydraulics but it is desirable to have a test that included the thermal behavior as well.

The simulation of a more common reactor design would be useful in further establishing some of the characteristics of the resultant code. Proper calculation of the multigroup parameters is required using an appropriate lattice cell code.

A possible course for further code development would be the combination of a nodal thermalhydraulics model with a nodal space-time neutron kinetics model. If the same nodal geometry was used in the reactor core for both the thermalhydraulics and the neutron kinetics, there would fewer ambiguities about heat production (fission heat source) and changes in material properties (due to changes in temperature and/or density) in a node. Such a simulation would be more difficult to implement but it would have a wider area of applicability.

11.7 Contribution

Contribution can be divided into three areas: the development of a thermalhydraulic simulation based on the rate form of the equation of state, a space-time dynamic simulation (neutronics and fuel temperature) with an integral thermalhydraulic module, and a study of the methods of solution of the linear algebraic problem for banded matrices encountered in the solution of certain PDEs.

Thermalhydraulics

The calculation of pressure is somewhat problematic when simulating thermalhydraulic systems. The use of the equation of state in a rate form is a relatively innovative concept for the calculation of pressure, one which has been used as the basis of a thermalhydraulic simulation utilizing the nodal form of the thermalhydraulic equations.[113] A problem encountered with the nodal form is the tendency for the pressure to drift, which requires an adjustment to counteract this effect. The nodal form assumes that the nodes are sections of pipe containing fluid where only pressure differentials with the neighboring nodes and the viscous drag of the walls will cause velocity changes. If the desire is to model fluid behavior within a cavity or tube, the viscous effects of the fluid surrounding a point of interest must be considered. This requires the use of a form of the equations other than the nodal form. The thermalhydraulic equations in the present work were derived using a finite difference approximation employing a staggered mesh. The necessity that the pressure be specified as a condition at a minimum of one of the boundaries provides this form of the equations

113 Garland and Sollychin (1987)

with a reference pressure and obviates the need for any form of pressure correction. The method of solution of the thermalhydraulic equations developed here represents a generalization of the earlier nodal method based on the rate form of the equation of state.

Time Integration

Time integration is the most CPU intensive part of this, and most any, simulation. Crank-Nicolson, explicit and semi-implicit Runge-Kutta are the working integration methods. Crank-Nicolson, the simplest integration method, was used for the hydraulics (fluid velocity and pressure). The method was modified to handle a partitioned matrix, the fluid velocity and the pressure were seperated, which facilitated the use of banded matrix inversion methods. The neutron kinetics was also solved utilizing a partitioned matrix, where the prompt and delayed components were seperated. A semi-implicit Runge-Kutta integration method was used for the prompt components and an explicit Runge-Kutta method was used for the delayed components.

The stiffness of the neutron kinetics and the nonlinearity of the thermalhydraulics requires that implicit and semi-implicit time integration methods be used for stability and efficiency. These methods utilize matrix inversion, therefore, there is strong motivation for finding the most efficient solution methods for the associated linear algebraic problems. The applicability of the three matrix inversion method types (direct, implicit and semi-implicit) were explored. The three method types were compared for various test matrices and, predictably, the methods that exploited the banded structure of the matrices performed appreciably better than the non-banded methods. Conjugate gradient methods have been suggested for the solution of some thermalhydraulic problems, hence their inclusion. However, their performance in the test problems indicated that further investigation was not warranted. While the individual methods are well known, specific combinations of algorithms, such as a semi-implicit Runge-Kutta algorithm using banded LU decomposition and matrix partitioning, are not. It was these combinations that proved to be the most valuable.

Efficient methods of storage for the banded linear systems were devised. A method of storage involving only the nonzero bands was utilized for the iterative (Gauss-Seidel) and semi-iterative (conjugate gradient) solver routines. Another storage method for use by direct methods (banded LU decomposition) was devised whereby only the central banded section of the matrix could be stored without matrix rearrangement. A third storage method for block tridiagonal matrices which incorporated only the block matrices was defined. Memory requirements could be reduced significantly depending upon the matrix size and bandwidth.

Reactor Physics - The Complete Simulation

A reactor simulation code is not presently known that has the tight coupling of the thermalhydraulics and neutron kinetics as the code presented here. This coupling is in both the temporal and spatial dimensions. This can be useful to show how significantly the dynamics of the coolant affects the reactor power. Several other combined simulations exist and are more practical for general safety analysis than the present simu-

lation because the whole reactor core can be represented. However, they cannot furnish the detail provided by the present software. If a question arose as to the dynamics of a fuel channel and a closer examination of the coolant-neutronics interaction is required, such a task would be best handled by the code presented here. The code in its present form would require further development before becoming suitable for commercial applications, but it is a useful tool for the exploration of the dynamics of certain problems. Problems associated with combined simulations were revealed, in particular, the estimation of a *K-effective* in static calculations where the coolant properties are not fixed. Ideally this code could be used to develop more simplified models of the reactor dynamics, of the coolant in particular, which would then be used in less computationally demanding simulations. The development of such simplified models however requires a detailed simulation for the purpose of comparison. Such a detailed simulation is presented here.

Appendices

A Thermalhydraulics Equations

This appendix contains algebraic simplifications of some of the thermalhydraulic equations.

1 The Conservation of Momentum Equation

The variable that is conserved here is the product of the density and the velocity. This is substituted into the microscopic conservation equation,

$$\frac{\partial(\rho\vec{V})}{\partial t}+\vec{\nabla}\cdot\rho\vec{V}\vec{V}=\rho\vec{g}+\vec{\nabla}\cdot\vec{\vec{\sigma}}$$

The stress tensor can be expanded into its normal component, which is pressure, and its shear stress component,

$$\vec{\vec{\sigma}}=-P\vec{\vec{I}}+\vec{\vec{\tau}}$$

When the time derivative term and the stress tensor term are expanded the result is

$$\rho\frac{\partial(\vec{V})}{\partial t}+\vec{V}\frac{\partial\rho}{\partial t}+\vec{\nabla}\cdot\rho\vec{V}\vec{V}=\rho\vec{g}-\vec{\nabla}P+\vec{\nabla}\cdot\vec{\vec{\tau}}$$

When the conservation of mass equation is used to substitute the derivative of density with respect to time and the identity

$$\begin{aligned}\vec{\nabla}\cdot\rho\vec{V}\vec{V}&=\vec{V}\vec{V}\cdot\vec{\nabla}\rho+\rho\vec{\nabla}\cdot\vec{V}\vec{V}\\&=\vec{V}\vec{V}\cdot\vec{\nabla}\rho+\rho\vec{V}\cdot\vec{\nabla}\vec{V}+\rho\vec{V}\vec{\nabla}\cdot\vec{V}\end{aligned}$$

is used. The result will be an equation that describes the rate of change of the velocity with respect to time,

$$\rho\frac{\partial\vec{V}}{\partial t}+\rho\vec{V}\cdot\vec{\nabla}\vec{V}=\rho\vec{g}-\vec{\nabla}P+\vec{\nabla}\cdot\vec{\vec{\tau}}$$

If the fluid is assumed to be Newtonian, the shear stress is a linear function of the rate of strain,

$$\vec{\vec{\tau}}=\lambda(\vec{\nabla}\cdot\vec{V})\vec{\vec{I}}+2\mu\vec{\vec{s}}$$

where λ and μ are two coefficients for viscosity and

$$\vec{\vec{s}}=\frac{1}{2}(\vec{\nabla}\vec{V}+(\vec{\nabla}\vec{V})^{T})$$

is the rate of strain tensor. The above equations can be used to calculate $\vec{\nabla}\cdot\vec{\vec{\tau}}$,

$$\vec{\nabla}\cdot\vec{\vec{\tau}}=\vec{\nabla}\cdot\lambda(\vec{\nabla}\cdot\vec{V})\vec{\vec{I}}+\vec{\nabla}\cdot\mu[\vec{\nabla}\vec{V}+(\vec{\nabla}\vec{V})^{T}]$$

$$\vec{\nabla}\cdot\lambda(\vec{\nabla}\cdot\vec{V})\vec{\vec{I}}=\lambda\vec{\nabla}(\vec{\nabla}\cdot\vec{V})+(\vec{\nabla}\cdot\vec{V})\vec{\nabla}\lambda$$

$$\vec{\nabla}\cdot\mu[\vec{\nabla}\vec{V}+(\vec{\nabla}\vec{V})^{T}]=2\vec{\vec{s}}\cdot\vec{\nabla}\mu+\mu\vec{\nabla}^{2}\vec{V}+\mu\vec{\nabla}(\vec{\nabla}\cdot\vec{V})$$

Note that $\vec{\nabla}\cdot(\vec{\nabla}\vec{V})^T = \vec{\nabla}(\vec{\nabla}\cdot\vec{V})$. The conservation of momentum equation now takes the form:

$$\rho\frac{\partial\vec{V}}{\partial t}+\rho(\vec{V}\cdot\vec{\nabla})\vec{V}=\rho\vec{g}-\vec{\nabla}P+\mu\nabla^2\vec{V}+(\lambda+\mu)\vec{\nabla}(\vec{\nabla}\cdot\vec{V})+(\vec{\nabla}\cdot\vec{V})\vec{\nabla}\lambda+2\vec{\tau}\cdot\vec{\nabla}\mu$$

If is assumed that the fluid is incompressible ($\vec{\nabla}\cdot\vec{V}=0$) and that the viscosity is constant ($\vec{\nabla}\mu=0$), the conservation of momentum equation becomes the following (after dividing by ρ)

$$\frac{\partial\vec{V}}{\partial t}+\vec{V}\cdot\vec{\nabla}\vec{V}=\vec{g}-\frac{1}{\rho}\vec{\nabla}P+\upsilon\nabla^2\vec{V}$$

The kinematic viscosity is defined as $\upsilon\equiv\mu/\rho$.

2 The Conservation of Energy Equation

Energy is the variable that is conserved here. Energy can be can be expressed as

$$\Psi=\rho\left(e+\frac{1}{2}V^2\right)$$

where $\rho e=\rho h-P$ is the internal energy per unit volume and $\frac{1}{2}\rho V^2$ is the kinetic energy per unit volume. The conservation of energy equation now appears as

$$\frac{\partial}{\partial t}\left[\rho\left(e+\frac{1}{2}V^2\right)\right]+\vec{\nabla}\cdot\rho\left(e+\frac{1}{2}V^2\right)\vec{V}=-\vec{\nabla}\cdot\vec{q}+S(\vec{r},t)+\rho\vec{g}\cdot\vec{V}+\vec{\nabla}\cdot(\vec{\sigma}\cdot\vec{V})$$

We substitute for the internal energy and expanding the left hand side of the above equation arrive at

$$\frac{\partial}{\partial t}(\rho h)+\vec{\nabla}\cdot\rho h\vec{V}+\frac{1}{2}V^2\left\{\frac{\partial\rho}{\partial t}+\vec{V}\cdot\vec{\nabla}\rho+\rho\vec{\nabla}\cdot\vec{V}\right\}-\frac{\partial P}{\partial t}-\vec{\nabla}\cdot P\vec{V}+\rho\vec{V}\cdot\left(\frac{\partial\vec{V}}{\partial t}+\vec{V}\cdot\vec{\nabla}\vec{V}\right)=R.H.S.$$

The conservation of mass equation can be used to set the terms in the curly brackets to zero. The equation

$$\rho\vec{V}\cdot\frac{\partial\vec{V}}{\partial t}+\rho\vec{V}\cdot(\vec{V}\cdot\vec{\nabla}\vec{V})=\vec{V}\cdot(\vec{\nabla}\cdot\vec{\sigma})+\rho\vec{V}\cdot\vec{g}$$

is the dot product of the velocity and the conservation of momentum equation. If this equation is subtracted from the equation above it, the result (with some manipulation) is

$$\frac{\partial}{\partial t}(\rho h)+\vec{\nabla}\cdot\rho h\vec{V}=-\vec{\nabla}\cdot\vec{q}+S(\vec{r},t)+\frac{\partial P}{\partial t}+\vec{\nabla}\cdot P\vec{V}+\vec{\nabla}\cdot(\vec{\sigma}\cdot\vec{V})-\vec{V}\cdot(\vec{\nabla}\cdot\vec{\sigma})$$

The identities

$$\vec{\nabla}\cdot(\vec{\sigma}\cdot\vec{V})=\vec{V}\cdot(\vec{\nabla}\cdot\vec{\sigma})+\vec{\sigma}:\vec{\nabla}\vec{V}$$

$$\vec{\sigma}:\vec{\nabla}\vec{V}=-P\vec{\nabla}\cdot\vec{V}+\vec{\tau}:\vec{\nabla}\vec{V}$$

$$\vec{\nabla}\cdot P\vec{V}=P\vec{\nabla}\cdot\vec{V}+\vec{V}\cdot\vec{\nabla}P$$

are of some value. When these identities are used the enthalpy equation takes the form

$$\frac{\partial}{\partial t}(\rho h)+\vec{\nabla}\cdot(\rho h\vec{V})=-\vec{\nabla}\cdot\vec{q}+S(\vec{r},t)+\vec{\tau}:\vec{\nabla V}+\frac{\partial P}{\partial t}+\vec{V}\cdot\vec{\nabla}P$$

The two terms on the right hand side can be expanded to get

$$\frac{\partial}{\partial t}(\rho h)=h\frac{\partial \rho}{\partial t}+\rho\frac{\partial h}{\partial t}$$

$$\vec{\nabla}\cdot(\rho h\vec{V})=h\vec{\nabla}\cdot\rho\vec{V}+\rho\vec{V}\cdot\vec{\nabla}h$$

These can then be substituted into the enthalpy equation with the result

$$h\left\{\frac{\partial \rho}{\partial t}+\vec{\nabla}\cdot\rho\vec{V}\right\}+\rho\left(\frac{\partial h}{\partial t}+\vec{V}\cdot\vec{\nabla}h\right)=R.H.S.$$

after some grouping of terms. The conservation of mass equation can be used to set the group of terms on the far left to zero. The result

$$\rho\frac{\partial h}{\partial t}+\rho\vec{V}\cdot\vec{\nabla}h=-\vec{\nabla}\cdot\vec{q}+S(\vec{r},t)+\vec{\tau}:\vec{\nabla V}+\frac{\partial P}{\partial t}+\vec{V}\cdot\vec{\nabla}P$$

is the simplified form of the energy equation.

B The Multigroup Coefficients

The following is a definition of the commonly used multigroup coefficients.

The group flux is

$$\Phi_g(\vec{r},t) = \int_{E_g}^{E_{g-1}} \Phi(\vec{r},E,t)dE$$

The group velocity is

$$\frac{1}{v_g} \equiv \frac{1}{\Phi_g}\int_{E_g}^{E_{g-1}} \frac{\Phi(\vec{r},E,t)}{v(E)}dE$$

The groups total cross section is

$$\Sigma_{tg}(\vec{r}) \equiv \frac{1}{\Phi_g}\int_{E_g}^{E_{g-1}} \Sigma_t(\vec{r},E)\Phi(\vec{r},E,t)dE$$

The group diffusion constant if a diffusion constant $D(\vec{r},E)$ is defined as

$$D_g(\vec{r}) \equiv \frac{\int_{E_g}^{E_{g-1}} D(\vec{r},E)\vec{\nabla}\Phi(\vec{r},E,t)dE}{\int_{E_g}^{E_{g-1}} \vec{\nabla}\Phi(\vec{r},E,t)dE}$$

The transport cross section may be given instead of the diffusion constant. The group transport cross section

$$\Sigma_{trg}(\vec{r}) \equiv \frac{1}{\Phi_g}\int_{E_g}^{E_{g-1}} \Sigma_{tr}(\vec{r},E)\Phi(\vec{r},E,t)dE$$

is calculated in the same manner as the total cross section. The group diffusion constant

$$D_g(\vec{r}) = \frac{1}{3\Sigma_{trg}(\vec{r})}$$

is defined as a function of the group transport cross section in this case.

The scattering cross section from group g' to g is

$$\Sigma_{sg'g}(\vec{r}) \equiv \frac{1}{\Phi_{g'}}\int_{E_g}^{E_{g-1}} dE \int_{E_{g'}}^{E_{g'-1}} \Sigma_s(\vec{r},E' \rightarrow E)\Phi(\vec{r},E',t)dE'$$

The number of fission neutrons produced because of neutrons in group g (i.e. The product of the fission cross section for group g and the number of neutrons produced per fission.) is

$$\nu_g \Sigma_{fg}(\vec{r}) \equiv \frac{1}{\Phi_g} \int_{E_g}^{E_{g-1}} \nu(E') \Sigma_f(\vec{r}, E') \Phi(\vec{r}, E', t) dE'$$

The fraction of fission neutrons that appear in group g is

$$\chi_g \equiv \int_{E_g}^{E_{g-1}} \chi(E) dE$$

C Analytical Calculation of K-effective

As a method of checking whether the results of the neutronic simulation were reasonable, an analytical calculation of *K-effective* for a two region, one and two dimensional reactor was undertaken. The results of these calculations could be then compared to numerical results.

One Dimensional Reactor

The Helmholtz equation is used in the static calculation of the flux within a reactor.

$$\nabla^2\Phi_i + \beta^2\Phi_i = 0$$

where the subscript i represents the region, f for in the fuel and c for in the coolant. The buckling coefficient β is dependent on the material properties of the region. In the fuel, the buckling coefficient is:

$$\beta_f^2 = \frac{\frac{\nu}{K_{eff}}\Sigma_f - \Sigma_t}{D_f}$$

where Σ_f is the fission cross section, Σ_t is the total cross section, D_f is the diffusion coefficient and, ν is the number of neutrons produced per fission. The general solution of the Helmhltz equation is the fuel is

$$\Phi_f(x) = A_f\cos(\beta_f x) + B_f\sin(\beta_f x)$$

At the left boundary, reflective boundary conditions are used and as a consequence, the coefficient B_f is zero,

$$\frac{d\Phi_f(0)}{dx} = 0 \Rightarrow B_f = 0$$

In the coolant, the buckling coefficient is

$$\beta_c^2 = \frac{\Sigma_t}{D_c}$$

and the general solution of the Helmholtz equation is

$$\Phi_c(x) = A_c\exp(\beta_c x) + B_c\exp(-\beta_c x)$$

The boundary conditions are: continuity of flux and current at the interface between the two regions,

$$D_f\frac{d\Phi_f(x_1)}{dx} = D_c\frac{d\Phi_c(x_1)}{dx}$$

$$\Phi_f(x_1) = \Phi_c(x_1)$$

and a reflective boundary at the right,

$$\frac{d\Phi_c(x_2)}{dx} = 0$$

where x_1 is the interface between the two regions and x_2 is the right boundary. These equations may be written as

$$D_f\beta_f A_f \sin(\beta_f x_1) + D_c\beta_c[A_c \exp(\beta_c x_1) - B_c \exp(-\beta_c x_1)] = 0$$

$$A_f \cos(\beta_f x_1) - A_c \exp(\beta_c x_1) - B_c \exp(-\beta_c x_1) = 0$$

$$\beta_c[A_c \exp(\beta_c x_2) - B_c \exp(-\beta_c x_2)] = 0$$

These three equations must be solved simultaneously. The coefficents multiplying the equations (A_f, A_c, B_c) are placed into a vector that multiplies a matrix containing the terms in the three equations. The result of this matrix equation is zero which implies that the matrix must be singular for the non-trivial solution (non-zero coefficients). The charateristic equation is the determinant of the matrix. This equation is equated to zero and used to solve for β_f. This resultant buckling coefficient is then used to caluculated the *K-effective* for the reactor.

Two Dimensional Reactor

The calculation of *K-effective* for the two dimensional reactor requires that buckling coefficients in the *Y* direction be determined. The Helmholtz equation now appears as

$$\frac{\partial^2\Phi_i}{\partial x^2} + \frac{\partial^2\Phi_i}{\partial y^2} + \beta^2\Phi_i = 0$$

Seperation of variables is used such that $\Phi(x, y) = X(x)Y(y)$. The seperated equation is

$$\frac{1}{X}\frac{d^2X(x)}{dx^2} + \beta^2 = \lambda^2 = -\frac{1}{Y}\frac{d^2Y(y)}{dy^2}$$

where λ^2 is the constant of seperation. The general solution of the *Y(y)* equation is

$$Y(y) = A \sin(\lambda y) + B \cos(\lambda y)$$

The boundary conditions are symmetric in the y direction which implies that the coefficient *A* is zero (*A = 0*). The general boundary condition at the top and bottom boundaries is the Cauchy boundary condition (see section 6.7.3),

$$D\frac{\partial\phi}{\partial y} + b\phi = 0$$

where *D* is the diffusion coefficient and *b* is some predetermined coefficient. Substituting *Y(y)* for φ in the above equation, the boundary condition equation now becomes

$$\frac{D}{b}\cos(\lambda a) - \lambda \sin(\lambda a) = 0$$

where *a* is the distance from the center of the reactor to the top or bottom boundary. This equation is used to solve for λ in both regions of the reactor (there are two lambda's).

The equation in the *X* direction

$$\frac{d^2X}{dx^2} + (\beta^2 - \lambda^2)X(x) = 0$$

is slightly modified to include the constant of seperation. Similarly, set $\beta'^2 = \beta^2 - \lambda^2$ and solve as before for β'.

The implementation of these calculations can be found in the program listing *critical* (critical calculation).

D Heat Transfer Expression

The conservation of energy expression can be written as

$$\int_v \frac{\partial T}{\partial t} dv = \oint_S (T\overline{V} - \alpha\overline{\nabla}T)d\overline{S} = Q_t$$

where S is the surface surrounding some volume of interest, v, and Q_t is the total heat flow through the surface. The volume of interest in this case is the volume that is interior to: two planes that are parallel to the hot and cold walls, which intersect the x axis at points a and b ($0 < a < b < 1$). The surface integral will vanish at the top and bottom walls due to the insulated boundary and zero flow velocity, leaving the surface integral over the two planes. The resultant heat flow expression is

$$Q_t = \int_0^1 \left(TV_x - \alpha \frac{\partial T}{\partial x} \right)_{x=a} dy \quad - \quad \int_0^1 \left(TV_x - \alpha \frac{\partial T}{\partial x} \right)_{x=b} dy$$

For the steady state condition, Q_t will be zero, which implies that the thermal energy flowing through the planes will be equal. The points a and b are arbitrary, hence the heat flow through the planes must be a constant.

E Matrix Properties

There are several matrix properties that are mentioned throughout. The definition of these matrix properties is outlined here.

Diagonally Dominant

A matrix is diagonally dominiant if the absolute value of the elements along the diagonal is greater than or equal to the sum of the absolute values of all the elements in the same column or the same row (excluding the diagonal element):

$$|a_{ii}| \geq \sum_{j=0, j\neq i}^{n} |a_{ij}|$$

Positive Definite, Positive Semi-definite and Negative definite

A matrix is positive definite, positive semi-definite and, negative definite if the inner product of a vector with the product of the matrix and the vector is greater than zero, greater than or equal to zero or, less than zero respectively,

$$(v, Av)\begin{cases} >0 & \text{positive definite} \\ \geq 0 & \text{positive semi-definite} \\ <0 & \text{negative definite} \end{cases}$$

where v is any nonzero vector.

Symmetric

A matrix is symmetric if it is equal to its transpose, $A^T = A$.

Singular

A matrix is singular if there exists a nonzero vector such that the product of that matrix and that vector is zero (a zero vector),

$$\exists \quad v \neq 0 \quad \ni \quad Av = 0$$

Null Space and Range

The null space of a matrix, *Null(A)*, is the space that is spanned by all the nonzero vectors *{v}* that satisfy $Av = 0$. A nonsingular matrix is a matrix who's null space is empty. The range of a matrix, *Range(A)*, is space that is spanned by all the vectors that satisfy $Av \neq 0$. Note that *dim{Range(A)} + dim{Null(A)}* = n, where n is the dimension of A.

F Program Listings

1 Main Reactor Simulation Routine

The important sections of the main routine are explained in general terms in section 9.1. As stated there, this is the routine that controls the simulation and calls all the necessary subroutines. The following is the direct listing of the C code.

```
/*
  Main routine for the reactor simulation program.

   return values
   0    normal termination
   1    memory allocation error
   2    file error
   4    band array error
   5    solver error
*/

#include <stdio.h>
#include <math.h>
#include <signal.h>
#include <string.h>
#include <fcntl.h>
#include <time.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <sys/times.h>
#include <mathG.h>
#include <react.h>
#include <ansi.h>

/* external variables */
int nx,ntTH,ntN,ntf,ntc,ntm,ntg,nts,nXN,nXC,nXTH,nXV,nPhi,nW;
int nxf,nxc,nxm,nyN,nyTH,ng;
```

```
int ntreg[3],nxreg[3];
double dy,dxf,dxc,dxm;
double cxf,cxc,cxm,cyy,cxv,cyv,cxh,cyh;

double Prandtl,THexp,mu,grav,G1,FT,Gtx,Gty;
double alphaf,wf,cxreg[3],dxreg[3];
double cxfuel,cyfuel;
double Kf,Kc,muPr;
double T_cold = 0.0;

/* delayed neutron data */
int iC;
double betat,beta1,Keff;
double lambda[6]={0.0127,0.0317,0.115,0.311,1.4,3.87};
double beta[6]={0.038,0.213,0.188,0.407,0.128,0.026};

/* criticality calculation - Chebyshev acceleration */
double Me=0.8,me=0.0,Chebyshev();
int IE,Ilim;

/* array pointers */
double   *BN,*BP,*BTf,*BTcN,*BTc,*BTH,*CN,*CTf,*CTH,*XN,*XTH,*XTHmean;
double   **AN,**ANw,**ATH,**ATHr;
double   *ATf[5],*ATcN[5],*ATc[5],*AV[5],*A12b[4],*A21b[4],
   **A12t,**A22;
double   **Phi,*Cn[6],*dCn[6],*XC,*XdC,*NuEfPhi;
double   **DenC,**DenCmean,**DenCold,*DenV,*Tf,*TcN,*Tc,*Tm,*Temp,**Cg,**Bg;
double   *Vx,*Vy,*P,*VyBC,*VxBC,*Pf,*M_err,*work[4],**workN;
double   *G,*Vg,*EfT,*EaT,*NuEf[3],*Et[3],*Etr[3],**Es[3],**Esv,*Est;
struct   BITriDiag BTD1,BTD2;
FILE     *fopen(),*fpbin,*fpdat,*fpout,*fpxsec,*fpstat,*fpplot;

char * head_fluid[3] = {" X velocity\n"," Y velocity\n"," Pressure\n"};
char * head_den[1] = {" Coolant Density\n"};
char * head_mass[1] = {" Mass Source\n"};
char * head_temp[2] =  {" Fuel Temperature\n"," Coolant Temperature\n"};
char * head_Cn[6] = {" Delayed Group 1\n"," Delayed Group 2\n",
" Delayed Group 3\n"," Delayed Group 4\n"," Delayed Group 5\n",
" Delayed Group 6\n"};

main(int nargs,char ** args)
{
   char fndat[20],fnxsec[30],*xspath,*getenv(),comment[80];
```

```
char fnflow[20],fnflux[20],fnTH[20];
struct tms t_sys, *tp= &t_sys;
long   tlocal;
float  t_cpu,clock_tick;
double Phi_mean,Phi_set,Phi_err,MEmean,MEvar;
double dt,t_sim,ct,tend,dtmp,rhogh,*dptr,c1,c2;
double XTHdotnm,XTHnm,XNdotnm,XNnm,*VN0,*VC0,*VN1,*VC1;
double Cnm,Cdotnm,dtC,dtN,dtTH,Vplot[6];
double Vxmean,Vymean,Tcmean,Tfmean,Tf0,Tc0,Vx0,Vy0,P0;
double Power,Power0,Pcoef;
double tolN,tolTH,epsN,epsTH,alpTH,dK,Keps=1.0e-5;
double XNchk=0,XTHchk=0,Cchk=0;
int        i,j,k,l,kg,kr,sscN,sscTH,ssc_int;
int   iter,nstep,nbl1,nbl2,*PivTH;
int   binio,view,err=0,BC_N,BC_TH,control,nplot;
int   solveN,ssg,ny1,ny2;
int   limN,iterN=0,nbAN,bandN[30],bandV[5],band12[4],band21[4];
int   k0N,k0TH=6,k0V,k012,k021,k0Tf;
int   limTH,nbATH=13,bandTH[13],impTH;

clock_tick = (float)CLK_TCK;
FT = 4.0/3.0;
/* interupt handler */
signal (SIGFPE, sig_handle);
signal (SIGINT, sig_handle);
signal (SIGQUIT,sig_handle);
printf ("\x07");

if (nargs > 1)
  strcpy (fndat,args[1]);
else
  strcpy (fndat,"react.dat");

/* input data */
if ((fpdat = fopen(fndat,"r")) == NULL)
  error_exit ("error opening data file",2);
fscanf (fpdat,"%d%d%d%d%d%d",&nxf,&nxc,&nxm,&ny1,&ny2,&ng);
fscanf (fpdat,"%le%le%le%le",&dxf,&dxc,&dxm,&dy);
fscanf (fpdat,"%d%le%le",&nstep,&dt,&tend);
fscanf (fpdat,"%d%le%le%d%d",&limN,&tolN,&epsN,&ssg,&solveN);
fscanf (fpdat,"%d%le%le%le",&limTH,&tolTH,&epsTH,&alpTH);
```

```
fscanf (fpdat,"%d%d%d",&binio,&view,&nplot);
fscanf (fpdat,"%d%d%d",&BC_N,&BC_TH,&control);
fscanf (fpdat,"%le%le%le%le%le%le",&Kc,&Prandtl,&THexp,&mu,&grav,&G1);
fscanf (fpdat,"%le%le%le",&Kf,&alphaf,&wf);
fscanf (fpdat,"%le%le%le",&Phi_set,&betat,&Keff);
fscanf (fpdat,"%le%le%le%d%d",&Keps,&Me,&me,&IE,&Ilim);
fscanf (fpdat,"%le%le%le%le%le",&Tf0,&Tc0,&Vx0,&Vy0,&T_cold);
do {
 if(fgets (comment,80,fpdat) == NULL) {
    strcpy (comment," no comment ");
    break;
 }
 if (strstr(comment,"Comment") != NULL) {
    fgets (comment,80,fpdat);
    break;
 }
} while (!feof(fpdat));
fclose (fpdat);

if (alpTH == 1.0) impTH = 1;
if (ssg > ng-1) ssg = ng-1;
if (solveN)              /* interval - step size control */
  ssc_int = 50;
else
  ssc_int = 10;

if (control & 2) {       /* 1D calculation */
  control |= 4;     /* no TH */
  BC_N |= 4;        /* 1D */
  ny1 = 1;
  ny2 = 1;
  nbAN = 3 + ng * 2;
  k0Tf = 1;
  ssc_int = 100;
 }
 else {              /* 2D calculation */
  nbAN = 5 + ng * 2;
  k0Tf = 2;
 }
 if (control & 4 || control & 8) {    /* no TH */
   view &= 21;
```

```
 sscTH = 0;
}
else
 sscTH = 1;
sscN = 1;
if (control & 16) {            /* no step size control */
 sscN = 0;
 sscTH = 0;
}
nyN = ny1;
nyTH = ny1 + ny2;
/* data for mesh and constants */
for (i=0; i < 6; ++i)
 beta[i] *= betat;
beta1 = 1.0 - betat;
grav = - fabs (grav);   /* gravity must be negative */
muPr = mu / Prandtl;
G1 *= 1.0e4;            /* adjust units to g / cm s^2 */
nx = nxf+nxc+nxm;
ntf = nxf * nyN;
ntc = nxc * nyN;
ntm = nxm * nyN;
ntN = nx * nyN;              /* # mesh points */
ntTH = nxc * nyTH;
ntg = 12 * ng;
nts = 3 * ng * ng;
nPhi = ng * ntN;             /* flux vector */
nXN = nPhi + ntN;
nXC = 6 * ntf;          /* precursors */
nXTH = 4 * ntTH;             /* V, P & Tc */
nXV = 2 * ntTH;
if (control & 1)        /* static calc */
 nW = nXN;
else
 nW = nXN + nXC;
cxf = 1.0/(dxf*dxf);
cxc = 1.0/(dxc*dxc);
cxm = 1.0/(dxm*dxm);
cyy = 1.0/(dy*dy);
```

```
cxv = mu * cxc;
cyv = mu * cyy;
cxh = mu * cxc / Prandtl;
cyh = mu * cyy / Prandtl;
cxfuel = alphaf * cxf;
cyfuel = alphaf * cyy;
Gtx = G1 * dxc;
Gty = G1 * dy;
nxreg[0] = nxf;
nxreg[1] = nxc;
nxreg[2] = nxm;
ntreg[0] = ntf;
ntreg[1] = ntc;
ntreg[2] = ntm;
dxreg[0] = dxf;
dxreg[1] = dxc;
dxreg[2] = dxm;
cxreg[0] = cxf;
cxreg[1] = cxc;
cxreg[2] = cxm;
Pcoef = wf * dxf * dy * ntf / 2.43 * Phi_set;
k0N = nbAN/2;   /* central band - neutronics */

/* MEMORY Allocation */

/* Neutronics */
workN = Mfmalloc (8,nW);
if (workN == NULL)
  error_exit ("\n workN memory allocation error\n",1);
VN0 = workN[6];
VC0 = &VN0[nXN];
VN1 = workN[7];
VC1 = &VN1[nXN];
BN  = (double *)calloc(nXN,sizeof(double));
CN  = (double *)calloc(nXN,sizeof(double));
XN  = (double *)calloc(nXN,sizeof(double));
Phi = (double **)calloc(ng,sizeof(double *));
Cg  = (double **)calloc(ng,sizeof(double *));
Bg  = (double **)calloc(ng,sizeof(double *));
```

```
/* Delayed precursors */
XC = (double *)calloc(nXC,sizeof(double));
XdC = (double *)calloc(nXC,sizeof(double));
for (i=0; i < 6; ++i) {
  Cn[i]  = &XC[i*ntf];
  dCn[i] = &XdC[i*ntf];
}
NuEfPhi = (double *)calloc(ntf,sizeof(double));   /* fission */
for (k=0; k < 4; ++k)
  work[k] = (double *)calloc(ntf,sizeof(double)); /* RK work */

/* Thermalhydraulics */
BTH  = (double *)calloc(nXTH,sizeof(double));
CTH  = (double *)calloc(nXTH,sizeof(double));
XTH  = (double *)calloc(nXTH,sizeof(double));
XTHmean = (double *)calloc(nXTH,sizeof(double));
DenV = (double *)calloc(ntTH,sizeof(double));
M_err = (double *)calloc(ntTH,sizeof(double));
PivTH = (int *)calloc(ntTH,sizeof(int));
DenC = (double **)calloc(nxc,sizeof(double *));
for (i=0; i < nxc; ++i)
  DenC[i] = &DenV[i*nyTH];
VxBC = (double *)calloc(nyTH,sizeof(double));
VyBC = (double *)calloc(nxc,sizeof(double));
Pf = (double *)calloc(nxc,sizeof(double));
DenCmean = Mfmalloc (nxc,nyTH);
DenCold = Mfmalloc (nxc,nyTH);
if (DenCmean == NULL || DenCold == NULL)
error_exit ("\n DenCmean memory allocation error\n",1);

/* cross sections */
G   = (double *)calloc(ntg,sizeof(double));  /* cross sections */
Est  = (double *)calloc(nts,sizeof(double));  /* scattering */
Esv = (double **)calloc(3*ng,sizeof(double *));

/* check allocation */
if  (BN == NULL || BTH == NULL || CN == NULL || CTH == NULL ||
  XN == NULL || XTH == NULL || DenV == NULL || M_err == NULL ||
  G == NULL)
  error_exit ("\n B,C,X,G memory allocation error\n",1);
```

```
/* cross sectional data */
Vg = G;
EfT = &Vg[ng];
EaT = &EfT[ng];
i = 3*ng;
for (k=0; k < 3; ++k) {
  j = (k+3)*ng;
  NuEf[k] = &G[j];
  Et[k] = &G[j+i];
  Etr[k] = &G[j+2*i];
  Es[k] = &Esv[k*ng];
  for (l=0; l < ng; ++l)
     Esv[k*ng+l] = &Est[(k*ng+l)*ng];
 }
 /* neutronics and temperature */
 BTf = &BN[nPhi];
 BTcN = &BTf[ntf];
 CTf = &CN[nPhi];
 for (k=0; k < ng; ++k) {   /* Phi[group]  */
   j = k*ntN;
   Phi[k] = &XN[j];
   Bg[k] = &BN[j];
   Cg[k] = &CN[j];
 }
 Temp= &XN[nPhi];
 Tf = Temp;
 TcN = &Tf[ntf];
 Tm = &TcN[ntc];
 /* thermalhydraulics */
 Vx = XTH;
 Vy = &Vx[ntTH];
 P = &Vy[ntTH];
 Tc = &P[ntTH];
 BP = &BTH[nXV];
 BTc = &BP[ntTH];
 /* initialize band arrays */
 /* Neutronics */
 if (control & 2) {  /* 1D problem */
   for (j=0; j < k0N-1; ++j)
```

```
    bandN[j] = (j-k0N+1) * ntN;
  bandN[k0N-1] = -1;
  bandN[k0N] = 0;
  bandN[k0N+1] = 1;
  for (j=k0N+2; j < nbAN; ++j)
    bandN[j] = (j-k0N-1) * ntN;
}
else {                  /* 2D problem */
  for (j=0; j < k0N-2; ++j)
    bandN[j] = (j-k0N+2) * ntN;
  bandN[k0N-2] = -nyN;
  bandN[k0N-1] = -1;
  bandN[k0N] = 0;
  bandN[k0N+1] = 1;
  bandN[k0N+2] = nyN;
  for (j=k0N+3; j < nbAN; ++j)
    bandN[j] = (j-k0N-2) * ntN;
}
if (CheckBand (bandN,nbAN,&k0N))
  error_exit ("bandN error\n",4);

if (control & 4)
  goto Matrix_Memory;
bandTH[0]  = -2*ntTH-nyTH;
bandTH[1]  = -2*ntTH;
bandTH[2]  = -ntTH-1;
bandTH[3]  = -ntTH;
bandTH[4]  = -nyTH;
bandTH[5]  = -1;
bandTH[6]  = 0;
bandTH[7]  = 1;
bandTH[8]  = nyTH;
bandTH[9]  = ntTH;
bandTH[10] = ntTH+1;
bandTH[11] = 2*ntTH;
bandTH[12] = 2*ntTH+nyTH;
if (CheckBand (bandTH,nbATH,&k0TH))
  error_exit ("bandTH error\n",4);
bandV[0]  = -nyTH;
```

```
 bandV[1] = -1;
 bandV[2] = 0;
 bandV[3] = 1;
 bandV[4] = nyTH;
 if (CheckBand (bandV,5,&k0V))
   error_exit ("bandV error\n",4);
 band12[0] = -ntTH;
 band12[1] = -ntTH+1;
 band12[2] = 0;
 band12[3] = nyTH;
 if (CheckBand (band12,4,&k012))
   error_exit ("band12 error\n",4);
 band21[0] = -nyTH;
 band21[1] = 0;
 band21[2] = ntTH-1;
 band21[3] = ntTH;
 if (CheckBand (band21,4,&k021))
   error_exit ("band21 error\n",4);

Matrix_Memory:
  /* neutronics */
  AN = Mbmalloc (bandN,nbAN,nXN);
  ANw = Mbmalloc (bandN,nbAN,nXN);
  if (AN == NULL || ANw == NULL)
    error_exit (" AN or ANw memory allocation error",1);
  /* temperature */
  if (control & 2) {
    for (k=0; k < 3; ++k)
      ATf[k] = &AN[k+k0N-1][nPhi];
  }
  else {
    for (k=0; k < 5; ++k) {
      ATf[k] = &AN[k+k0N-2][nPhi];
      ATcN[k] = &ATf[k][ntf];
    }
  }
  if (control & 4)
    goto Cross_Sections;
```

```
/* thermalhydraulics */
ATH = Mbmalloc (bandTH,nbATH,nXTH);
if (ATH == NULL)
  error_exit (" ATH memory allocation error",1);
if (!impTH) {
  ATHr = Mbmalloc (bandTH,nbATH,nXTH);
  if (ATHr == NULL)
     error_exit (" ATHr memory allocation error",1);
}
A12t = Mfmalloc (ntTH,nXV);
A22 = Mfmalloc (ntTH,ntTH);
if (A12t == NULL || A22 == NULL)
  error_exit (" A12 or A22 memory allocation error",1);
for (k=0; k < 5; ++k) {
  AV[k] = ATH[k+4];
  ATc[k] = &ATH[k+4][3*ntTH];
}
for (k=0; k < 4; ++k) {
  A12b[k] = ATH[k+9];
  A21b[k] = &ATH[k][nXV];
}
/* thermalhydraulics: Block Tridiagonal solver - V & T */
nbl1 = 2*nxc;
BTD1 = BlTriLUInit (AV,nyTH,nbl1);
if (BTD1.err)
  error_exit ("\n BTD - V  memory allocation error\n",1);
nbl2 = nxc;
BTD2 = BlTriLUInit (ATc,nyTH,nbl2);
if (BTD2.err)
  error_exit ("\n BTD - T  memory allocation error\n",1);
ZeroMb (ATH,bandTH,nbATH,nXTH);
ZeroV (BTH,nXTH);
ZeroV (CTH,nXTH);
ZeroV (VxBC,nyTH);
ZeroV (VyBC,nxc);

Cross_Sections:
  /* input cross sectional data; 3 Regions: fuel, coolant & moderator */
  if ((xspath = getenv("XSECTION")) == NULL)
```

```
  error_exit ("cross section environment error",6);
sprintf (fnxsec,"%s/macxs.%d",xspath,ng);
if ((fpxsec = fopen(fnxsec,"rb")) == NULL)
  error_exit("\n cross section file error",2);
fread (G,sizeof(double),ntg,fpxsec);
fread (Est,sizeof(double),nts,fpxsec);
fclose (fpxsec);
for (kg = 0; kg < ng; ++kg) {
  if (kg < ssg)
     Vg[kg] = Vg[ssg];
}
if (solveN & 2 && control & 1) {        /* static calc - relative velocity */
  for (kg = 0; kg < ng; ++kg)
     Vg[kg] /= Vg[ng-1];
}
/* INITIALIZE */
ZeroV (BN,nXN);
ZeroV (CN,nXN);
if (binio & 1) {
  if ((fpbin = fopen("flux.bin","rb")) == NULL)
     error_exit ("error opening binary file",2);
  fread (XN,sizeof(double),nXN,fpbin);
  fread (XC,sizeof(double),nXC,fpbin);
  fclose(fpbin);
 }
 else {
  CfillV (Tf,Tf0,ntf);
  CfillV (Phi[0],1.0/ng,nPhi);
  /* precursors */
  ZeroV (NuEfPhi,ntf);
  for (kg = 0; kg < ng; ++kg)
     V_addCV (NuEfPhi,NuEf[0][kg]/Keff,Phi[kg],ntf);
  for (iC=0; iC < 6; ++iC) {
     V_CmulV (Cn[iC],beta[iC]/lambda[iC],NuEfPhi,ntf);
     V_CVaddCV (dCn[iC],beta[iC],NuEfPhi,-lambda[iC],Cn[iC],ntf);
   }
 }
 rhogh = grav * dy;
 P0 = -rhogh*nyTH;
```

```
CfillV (Pf,rhogh*nyTH+P0,nxc);
if (binio & 4) {
  if ((fpbin = fopen("flow.bin","rb")) == NULL)
     error_exit (" error opening flow file",2);
  fread (XTH,sizeof(double),nXTH,fpbin);
  fread (DenV,sizeof(double),ntTH,fpbin);
  fclose (fpbin);
}
else {
  CfillV (Tc,Tc0,ntTH);
  CfillV (Vx,Vx0,ntTH);
  CfillV (Vy,Vy0,ntTH);
  CfillV (DenV,1.0,ntTH);
  for (j=0; j < nyTH; ++j) {
     dtmp = rhogh * j + P0;
     for (i=0; i < nxc; ++i)
        P[i*nyTH+j] = dtmp;
  }
}
for (k=0; k < nxc; ++k)
  CopyV (&TcN[k*nyN],&Tc[k*nyTH],nyN); /* initialize TcN */

ZeroV (VN0,ntf);
for (kg = 0; kg < ng; ++kg)
  V_addV (VN0,Phi[kg],ntf);
Phi_mean = Mean (VN0,ntf);
if (control & 1) {
  /* adjust the mean flux - normalized */
  Phi_err =  1.0 / Phi_mean;
  for (kg=0; kg < ng; ++kg)
     V_mulC (Phi[kg],Phi_err,ntN);
  V_mulC (XC,Phi_err,nXC);
}
  /*  THE MAIN LOOP  - INITIALIZE */
t_sim = 0.0;
ct = 1.0/dt;
dtN = dt;
dtTH = dt;
iter = 0;
Tfmean = Mean (Tf,ntf);
```

```
Tfmean = Mean (Tf,ntf);
Tcmean = Mean (Tc,ntTH);
Vxmean = Mean (Vx,ntTH);
Vymean = Mean (Vy,ntTH);
if ((fpplot = fopen("react.plt","wb")) == NULL)
  error_exit ("\n error opening binary plot file\n",2);
ZeroV (NuEfPhi,ntf);
for (kg = 0; kg < ng; ++kg)
  V_addCV (NuEfPhi,NuEf[0][kg]/Keff,Phi[kg],ntf);
Power0 = Mean (NuEfPhi,ntf);
Vplot[0] = t_sim;
if (control & 1)
  Vplot[1] = Keff;
else
  Vplot[1] = 1.0;
Vplot[2] = Tfmean;
Vplot[3] = Tcmean;
Vplot[4] = Vxmean;
Vplot[5] = Vymean;
fwrite (Vplot,sizeof(double),6,fpplot);

if (view & 1)      /* clear the screen */
  printf("%s",Clear);
else {
  if ((fpstat = fopen("react.st","w")) == NULL)
    error_exit(" status file error",2);
}

Flux (AN,k0N,BN,Keff,Phi_set,BC_N);
Thermal (ATf,k0Tf,BTf,BC_N);

/* thermal feedback */
if (control & 2) {  /* 1D problem  - EaT */
  for (kg=0; kg < ng; ++kg)     /* Doppler - absorption */
    V_CmulV (&AN[k0N+ng-kg+1][kg*ntN],-EaT[kg],Phi[kg],ntf);
  for (kg=0; kg < ng; ++kg)     /* Doppler - fission */
    V_addCV (AN[k0N+ng+1],EfT[kg],Phi[kg],ntf);
}
else {              /* 2D problem */
  Density (DenV,THexp,Tc,ntTH);
  XSCoolant (AN,k0N,DenC,BC_N,Keff);
```

```
  for (kg=0; kg < ng; ++kg)     /* Doppler - absorption */
    V_CmulV (&AN[k0N+ng-kg+2][kg*ntN],-EaT[kg],Phi[kg],ntf);
  for (kg=0; kg < ng; ++kg)     /* Doppler - fission */
    V_addCV (AN[k0N+ng+2],EfT[kg],Phi[kg],ntf);
 }
if (!nstep) goto react_exit;
 /******* THE MAIN LOOP *******/
do {
 CopyMb (ANw,AN,bandN,nbAN,nXN);
 ZeroV (BN,nXN);
 if (!(control & 4)) {
   ZeroV (BTH,nXTH);
   Fluid (ATH,BTH,Vx,Vy,DenV,VxBC,VyBC,Pf,ATf,BTf,ATcN,BTcN,
     ATc,BTc,BC_TH);
 }
 /* Neutronics */
 if (!(control & 1) || solveN & 2)
   GroupVelocity(nbAN,bandN);
 /* STEP SIZE CONTROL */
 /* neutronics & temperature */
 if (control & 1) {    /* Static Calculation */
   ZeroV (NuEfPhi,ntf);
   for (kg = 0; kg < ng; ++kg) /* fission production */
     V_addCV (NuEfPhi,NuEf[0][kg]/Keff,Phi[kg],ntf);
   /* delayed neutron - steady state */
   for (i=0; i < 6; ++i) {
     V_CmulV (Cn[i],beta[i]/lambda[i],NuEfPhi,ntf);
     V_addCV (Bg[0],lambda[i],Cn[i],ntf);
   }
   if (solveN & 1)
     err = RKSI_GS (XN,XN,nXN,dt,ANw,bandN,nbAN,k0N,BN,
       workN,tolN,limN,&iterN);
   else
     err = RKSI_LUb (XN,XN,nXN,dt,ANw,bandN,nbAN,k0N,BN,workN);
```

```
if (sscN && iter > 0 && iter % ssc_int == 0) {

    /* two half steps - step size control */
  if (solveN & 1)
    err= RKSI_GS (XN,VN0,nXN,dt/2,ANw,bandN,nbAN,k0N,BN,
      workN,tolN,limN,&iterN);
  else
    err= RKSI_LUb(XN,VN0,nXN,dt/2,ANw,bandN,nbAN,k0N,BN,workN);
  CopyMb (ANw,AN,bandN,nbAN,nXN);
  if (!(control & 1) || solveN & 2)
    GroupVelocity(nbAN,bandN);

  if (solveN & 1)
    err= RKSI_GS (VN0,VN1,nXN,dt/2,ANw,bandN,nbAN,k0N,BN,
      workN,tolN,limN,&iterN);
  else
    err= RKSI_LUb(VN0,VN1,nXN,dt/2,ANw,bandN,nbAN,k0N,BN,workN);
  V_VsubV (CN,XN,VN0,nXN);
  XNchk = 0;
  for (kg=0; kg < ng; ++kg) {
    XNnm = VNorm (Phi[kg],ntN);          /* flux */
    XNdotnm = VNorm (Cg[kg],ntN);
    dtmp = dt * XNdotnm / XNnm;
    XNchk = (dtmp > XNchk) ? dtmp : XNchk;
  }
  XNnm = VNorm (Tf,ntf);                 /* Tf */
  if (XNnm > 1.0e-6) {
    XNdotnm = VNorm (CTf,ntf);
    dtmp = dt * XNdotnm / XNnm;
    XNchk = (dtmp > XNchk) ? dtmp : XNchk;
  }
  dtN = dt * sqrt (epsN / XNchk);
 }
 if (err)
  goto react_exit;
}
else {          /* transient calc */
 if (sscN && (iter % 25 == 0 && iter > 0)) {
  if (solveN & 1) {
```

```
  /* one step */
  err = RKSI_GS_N (XN,VN0,nXN,XC,VC0,nXC,dt,
   ANw,bandN,nbAN,k0N,BN,workN,tolN,limN,&iterN,BC_N,ssg,Keff);
  /* two half steps */
  err = RKSI_GS_N (XN,VN1,nXN,XC,VC1,nXC,dt/2,
   ANw,bandN,nbAN,k0N,BN,workN,tolN,limN,&iterN,BC_N,ssg,Keff);
  err = RKSI_GS_N (VN1,XN,nXN,VC1,XC,nXC,dt/2,
   ANw,bandN,nbAN,k0N,BN,workN,tolN,limN,&iterN,BC_N,ssg,Keff);
}
else {
  /* one step */
  err = RKSI_LU_N (XN,VN0,nXN,XC,VC0,nXC,dt,
   ANw,bandN,nbAN,k0N,BN,workN,tolN,limN,&iterN,BC_N,ssg,Keff);
  /* two half steps */
  err = RKSI_LU_N (XN,VN1,nXN,XC,VC1,nXC,dt/2,
   ANw,bandN,nbAN,k0N,BN,workN,tolN,limN,&iterN,BC_N,ssg,Keff);
  err = RKSI_LU_N (VN1,XN,nXN,VC1,XC,nXC,dt/2,
   ANw,bandN,nbAN,k0N,BN,workN,tolN,limN,&iterN,BC_N,ssg,Keff);
}
if (err)
  goto react_exit;
V_VsubV (CN,XN,VN0,nXN);
V_VsubV (XdC,XC,VC0,nXC);

XNchk = 0;
Cchk= 0;
for (kg=0; kg < ng; ++kg) {
  XNnm = VNorm (Phi[kg],ntN);
  XNdotnm = VNorm (Cg[kg],ntN);
  dtmp = dt * XNdotnm / XNnm;
  XNchk = (dtmp > XNchk) ? dtmp : XNchk;
}
XNnm = VNorm (Tf,ntf);
if (XNnm > 1.0e-6) {
  XNdotnm = VNorm (CTf,ntf);
  dtmp = dt * XNdotnm / XNnm;
  XNchk = (dtmp > XNchk) ? dtmp : XNchk;
}
dtN = dt * sqrt (epsN / XNchk);
```

```
      for (iC = 0; iC < 6; ++iC) {
        Cnm = VNorm (Cn[iC],ntf);
        Cdotnm = VNorm (dCn[iC],ntf);
        dtmp = dt * Cdotnm / Cnm;
        Cchk = (dtmp > Cchk) ? dtmp : Cchk;
      }
      dtC = dt * sqrt (epsN / Cchk);
      dtN = (dtN < dtC) ? dtN : dtC;
    }
    else {
      ZeroM (workN,8,nW);     /***********/
      if (solveN & 1) {
       err = RKSI_GS_N (XN,XN,nXN,XC,XC,nXC,dt,ANw,bandN,nbAN,k0N,BN,
         workN,tolN,limN,&iterN,BC_N,ssg,Keff);
      }
      else {
       err = RKSI_LU_N (XN,XN,nXN,XC,XC,nXC,dt,ANw,bandN,nbAN,k0N,BN,
         workN,tolN,limN,&iterN,BC_N,ssg,Keff);
      }
      if (err)
        goto react_exit;
    }
  }
  if (control & 2) {    /* 1D problem */
    for (kg=0; kg < ng; ++kg)  /* Doppler - absorption */
      V_CmulV (&AN[k0N+ng-kg+1][kg*ntN],-EaT[kg],Phi[kg],ntf);
    for (kg=0; kg < ng; ++kg)  /* Doppler - fission */
      V_addCV (AN[k0N+ng+1],EfT[kg],Phi[kg],ntf);
  }
  else {
    for (kg=0; kg < ng; ++kg)  /* Doppler - absorption */
      V_CmulV (&AN[k0N+ng-kg+2][kg*ntN],-EaT[kg],Phi[kg],ntf);
    for (kg=0; kg < ng; ++kg)  /* Doppler - fission */
      V_addCV (AN[k0N+ng+2],EfT[kg],Phi[kg],ntf);
  }
  ZeroV (VN0,ntf);
  for (kg = 0; kg < ng; ++kg)
    V_addV (VN0,Phi[kg],ntf);
  Phi_mean = Mean (VN0,ntf);
```

```
ZeroV (NuEfPhi,ntf);
for (kg = 0; kg < ng; ++kg)
   V_addCV (NuEfPhi,NuEf[0][kg]/Keff,Phi[kg],ntf);
Power = Mean (NuEfPhi,ntf) / Power0;     /* normalize power */

/* Keff - static calculation */
if (control & 1) {
   dK = Power - 1.0;
   Keff = Chebyshev (Keff,dK);
   if ((fabs(dK)/Keff < (1-Me)*Keps) && (iter > 100))
      goto react_exit;
   AdjustK (AN,k0N,Keff,BC_N,DenC);

   /* adjust flux */
   Phi_err =  1.0 / Phi_mean;
   for (kg=0; kg < ng; ++kg)
      V_mulC (Phi[kg],Phi_err,ntN);
   V_mulC (XC,Phi_err,nXC);
   /* new power */
   ZeroV (NuEfPhi,ntf);
   for (kg = 0; kg < ng; ++kg)
      V_addCV (NuEfPhi,NuEf[0][kg]/Keff,Phi[kg],ntf);
   Power0 = Mean (NuEfPhi,ntf);
}

if (control & 4 || control & 8) /* no TH solver */
   goto ENDiter;

/* Thermalhydraulics */
V_MbmulV (CTH,nXTH,ATH,bandTH,nbATH,k0TH,XTH,nXTH);
V_addV (CTH,BTH,nXTH);
XTHchk = 0;
for (k=0; k < 4; ++k) {    /* V, P & Tc */
   XTHnm = VNorm (&XTH[k*ntTH],ntTH);
   if (XTHnm == 0)
      continue;
   XTHdotnm = VNorm (&CTH[k*ntTH],ntTH);
   dtmp = dt * XTHdotnm / XTHnm;
   XTHchk = (dtmp > XTHchk) ? dtmp : XTHchk;
}
dtTH = dt * (0.5 + 0.5 * sqrt (epsTH / XTHchk));
```

```
if (impTH) {
   V_VaddCV (CTH,BTH,ct,XTH,nXTH);
   NegV (BTH,CTH,nXTH);
   V_addC (ATH[k0TH],-ct,nXTH);
}
/* (I - dt*alpha*A) X(k+1) = (I + dt*(1-alpha)A) X(k) + dt*B */
else {
   CopyM (ATHr,ATH,nbATH,nXTH);
   M_mulC (ATH,-dt*alpTH,nbATH,nXTH);
   M_mulC (ATHr,dt*(1.0-alpTH),nbATH,nXTH);
   V_addC (ATH[k0TH],1.0,nXTH);
   V_addC (ATHr[k0TH],1.0,nXTH);
   V_MbmulV (CTH,nXTH,ATHr,bandTH,nbATH,k0TH,XTH,nXTH);
   V_VaddCV (BTH,CTH,dt,BTH,nXTH);
}

/* BTD solver: V & P */
if (BlTriLU (BTD1,nyTH,nbl1))
   error_exit ("\n Block Tridiag Velocity\n",5);
BlTriLUSolve (BTD1,BTH,nyTH,nbl1);   /* B1' */
ZeroM (A12t,ntTH,nXV);
BtoFt (A12b,band12,4,A12t,nXV,ntTH);
BlTriLUSolveM (BTD1,A12t,ntTH,nyTH,nbl1); /* A12' */
k=0;
do {
   V_MbmulV (CTH,ntTH,A21b,band21,4,k021,A12t[k],nXV);
      M_ColV (A22,k,CTH,ntTH);
} while (++k < ntTH);
M_DsubM (A22,&ATH[k0TH][nXV],A22,ntTH); /* A22 - A21*A12' */
V_MbmulV (CTH,ntTH,A21b,band21,4,k021,BTH,nXV); /* A21*B1' */
V_VsubV (CTH,BP,CTH,ntTH); /* B2 - A21*B1' */
if (LU (A22,PivTH,ntTH))
   error_exit ("\n A22 Decomposition \n",5);
LUSolve (A22,P,CTH,PivTH,ntTH);
V_MtmulV (CTH,nXV,A12t,P,ntTH);    /* A12'* X2 */
V_VsubV (XTH,BTH,CTH,nXV); /* B1' - A12'* X2 */
```

```
    /* BTD Temperature */
    CopyV (Tc,BTc,ntTH);
    if (BTriLU (BTD2,nyTH,nbl2))
      error_exit ("\n Block Tridiag Temperature\n",2);
    BTriLUSolve (BTD2,Tc,nyTH,nbl2);

ENDiter:
    if (control & 2)        /* 1D */
      goto INC_TIME;
    for (k=0; k < nxc; ++k)
      CopyV (&Tc[k*nyTH],&TcN[k*nyN],nyN);

    /* adjust the coolant density and cross sections */
    CopyM (DenCold,DenC,nxc,nyTH);        /* save old density */
    Density (DenV,THexp,Tc,ntTH);     /* new density */
    if (control & 1) {
      if (t_sim == 0) {
        CopyM (DenCmean,DenC,nxc,nyTH);
        CopyV (XTHmean,XTH,nXTH);
        goto INC_TIME;
      }
      c1 = 1.0 / (1 + dt / t_sim);
      c2 = (1 - c1);
      V_CVaddCV (XTHmean,c1,XTHmean,c2,XTH,nXTH);
      c1 *= 0.999;
      c2 = (1 - c1) * 0.5;
      for (i=0; i < nxc; ++i) {        /* mean density w.r.t. time */
        for (j=0; j < nyTH; ++j) {
          DenCmean[i][j] *= c1;
          DenCmean[i][j] += c2 * (DenC[i][j] + DenCold[i][j]);
        }
      }
    }
    else {
      for (i=0; i < nxc; ++i) {
        for (j=0; j < nyTH; ++j)
          DenCmean[i][j] = (DenC[i][j] + DenCold[i][j]) / 2;
      }
    }
    XSCoolant (AN,k0N,DenCmean,BC_N,Keff);
INC_TIME:
```

```
/* ADVANCE TIME */
t_sim += dt;
++iter;
Tfmean = Mean (Tf,ntf);
Tcmean = Mean (Tc,ntTH);
Vxmean = Mean (Vx,ntTH);
Vymean = Mean (Vy,ntTH);
/* plot file - transient runs */
if (iter % nplot == 0) {
   Vplot[0] = t_sim;
   if (control & 1)
      Vplot[1] = Keff;
   else
      Vplot[1] = Power;
   Vplot[2] = Tfmean;
   Vplot[3] = Tcmean;
   Vplot[4] = Vxmean;
   Vplot[5] = Vymean;
   fwrite (Vplot,sizeof(double),6,fpplot);
   fflush (fpplot);
 }
 if (view & 1) {
   printf ("%s",Home);
   printf (" iteration %4.0d  dt = %8.3e, total time %8.4e\n",
      iter,dt,t_sim);
   printf (" GSSOR N: %d, Mean flux %9.5e, Power %g, K_eff %9.6f \n",
      iterN,Phi_mean,Power,Keff);
   printf (" Me %g, me %g  \n",Me,me);
   printf (" X check N: %g; C: %g; TH: %g  \n",XNchk,Cchk,XTHchk);
   printf (" T mean fuel: %g; coolant: %g  \n",Tfmean,Tcmean);
   printf (" Vx mean = %8.4g, Vy mean = %8.4g \n",Vxmean,Vymean);
   if (view & 2) {
      Mass (Vx,Vy,VyBC,VxBC,DenV,M_err);
      MEmean = Mean (M_err,ntTH);
      MEvar = Variance (M_err,ntTH,MEmean);
      printf (" Mass Error: Mean = %8.4g, Variance = %8.4g \n",
         MEmean,MEvar);
   }
 }
```

```
    else {
      /* status file */
      rewind (fpstat);
      fprintf (fpstat," iteration %4.0d  dt = %8.3e, total time %g\n",
            iter,dt,t_sim);
      fprintf (fpstat," GSSOR %d, Mean flux %g, Power %g, K_eff %9.6f\n",
            iterN,Phi_mean,Power,Keff);
      fprintf (fpstat," Me %g, me %g  \n",Me,me);
      fprintf (fpstat," X check N: %g; C: %g; TH: %g,  \n",
            XNchk,Cchk,XTHchk);
      fprintf (fpstat," T mean fuel: %g; coolant: %g\n",Tfmean,Tcmean);
      Mass (Vx,Vy,VyBC,VxBC,DenV,M_err);
      MEmean = Mean (M_err,ntTH);
      MEvar = Variance (M_err,ntTH,MEmean);
      fprintf (fpstat," Mass Err: Mean = %8.4g, Var = %8.4g \n",
        MEmean,MEvar);
      fprintf (fpstat," Vx mean = %8.4g, Vy mean = %8.4g \n",
        Vxmean,Vymean);
      fflush (fpstat);
    }
    if (sscN & sscTH)
      dt = (dtN > dtTH) ? dtTH : dtN;
    else {
      if (sscN)
        dt = dtN;
      if (sscTH)
        dt = dtTH;
    }
  } while ((iter < nstep) && (t_sim < tend));
react_exit:
  fclose(fpplot);
  if (err) {
    save_flux ("flux.dmp");
    save_flow ("flow.dmp");
    printf ("\n Application terminated, data array dumped\n");
    fprintf (fpstat,"Error Termination: Solver Error; GSSOR=%d\n",iterN);
    fclose (fpstat);
    error_exit ("Solver Error",5);
  }
```

```
Mass (Vx,Vy,VyBC,VxBC,DenV,M_err);
MEmean = Mean (M_err,ntTH);
MEvar = Variance (M_err,ntTH,MEmean);

if (times (tp) != -1)
  t_cpu = (float)t_sys.tms_utime / clock_tick;
tlocal = time ((long *)0);
if (view & 1) {
  printf("%s",Clear);
  printf (" GS N: %d, Mean flux %8.4e, K_effective %lf\n",
     iterN,Phi_mean,Keff);
  printf (" Me %g, me %g  \n",Me,me);
  printf (" T mean fuel: %g; coolant: %g\n",Tfmean,Tcmean);
  printf(" Elapsed CPU time (sec) = %8.2f",t_cpu);
  printf(" total simulation time = %8.4g\n",t_sim);
 }
 else
  printf (" Reactor simulation complete \n");
 if (binio & 2)
  save_flux ("flux.bin");
 if (binio & 8)
  save_flow ("flow.bin");
 if ((fpbin = fopen("flowm.bin","wb")) == NULL)
  error_exit ("error opening flowm file",3);
 fwrite (XTHmean,sizeof(double),nXTH,fpbin);
 for (i=0; i < nxc; ++i)
    fwrite (DenCmean[i],sizeof(double),nyTH,fpbin);
 fclose(fpbin);

 /* ascii output */
 if ((fpout = fopen("react.lp","w")) != NULL) {
   fprintf (fpout," Unix Reactor Simulation  %s\n",ctime(&tlocal));
   fprintf (fpout,"  %s\n",comment);
   fprintf (fpout," Elapsed CPU time (sec) = %g\n",t_cpu);
   fprintf (fpout," iteration %4.0d, dt = %g, total time = %g\n",
        iter,dt,t_sim);
   fprintf (fpout," Neutronics: tolerance = %g\n",tolN);
   fprintf (fpout," GS: Neutronics  %d\n",iterN);
   fprintf (fpout," Mean flux %10.6e, K_effective %11.7f, Power %g\n",
        Phi_mean,Keff,Power);
   fprintf (fpout," Me %g, me %g  \n",Me,me);
```

```
fprintf (fpout," parameters: G1 = %6.2lg\n",G1);
fprintf (fpout,"   alpha = %6.2lg, tolerance = %g\n",alpTH,tolTH);
fprintf (fpout," Mass Error: Mean = %8.4g, Variance = %8.4g\n",
      MEmean,MEvar);
fprintf (fpout," Temperature: fuel: %g; coolant: %g; Tcold: %g\n",
      Tfmean,Tcmean,T_cold);
fprintf (fpout," Velocity: X = %8.4g, Y = %8.4g \n",Vxmean,Vymean);
fprintf (fpout,"\n mesh data\n");
fprintf (fpout," nxf = %d, nxc = %d, nxm = %d, nyN = %d, nyTH = %d\n",
      nxf,nxc,nxm,nyN,nyTH);
fprintf (fpout," dxf = %g, dxc = %g, dxm = %g, dy = %g\n",
      dxf,dxc,dxm,dy);
fprintf (fpout,"\n mean flux per region\n");
for (kg=0; kg < ng; ++kg) {
   fprintf (fpout," group # %d\n",kg);
   dptr = Phi[kg];
   for (kr=0; kr < 3; ++kr) {
      k = ntreg[kr];
      fprintf (fpout," %14.6e  ",Mean(dptr,k));
      dptr = &dptr[k];
   }
   fprintf (fpout,"\n");
 }
 fclose(fpout);
}
if (view & 4) {
 if ((fpout = fopen("flux.lp","w")) == NULL)
   error_exit("\n error opening flux file",2);
 fprintf (fpout," Unix Reactor Simulation  %s\n",ctime(&tlocal));
 if (control & 2) {
   fprintf (fpout," Neutron Flux\n");
   for (k=0; k < ntN; ++k) {
      if (k % 8 == 0)
         fprintf (fpout,"\n");
      fprintf (fpout,"% 10.3e",XN[k]);
   }
   fprintf (fpout,"\n\n");
   for (j=0; j < 6; ++j) {
      fprintf (fpout," Precursor %d\n",j);
```

```
        for (k=0; k < ntf; ++k) {
          if (k % 8 == 0)
            fprintf (fpout,"\n");
          fprintf (fpout,"% 10.3e",Cn[j][k]);
        }
        fprintf (fpout,"\n\n");
      }
    }
    else {
      printdat (fpout,XC,nxf,nyN,6,head_Cn);
      fprintf (fpout,"\f");
      printflux (fpout);
    }
    fclose(fpout);
  }
  if (view & 8) {
    if ((fpout = fopen("flow.lp","w")) == NULL)
      error_exit("\n error opening flow file",2);
    fprintf (fpout," Unix Reactor Simulation  %s\n",ctime(&tlocal));
    printdat (fpout,XTH,nxc,nyTH,3,head_fluid);
    printdat (fpout,DenV,nxc,nyTH,1,head_den);
    printdat (fpout,M_err,nxc,nyTH,1,head_mass);
    fclose(fpout);
  }
  if (view & 16) {
    if ((fpout = fopen("temp.lp","w")) == NULL)
      error_exit("\n error opening temperature file",2);
    fprintf (fpout," Unix Reactor Simulation  %s\n",ctime(&tlocal));
    printdat (fpout,Tf,nxf,nyN,1,head_temp);
    printdat (fpout,Tc,nxc,nyTH,1,&head_temp[1]);
    fclose(fpout);
  }
  printf("\n normal termination\n");
  fprintf (fpstat,"Normal Termination\n");
  fclose (fpstat);
  exit (0);
}
```

```
int GroupVelocity (int nbA,int * band)
{
  int kb,kg,ks,ke;
  V_mulC (Bg[0],Vg[0],ntf);   /* group () velocity */
  for (kg=0; kg < ng; ++kg) {
    for (kb=0; kb < nbA; ++kb) {
      ks = kg * ntN;
      ke = ks + ntN;
      if (band[kb] < 0) {
        if (ke <= -band[kb])
          continue;
      }
      else
        if (ks+band[kb] >= nXN)
          continue;
      V_mulC (&ANw[kb][ks],Vg[kg],ntN);
    }
  }
}
/* adjust the coolant density */
int Density (double * DenV,double THexp,double * Tc,int ntTH)
{
  int k;
  double dtmp;
  k=0;
  do {
    dtmp = 1.0 - THexp * Tc[k];
    DenV[k] = (dtmp < 0.01) ? 0.01 : dtmp;
  } while (++k < ntTH);
}
int save_flux (char * fname)
{
  FILE *fpbin;
  if ((fpbin = fopen(fname,"wb")) == NULL)
    error_exit ("error opening binary file",3);
  fwrite (XN,sizeof(double),nXN,fpbin);
  fwrite (XC,sizeof(double),nXC,fpbin);
  fclose (fpbin);
}
```

```
int save_flow (char * fname)
{
  FILE *fpbin;
  if ((fpbin = fopen(fname,"wb")) == NULL)
    error_exit ("error opening flow file",3);
  fwrite (XTH,sizeof(double),nXTH,fpbin);
  fwrite (DenV,sizeof(double),ntTH,fpbin);
  fclose(fpbin);
}
int sig_handle (int sig)
{
  save_flux ("flux.dmp");
  save_flow ("flow.dmp");
  switch (sig) {
    case 2:
        fprintf (fpstat,"Error Termination: Iterrupt\n");
        printf ("\n interrupt\n");
        break;
    case 3:
        fprintf (fpstat,"Error Termination: Quit\n");
        printf ("\n quit\n");
        break;
    case 8:
        fprintf (fpstat,"Error Termination: Floating Point\n");
        printf ("\n floating point exception\n");
  }
  printf ("\n Application terminated, data array dumped\n");
  fclose (fpstat);
  exit (sig);
}
void printdat (FILE * fp,double * X,int nx,int ny,int nbl,char ** head)
{
  int j,k=0,l,n;
  double * dptr = X;
  n = nx*ny;
  for (k=0; k < nbl; ++k) {
    fprintf (fp,head[k]);
    j=0;
    do {
```

```
        for (l=0; l < ny; ++l)
            fprintf (fp,"% 10.3e",dptr[j++]);
        fprintf (fp,"\n");
      } while (j < n);
      dptr = &dptr[n];
    }
}
void printflux (FILE * fp)
{
    int kr,kg,i,j,k;
    double *dptr = XN;
    if (nyN > 8) goto page2;
    for (kg=0; kg < ng; ++kg) {
      for (kr=0; kr < 3; ++kr) {
        fprintf (fp," Flux in region %d, group %d\n",kr,kg);
        j = 0;
        do {
          i=0;
          do {
            fprintf (fp,"% 10.3e",dptr[j++]);
          } while (++i < nyN);
          fprintf (fp,"\n");
        } while (j < ntreg[kr]);
        dptr = &dptr[ntreg[kr]];
      }
    }
    return;
page2:
    for (kg=0; kg < ng; ++kg) {
      for (kr=0; kr < 3; ++kr) {
        fprintf (fp," Flux in region %d, group %d\n",kr,kg);
        j=0;
        do {
          k = j*nyN;
          i=0;
          do {
            fprintf (fp,"% 10.3e",dptr[k+i]);
          } while (++i < 8);
          fprintf (fp,"\n");
```

```
        } while (++j < nxreg[kr]);
        dptr = &dptr[ntreg[kr]];
      }
      dptr = &dptr[-ntN];      /* backup pointer */
      fprintf (fp,"\f");
      for (kr=0; kr < 3; ++kr) {
        fprintf (fp," Flux in region %d, group %d, page 2\n",kr,kg);
        j=0;
        do {
          k = j*nyN;
          i=8;
          do {
            fprintf (fp,"% 10.3e",dptr[k+i]);
          } while (++i < nyN);
          fprintf (fp,"\n");
        } while (++j < nxreg[kr]);
        dptr = &dptr[ntreg[kr]];
      }
      fprintf (fp,"\f");
    }
}
```

2 Multigroup Jacobian Routine

This routine calculates the coefficients of the Jacobian matrix for the multigroup neutron diffusion simulation and stores them in the appropriate locations. The Jacobian matrix is modified in the main routine prior to the system being solved according to the time step size and the multigroup eigenvalue (K effective). The routines *Coolant*, which adjusts the cross-sections and Jacobian coefficients due to changes in coolant density, and *AdjustK*, which adjusts the Jacobian coefficients due to changes in *K-effective*, are derivatives of this routine and will not be listed.

```
/*
This routine calculates the coefficients for the Neutronics Jacobian
that use the Neutron Multigroup Diffusion parameters and stores then
in the appropriate locations.
```

```
ng   number of groups - 3 regions (fuel, coolant, moderator)
Phi   neutron flux
Pre   precursors
DenC  density of the coolant
DenM  density of the moderator
Tf   fuel Temperature
H    enthalpy (coolant and moderator)
Df   heat Diffusion coefficient
wf   heat produced per fission /Cc(fuel)*density(fuel)
   GROUP CONSTANTS
Vg   group velocity       Vg[group]
NuEf   fission       cross section          NuEf[region][group]
Et   total          "    "
Etr  transport      "    "
Es   scattering cxoss section;  Es[region][group1][group2]
*/

#include <stdio.h>
#include <math.h>

int  nx,ntf,ntc,ntm,ntg,nXN,nPhi;
int  ntN,nyN,nxf,nxc,nxm,ng;
int  ntreg[3], nxreg[3];
double cxreg[3],dxreg[3];
double wf,beta1;
double *Vg,*NuEf[3],*Et[3],*Etr[3],**Es[3]; /* 3 regions */
double dy,dxf,dxc,dxm;
double cyy,cxf,cxc,cxm;

Flux (double ** A,int k0,double * B,double Keff,double Phi_set,int BC_N)
{
   int ntr;
   int i,j,k,kg,kr,ig,kband;
   int km1,km2,kp1,kp2,ns;
   double Ae,Aw,An,As,Ap,cx,Sf,dxr,ryr,dxb;
   double Aeb,Awb,Anb,Asb,Apb;
   double Sigtr,Sigt,Sigf,Sigs,Diff,Diffb,cxb;

   km1 = k0-1;
   km2 = k0-2;
   kp1 = k0+1;
```

```
kp2 = k0+2;
if (BC_N & 4)
  goto OneDim;

  /* top and bottom BCs */
ryr = 0.01 / dy;          /* H2O reflector */
if (BC_N & 1)
  ryr = 1.0 / dy;         /* Marshak BC */
if (BC_N & 2)
  ryr = 0;          /* reflective BC */

 for (kg=0; kg < ng; ++kg) {
  k = kg*ntN;                  /* index of position in array */
  for (kr=0; kr < 3; ++kr) {
     dxr = dxreg[kr];
     cx = cxreg[kr];
     ntr = ntreg[kr];
     Diff = 1.0 / (3.0 * Etr[kr][kg]);
     Sigt = Es[kr][kg][kg] - Et[kr][kg];
     Sigf = beta1 * NuEf[kr][kg] / Keff;
     if (kg == 0 && kr == 0)       /* fission neutrons appear in group 0 */
       Sigt += Sigf;
     Ae = Diff*cx;
     Aw = Ae;
     An = Diff*cyy;
     As = An;
     Ap = -Ae - Aw - An - As + Sigt;

     /* five central bands subject to Boundary Conditions */
     /* i=0 */
     if (kr) {
        /* j = 0 */
        A[km2][k] = Aw;
        A[k0][k]  = Ap - ryr;
        A[kp1][k] = An + As;
        A[kp2][k] = Ae;
        ++k;
        for (j=1; j < nyN-1; ++j) {
           A[km2][k] = Aw;
           A[km1][k] = As;
           A[k0][k]  = Ap;
```

```
      A[kp1][k] = An;
      A[kp2][k] = Ae;
      ++k;
    }
    /* j = nyN-1 */
    A[km2][k] = Aw;
    A[km1][k] = As + An;
    A[k0][k]  = Ap - ryr;
    A[kp2][k] = Ae;
    ++k;
  }
  else {
    A[k0][k]  = Ap - ryr;
    A[kp1][k] = An + As;
    A[kp2][k] = Ae + Aw; /* left reflective BC */
    ++k;
    for (j=1; j < nyN-1; ++j) {
      A[km1][k] = As;
      A[k0][k]  = Ap;
      A[kp1][k] = An;
      A[kp2][k] = Aw + Ae; /* left reflective BC */
      ++k;
    }
    A[km1][k] = As + An;
    A[k0][k]  = Ap - ryr;
    A[kp2][k] = Ae + Aw; /* left reflective BC */
    ++k;
  }
  for (i=1; i < nxreg[kr]-1; ++i) {
    A[km2][k] = Aw;
    A[k0][k]  = Ap - ryr;
    A[kp1][k] = An + As;
    A[kp2][k] = Ae;
    ++k;
    for (j=1; j < nyN-1; ++j) {
      A[km2][k] = Aw;
      A[km1][k] = As;
      A[k0][k]  = Ap;
      A[kp1][k] = An;
```

```
        A[kp2][k] = Ae;
        ++k;
      }
      A[km2][k] = Aw;
      A[km1][k] = As + An;
      A[k0][k]  = Ap - ryr;
      A[kp2][k] = Ae;
      ++k;
    }
/***** conservation of current BC_N *****/
    /* i = nxreg[kr]-1          BCs for last column */
    if (kr < 2) {
      Diffb = 1.0 / (3.0 * Etr[kr+1][kg]);
      dxb = dxreg[kr+1];
      cxb = 2.0 / (dxr + dxb);

      Awb = cxb * Diff / dxr;
      Aeb = cxb * Diffb / dxb;
      Anb = Diff * cyy;
      Asb = Anb;
      Apb = -Aeb - Awb - Anb - Asb + Sigt;

      /* store coefficients */
      A[km2][k] = Awb;
      A[k0][k]  = Apb - ryr;
      A[kp1][k] = Anb + Asb;
      A[kp2][k] = Aeb;
      ++k;
      for (j=1; j < nyN-1; ++j) {
        A[km2][k] = Awb;
        A[km1][k] = Asb;
        A[k0][k]  = Apb;
        A[kp1][k] = Anb;
        A[kp2][k] = Aeb;
        ++k;
      }
      A[km2][k] = Awb;
      A[km1][k] = Asb + Anb;
      A[k0][k]  = Apb - ryr;
      A[kp2][k] = Aeb;
      ++k;
```

```
    }
    else {
      A[km2][k] = Aw + Ae; /* right reflective BC */
      A[k0][k]  = Ap - ryr;
      A[kp1][k] = An + As;
      ++k;
      for (j=1; j < nyN-1; ++j) {
        A[km2][k] = Aw + Ae;        /* right reflective BC */
        A[km1][k] = As;
        A[k0][k]  = Ap;
        A[kp1][k] = An;
        ++k;
      }
      A[km2][k] = Aw + Ae; /* right reflective BC */
      A[km1][k] = As + An;
      A[k0][k]  = Ap - ryr;
      ++k;
    }
    /* group to group scattering and fission */
    i=0;     /* beginning of region */
    for (j=0; j < kr; ++j)
      i += ntreg[j];
      /* alternate k - ntr */
    ns = ntN*kg + i;
    for (ig=0; ig < ng; ++ig) {
      if (ig == kg) continue;
      if (ig > kg)
        kband = k0 + 2 + ig - kg;
      else
        kband = k0 - 2 + ig - kg;
      /* group ig to group kg */
      Sigs = Es[kr][kg][ig];
      if (kg == 0)
        Sigs += beta1 * NuEf[kr][ig] / Keff;
      CfillV (&A[kband][ns],Sigs,ntr);
    }
  }
}
/* fission heat source -- region 0 only (fuel) */
```

```
  for (kg=0; kg < ng; ++kg) {
    Sf = Phi_set * wf * NuEf[0][kg] / 2.43;
    CfillV (&A[k0+kg-ng-2][nPhi],Sf,ntf);
  }
  return (0);

    /* One Dimensional Calculation */
OneDim:
  for (kg=0; kg < ng; ++kg) {
    k = kg*ntN;                    /* index of position in array */
    for (kr=0; kr < 3; ++kr) {
      dxr = dxreg[kr];
      cx = cxreg[kr];
      ntr = ntreg[kr];
      Diff = 1.0 / (3.0 * Etr[kr][kg]);
      Sigt = Es[kr][kg][kg] - Et[kr][kg];
      if (kg == 0)
        Sigt += beta1 * NuEf[kr][kg] / Keff;

      Ae = Diff*cx;
      Aw = Ae;
      Ap = -Ae - Aw + Sigt;

      /* central bands subject to Boundary Conditions */
      /* i=0 */
      if (kr == 0) {
        A[k0][k]  = Ap;
        A[kp1][k] = Aw + Ae; /* left reflective BC */
        ++k;
      }
      else {
        A[km1][k] = Aw;
        A[k0][k]  = Ap;
        A[kp1][k] = Ae;
        ++k;
      }
      for (i=1; i < nxreg[kr]-1; ++i) {
        A[km1][k] = Aw;
        A[k0][k]  = Ap;
        A[kp1][k] = Ae;
        ++k;
```

```
    }
    /* i = nxreg[kr]-1          BCs for last column */
    if (kr < 2) {
      Diffb = 1.0 / (3.0 * Etr[kr+1][kg]);
      dxb = dxreg[kr+1];
      cxb = 2.0 / (dxr + dxb);
      Awb = cxb * Diff / dxr;
      Aeb = cxb * Diffb / dxb;
      Apb = -Aeb - Awb + Sigt;
      A[km1][k] = Awb;
      A[k0][k]  = Apb;
      A[kp1][k] = Aeb;
      ++k;
    }
    else {
      A[km1][k] = Aw + Ae;     /* right reflective BC */
      A[k0][k]  = Ap;
      ++k;
    }
    /* group to group scattering and fission */
    i=0;      /* beginning of region */
    for (j=0; j < kr; ++j)
      i += ntreg[j];
      /* alternate k - ntr */
    ns = ntN*kg + i;
    for (ig=0; ig < ng; ++ig) {
      if (ig == kg) continue;
      if (ig > kg)
        kband = k0 + 1 + ig - kg;
      else
        kband = k0 - 1 + ig - kg;
      /* group ig to group kg */
      Ap = Es[kr][kg][ig];
      if (kg == 0)
        Ap += beta1 * NuEf[kr][ig] / Keff;
      CfillV (&A[kband][ns],Ap,ntr);
    }
  }
}
```

```
  /* fission heat source -- region 0 only (fuel) */
  for (kg=0; kg < ng; ++kg) {
    Sf = Phi_set * wf * NuEf[0][kg] / 2.43;
    CfillV (&A[k0+kg-ng-1][nPhi],Sf,ntf);
  }
  return (0);
}
```

3 Thermal Conduction Routine

The Jacobian coefficients for thermal conduction in the fuel are calculated and stored in this routine.

```
/*
        Thermal conduction in the fuel

DenF  density of the fuel
DenC  density of the coolant
DenM  density of the moderator
Tf    fuel Temperature
Tc    coolant Temperature
Tm    moderator Temperature

The default right boundary is a fixed temperature otherwise,
the fuel temperature at the coolant boundary is handled in the
routine Fluid.

*/

#include <stdio.h>
#include <math.h>

int  nx,ntf,ntc,ntm,ntg,nXN;
int  ntN,nyN,nxf,nxc,nxm,ng;
double cxfuel,cyfuel;
double dy,dxf,dxc,dxm;
double cyy,cxf,cxc,cxm;

Thermal (double ** AT,int k0,double * B,int BC_N)
{
```

```
int i,j,k;
int km1,km2,kp1,kp2;
double Ae,Aw,An,As,Ap;

km1 = k0-1;
km2 = k0-2;
kp1 = k0+1;
kp2 = k0+2;

/* fuel temperature - thermal conduction */

Ae = cxfuel;
Aw = cxfuel;
An = cyfuel;
As = cyfuel;
if (BC_N & 4) {
   Ap = -Ae - Aw;
  goto OneDim;
}
else
   Ap = -Ae - Aw - An - As;

/* five central bands */
k = 0;
AT[k0][k]  = Ap;
AT[kp2][k] = Ae + Aw;          /* reflective left BC */
AT[kp1][k] = An + As;          /* bottom reflective */
++k;
for (j=1; j < nyN-1; ++j) {
  AT[km1][k] = As;
  AT[k0][k]  = Ap;
  AT[kp2][k] = Ae + Aw;    /* reflective left BC */
  AT[kp1][k] = An;
  ++k;
}
AT[km1][k] = An + As;          /* top reflective */
AT[k0][k]  = Ap;
AT[kp2][k] = Ae + Aw;          /* reflective left BC */
++k;
for (i=1; i < nxf-1; ++i) {
   AT[km2][k] = Aw;
  AT[k0][k]  = Ap;
```

```
AT[kp1][k] = An + As;     /* bottom reflective */
AT[kp2][k] = Ae;
++k;
for (j=1; j < nyN-1; ++j) {
   AT[km2][k] = Aw;
   AT[km1][k] = As;
   AT[k0][k]  = Ap;
   AT[kp1][k] = An;
   AT[kp2][k] = Ae;
   ++k;
 }
 AT[km2][k] = Aw;
 AT[km1][k] = An + As;    /* top reflective */
 AT[k0][k]  = Ap;
 AT[kp2][k] = Ae;
 ++k;
}
/* right BC - coolant or heat sink */
AT[km2][k] = Aw;
AT[k0][k]  = Ap;
AT[kp1][k] = An + As;       /* bottom reflective */
AT[kp2][k] = Ae;
++k;
for (j=1; j < nyN-1; ++j) {
  AT[km2][k] = Aw;
  AT[km1][k] = As;
  AT[k0][k]  = Ap;
  AT[kp1][k] = An;
  AT[kp2][k] = Ae;
  ++k;
 }
 AT[km2][k] = Aw;
 AT[km1][k] = An + As;      /* top reflective */
 AT[k0][k]  = Ap;
 AT[kp2][k] = Ae;
 return (0);
```

```
OneDim:
  k = 0;
  AT[k0][k]  = Ap;
  AT[kp1][k] = Ae + Aw;      /* reflective left BC */
  ++k;
  for (i=1; i < nxf-1; ++i) {
    AT[km1][k] = Aw;
    AT[k0][k]  = Ap;
    AT[kp1][k] = Ae;
    ++k;
  }
  /* right BC - coolant or heat sink */
  AT[km1][k] = Aw;
  AT[k0][k]  = Ap;
  AT[kp1][k] = Ae;
  ++k;
  return (0);
}
```

4 Thermalhydraulic Jacobian Routine

The coefficients of the thermalhydraulic Jacobian matrix are calculated and stored in this routine. Like the multigroup Jacobian, the thermalhydraulic Jacobian is modified before the system is solved but only according to the time step size. The coefficients for convective heat transfer in the coolant are calculated in this routine, which are then stored in the appropriate locations in the neutronics/heat-transfer Jacobian.

```
/*
      THERMALHYDRAULIC SUBROUTINE  for use with the reactor simulation
*/
#include <math.h>
#include <mathG.h>
/* external variables */
int     nxf,nxc,nyN,nyTH,ntc,ntf,ntTH;
double  dy,dxf,dxc,FT;
double  grav,G1;
double  cxc,cyy,cxv,cyv,cxh,cyh;
```

```
double  Gtx,Gty,cxfuel,cyfuel;
double  T_cold,*Tf;
double  Kc,Kf,alphaf,muPr;

Fluid (double ** A,double * B,double * Vx,double * Vy,double * DenC,
       double * VxBC,double * VyBC,double * Pf, double ** ATf,
       double * BTf,double ** ATcN,double * BTcN,double ** ATc,
       double * BTc,int BCs)
{
   int i,j,k,k1,k2,k3,kN;
   double Ux,Uz,Hx,Hy,Ae,Aw,An,As,Ap,
          Apx,Aex,Awx,Anx,Asx,
          Apy,Aey,Awy,Any,Asy,
          Aph,Aeh,Awh,Anh,Ash,
          Aep,Awp,Asp,Anp;
   double rho,Dpx,Dpy,alphac,den,*Tw;
   double fterm,cterm;

   /* conservation of thermal flux
        at fuel - coolant interface */

   An = cyfuel;
   As = cyfuel;
   Tw = &Tf[ntf-nyN];/* wall temperature */

   i = 0;
   k = ntf - nyN; /* points along boundary */
   rho = DenC[i++];   /* density at wall */
   alphac = muPr / rho;
   fterm = Kf * dxf / alphaf;
   cterm = Kc * dxc;
   den = fterm + cterm / alphac;
   Aw = 2 * Kf / (dxf * den);
   Ae = 2 * Kc / (dxc * den);
   Ap = -Ae - Aw - An - As;
   ATf[0][k] = Aw;
   ATf[2][k] = Ap;
   ATf[3][k] = An + As;
   ATf[4][k] = Ae;
   ++k;
   for (j=1; j < nyN-1; ++j) {
        rho = DenC[i++];
```

```
    alphac = muPr / rho;
    den = fterm + cterm / alphac;
    Aw = 2 * Kf / (dxf * den);
    Ae = 2 * Kc / (dxc * den);
    Ap = -Ae - Aw - An - As;
    ATf[0][k] = Aw;
    ATf[1][k] = As;
    ATf[2][k] = Ap;
    ATf[3][k] = An;
    ATf[4][k] = Ae;
    ++k;
}
rho = DenC[i++];
alphac = muPr / rho;
den = fterm + cterm / alphac;
Aw = 2 * Kf / (dxf * den);
Ae = 2 * Kc / (dxc * den);
Ap = -Ae - Aw - An - As;
ATf[0][k] = Aw;
ATf[1][k] = An + As;
ATf[2][k] = Ap;
ATf[4][k] = Ae;

if (BCs) goto Channel;

/*  Setup Jacobian Matices - Cavity */
i = 0;
do {
    j = 0;
    do {
       kN = i * nyN + j;
       k1 = i * nyTH + j;
       k2 = k1 + ntTH;
       k3 = k2 + ntTH;
       rho = DenC[k1];
       Dpx = 1.0 / (rho * dxc);   /* pressure gradient */
       Dpy = 1.0 / (rho * dy);
       Asp = rho * Gty;
       Anp = -rho * Gty;
       Aep = -rho * Gtx;
       Awp = rho * Gtx;
```

```
Hx = cxh/rho;
Hy = cyh/rho;
Ux = cxv/rho;
Uz = cyv/rho;
/* upwinding */
Ae = Dmax (-Vx[k1]/dxc, 0);
Aw = Dmax (Vx[k1]/dxc, 0);
An = Dmax (-Vy[k1]/dy, 0);
As = Dmax (Vy[k1]/dy, 0);
Aex = Ux + Ae;
Awx = Ux + Aw;
Anx = Uz + An;
Asx = Uz + As;
Aey = Aex;
Awy = Awx;
Any = Anx;
Asy = Asx;
Aeh = Hx + Ae;
Awh = Hx + Aw;
Anh = Hy + An;
Ash = Hy + As;
/* BOUNDARY CONDITIONS */
/* part 1, assign the coefficients */
if (i)
Awp = 0.5 * (rho + DenC[k1-nyTH]) * Gtx;
else {    /* left boundary */
Ux *= FT;
Aey = Ux + Ae;
Awy = 2.0 * (Ux + Aw);
Hx *= FT;
Aeh = Hx + Ae;
Awh = 2.0 * (Hx + Aw);
}
if (i < nxc-1)
Aep = -0.5 * (rho + DenC[k1+nyTH]) * Gtx;
else {    /* right boundary */
Awx = 0;
Aex = 0;
Anx = 0;
```

```
Asx = 0;
Ux *= FT;
Acy = 2.0 * (Ux + Ac);
Awy = Ux + Aw;
Hx *= FT;
Ach = 2.0 * (Hx + Ac);
Awh = Hx + Aw;
  }
  if (j)
Asp = 0.5 * (rho + DenC[k1-1]) * Gty;
  else {    /* bottom boundary - insulated */
Uz *= FT;
Anx = Uz + An;
Asx = 2.0 * (Uz + As);
Ash = 0;
  }
  if (j < nyTH-1)
Anp = -0.5 * (rho + DenC[k1+1]) * Gty;
  else {    /* top boundary - insulated */
Acy = 0;
Awy = 0;
Any = 0;
Asy = 0;
Uz *= FT;
Anx = 2.0 * (Uz + An);
Asx = Uz + As;
Anh = 0;
  }
  Apx = - Asx - Anx - Awx - Acx;
  Apy = - Asy - Any - Awy - Acy;
  Aph = - Ash - Anh - Awh - Ach;
  /* part 2, fill arrays */
  ATcN[0][kN] = Awh;  /* contiguous boundary */
  if (i) {
A[4][k1] = Awx;
A[4][k2] = Awy;
A[0][k3] = Awp;
  }
  else {    /* left boundary, fuel */
```

```
B[k1] = VxBC[j] * Awx;
B[k3] = VxBC[j] * Awp;
  }
  if (i < nxc-1) {
A[8][k1] = Aex;
A[11][k1] = Dpx;   /* (dP/dx)/rho */
A[12][k1] = -Dpx;
A[8][k2] = Aey;
ATcN[4][kN] = Aeh;
  }
  else {     /* right boundary, fixed temperature */
BTcN[kN] = Aeh * T_cold;
  }
  if (j) {
A[5][k1] = Asx;
A[5][k2] = Asy;
A[2][k3] = Asp;
ATcN[1][kN] = Ash;
  }
  else {     /* bottom boundary, insulated */
B[k2] = VyBC[i] * Asy;
B[k3] = VyBC[i] * Asp;  /* inlet velocity */
  }
  if (j < nyTH-1) {
A[7][k1] = Anx;
A[7][k2] = Any;
A[9][k2] = Dpy;   /* (dP/dy)/rho */
A[10][k2] = -Dpy;
ATcN[3][kN] = Anh;
B[k2] = grav;
  }  /* top boundary, insulated */
  A[6][k1] = Apx;
  A[6][k2] = Apy;
  A[6][k3] = 0;
  ATcN[2][kN] = Aph;
  A[1][k3] = Aep;
  A[3][k3] = Anp;
```

```
        } while (++j < nyTH);
    } while (++i < nxc);
    return(0);

    /*  Setup Jacobian Matrix */
Channel:/* flow channel */
    i = 0;
    do {
        j = 0;
        do {
            kN = i * nyN + j;
            k1 = i * nyTH + j;
            k2 = k1 + ntTH;
            k3 = k2 + ntTH;
            rho = DenC[k1];
            Dpx = 1.0 / (rho * dxc);    /* pressure gradient */
            Dpy = 1.0 / (rho * dy);
            Asp = rho * Gty;
            Anp = -rho * Gty;
            Aep = -rho * Gtx;
            Awp = rho * Gtx;
            Hx = cxh/rho;
            Hy = cyh/rho;
            Ux = cxv/rho;
            Uz = cyv/rho;
            /* upwinding */
            Ae = Dmax (-Vx[k1]/dxc, 0);
            Aw = Dmax (Vx[k1]/dxc, 0);
            An = Dmax (-Vy[k1]/dy, 0);
            As = Dmax (Vy[k1]/dy, 0);
            Aex = Ux + Ae;
            Awx = Ux + Aw;
            Anx = Uz + An;
            Asx = Uz + As;
            Aey = Aex;
            Awy = Awx;
            Any = Anx;
            Asy = Asx;
            Aeh = Hx + Ae;
```

```
  Awh = Hx + Aw;
  Anh = Hy + An;
  Ash = Hy + As;
  /* BOUNDARY CONDITIONS */
  /* part 1, assign the coefficients */
  if (i)
Awp = 0.5 * (rho + DenC[k1-nyTH]) * Gtx;
  else {     /* left boundary */
Ux *= FT;
Aey = Ux + Ae;
Awy = 2.0 * (Ux + Aw);
if (j < nyN) {   /* hot wall */
   Hx *= FT;
   Aeh = Hx + Ae;
   Awh = 2.0 * (Hx + Aw);
}
else {/* insulated */
   Awh = 0;
}
   }
   if (i < nxc-1)
 Aep = -0.5 * (rho + DenC[k1+nyTH]) * Gtx;
   else {     /* right boundary */
 Awx = 0;/* zero flow */
 Aex = 0;
 Anx = 0;
 Asx = 0;
 Ux *= FT;
 Aey = 2.0 * (Ux + Ae);
 Awy = Ux + Aw;
 Aeh = 0;/* insulated */
   }
   if (j < nyTH-1)
 Anp = -0.5 * (rho + DenC[k1+1]) * Gty;
   else {     /* top boundary */
 Anh = 0;/* insulated */
 Anx = 0;
 Any = 0;
   }
```

```
  if (j)
Asp = 0.5 * (rho + DenC[k1-1]) * Gty;
  else {/* bottom boundary */
Asy = 0;/* dVy/dy = 0 */
Anp = 0;/* dP/dy = 0 */
Asp = 0;
  }
  Apx = - Asx - Anx - Awx - Aex;
  Apy = - Asy - Any - Awy - Aey;
  Aph = - Ash - Anh - Awh - Aeh;

  /* part 2, fill arrays */
  if (j < nyN)
ATcN[0][kN] = Awh;  /* contiguous boundary */
  if (i) {
A[4][k1]  = Awx;
A[4][k2]  = Awy;
A[0][k3]  = Awp;
ATc[0][k1] = Awh;
  }
  else {    /* left boundary, fuel */
B[k1]  = VxBC[j] * Awx;
B[k3]  = VxBC[j] * Awp;
BTc[k1] = Awh * Tw[j];
  }
  if (i < nxc-1) {
A[8][k1]  = Aex;
A[11][k1] = Dpx;   /* (dP/dx)/rho */
A[12][k1] = -Dpx;
A[8][k2]  = Aey;
if (j < nyN)
  ATcN[4][kN] = Aeh;
ATc[4][k1] = Aeh;
  }  /* right boundary */
  if (j) {
A[5][k1] = Asx;
A[5][k2] = Asy;
A[2][k3] = Asp;
if (j < nyN)
  ATcN[1][kN] = Ash;
```

```
      ATc[1][k1] = Ash;
        }
        else {      /* bottom boundary */
      BTcN[kN] = Ash * T_cold; /* fixed temperature */
      BTc[k1] = Ash * T_cold; /* fixed temperature */
        }
        if (j < nyTH-1) {
      A[7][k1] = Anx;
      A[7][k2] = Any;
      if (j < nyN-1)
        ATcN[3][kN] = Anh;
      ATc[3][k1] = Anh;
      A[9][k2] = Dpy;   /* (dP/dy)/rho */
      A[10][k2] = -Dpy;
      B[k2] = grav;
        }
        else {      /* top boundary */
      A[9][k2] = Dpy; /* (dP/dy)/rho */
      B[k2] = grav - Dpy * Pf[i];    /* outlet pressure */
        }
        A[6][k1]  = Apx;
        A[6][k2]  = Apy;
        A[6][k3]  = 0;
        if (j < nyN)
      ATcN[2][kN] = Aph;
        ATc[2][k1] = Aph;
        A[1][k3]  = Aep;
        A[3][k3]  = Anp;
      } while (++j < nyTH);
    } while (++i < nxc);
    return(0);
}
```

5 Integration Routines

The integration routines that are used for the time integration of the neutronics/heat-transfer problem are of the *Semi-implicit Runge Kutta* (RK-SI) variety. A slightly different implementation is used for the static and transient simulations due to the treatment of delayed neutron precursors. For the static calculation, it is

assumed that the delayed neutron precursors are not changing in time and hence are not calculated using time integration but instead their steady state concentrations are calculated using neutron flux. For the transient calculation, their concentrations are explicitly calculated in time.

The Gauss-Seidel version of the (RK-SI) integration routine is most commonly used in the static calculations and the LU-decomposition version of the (RK-SI) routine is most commonly used for the transient calculation therefore, these are the two routines that are listed.

a) Semi-Implicit Runge Kutta Routine: Gauss-Seidel Version for Static Calculations

```
/*
    Semi Implicit Runge Kutta solver - GSSOR version
*/

#include <mathG.h>
#include <math.h>

RKSI_GS (Y,Yout,n,h,A,band,nbA,k0A,B,work,tol,itmax,it)
double *Y,h,*Yout,**A,*B,**work,tol;
int n,*band,nbA,k0A,itmax,*it;
{
   double a1=0.788675134595,b1=-1.15470053838,w1=0.75,w2=0.25;
   double *K1=work[0],*K2=work[1],*B1=work[2],*B2=work[3],
     *A0=work[4],*A1=work[5],dtmp,a1r;
   int k,err;

   a1r = -1.0 / a1;
   CopyV (A0,A[k0A],n);            /* store central band */
   V_MbmulV (B1,n,A,band,nbA,k0A,Y,n);
   V_addV (B1,B,n);
   V_mulC (B1,a1r,n);
   dtmp = a1r / h;
   V_addC (A[k0A],dtmp,n);
   CopyV (A1,A[k0A],n);            /* store new central band */

   err = GSSORb (A,band,nbA,k0A,K1,B1,n,tol,itmax,it);
   if (err) return (1);
```

```
  V_VaddCV (B1,Y,b1,K1,n);
  CopyV (A[k0A],A0,n);              /* restore central band */
  V_MbmulV (B2,n,A,band,nbA,k0A,B1,n);
  V_addV (B2,B,n);
  V_mulC (B2,a1r,n);
  CopyV (A[k0A],A1,n);              /* restore new central band */
  err = GSSORb (A,band,nbA,k0A,K2,B2,n,tol,itmax,it);
  if (err) return (1);
  V_VaddCVaddCV (Yout,Y,w1,K1,w2,K2,n);
  return (0);
}
```

b)) Semi-Implicit Runge Kutta Routine: LU-decomposition Version for Transient Calculations

```
/*
    Semi Implicit Runge Kutta solver - LUb version
    adapted for neutron kinetics - partitioned
*/
#include <mathG.h>
#include <math.h>
#include <stdio.h>
int rksilu_init=1,*Piv;
double **Ap,*Apptr,**PhiX1,**PhiK1,**PhiK2,*CnX2[6],*CnK1[6],*CnK2[6],*CnB2[6];
double *K1,*K12,*K2,*K22,*BK,*X1RK,*X2RK,*B2RK,**AfRK;
double *NuEf[3],*Vg,*NuEfPhi,Vg0;
double lambda[6],beta[6];
int ng,ntf,ntN;
RKSI_LU_N (double * X1,double * X1out,int nx1,double * X2,double * X2out,
    int nx2,double h,double ** A,int * band,int nbA,int k0A,double * B,
    double ** work,double tol,int itmax,int * it,int BC_N,int ssg,
    double Keff)
{
  double a1=0.788675134595,b1= -1.15470053838,w1=0.75,w2=0.25;
  double dump,a1r,*Vf;
  double a12[6],a21[6],a22[6];
  int k,kg,err,nb,nb1,bl= -band[0],bu= band[nbA-1];
```

```
if (rksilu_init) {
  K1  = work[0];
 K12 = &K1[nx1];
 K2  = work[1];
 K22 = &K2[nx1];
 BK  = work[2];
 B2RK  = &BK[nx1];
 X1RK = work[3];
 X2RK = &X1RK[nx1];
 Apptr = (double *)calloc(nx1*nbA,sizeof(double));
 Ap = (double **)calloc(nbA,sizeof(double *));
 PhiX1 = (double **)calloc(ng,sizeof(double *));
 PhiK1 = (double **)calloc(ng,sizeof(double *));
 PhiK2 = (double **)calloc(ng,sizeof(double *));
 if (Ap == NULL || Apptr == NULL || PhiX1 == NULL ||
       PhiK1 == NULL || PhiK2 == NULL)
   error_exit (" RKSI_LU_N memory allocation error",2);
 for (k=0; k < nbA; ++k)
   Ap[k] = &Apptr[k*nx1];
 for (k=0; k < ng; ++k) {
   PhiX1[k] = &X1RK[k*ntN];
   PhiK1[k] = &K1[k*ntN];
   PhiK2[k] = &K2[k*ntN];
 }
 for (k=0; k < 6; ++k) {
   CnX2[k] = &X2RK[k*ntf];
   CnB2[k] = &B2RK[k*ntf];
   CnK1[k] = &K12[k*ntf];
   CnK2[k] = &K22[k*ntf];
 }
  if (ssg)
    Vg0 = Vg[ssg];
  else
    Vg0 = Vg[0];
 AfRK = Mlubmalloc (bl,bu,nx1);
 Piv = (int *)calloc(nx1,sizeof(int));
 if (AfRK == NULL || Piv == NULL)
```

```
    error_exit (" RKSI_LU_N memory allocation error",2);
  rksilu_init = 0;
}
Vf = NuEfPhi;
a1r = -1.0 / a1;
CopyM (Ap,A,nbA,nx1);
V_addC (Ap[k0A],a1r/h,nx1);
for (k=0; k < 6; ++k) {
  a22[k] = 1.0 / (lambda[k] - a1r / h);   /* - 1 / A22 */
  a12[k] = Vg0 * lambda[k] * a22[k];  /* - A12 / A22 */
  a21[k] = beta[k] * a22[k];
 }
 if (BC_N & 4)
   nb1 = 1;
 else
   nb1 = 2;
/* A11 - A12 / A22 * A21 */
 for (kg=0; kg < ng; ++kg) {
   if (kg)
      nb = k0A + nb1 + kg;
   else
      nb = k0A;
   dtmp = 0;
   for (k=0; k < 6; ++k)
      dtmp += beta[k] * NuEf[0][kg] / Keff * a12[k];
   V_addC (Ap[nb],dtmp,ntf);
 }
 ZeroMlub (AfRK,bl,bu,nx1,nx1);
 BtoF (Ap,band,nbA,AfRK,nx1,nx1);
 err = LUb (AfRK,bl,bu,Piv,nx1);
 if (err) return (1);
   /* stage one */
 CopyV (X1RK,X1,nx1);
 CopyV (X2RK,X2,nx2);
```

```
/* BK = A11 * X1  + A12 * X2 + B1 */
V_MbmulV (BK,nx1,A,band,nbA,k0A,X1RK,nx1);    /* A11 * X1 */
ZeroV (Vf,ntf);
for (k=0; k < 6; ++k)
  V_addCV (Vf,Vg0*lambda[k],CnX2[k],ntf);/* + A12 * X2 */
V_addV (BK,Vf,ntf);
V_addV (BK,B,nx1);                    /* + B1 */
V_mulC (BK,a1r,nx1);                  /* = B1' */

/* B2 = A21 * X1 + A22 * X2 */
ZeroV (NuEfPhi,ntf);
for (kg=0; kg < ng; ++kg)
  V_addCV (NuEfPhi,NuEf[0][kg]/Keff,PhiX1[kg],ntf);      /* A21 * X1 */
for (k=0; k < 6; ++k) {
   V_CVaddCV (CnB2[k],beta[k],NuEfPhi,-lambda[k],CnX2[k],ntf);
  V_mulC (CnB2[k],a1r,ntf);       /* B2' = A21 * X1 + A22 * X2 */
  V_addCV (BK,a12[k],CnB2[k],ntf);
}

err = LUbSolve (AfRK,bl,bu,K1,BK,Piv,nx1);
if (err) return (1);

/* K1[2] = 1/A22 (B2 - A21 * X1) */
ZeroV (NuEfPhi,ntf);
for (kg=0; kg < ng; ++kg)
  V_addCV (NuEfPhi,NuEf[0][kg]/Keff,PhiK1[kg],ntf);
for (k=0; k < 6; ++k)
  V_CVaddCV (CnK1[k],-a22[k],CnB2[k],a21[k],NuEfPhi,ntf);

 /* stage two */

V_VaddCV (X1RK,X1,b1,K1,nx1);
V_VaddCV (X2RK,X2,b1,K12,nx2);

/* BK = A11 * X1  + A12 * X2 + B1 */
V_MbmulV (BK,nx1,A,band,nbA,k0A,X1RK,nx1);    /* A11 * X1 */
ZeroV (Vf,ntf);
for (k=0; k < 6; ++k)
  V_addCV (Vf,Vg0*lambda[k],CnX2[k],ntf);/* + A12 * X2 */
V_addV (BK,Vf,ntf);
V_addV (BK,B,nx1);                    /* + B1 */
V_mulC (BK,a1r,nx1);
```

```
  /* B2 = A21 * X1 + A22 * X2 */
  ZeroV (NuEfPhi,ntf);
  for (kg=0; kg < ng; ++kg)
    V_addCV (NuEfPhi,NuEf[0][kg]/Keff,PhiX1[kg],ntf);
  for (k=0; k < 6; ++k) {
     V_CVaddCV (CnB2[k],beta[k],NuEfPhi,-lambda[k],CnX2[k],ntf);
    V_mulC (CnB2[k],a1r,ntf);       /* B2' = A21 * X1 + A22 * X2 */
    V_addCV (BK,a12[k],CnB2[k],ntf);
  }
  err = LUbSolve (AfRK,bl,bu,K2,BK,Piv,nx1);
  if (err) return (1);
  /* K1[2] = 1/A22 (B2 - A21 * X1) */
  ZeroV (NuEfPhi,ntf);
  for (kg=0; kg < ng; ++kg)
    V_addCV (NuEfPhi,NuEf[0][kg]/Keff,PhiK2[kg],ntf);
  for (k=0; k < 6; ++k)
    V_CVaddCV (CnK2[k],-a22[k],CnB2[k],a21[k],NuEfPhi,ntf);
  V_VaddCVaddCV (X1out,X1,w1,K1,w2,K2,nx1);
  V_VaddCVaddCV (X2out,X2,w1,K12,w2,K22,nx2);
  return (0);
}
```

References

1. Adebiyi S.A.
The Hierarchy and Geometry of Fission Reactor Dynamics
PhD Thesis, Department of Engineering Physics, McMaster University, 1988.

2. Agresta J. & L.B. Borst
Space Dependent Prompt Kinetics of a Subcritical Reactor
Nucl. Sci. and Eng. 7, 64-68, 1960.

3. Alcouffe R.E. & R.W. Albrecht
A Generalization of the Finite Difference Approximation Method with an Application to Space-Time Nuclear Reactor Kinetics
Nucl. Sci. and Eng. 39, 1, 1970.

4. Ames W.F.
Numerical Methods for Partial Differential Equations
Academic Press, 1977.

5. Anderson D.A., J.H. Tannehill & R.H. Pletcher
Computational Fluid Mechanics and Heat Transfer
Hemisphere Publishing Corp., 1984.

6. Andrews J.B. & K.F. Hansen
Numerical Solution of the Time-Dependent Multigroup Diffusion Equations
Nucl. Sci. and Eng. 31, 304, 1968.

7. Askew J.R., F.J. Fayers & P.B. Kemshell
A General Description of the Lattice Code WIMS
Journal of British Nuclear Engineering, pp. 564-585, 1964.

8. Bell G.I. & S. Glasstone
Nuclear Reactor Theory
Van Nostrand Reinhold Co., 1970.

9. Bird R.B., W.E. Stewart & E.N. Lightfoot
Transport Phenomena
J. Wiley & Sons Inc., 1960.

10. Buckner M.R. & J.W. Stewart
Multidimensional Space-Time Nuclear Reactor Kinetics Studies - Part I: Theoretical
Nucl. Sci. & Eng. 59, 289-297, 1976.

11. Bunch J.R. & D.J. Rose, Ed..
Sparse Matrix Computations
Academic Press, 1976.

12. Butcher J.C.
Implicit Runge-Kutta Processes
Math. Comp. 18, pp. 50, 1964.

13. Butkov E.
Mathematical Physics
Addison-Wesley, 1968.

14. Calahan D.A.
A Stable, Accurate Method of Numerical Integration for Nonlinear Systems
Proc. of the IEEE, 56, pp. 744, 1968.

15. Carnahan B., H.A. Luther & J.O. Wilkes
Applied Numerical Methods
John Wiley & Sons Inc., 1969.

16. Carver M.B. & H.W. Hinds
The Method of Lines and the Advective Equation
AECL-6155, 1978.

17. Carver M.B.
Method of Lines Solution of Partial Differential Equations: Some Standard and Devious Applications
ANS Topical Meeting on Computational Methods in Nuclear Engineering, 23-25 April, 1979

18. Carver M.B.
Pseudo Characteristic Method of Lines Solution of Conservation Equations
J. of Computational Physics, **35**, 57, 1980.

19. Carver M.B.
Pseudo Characteristic Method of Lines Solution of First-Order Hyperbolic Equation Systems
OECD/CSNI Conference on Two Phase Flow Computation, Pasadena, Calif., 23-25 March 1981

20. Chatoorgoon, V.
SPORTS - A Simple Non-Linear Thermalhydraulic Stability Code
Nucl. Eng. and Design **93**,, 1986. p.51-67 or AECL - 9214

21. Chorin A.J.
A Numerical Method for Solving Incompressible Viscious Flow Problems
J. of Comp. Physics, **2**, 12-26, 1967.

22. Chorin A.J.
Numerical Solution ot the Navier-Stokes Equations
Math. Comp., **22**, 745, 1968.

23. Chow H.C. & M.H.M. Roshd
SHETAN - a three dimensional intergal transport code for reactor analysis.
AECL 6878, 1980.

24. Dahlquist G.& Åke Björck
Numerical Methods
Prentice Hall, 1974.

25. Dastur A.R. & D.B. Buss
Space-Time Kinetics of CANDU Systems
AECL-5181, 1975.

26. Dastur A.R. & D.B. Buss
MULTICELL - a 3D Program for the Simulation of Reactivity Devices in CANDU Reactors
AECL-7544, 1983.

27. De Vahl Davis,G. & I.P. Jones
Natural Convection in a Square Cavity: A Comparison Exercise
Int. Journal for Numerical Methods in Fluids, **3**, pp 227-248, 1983.

28. De Vahl Davis,G.
Natural Convection in a Square Cavity: A Benchmark Numerical Solution
Int. Journal for Numerical Methods in Fluids, **3**, pp 249-263, 1983.

29. Devooght J. & E. Mund
Generalized Quasi-Static Method for Nuclear Reactor Space-Time Kinetics
Nucl. Sci. and Eng. **76**, 10, 1980.

30. Donnelly J.V.
Progress in the Development of WIMS at Chalk River
6th Annual CNS Conference, 1985.

31. Donnelly J.V.
WIMS-CRNL: A User's Manual for the Chalk River Version of WIMS
AECL-8955, 1986.

32. Duderstadt J.J. & L.J. Hamilton
Nuclear Reactor Analysis
John Wiley & Sons, 1976.

33. Eisenstat S.C.
Efficient Implementation of a Class of Preconditioned Conjugate Gradient Methods
SIAM J. Sci. Stat. Comp. **2**, no. 1, 1981.

34. Ellis R.J., P.A. Carlson & H.J. Smith
Comparisons of the Steady State and Transient Thermal Behaviors of UO_2, U-Al and U_3 Si-Al MAPLE Research Reactor Fuels
Proc. of the 14th Annual Nuclear Simulation Symposium, Pinawa, Manitoba, 1988.

35. Ellis R.J., H.J. Smith & P.A. Carlson
TANK - A Computer Code for the Two Dimensional Modelling of Transient Behavior in Research Reactors
Proc. of the 28th Annual CNA/ 9th Annual CNS Conference, Winnipeg, Manitoba, 1988.

36. Foulke L.R.
A Modal Expansion Technique for Space-Time Reactor Kinetics
Ph.D. Thesis, Department of Nuclear Engineering, MIT, 1966.

37. Foulke L.R. & E.P. Gyftopoulos
Application of the Natural Mode Approximations to Space-Time Reactor Problems
Nucl. Sci. & Eng. **30**, 419-435, 1967.

38. Fowler T.B., D.R. Vondy & G.W. Cunningham
Nuclear Reactor Core Analysis Code: CITATION
ORNL-TM-2496, Revision 2, Oak Ridge National Laboratory, 1971.

39. Fuller E.L. & D.A. Meneley
Weighted-Residual Methods in Space-Dependent Reactor Dynamics
Nucl. Sci. and Eng. **40**, 206-233, 1970.

40. Garbow B.S. et al
Matrix Eigensystem Routines: EISPACK Guide Extension,
Springer-Verlag, 1972.

41. Garland W.J.
Basic Equations for Thermalhydraulic System Analysis
Dept. of Engineering Physics, McMaster University

42. Garland W.J. and B.J. Hand
Simple Functions for the Fast Approximation of Light Water Thermodynamic Properties
Nuclear Engineering and Design, to be published., 1989.

43. Garland W.J. and J.D. Hoskins
Approximate Functions for the Fast Calculation of Light Water Properties at Saturation
Int. Journal of Multiphase Flow, **14**, No.3, pp 333-348, 1988.

44. Garland W.J. and R. Sollychin
The Rate Form of the Equation of State for Thermalhydraulic Systems: Numerical Considerations
Eng. Comput., 4, December, 1987.

45. Garland W.J. and R. Sollychin
Generalized Rate Form of the Equation of State for Thermalhydraulics
Dept. of Engineering Physics, McMaster University

46. Gold M.
SMOKIN - A Family of Codes for Reactor Space-Time Neutronics Calculations Based on Modal Kinetics - Theory Manual
Report No. 90133, Nuclear Safety Department, Ontario Hydro, September, 1990.

47. Golub G.H. and C.F. Van Loan
Matrix Computations
The Johns Hopkins University Press, 1983.

48. Gosman A.D., W.M. Pun, A.K. Runchal, D.B. Spalding, & M. Wolfstein
Heat and Mass Transfer in Recirculating Flows
Academic Press, 1969.

49. Guckenheimer, J. & P. Holmes
Nonlinear Oscillations, Dynamical Systems & Bifurcation of Vector Fields
Springer Verlag, 1983.

50. Hageman L.A. and J.B. Yasinsky
Comparison of Alternating Direction Time Differencing Methods with Other Implicit Methods fc the Solution of the Neutron Group Diffusion Equations
Nucl. Sci. and Eng. 38, 8-32, 1969.

51. Hageman L.A. and D.M. Young
Applied Iterative Methods
Academic Press, 1981.

52. Hansen K.F. & S.R. Johnson
GAKIN, A Program for the Solution of the One-Dimensional Multigroup, Space-Time Dependent Diffusion Equations
GA-7543, General Atomic, 1967.

53. Harlow F.H. & J.E. Welch
Numerical Calculation of Time-Dependent Viscous Incompressible Flow of Fluid with Free Surface
GA-7543, General Atomic, 1967.

54. Henry A.F.
The Application of Reactor Kinetics to the Analysis of Experiments
Nucl. Sci. and Eng. 3, 52, 1958.

55. Henry A.F.
Review of Computational Methods for Space-Time Kinetics
Proceedings of the Conference on the Effective Use of Computers in Nuclear Industry, Knoxville, Tenn., 1969.

56. Henry A.F.
Nuclear Reactor Analysis
The M.I.T. Press, 1975.

57. Henry A.F. & K.S. Smith
Modern Nodal Methods for Light Water Reactors
MIT Course notes,, June 1989.

58. Hestenes M.R. & E. Stiefel
Method of Conjugate Gradients for Solving Linear Systems
J. Res. Nat. Bur. Std. 49, 409, 1952.

59. Kailath T.
Linear Systems
Prentice Hall, 1980.

60. Kaplan S.
The Property of Finality and the Analysis of Problems in Reactor Space-Time Kinetics by Various Modal Expansions
Nucl. Sci. and Eng. 9, 357-361, 1961.

61. Kaplan S.
Synthesis Methods in Reactor Analysis
Advances in Nuclear Science and Technology
P.R. Greebler, Ed., 3, Academic Press, 1965.

62. Kelber C.N., L.C. Just & N.F. Morehouse Jr.
The Solution of Many Region Reactor Kinetics Problems on an Analog Computer
Nucl. Sci. and Eng. 11, 285-289, 1961.

63. Kugler G. & A.R. Dastur
Accuracy of the Improved Quasistatic Space-Time Method Checked with Experiment
AECL-5553, 1976.

64. Laidus L. & J.H. Seinfeld
Numerical Solution of Ordinary Differential Equations
Academic Press, 1971.

65. Lawrence R.D.
Progress in Nodal Methods for the Solution of the Neutron Diffusion and Transport Equations
Prog. in Nucl. Eng., 17, No. 3, 271-301, 1986.

66. Lee C.E. & S. Rottler
Analytic Solutions of the Multigroup Space-Time Reactor Kinetics Equations - I
Ann. Nucl. Energy, 13, No. 5, 245-268, 1986.

67. Lewis E.E. & W.F. Miller Jr.
Computational Methods of Neutron Transport
John Wiley & Sons Inc., 1984.

68. Lin M.R., S.Prawirosoehardjo, D.F.Rennick, E.Wessman, R.Blain, & J.M.Wilson
FIREBIRD-III Program Description
AECL-7533, 1979.

69. March-Leuba J., D.G. Cacuci & R.B. Perez
Universality and Aperiodic Behavior of Nuclear Reactors
Nucl. Sci. and Eng., 86, 401, 1984.

70. March-Leuba J., D.G. Cacuci & R.B. Perez
Nonlinear Dynamics and Stability of Boiling Water Reactors
Nucl. Sci. and Eng., 93, 111, 1986.

71. McDonnell F.N., A.P. Baudouin, P.M. Garvey, & J.C. Luxat
CANDU Reactor Kinetics Benchmark Activity
Nucl. Sci. and Eng. 64, 95, 1977.

72. Meuller A. & M.R. Wagner
Use of Collision Probability Method for Multidimensional Neutron Flux Calculations
Trans. Am. Nucl. Soc., 15, No.1,, 1972.

73. Mills P.J. & S.Y. Shin
SPORTS-M Prediction of the Transition from Forced to Natural Convection in a MAPLE Reactor
15 Annual Nuclear Simulation Symposium, Sheridan Park, Mississauga, Ontario,, 1989.

74. Moin P. & J. Kim
On the Numerical Solution of Time-Dependent Viscous Incompressible Fluid Flows Involving Solid Boundaries
J. of Comp. Physics, **35**, pp 381, 1980.

75. Nahavandi A.N. & R.F. Von Hollen
A Digital Computer Solution for Space-Dependent Neutron Kinetics Equations
Nucl. Sci. and Eng. **18**, 335-350, 1964.

76. Ott K.O. & D.A. Meneley
Accuracy of the Quasistatic Treatment of Spatial Reactor Kinetics
Nucl. Sci. and Eng. **36**, 402, 1969.

77. Ott K.O. & R.J. Neuhold
Nuclear Reactor Dynamics
American Nuclear Society, 1985.

78. Patankar S.V. and D.B. Sterling
A Calculation Procedure for Heat, Mass and Momentum Transfer in Three-Dimensional Parabolic Flows
International Journal of Heat and Mass Tranfer, **15**, pp 1787-1806, 1972.

79. Patankar S.V.
Numerical Heat Transfer and Fluid Flow
Hemisphere Publishing,, 1980.

80. Patankar S.V.
A Calculation Procedure for Two-Dimensional Elliptic Situations
Numerical Heat Transfer, **4**, pp 409-425, 1981.

81. Peaceman, D.W. & H.H. Rachford Jr.
J. Soc. ind. appl. Math., **3**, 42, 1955.

82. Phillips G.J. & J. Griffons
LATREP Users Manual
AECL - 3857, 1971.

83. Porsching T.A., J.H. Murphy, and J.A. Redfield
Stable Numerical Integration of Conservation Equations for Hydraulic Networks
Nucl. Sc. and Eng., 43, 218-225,, 1971.

84. Press W.H., B.P. Flannery, S.A. Teukolsky and W.T. Vetterling
Numerical Recipes in C: The Art of Scientific Computing
Cambridge University Press, 1988.

85. Raithby G.D. and G.E. Schneider
Numerical Solution of Problems in Incompressible Flow: Treatment of the Velocity-Pressure Coupling
Numerical Heat Transfer, **2**, pp 417-440,, 1979.

86. Reed W.H. & K.F. Hansen
Alternating Direction Methods for the Reactor Kinetics Equations
Nucl. Sci. & Eng. **41**, 431-442, 1970.

87. Riese J.W., G. Collier & C.E. Rieck
VARI-QUIR: A Two-Dimensional, Time-Dependent Multigroup Diffusion Code
WANL-TNR-133-Revised, Westinghouse Astronuclear Laboratory, 1965.

88. Roshd M.H.M. & H.C. Chow
GETRANS: A Two Dimensional Integral Transport Code
AECL - 6359, 1978.

89. Saad Y., & M.H. Schultz
GMRES: A Generalized Minimal Residual Algorithm for Solving Nonsymmetric Linear Systems
SIAM J. Sci. Stat. Comp. 7, no. 3,, 1986.

90. Saphier D.
A Hybrid Method for Solving the Reactor Space-Time Dependent Equations
Nucl. Sci. and Eng. 47, 275-289, 1972.

91. Shih T.M. & C.H. Tan
Equivalence of Artificial Compressibility Method and Penalty Function Method
Num. Heat Transfer, Part B, 15, pp.127-130, 1989.

92. Smith B.T. et al
Matrix Eigensystem Routines: EISPACK Guide, 2nd ed.,
Springer-Verlag, 1976.

93. Soh W.Y.
Time-Marching Solution of Incompressible Navier-Stokes Equations for Internal Flow
J. of Comp. Physics, 70, pp 232, 1987.

94. Soh W.Y. & J.W. Goodrich
Unsteady Solution of Incompressible Navier-Stokes Equations
J. of Comp. Physics, 79, pp 113, 1988.

95. Stacey W.M. Jr.
Space-Time Nuclear Reactor Kinetics
Academic Press, 1969.

96. Stewart J.W.
Recent Development of Methods for Multidimensional Reactor Dyanmics
Conference on Mathematical Models and Computational Techniques for Analysis of Nuclear Systems, 1973.

97. Temitope A.T. & A.F. Henry
Perturbation Theory Based on a Nodal Model
Nucl. Sci. and Eng. 92, 34-41, 1986.

98. Tsutomu H. & J. Wakabayashi
New Approximate Solution of Space and Energy Dependent Reactor Kinetics
Nucl. Sci. & Eng. 23, 170-182, 1965.

99. Tuttle R.J.
Delayed Neutron Data for Reactor Physics Analysis
Nucl. Sci. and Eng. 56,, 1975.

100. Van Doormaal J.P. and G.B. Raithby
Enhancements of the SIMPLE Method for Predicting Incompressible Flow
Numerical Heat Transfer, 7, pp 147-163, 1984.

101. Varga R.S.
Matrix Iterative Analysis
Prentice-Hall, 1962.

102. Vatsya S.R.
An Iterative Method to Solve Large Sparse Linear Systems
Presented at the 15th annual Nuclear Simulation Symposium, Sheridan Park Conference Center, Mississaugua, Ontario, 1989.

103. Vigil J.C.
3DDT, A Three Dimensional Multigroup Diffusion-Burnup Program
Los Alamos Scientific Laboratory, LA 4396, 1970.

104. Wachspress E.L.
Iterative Solution of Elliptic Systems
Prentice-Hall, 1966.

105. Westlake J.R.
A Handbook of Numerical Matrix Inversion and Solution of Linear Systems
John Wiley & Sons, 1968.

106. Wight A.L., K.F. Hansen & D.R. Ferguson
Application of Alternating Direction Implicit Methods to the Space Dependent Kinetics Equations
Nucl. Sci. and Eng. **44**, 239-251, 1971.

107. Wilkinson J.H.
The Algebraic Eigenvalue Problem
Clarendon Press, Oxford, 1965.

108. Wilkinson J.H. and C. Reinsch
Handbook for Automatic Computation, Volume 2, Linear Algebra
Springer-Verlag,, 1971.

109. Yasinsky J.B. & A.F Henry
Some Numerical Experiments Concerning Space-Time Reactor Kinetics Behavior
Nucl. Sci. and Eng. **22**, 171-181, 1965.

110. Yasinsky J.B.
The Solution of the Space-Time Neutron Group Diffusion Equations by a Time-Discontinuous Synthesis Method
Nucl. Sci. and Eng. **29**, 381-391, 1967.

111. Yasinsky J.B., M. Natelson & L.A. Hageman
TWIGL - A Program to Solve the Two-Dimensional, Two Group, Space-Time Neutron Diffusion Equations with Temperature Feedback
WAPD-TM 743, Bettis Atomic Power Laboratory, 1968.

112. Younis M.H., P. Soedijono, R. Calabrese & A.R. Dastur
The Effect of Abnormal Flux Shapes on the Performance of CANDU Shutdown System
Presented at the CNS 15 Annual Nuclear Simulation Symposium, AECL CANDU Operations, Mississauga, Ontario, 30 April, 1988.

113. Yung Y.A.
A Theory of Space-Time Reactor Kinetics
Nucl. Sci. and Eng. **80**, 476, 1982.

Abbreviations

BCG	Boosted Conjugate Gradient (method)
BWR	Boiling Water Reactor
CANDU	Canadian Deuterium Uranium (Reactor)
CG	Conjugate Gradient (method)
CP	Collision Probability (method)
CRNL	Chalk River Nuclear Laboratories
CNS	Canadian Nuclear Society
GCG	Generalized Conjugate Gradient (method)
GS	Gauss-Seidel (method)
GSSOR	Gauss-Seidel with successive over relaxation (method)
IQS	Improved Quasistatic (method)
LU (decomposition)	Lower-Upper (Tridiagonal) decomposition
MC	Monte Carlo (method)
NTE	Neutron Transport Equation
ODE	Ordinary Differential Equation
PDE	Partial Differential Equation
Pn	Pn Approximation method
PWR	Pressurized Water Reactor
Sn	Sn method - Discrete Ordinates method
SOR	Successive Over Relaxation

List of Symbols

Symbol	Description
$C_i(\vec{r},t)$	The density of the 'i'th neutron precursor
C_p	Specific heat
D_g	Diffusion coefficient for group g
e	Internal energy
E	Energy (neutronics equations)
E_g	Lower energy limit of the 'g'th energy group
G_1	$\partial P/\partial \rho$
$\tilde{h}$	Convection coefficient
$\vec{g}$	Gravity
G	Number of energy groups
h, H	Enthalpy and total enthalpy
$k(\vec{r})$	Coefficient of heat conduction
$\vec{n}$	Surface normal
Nu	Nusselt number
$\vec{q}$	Thermal flux
P	Pressure
$P_l(x)$	Legendre Polynomial
Pr	Prandtl number
$\vec{r}$	Spatial position vector
Ra	Rayleigh number
$S(\vec{r},t)$	Thermal source term
$\vec{\vec{s}}$	Rate of strain tensor
t	Time
$T(\vec{r},t)$	Temperature
$\vec{V}$	Velocity (of the fluid)
α	Thermal diffusivity
β	Delayed neutron fraction (total), thermal expansion coefficient

β_i^j	Delayed neutron fraction from the 'i'th precursor that was produced by the fission of isotope j.
λ_i	Decay constant of the 'i'th precursor
Λ	Neutron generation time
μ	Viscosity, cosine of the scattering angle
ν	Number of neutrons produced per fission
ρ	Density
$\bar{\bar{\sigma}}$	Stress tensor
σ_x	Microscopic cross section for interaction x
Σ_x	Macroscopic cross section for interaction x
$\bar{\bar{\tau}}$	Shear stress tensor
$\Phi(\vec{r},E,t)$	Neutron flux
$\Phi_g(\vec{r},t)$	Group flux (multigroup formulation)
$\chi(E)$	Fission neutron spectrum
$\nu_i(E)$	Neutron emission spectrum for the 'i'the precursor
ω_f	Fission heat production coefficient
$\vec{\Omega}$	Neutron direction vector
Ψ	Variable of interest (conservation equation)
$(\cdot,\cdot)$	Inner (scalar) product of two vectors