A MULTIGRID METHOD
APPLIED TO REACTOR KINETICS

In memory of Mom, Dad and forever young Bro

A MULTIGRID METHOD APPLIED TO REACTOR KINETICS

By
NGUYỄN, THÁI SINH
B. ENG., M. ENG.

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 (2005) McMaster University
(Engineering Physics) Hamilton, Ontario

TITLE: A Multigrid Method Applied To Reactor Kinetics

AUTHOR: Thai Sinh NGUYEN
B.Eng. (Zaporozhye Industrial Institute, Zaporozhye, USSR, 1984)
M.Eng. (Chulalongkorn University, Bangkok, Thailand, 1999)

ADVISOR: Dr. William J. GARLAND
Professor, McMaster University

NUMBER OF PAGES: xiii, 180

ABSTRACT

The control and safety analysis of a nuclear reactor strongly relies on numerical simulation of reactor dynamics, in which the neutronics computation is one of the most important tasks. It is necessary to utilize a full three-dimensional model of neutron kinetics for satisfactory results but this requires an extensive computation. The purpose of this research is to explore an efficient method for accurate solution of the spatial neutron kinetics problem.

The kinetics of neutrons in a nuclear reactor of practical interest is adequately represented by the few-group diffusion equations with delayed neutron effects taken into account. For solving such a space-time equation system, finite difference methods, though the simplest, must work with a very fine-mesh grid, resulting in an extremely large algebraic system whose solution by basic numerical methods encounters inefficiency. Coarse-mesh methods increase computational efficiency by reducing the number of discretized equations. However, by adding more complexity and limitations, the coarse-mesh computation is still rather time-consuming.

Multigrid methods may provide an optimal solution for a large, sparse algebraic system arising from discretization of a partial differential equation or system but have not found many applications in reactor physics due to inherent difficulties.

In this research, a finite difference method is used for discretization of the kinetics equations and a multigrid solver is developed to solve the discretized equation system. The Additive Correction Multigrid, the simplest and cheapest method in the multigrid family, is used for grid coarsening, allowing for reaching the coarsest grid without any difficulties. By avoiding the singularity and indefiniteness of the discretized system, the Red-Black Gauss-Seidel method is suited for multigrid smoothing and favours an implementation of parallel computation.

Numerical experiments show that our multigrid solver is not only much faster than any basic iterative method but also tends to be mesh-independent or optimal for solving a practical kinetics problem.

ACKNOWLEDGEMENTS

I would like to express my sincere appreciation to my supervisor, Dr. Bill Garland, for his guidance and generous support during all the course of my graduate studies resulting in completing of this thesis.

I also would like to thank the other members of my Supervisory Committee, Drs. Marilyn Lightstone and Skip Poehlman, for their interest and helpful discussions.

The financial assistance from the School of Graduate Studies and the Department of Engineering Physics, McMaster University, through a scholarship, bursary and teaching assistantship is gratefully acknowledged.

Thanks are due to Simon Day for providing and discussing some useful data of the McMaster Nuclear Reactor that are used in this research.

My thanks also go to my former boss, Dr. Vũ Hải Long, my friend Phú Lê, as well as my siblings, Hà, Chí and Dân, for their encouragement and willingness to support whenever I need.

Finally, I must thank my wife, Như Thủy, and sons, Linh Giang and Tuệ Giang, without whose encouragement, compassion, impatience, inspiration, support and love this thesis could not have been completed.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

NOMENCLATURE

A, [A]	algebraic matrix
a	absorption cross-section index; algebraic coefficient; constant
$\tilde{a}$	extrapolated dimension of a slab reactor
B	boundary point; reactor buckling
[B]	vector of group sources
b	algebraic source vector; constant
C	constant
C_i	concentration of delayed neutron precursors in group i
$\overline{C}_i^P$	node-averaged concentration of delayed neutron precursors in group i
$\overline{C}_{iu}^P$	node transverse concentration of group-i delayed precursors in u-direction
c	constant
c_i	weighted precursor concentration in group i
D	diffusion coefficient; dimension(s); diagonal matrix
d	delayed neutron index; dimension index
E	neutron energy; error amplification matrix; east node index
[E]	vector of group errors
e	error vector; east face of a node
ext	external source index
$\hat{e}_s$	unit vector normal to the surface S
[F]	neutron fission operator
f	function; fission index; face index
f_i	discrete value of the function at a point i
G	total number of neutron energy groups; iteration matrix
g	neutron energy group index; Fourier mode amplification factor; geometric
H	horizontally-split matrix; coarse grid mesh length and index
h	mesh length; fine grid index
I	x-dimension size
[I]	unity matrix
i	index; x-dimension index
J	neutron current component; y-dimension size
$\overline{J}_{u\pm}^p$	node face-averaged current at the right (+)/left(-) face in u-direction
$\overline{J}_u^p$	node transverse current in u-direction

$\vec{J}$	neutron current vector
j	index; y-dimension index
$\bar{j}_{u\pm}^{\pm p}$	node face-averaged in(+)/out(-) partial current
K	z-dimension size
K_A, K_S	smoothing and approximation constants
K_m	m-th Krylov subspace
k	known vector; wave number; z-dimension index ; multiplication factor
L	elliptic operator; lower triangular matrix; coarsest grid level; lower node
$\bar{L}_u^p$	node transverse leakage in u-direction
l	lower face of a node
ℓ	coarse grid index
[M]	neutron removal operator
m	rest mass of a neutron; iteration number
m_1, m_2	numbers of pre- and post-smoothing steps in multigrid
m_c	multigrid strategy number
N	number of delayed neutron groups; size; north node index
n	neutron density; index; north face of a node
n_P, n_R	orders of a prolongation and a restriction operators
n_E	order of a partial differential equation
nb	neighbouring node index
P	a point under consideration; node index; prolongation operator
p	amplitude function; direction vector; prompt neutron index
Q	multigrid iteration matrix
q	norm order, e.g. $\|\cdot\|_q$, q = 1, 2 or ∞
Q_m	matrix/algebraic polynomial of order m
P_k	k-order polynomial function
R	restriction operator
[R]	vector of group residuals
r	residual vector
$\vec{r}$	position; general coordinate
$\vec{r}_s$	position on the boundary surface S
S	neutron source; smoothing operator; south node index
s	angular density of neutron source; scattering index; south face of a node
[S]	vector of group sources

s_o	weighted extra neutron source
$\overline{S}^P$	node-averaged neutron source
$\overline{S}_u^P$	node transverse source in u-direction
t	time; total cross-section index
tr	transport cross-section index
U	upper triangular matrix; upper node index; unigrid iteration cost unit
u	either of the coordinates x,y,z; upper face of a node
V	volume; vertically-split matrix
$[V^{-1}]$	diagonal matrix of group neutron inverse speeds
v	neutron speed; eigenvector
$\vec{v}$	neutron velocity
W	west node index
[W]	vector of weighting functions
w	west face of a node
X	basis solution function
x	coordinate; unknown vector
$\overline{x}$	exact solution vector
$\widetilde{x}_s$	extrapolated boundary
Y	basis solution function
y	coordinate
[y]	unknown function
z	coordinate
z_0	extrapolated length

Greeks

α	coefficient; exponent; index
β	total fraction of delayed neutrons; coefficient
β_i	fraction of delayed neutrons in group i
γ	iteration parameter
δ	iteration parameter
$[\chi]$	diagonal matrix of group prompt neutron spectrums
χ	spectrum of prompt neutrons
$[\chi_i^d]$	diagonal matrix of group delayed neutron spectrum
χ_i^d	spectrum of delayed neutrons in the precursor group i

ε	accuracy number
ε_0	float-point precision
$[\Phi]$	vector of group neutron fluxes
ϕ	neutron (scalar) flux; field variable; unknown vector
$\overline{\phi}$	exact solution vector
$\overline{\phi}^P$	node-averaged flux
$\overline{\phi}_u^p$	node transverse flux in u-direction
$[\Phi_0^*]$	vector of static adjoint group fluxes
η	function
φ	angular neutron flux
κ	switch for the Jacobi or Gauss-Seidel scheme
Λ	neutron generation time
λ	eigenvalue
λ_i	decay constant of delayed neutron precursors in group i
$\overline{\mu}_0$	average cosine of the scattering angle
ν	fission neutron yield
θ	time-integration parameter
θ_k	parameter of a Fourier mode
ρ	reactivity; spectral radius of a matrix; iteration contraction number
$\overline{\rho}$	virtual spectral radius; average iteration contraction number
Σ_α	neutron macroscopic cross-section of type α
τ	iteration parameter
Ω	computational grid
$\hat{\Omega}$	unit angular vector
ω	relaxation parameter for an iterative method
ω_b	optimum relaxation parameter for the successive over-relaxation method
$[\Psi]$	vector of shape functions or expansion modes
ψ	shape function
ψ_i	expansion mode

Chapter 1

INTRODUCTION

1.1 Thesis Overview

Nuclear industry is now well into the sixth decade of nuclear power, which has emerged as one of the principal sources of energy in many industrially developed countries. However, public skepticism is still a major factor guiding the future of nuclear power everywhere. For the industry to survive, the nuclear community has to be devoted to restoring the public confidence in the integrity of the industry. At the same time, nuclear researchers and engineers are devoted to economical and safe operations of nuclear reactors, as well as to development of more advanced designs. Reactor safety analysis has played a crucial role in these processes and will continue to do so.

The control and safety analysis of a nuclear reactor strongly relies on the prediction of transient behaviour of the reactor system under both normal operating conditions and accident situations. The analysis of such transients has been traditionally based on numerical simulation of coupled neutron kinetics and thermalhydraulics. While most reactor transient codes (e.g. CATHENA [Hanna (1997)] or RELAP5 [RELAP5 (1995)]) now employ the modern thermalhydraulic models that reflect the up-to-date knowledge of governing phenomena (i.e. heat transfer and hydrodynamics), these codes still depend on simplified neutron kinetics models (e.g. point reactor or dimensionally reduced models) whose results tend to be not only inaccurate but also non-conservative for many important cases of accident analyses. The incorporation of a full three-dimensional core model of neutron kinetics into the system code will allow "best-estimate" simulations of reactor dynamics but this still requires an extensive computation [Jackson et al. (1999)]. *The purpose of this research is to investigate an efficient method for accurate solution of the spatial neutron kinetics problem*.

In a nuclear reactor, neutrons are used to induce fission reactions on heavy nuclei of fissile materials (e.g. U-235 or Pu-241), accompanied by the release of energy and radiation plus additional neutrons. These fission neutrons can then be utilized to induce still further fission reactions, thereby inducing a chain of fission events. The neutron, hence, plays the role of the chain carrier while the fission reactions supply the desired energy. It is important that one be able to determine the neutron distribution through out the reactor core at all times in order to monitor and control the rates at which various neutron-nuclear reactions occur within the reactor.

The fundamental and most exact description of neutron behaviour in a nuclear reactor is provided by the neutron transport equation [Duderstadt & Hamilton (1976)]. For practical nuclear reactors, modeling of neutron kinetics in the framework of the transport theory is, however, prohibitively costly and is, therefore, usually simplified to be more

manageable. It is commonly agreed that the kinetics of neutrons in a nuclear reactor is adequately represented by the group diffusion theory [Henry (1975)]. As a result, we have a system of parabolic partial differential equations to solve for the neutron energy group fluxes and delayed precursor group concentrations in space and time.

Typically, finite difference methods are the simplest and most straightforward approach to the numerical solution of space-time problems. The great advantage of the finite difference method is that it gives the most accurate solution for sufficiently fine grid spacing and small time steps. The disadvantage, however, is that, for the neutron diffusion problem, the finite difference method must work with a very fine grid to obtain acceptable accuracy [Wachspress (1966)]. This requirement will result in an extremely large number of algebraic equations to be solved at each time step. If an implicit scheme is used to avoid instability and inconsistency of numerical solution, the equations in the discretized system are coupled in both space (between grid points) and energy (between energy groups). Basic numerical methods are quite inefficient for inverting such a large algebraic system as it takes so many steps (iterations) to propagate the error through all the grid points [Saad & Vorst (2000)]. Although one can utilize parallel computation on a multiprocessor computer system to reduce computing time at each iteration step, the total number of such iteration steps required for the solution to converge would remain unchanged and one also is likely to face the problem of false convergence.

The inefficiency of the finite difference methods has led to the development of various coarse mesh methods for dealing with the space problem of reactor kinetics, of which the nodal methods have received the greatest acceptance within the reactor physics community [Lawrence (1986)]. The nodal methods allow for spatial discretization of the neutron diffusion equations by using large grid spacing without losing accuracy. In fact, the nodal methods increase the computational efficiency by reducing the number of equations in the discretized system. However, in addition to the great complexity in derivation and difficulty in error analysis for nodal methods, it is difficult to accelerate the convergence for solution of the nodal discretized system. Consequently, the nodal kinetics computation is still rather time-consuming.

Fortunately, there is a class of multigrid methods which use a series of grids of different scales and inter-grid communication to propagate the error over the whole grid just in a single step [Wesseling (1992)]. The multigrid is among the fastest iterative methods known today for solving a large, sparse algebraic system arising from the discretization of partial differential equations. Unlike the basic iterative methods, the multigrid exhibits a convergence rate that is independent of the number of equations in the discretized system. It is, therefore, an optimal method. This great property of the multigrid suggests that we should consider its application to the solution of spatial kinetics equations. However, applications of the multigrid to neutronics computation have reported to encounter a number of difficulties.

The *primary goal* of our research is to develop a multigrid solver for space-energy neutron kinetics equations that, as a kinetics module, can be incorporated into a reactor dynamics simulation code. The simulation of neutron kinetics is, perhaps, the most important and demanding task because it provides the necessary input for computation of other processes and usually requires the largest portion of CPU time. It is expected that, an optimal multigrid solver (i.e. its convergence rate is independent of the grid size) would easily handle the "deep" problem (i.e. the convergence) of the spatial reactor kinetics, while the parallel computation is supposed to effectively deal with the "wide" problem (i.e. the sweep though a large number of algebraic equations in the system).

1.2 Related Works

The importance of spatial treatment of neutron kinetics has long been recognized. Yasinsky and Henry (1965), studying the neutron kinetics behaviour in slab reactors, indicated that point kinetics results were very inaccurate and non-conservative when applied to large reactors. Even in a small tightly-coupled core, for a prompt critical excursion, the power peak predicted by the point kinetics was too small compared with the spatial kinetics solution. This discrepancy was evidently due to the slight change in the flux shape associated with the correct spatial solution. Dubois (1975) extended a similar study to a one-dimensional kinetics model and came to a conclusion that a spatially reduced kinetics model did not yield better results than the point kinetics. According to Ott and Neuhold (1985), the analyses of many safety problems actually require a solution of three-dimensional kinetics problems.

Sutton and Aviles (1996) gave a good overview of spatial methods that have been used more or less for calculations of time-dependent diffusion equations over the past four decades. In particular, a family of nodal methods [Finnemann et al. (1977), Shober et al. (1977), Gupta (1981), Brega et al. (1984), Hebert (1987), Hennart (1986, 1988), Ougouag & Rajic (1988), Song & Kim (1993), Noh & Cho (1994), Verdu et al. (1995), Zhang et al. (1995), Zimin & Ninokata (1996, 1998), Koclas (1998), Penland et al. (1997), Jatuff & Gho (1999), Dahmani et al. (2001), Ikeda & Takeda (2001), Guessous & Akhmouch (2002), Zimin & Baturi (2002)] has demonstrated the best performance in both computational efficiency and physical accuracy; therefore, applications of many other methods (such as the finite difference methods [Alcouffe & Albrecht (1970), Hansen (1972), Mitchell & Griffiths (1980)], coarse mesh methods [Kang & Hansen (1973), Langenbuch et al. (1977), Takeda & Saji (1980), Ackroyd (1981), Schmidt & Fremd (1981), Kavenoky & Lautard (1986), Walters (1986), Montagnini et al. (1996), Cavdar & Ozgener (2004)], or modal methods [Kaplan et al. (1964), Stacey (1971), Miro et al. (2002)]) for reactor simulations have significantly diminished or even vanished. The 'early' nodal methods, developed in 1960s and early 1970s, have been criticized for being inconsistent with the neutron diffusion equations. The 'modern' nodal methods (also known as the transverse-integrated nodal methods) are consistent but add more complexity so that they are normally restricted to two energy groups only [Fu & Cho (2002)]. In addition, the error of nodal discretization is difficult to analyze. Finally, the unusual choice of nodal unknowns, the node-averaged and face-averaged quantities, makes the resulting

discretized system incompatible with fast iterative methods [Whitlock (1991), Moulton (1996)].

Gerard (1997), in developing the MACSIM for simulation of the McMaster Nuclear Reactor, utilized the finite difference method for solving the 3D neutron kinetics equations. Although the source term is treated explicitly in time (to avoid source iteration) and the grid is not fine enough (to reduce the computational cost), MACSIM's time performance is rather poor. The reason is that the simple Gauss-Seidel solver used in the code is too slow for solving the system of several thousands of algebraic equations. The property that the solution convergence strongly deteriorates with an increasing number of equations (or unknowns) in the solving algebraic system is inherent in most of the iterative methods known today [Young & Gregory (1973), Nukamura (1977), Hageman & Young (1981), Axelsson (1994), Saad (1996)].

Multigrid methods were first used by Fedorenko (1961) for solving Poisson's equation on the square domain but the paper by Brandt (1977) is still considered the origin of the modern multigrid by many. The literature on mutigrid methods is very extensive [Douglas (2003)]. Unlike usual iterative methods, multigrid methods provide the convergence that is independent of the number of equations in the solving algebraic system [Bakhvalov (1966), Hackbusch (1982), Braess (1984), Greenbaum (1984), Bank & Douglas (1985), Thole & Trottenberg (1986), Brandt (1994), Reusken (1994), Yavneh (1995, 1996), Bramble et al. (1996), Kang & Kwak (1997), Zhang (1998, 2000), Stuben (2001), Wiennands & Oosterlee (2001), Brenner (2002), Oosterlee & Wienands (2002)].

Although the multigrid has now become a quite standard iterative method in science and engineering [Douglas (1997)], there have been not so many publications of multigrid application to reactor physics found in the literature and almost all of these works [Alcouffe et al. (1983), Phillips & Schmidt (1984), Finnemann et al. (1988), Zalavsky (1993, 1995), Al-Chalabi & Turinsky (1994), Ginestar (2001)] are designated for reactor static (eigenvalue) calculations. Yet, there are only a few papers addressing reactor kinetics problems. In one of such rare papers, Scheilchl (2000) presented a parallel multigrid solver for the transient multigroup diffusion equations. Due to the difficulty in grid coarsening, only finer grids could be formed, leaving thousands of equations to solve for on the coarsest grid. Even with this drawback, the speedup by this solver was outstanding and even super-linear in some cases as reported. In fact, Scheilchl's work is more related to parallel computation rather than to multigrid application. Another quite similar work was reported by Kaveh et al. (2000), in which any coarse grid structure is defined as far as it can be spatially homogenized using nodal equivalence theory [Kollas & Henry (1976), Smith (1986)]. Since it would be not only enormously expensive but seemingly meaningless to homogenize large core regions on a very coarse grid, this scheme was limited to only a few coarse-grid levels, leaving again not a small number of equations to solve for on the coarsest grid.

1.3 Thesis Structure

The mathematical model of reactor kinetics is given in Chapter 2, with a brief introduction to the neutron transport theory and its diffusion approximation. In Chapter 3, an overview of various diffusion methods for reactor kinetics is presented. Numerical solution of algebraic systems is provided in Chapter 4, with a review of the basic direct and iterative methods. An introduction to multigrid methods for solving algebraic systems arising from discretization of partial differential equations is given in Chapter 5. In Chapter 6, the specific issues of multigrid application in reactor physics are addressed, and as a result, a multigrid solver for the neutron kinetics system is constructed. A numerical study of our multigrid solver is presented in Chapter 7, including the simplest cases of the one- and two-group neutronics model in a slab reactor, and a more general three-dimensional multigroup kinetics problem in the McMaster Nuclear Reactor core as well. Finally, conclusions and considerations for future research are offered in Chapter 8.

Chapter 2

REACTOR PHYSICS

The central problem of *reactor physics* is to determine *neutron distribution* in space, energy and time throughout a reactor core, for it is the neutron distribution that determines the rate at which various neutron-nuclear reactions will take place at any given time and location within the reactor. Transport theory provides the fundamental and most exact description of the neutron behaviour in a nuclear reactor, but it is extremely difficult to solve the *neutron transport equation* for any but the simplest modelled problems. Diffusion theory, in which the neutron motion is treated as a diffusion process of a gas, simplifies the transport equation to the *neutron diffusion equation* and is usually found adequate for many practical reactor applications.

2.1 Transport Theory

2.1.1 The Neutron Transport Equation

To determine the distribution of neutrons in a nuclear reactor core requires accounting for their motion about the core and their interactions with medium materials within the core.

A neutron at any time is specified by its position $\vec{r}$ and its velocity $\vec{v} = v\hat{\Omega}$ (Fig.2.1.1); here, $v = |\vec{v}|$ is the speed and $\hat{\Omega}$ is the unit vector in the direction of motion of the neutron. It is more frequent to use the kinetic energy of the neutron, $E = \frac{1}{2}mv^2$, instead of its speed v.

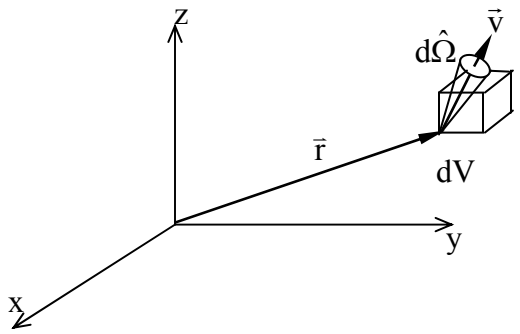

Figure 2.1.1. The position and velocity of a neutron

The fundamental quantity that describes the neutron population is the *angular neutron density* $n(\vec{r}, E, \hat{\Omega}, t)$, which is defined in such a way that

$n(\vec{r}, E, \hat{\Omega}, t)dV dE\, d\hat{\Omega}$ is the number of neutrons at time t in volume element dV surrounding the point $\vec{r}$ and in energy band dE about E, moving in direction $\hat{\Omega}$ in solid angle $d\hat{\Omega}$.

In most cases of interest, the neutron population is so large (typically, ~10^8 neutrons/cm^3) that the neutrons can be treated as a continuum. At the same time, the density of neutrons is sufficiently low compared to the atomic density of the medium (~10^{22} atoms/cm^3) that neutron-neutron interactions can be ignored.

While migrating in a reactor core, the neutrons interact with nuclei of the core materials until they are either absorbed or leak out. The neutron-nuclear interactions are often characterized by the *macroscopic cross section* Σ_α, which specifies the probability per unit distance of travel that a neutron will suffer a collision leading to a reaction of type α ('α' can be absorption 'a', fission 'f', scattering 's', etc.). Knowing the neutron distribution and the macroscopic cross section, one is able to compute the *reaction rate*

$v\Sigma_\alpha(\vec{r}, E)n(\vec{r}, E, \hat{\Omega}, t)dV dE\, d\hat{\Omega}$, which is the number of interactions of type α per unit time with the material nuclei in $dV dE\, d\hat{\Omega}$ at time t that the neutrons at $(\vec{r}, E, \hat{\Omega}, t)$ undergo.

The *neutron transport equation* [Duderstadt & Hamilton (1976), Henri (1980)] is essentially an expression of conservation for the field variable $n(\vec{r}, E, \hat{\Omega}, t)$ - the *neutron density* - within an arbitrary volume V about $\vec{r}$; that is, the time rate of change of the neutron density equals to the net sum of all local sources and sinks of neutrons within volume V

$$\frac{\partial n}{\partial t} = -\nabla\cdot\hat{\Omega}vn - v\Sigma_t(\vec{r}, E)n(\vec{r}, E, \hat{\Omega}, t) + \int_{4\pi} d\hat{\Omega}' \int_0^\infty dE'[v(E')\Sigma_s(\vec{r}, E' \to E, \hat{\Omega}' \to \hat{\Omega})n(\vec{r}, E', \hat{\Omega}', t)] + s(\vec{r}, E, \hat{\Omega}, t) \qquad (2.1.1)$$

where

$s(\vec{r}, E, \hat{\Omega}, t)$ - any neutron sources, including fissions;

$\Sigma_s(\vec{r}, E' \to E, \hat{\Omega}' \to \hat{\Omega})$ - the scattering macroscopic cross-section, characterizing the probability that the neutron will scatter from $(E', \hat{\Omega}')$ to $(E, \hat{\Omega})$;

$\Sigma_t(\vec{r}, E)$ - the total macroscopic cross-section, characterizing the probability that the neutron will suffer a collision at point $\vec{r}$.

It is customary to define the product $vn(\vec{r}, E, \hat{\Omega}, t)$ as a new quantity $\varphi(\vec{r}, E, \hat{\Omega}, t)$, referred to as the *angular neutron flux*, and equation (2.1.1) can be rewritten in term of this angular flux as

$$\frac{1}{v}\frac{\partial \varphi}{\partial t} = -\hat{\Omega}\cdot\nabla\varphi - v\Sigma_t(\vec{r}, E)\varphi(\vec{r}, E, \hat{\Omega}, t) + \int_{4\pi} d\hat{\Omega}' \int_0^\infty dE' \left[\Sigma_s(\vec{r}, E' \to E, \hat{\Omega}' \to \hat{\Omega})\varphi(\vec{r}, E', \hat{\Omega}', t)\right] + s(\vec{r}, E, \hat{\Omega}, t) \qquad (2.1.2)$$

To complete the mathematical description of the neutron transport problem we must provide appropriate initial and boundary conditions for the angular flux. The *initial condition* is usually chosen as

$$\varphi(\vec{r}, E, \hat{\Omega}, 0) = \varphi_0(\vec{r}, E, \hat{\Omega}), \quad \text{for all } \vec{r}, E, \hat{\Omega} \qquad (2.1.3)$$

The boundary condition will depend on the particular problem of interest. The *vacuum boundary condition* is the most common in transport problems, with the assumption that if a neutron leaks out it cannot return into the system, i.e.

$$\varphi(\vec{r}_s, E, \hat{\Omega}, t) = 0, \qquad \text{if} \quad \hat{\Omega}\cdot\hat{e}_s < 0 \qquad (2.1.4)$$

where $\vec{r}_s$ denotes a point on the boundary surface S.

The source term $s(\vec{r}, E, \hat{\Omega}, t)$ can be separated into a fission source term and a term for any external sources. The fission source term, in turn, is composed of *prompt* neutrons (which appear almost instantaneously following the fission event) and *delayed* neutrons (which appear with an appreciable time delay from the subsequent decay of certain fission products - the *delayed neutron precursors*). If we assume that the fission neutrons are emitted isotropically, then

$$s(\vec{r},E,\hat{\Omega},t) = \frac{\chi(E)}{4\pi}(1-\beta)\int_{4\pi}d\hat{\Omega}'\int_{0}^{\infty}dE'[\nu(E')\Sigma_f(\vec{r},E')\varphi(\vec{r},E',\hat{\Omega}',t)] + \sum_{i=1}^{N}\lambda_i C_i(\vec{r},t)\frac{\chi_i^d(E)}{4\pi} + s_{ext}(\vec{r},E,\hat{\Omega},t) \qquad (2.1.5)$$

where

$\Sigma_f(\vec{r},E)$ - the fission macroscopic cross-section, characterizing the probability that the neutron of energy E will induce a fission event at point $\vec{r}$;

$\nu(E)$ - the average number of neutrons produced by a fission event induced by a neutron of energy E;

$\chi(E)$ - the spectrum of prompt fission neutrons;

β - the total fraction of all delayed neutrons emitted by the decay of their precursors;

$C_i(\vec{r},t)$ and λ_i - the concentration and decay constant of the delayed neutron precursors in group i (i = 1,..,N);

$\chi_i^d(E)$ - the spectrum of delayed neutrons for each precursor group.

The concentration of each group of delayed precursors satisfies the equation:

$$\frac{\partial}{\partial t}C_i(\vec{r},t) = -\lambda_i C_i(\vec{r},t) + \beta_i\int_{4\pi}d\hat{\Omega}'\int_{0}^{\infty}dE'\left[\nu(E')\Sigma_f(\vec{r},E')\varphi(\vec{r},E',\hat{\Omega}',t)\right], \quad i = 1,..,N \qquad (2.1.6)$$

where β_i is the fraction of the fission neutrons produced from the decay of precursors in group i, and $\beta = \sum_{i=1}^{N}\beta_i$.

The neutron transport equation (2.1.3) provides an exact description of the neutron population in a nuclear reactor, provided that appropriate cross-section data are supplied. Its solution would yield the angular flux $\varphi(\vec{r},E,\hat{\Omega},t)$ as a function of position, energy, direction and time. Unfortunately, even numerical solutions of this equation in its general form are extremely difficult for any practical problems of nuclear reactor analysis. Therefore, suitable approximations to the transport equation are required to reduce it to a more manageable form.

2.1.2 Methods for Solution of the Neutron Transport Equation

The neutron transport equation (2.1.2) is rather simple to derive, but it is extremely difficult to be solved for any but the simplest modelled problems [Lewis & Miler (1993), Adams & Larsen (2002)]. For any realistic problems of nuclear reactor analysis, attempts to solve the neutron transport equation analytically are impossible and even the numerical solution is not easy either.

To numerically solve the neutron transport equation, one first has to convert it into a system of algebraic equations that can then be coded and solved by using a computer. This is accomplished by discretizing each of the variables in the transport equation, i.e. by replacing functions of continuous variables by a discrete set of values at a discrete set of points. The derivatives and integrals appearing in the equation must also be replaced by a corresponding discrete representation.

Like any integral/differential equations, the neutron transport equation can be discretized by using either *discrete ordinates methods* or *function expansions* [Duderstadt & Hamilton (1976)]. In the discrete ordinate approach, the unknown function f(x) is represented only by its values at a discrete set of points $\{x_i\}$ of the independent variable x as

$$f(x) \rightarrow \{f(x_i)\} \equiv \{f_i \mid i=1,..,N\} \tag{2.1.7}$$

Derivatives and integrals are approximated by *finite differences* and *sums*, respectively. By that way, one would arrive at an algebraic system of N discretized equations for N unknowns $\{f_i\}$.

In the function expansion approach, the unknown function f(x) is expanded in a series of known functions $\psi_i(x)$ - the *expansion modes* – as

$$f(x) \cong \sum_{i=1}^{N} f_i \psi_i(x) \tag{2.1.8}$$

That is, once again, the function f(x) is represented by a set $\{f_i\}$- the *expansion coefficients*. One now can substitute the expansion (2.1.8) into the original equation and use one of numerous techniques to obtain a set of algebraic equations for the expansion coefficients $\{f_i\}$.

The primary variable in the transport equation that distinguishes it from its approximations is the directional variable $\hat{\Omega}$, which can be discretized by using either of the two aforementioned methods. In the discrete ordinate approach, $\hat{\Omega}$ is represented by a discrete set of directions

$$\hat{\Omega} \equiv \{\hat{\Omega}_n \mid n=1,..,N\}$$

and the flux is represented by only its values at each of these mesh directions

$$\varphi(\hat{\Omega}) = \{\varphi(\hat{\Omega}_n)\} \equiv \{\varphi_n \mid n=1,..,N\}.$$

This approach reduces the transport equation to a coupled set of N equations, commonly referred to as the *S_n equations*

$$\frac{1}{v}\frac{\partial \varphi_n}{\partial t} = -\hat{\Omega}_n \cdot \nabla \varphi_n - v\Sigma_t \varphi_n(\vec{r}, E, t) + \sum_{n'=1}^{N} \int_0^{\infty} dE' [\Sigma_s(\vec{r}, E' \to E, \hat{\Omega}_{n'} \to \hat{\Omega}_n)\varphi_n(\vec{r}, E', t)] + s_n(\vec{r}, E, t), \quad n = 1,..,N \tag{2.1.9}$$

An alternative way to discretize the angular variable is to use the function expansion method. In general, the angular flux, and other angular quantities as well, is expanded in a finite series of spherical harmonics $Y_{\ell m}(\hat{\Omega})$ as:

$$\varphi(\vec{r}, E, \hat{\Omega}, t) = \sum_{\ell=0}^{N} \sum_{m=-\ell}^{+\ell} \varphi_{\ell m}(\vec{r}, E, t) Y_{\ell m}(\hat{\Omega}) \tag{2.1.10}$$

By substituting the expansion (2.1.10) into the original equation, multiplying by modes of different order $Y_{\ell' m'}(\hat{\Omega})$, and integrating over the angular variable, one then can use orthogonality to obtain a coupled set of equations for the expansion coefficients $\varphi_{\ell m}$. This set of equations is known as the *P_N equations*. If the expansion in spherical harmonics is truncated after two terms, i.e. N = 1, the expansion for the angular flux takes form:

$$\varphi(\vec{r}, E, \hat{\Omega}, t) \cong \frac{1}{4\pi}\varphi_0(\vec{r}, E, t) + \frac{3}{4\pi}\vec{\varphi}_1(\vec{r}, E, t) \cdot \hat{\Omega} \tag{2.1.11}$$

We will find shortly that this *P_1 approximation* to the angular flux is closely related to *neutron diffusion theory*. The higher order P_N equations are rarely used in nuclear reactor analysis, rather one usually relies on S_N equations if a more detailed treatment of the neutron directional dependence is required.

Other independent variables can be discretized in very similar techniques. However, the discrete ordinate approach is most common for the energy variable E, resulting in a set of *multigroup* equations. Since the multigroup derivation is the same for both transport and diffusion equations, we will perform it in the next section for the latter.

In practice, almost all transport problem calculations are static calculations, i.e. all quantities in the transport equations are considered time-independent. Furthermore, the geometry of the transport problem is usually simplified by considering only planar or spherical symmetry (i.e. 1D geometry). In fact, even after a number of such approximations, the transport equation in a rather simplified form (i.e. steady-state, 1D but directionally-dependent) can still only be solved with great effort.

2.2 Diffusion Theory

2.2.1 Diffusion Approximation

It is apparent that, for most reactor calculations, the details of the angular dependence of the flux are not necessary as we only need to know the angle-integrated (scalar) *neutron flux*, which is defined by

$$\phi(\vec{r}, E, t) = \int_{4\pi} \varphi(\vec{r}, E, \hat{\Omega}, t) d\hat{\Omega} \tag{2.2.1}$$

since knowing the flux $\phi(\vec{r}, E, t)$ is sufficient to calculate the rate of almost all neutron-nuclear reactions that occur in a nuclear reactor. By integrating the transport equation over the solid angle $\hat{\Omega}$, we can obtain an equation for the flux $\phi(\vec{r}, E, t)$ as

$$\frac{1}{v}\frac{\partial \phi}{\partial t} = -\nabla \cdot \vec{J} - \Sigma_t(\vec{r}, E)\phi(\vec{r}, E, t) + \int_0^{\infty} \Sigma_s(\vec{r}, E' \to E)\phi(\vec{r}, E', t) dE' + S(\vec{r}, E, t) \tag{2.2.2}$$

where

$\vec{J}(\vec{r}, E, t) \equiv \int_{4\pi} \hat{\Omega}\varphi(\vec{r}, E, \hat{\Omega}, t) d\hat{\Omega}$, which is referred to as the *neutron current*

$$\Sigma_s(\vec{r}, E' \to E) \equiv \int_{4\pi} \Sigma_s(\vec{r}, E' \to E, \hat{\Omega}' \to \hat{\Omega}) d\hat{\Omega}$$

$$S(\vec{r}, E, t) \equiv \int_{4\pi} s(\vec{r}, E, \hat{\Omega}, t) d\hat{\Omega}$$

Equation (2.2.2) is known as the *neutron continuity equation* and it is still quite exact since no approximations have been introduced. Unfortunately, it contains two unknowns, the flux $\phi(\vec{r}, E, t)$ and the current $\vec{J}(\vec{r}, E, t)$, which cannot be related to each other in a general and exact manner (although they are both expressed in terms of an angular

integral of the angular flux). One must resort to an approximate relationship between them by using the so called *Fick's law*: for the neutron treated as a gas diffusing in a reactor core, its current density is proportional to the negative spatial gradient of its density (i.e. the scalar flux)

$$\vec{J}(\vec{r},E,t) = -D(\vec{r},E)\nabla\phi(\vec{r},E,t) \tag{2.2.3}$$

where the constant of proportionality $D(\vec{r},E)$ is known as the *diffusion coefficient*. There exist various methods for finding this diffusion coefficient. In the following, the *P_1 approximation* [Duderstadt & Hamilton (1976), Henry (1980)] is used.

First, we multiply the transport equation by $\hat{\Omega}$ and integrate it over the solid angle to arrive at

$$\frac{1}{v}\frac{\partial\vec{J}}{\partial t} = -\nabla\cdot\int_{4\pi}\hat{\Omega}\hat{\Omega}\varphi(\vec{r},E,\hat{\Omega},t)d\hat{\Omega} - \Sigma_t(\vec{r},E)\vec{J}(\vec{r},E,t) + \int_0^\infty \Sigma_{s_1}(\vec{r},E'\to E)\vec{J}(\vec{r},E',t)dE' + \vec{S}_1(\vec{r},E,t) \tag{2.2.4}$$

where

$$\vec{S}_1(\vec{r},E,t) \equiv \int_{4\pi} s(\vec{r},E,\hat{\Omega},t)\hat{\Omega}d\hat{\Omega}$$

$$\Sigma_{s_1}(\vec{r},E'\to E) \equiv \int_{4\pi}\Sigma_s(E'\to E,\hat{\Omega}'\to\hat{\Omega})(\hat{\Omega}'\cdot\hat{\Omega})d\hat{\Omega}$$

Next, recall the P_1-approximation (2.1.11) that treats the angular flux as *linearly anisotropic* with the zero and first moments replaced by the flux and current, respectively

$$\varphi(\vec{r},E,\hat{\Omega},t) \cong \frac{1}{4\pi}\phi(\vec{r},E,t) + \frac{3}{4\pi}\vec{J}(\vec{r},E,t)\cdot\hat{\Omega} \tag{2.2.5}$$

With this approximation to the angular flux, it is easily verified that

$$\int_{4\pi}\varphi(\vec{r},E,\hat{\Omega},t)d\hat{\Omega} = \frac{1}{4\pi}\phi(\vec{r},E,t)\int_{4\pi}d\hat{\Omega} + \frac{3}{4\pi}\vec{J}(\vec{r},E,t)\cdot\int_{4\pi}\hat{\Omega}d\hat{\Omega} = \phi(\vec{r},E,t)$$

and

$$\int_{4\pi} \varphi(\vec{r}, E, \hat{\Omega}, t)\hat{\Omega}d\hat{\Omega} = \frac{1}{4\pi}\phi(\vec{r}, E, t)\int_{4\pi}\hat{\Omega}d\hat{\Omega} + \frac{3}{4\pi}\vec{J}(\vec{r}, E, t)\cdot\int_{4\pi}\hat{\Omega}\hat{\Omega}d\hat{\Omega}$$
$$= \vec{J}(\vec{r}, E, t).$$

Also,

$$\int_{4\pi} \hat{\Omega}\hat{\Omega}\varphi(\vec{r}, E, \hat{\Omega}, t)d\hat{\Omega} = \frac{1}{4\pi}\phi(\vec{r}, E, t)\int_{4\pi}\hat{\Omega}\hat{\Omega}d\hat{\Omega} + \frac{3}{4\pi}\vec{J}(\vec{r}, E, t)\cdot\int_{4\pi}\hat{\Omega}\hat{\Omega}\hat{\Omega}d\hat{\Omega}$$
$$= \frac{1}{3}\phi(\vec{r}, E, t) \qquad (2.2.6)$$

(We note that $\int_{4\pi} d\hat{\Omega} = 4\pi$,

$$\int_{4\pi} \hat{\Omega}\hat{\Omega}d\hat{\Omega} = \frac{4\pi}{3},$$

and

$$\int_{4\pi} \hat{\Omega}d\hat{\Omega} = \int_{4\pi} \hat{\Omega}\hat{\Omega}\hat{\Omega}d\hat{\Omega} = 0).$$

Further, we introduce two more approximations. The first approximation is the assumption that the neutron source is isotropic, which implies

$$\vec{S}_1(\vec{r}, E, t) \equiv \int_{4\pi} s(\vec{r}, E, \hat{\Omega}, t)\hat{\Omega}d\hat{\Omega} = \frac{1}{4\pi}S(\vec{r}, E, t)\int_{4\pi}\hat{\Omega}d\hat{\Omega} = 0.$$

This approximation is usually of reasonable validity in nuclear reactor studies since it is the fission neutrons that mostly contribute to the neutron source. The other approximation is the assumption that the rate of time variation of the current is much slower than the collision frequency, i.e.

$$\frac{1}{|\vec{J}|}\frac{\partial|\vec{J}|}{\partial t} << v\Sigma_t$$

Since $v\Sigma_t$ is typically of order 10^5 sec^{-1} or larger, only an extremely rapid time variation of the current would invalidate this assumption. Such rapid changes are very rarely encountered in reactor dynamics. Consequently, the time derivative term in equation (2.2.4) can be neglected in comparison with the remaining terms, that is

$$\frac{1}{v}\frac{\partial \vec{J}}{\partial t} \cong 0$$

Hence we can rewrite equation (2.2.4) as

$$\vec{J}(\vec{r}, E, t) = -\frac{1}{3}\left[\Sigma_t(\vec{r}, E) - \frac{\int_0^\infty \Sigma_{s_1}(\vec{r}, E' \to E) J(\vec{r}, E', t) dE'}{J(\vec{r}, E', t)}\right] \nabla\phi(\vec{r}, E, t) \qquad (2.2.7)$$

Comparing (2.2.7) with the Fick's law (2.2.3), we can find the diffusion coefficient

$$D(\vec{r}, E) = \frac{1}{3}\left[\Sigma_t(\vec{r}, E) - \frac{\int_0^\infty \Sigma_{s_1}(\vec{r}, E' \to E) J(\vec{r}, E', t) dE'}{J(\vec{r}, E', t)}\right]$$

If we neglect the anisotropic contribution to energy transfer in a scattering collision by setting

$$\Sigma_{s_1}(E' \to E) = \Sigma_{s_1}(E)\,\delta(E' - E)$$

so that

$$\int_0^\infty \Sigma_{s_1}(E' \to E) J(\vec{r}, E', t) dE' = \bar{\mu}_0 \Sigma_s(E) J(\vec{r}, E, t),$$

then we find a natural generalization of the diffusion coefficient:

$$D(\vec{r}, E) = \frac{1}{3[\Sigma_t(\vec{r}, E) - \bar{\mu}_0 \Sigma_s(\vec{r}, E)]} = \frac{1}{3\Sigma_{tr}(\vec{r}, E)} \qquad (2.2.8)$$

where

$\overline{\mu}_0 = \overline{\hat{\Omega}\cdot\hat{\Omega}'} \equiv \overline{\cos\theta}$ - the average scattering angle cosine;

$\Sigma_{tr}(\vec{r}, E) \equiv \Sigma_t(\vec{r}, E) - \overline{\mu}_0 \Sigma_s(\vec{r}, E)$ - the *macroscopic transport cross section*.

Combining the continuity equation (2.2.2) and the diffusion approximation (2.2.3), we arrive at the *neutron diffusion equation* for the scalar neutron flux

$$\frac{1}{v}\frac{\partial\phi}{\partial t} = \nabla\cdot D(\vec{r}, E)\nabla\phi - \Sigma_t(\vec{r}, E)\phi(\vec{r}, E, t) + \int_0^\infty \Sigma_s(\vec{r}, E' \to E)\phi(\vec{r}, E', t)dE' + S(\vec{r}, E, t) \tag{2.2.9}$$

Often, it is convenient to write these two equations separately in a sense that equation (2.2.2) is an exact equation and only equation (2.2.3) is an approximation.

The appropriate *initial condition* for the diffusion equation can be obtained by integrating the transport condition over angle as

$$\phi(\vec{r}, E, 0) = \phi_0(\vec{r}, E) \tag{2.2.10}$$

There are several types of *boundary conditions*, depending on the particular physical problem of interest. At an interface between two regions of differing cross sections, by integrating the transport condition of the continuity of the angular flux, one obtains the continuity of the flux and the current across the interface (the *interface boundary conditions*) as

$$\phi_1(\vec{r}_s, E, t) = \phi_2(\vec{r}_s, E, t) \tag{2.2.11}$$

$$\vec{J}_1(\vec{r}_s, E, t) = \vec{J}_2(\vec{r}_s, E, t) \tag{2.2.12a}$$

or

$$D_1(\vec{r}_s, E)\nabla\phi_1(\vec{r}_s, E, t) = D_2(\vec{r}_s, E)\nabla\phi_2(\vec{r}_s, E, t) \tag{2.2.12b}$$

However, the diffusion theory can only approximate the *vacuum boundary condition* by setting to zero all incoming neutrons, that is

$$\int_{2\pi^-} \varphi(\vec{r}, E, \hat{\Omega}, t)\hat{\Omega}\cdot\hat{e}_s d\hat{\Omega} \cong \frac{1}{4}\phi(\vec{r}_s, E, t) + \frac{D}{2}\hat{e}_s\cdot\nabla\phi(\vec{r}_s, E, t) = 0 \tag{2.2.13}$$

It is convenient to consider this boundary condition applied to 1D geometry with the boundary at $x = x_s$. If the flux is assumed to decrease linearly beyond the boundary, then it would vanish at some point $\tilde{x}_s > x_s$

$$\phi(\tilde{x}_s) = 0, \qquad \tilde{x}_s \equiv x_s + z_0 = x_s + 2D \tag{2.2.14}$$

where $\tilde{x}_s$ is referred to as the extrapolated boundary. More advanced transport theory calculations of the extrapolated boundary indicate that one should choose the *extrapolated length* z_0 for plane geometry equal to

$$z_0 = \frac{0.7104}{\Sigma_{tr}} \tag{2.2.15}$$

The source term in the diffusion equation in terms of the neutron flux can be obtained by integrating equation (2.1.5) over the angle to get

$$S(\vec{r}, E, t) = \chi(E)(1-\beta)\int_0^\infty \nu(E')\Sigma_f(\vec{r}, E')\phi(\vec{r}, E', t)dE' + \sum_{i=1}^{N}\lambda_i C_i(\vec{r}, t)\chi_i^d(E) + S_0(\vec{r}, E, t) \tag{2.2.16}$$

And the precursor equations can be written in terms of the flux as well:

$$\frac{\partial}{\partial t}C_i(\vec{r}, t) = -\lambda_i C_i(\vec{r}, t) + \beta_i \int_0^\infty \nu(E')\Sigma_f(\vec{r}, E')\phi(\vec{r}, E', t)dE' \tag{2.2.17}$$

2.2.2 The Multigroup Diffusion Equations

It is known that the neutron-nuclear cross sections present in the diffusion equation depend rather sensitively on the incident neutron energy which spans all over the range from 10^7eV (of the *fast neutrons*) down to the 10^{-3}eV (of the *thermal neutrons*) [Duderstadt & Hamilton (1976)]. It is customary to use the discrete ordinate method for discretizing the energy dependence of the flux. Thus, the neutron energy range is divided into a number of energy intervals or *groups*

$$E = \{E_g \mid g = 0,..,G\}, \quad 0 < E_G < \ldots < E_0 < \infty \tag{2.2.18}$$

The continuous energy-dependent flux $\phi(\vec{r}, E, t)$ is integrated over the energies of each group to form the *multigroup fluxes*, which represent the total flux of all neutrons with energies E in the group $E_g < E < E_{g-1}$

$$\phi_g(\vec{r}, t) = \int_{E_g}^{E_{g-1}} \phi(\vec{r}, E, t) dE \tag{2.2.19}$$

If the diffusion equation is integrated over a given group, the *multigroup diffusion equations* are obtained

$$\frac{1}{v_g}\frac{\partial \phi_g}{\partial t} = \nabla \cdot D_g(\vec{r}) \nabla \phi_g(\vec{r}, t) - \Sigma_{tg}(\vec{r}) \phi_g(\vec{r}, t) + \sum_{g'=1}^{G} \Sigma_{sg'g} \phi_{g'} + S_g(\vec{r}, t), \quad g = 1,..,G \tag{2.2.20a}$$

where

$$S_g = (1-\beta)\chi_g^p \sum_{g'=1}^{G} \nu\Sigma_{fg'}\phi_{g'} + \sum_{i=1}^{N} \lambda_i C_i \chi_{i,g}^d + S_g^0,$$

as well as the precursor equations

$$\frac{\partial}{\partial t} C_i(\vec{r}, t) = -\lambda_i C_i(\vec{r}, t) + \beta_i \sum_{g'=1}^{G} \nu\Sigma_{fg'}\phi_{g'}, \qquad i = 1,..,N \tag{2.2.20b}$$

The *group constants* appearing in the multigroup equations are defined as

$$\Sigma_{tg} \equiv \frac{1}{\phi_g} \int_{E_g}^{E_{g-1}} \Sigma_t(E) \phi(\vec{r}, E, t) dE$$

$$\frac{1}{v_g} \equiv \frac{1}{\phi_g} \int_{E_g}^{E_{g-1}} \frac{1}{v(E)} \phi(\vec{r}, E, t) dE$$

$$\Sigma_{sg'g} \equiv \frac{1}{\phi_g} \int_{E_g}^{E_{g-1}} dE \int_{E_{g'}}^{E_{g'-1}} dE' \left[\Sigma_s(E' \to E) \phi(\vec{r}, E', t)\right], \quad \text{i.e. } \Sigma_s(g' \to g)$$

$$D_g \equiv \frac{\int_{E_g}^{E_{g-1}} D(E)\nabla\phi(\vec{r},E,t)dE}{\int_{E_g}^{E_{g-1}} \nabla\phi(\vec{r},E,t)dE}$$

$$\chi_g^{(\cdot)} \equiv \int_{E_g}^{E_{g-1}} \chi^{(\cdot)}(E)dE\,, \qquad \text{for both prompt and delayed neutron spectrums}$$

$$\nu\Sigma_{fg} \equiv \frac{1}{\phi_g}\int_{E_g}^{E_{g-1}} \nu(E)\Sigma_f(E)\phi(\vec{r},E,t)dE$$

Sometimes, it is convenient to write the multigroup diffusion equations in matrix notation for the group fluxes [Φ] as

$$[V^{-1}]\frac{\partial}{\partial t}[\Phi] = [F_p - M][\Phi] + \sum_{i=1}^{N}\lambda_i C_i(\vec{r},t)[\chi_i^d] + [S_0] \qquad (2.2.21a)$$

$$\frac{\partial}{\partial t}C_i(\vec{r},t) = -\lambda_i C_i(\vec{r},t) + \beta_i[F][\Phi], \qquad i = 1,..,N \qquad (2.2.21b)$$

where

$$[\Phi] \equiv \begin{bmatrix} \phi_1(\vec{r},t) \\ \phi_2(\vec{r},t) \\ \cdots \\ \phi_G(\vec{r},t) \end{bmatrix} \text{ - the } \textit{flux vector}$$

$$[S_0] \equiv \begin{bmatrix} S_1^{ext}(\vec{r},t) \\ S_2^{ext}(\vec{r},t) \\ \cdots \\ S_G^{ext}(\vec{r},t) \end{bmatrix} \text{ - the } \textit{source vector}$$

$$[V^{-1}] \equiv \begin{bmatrix} v_1^{-1} & & & \\ & v_2^{-1} & & \\ & & \cdots & \\ & & & v_G^{-1} \end{bmatrix},$$

$$[\chi_i^d] \equiv \begin{bmatrix} \chi_{i,1}^d & & & \\ & \chi_{i,2}^d & & \\ & & \cdots & \\ & & & \chi_{i,G}^d \end{bmatrix},$$

$$[F_p] = (1-\beta)[\chi][F] \equiv (1-\beta) \begin{bmatrix} \chi_1 & & & \\ & \chi_2 & & \\ & & \cdots & \\ & & & \chi_G \end{bmatrix} \begin{bmatrix} \nu\Sigma_{f1} & \nu\Sigma_{f2} & \cdots & \nu\Sigma_{fG} \\ \nu\Sigma_{f1} & \nu\Sigma_{f2} & \cdots & \nu\Sigma_{fG} \\ \cdots & \cdots & \cdots & \cdots \\ \nu\Sigma_{f1} & \nu\Sigma_{f2} & \cdots & \nu\Sigma_{fG} \end{bmatrix},$$

and

$$[M] \equiv \begin{bmatrix} -\nabla \cdot D_1 \nabla + \Sigma_{t1} - \Sigma_{s11} & -\Sigma_{s21} & \cdots & -\Sigma_{sG1} \\ -\Sigma_{s12} & -\nabla \cdot D_2 \nabla + \Sigma_{t2} - \Sigma_{s22} & \cdots & -\Sigma_{sG2} \\ \cdots & \cdots & \cdots & \cdots \\ -\Sigma_{s1G} & -\Sigma_{s2G} & \cdots & -\nabla \cdot D_G \nabla + \Sigma_{tG} - \Sigma_{sGG} \end{bmatrix}.$$

Within the diffusion approximation, the multigroup diffusion equations are still quite exact, but the group constants present in these equations are yet to be determined. It appears that these group constants depend on the group fluxes $\phi_g(\vec{r}, t)$ which have yet to be determined. They are rigorously constants only when the energy dependence of the flux is separable such that

$$\phi(\vec{r}, E, t) = \psi(\vec{r}, t)\eta(E) \tag{2.2.22}$$

However, this is generally not the case in practical reactor calculations. In fact, for an extremely fine group structure (G ~ 1000, for example), the cross sections and hence the flux tend to be smoothly varying over each group. In this case, such a separable approximation to the flux would be acceptable. In practical reactor calculations, one usually works with from 2-4 (for thermal reactors) to 15-20 groups (for fast reactors) [Henry (1975)]. Such *few group* calculations can only be effective with reasonably accurate estimates of the group constants. The most common approach is to actually perform two

multigroup calculations. In the first of these calculations, the spatial and time dependence is ignored (or very crudely approximated), and a very finely structured multigroup calculation is performed to calculate the intragroup fluxes. The group constants for this fine spectrum calculation are frequently to be just tabulated cross section data averaged over each of the fine groups. These intragroup fluxes are then used to calculate the group constants for a coarse group calculation. This scheme of first calculating a neutron spectrum and then collapsing the cross section data over this spectrum to generate few group constants is the most common method in use today.

Since our work is concentrated in numerical solution of the diffusion equations, we will not involve in a detailed description of mutigroup constant generation. Here, we assume that the appropriate group constants are provided initially in space but subject to adjustment in time for a given reactor core.

*
* *

In conclusion to this chapter, we note that although the neutron transport equation provides the most exact mathematical model of neutron behaviour in a nuclear reactor, its solution is prohibitively costly even on a modern computer. The neutron diffusion equation, an approximation to the neutron transport equation, is more computationally manageable but still provides adequately reliable results for most practical reactor calculations. Despite being much simpler than their transport counterpart, the multigroup neutron diffusion equations still require significant efforts for their solution, especially, in space-time reactor calculations. With this to be kept in mind, we will proceed to review the diffusion methods for solving the neutron kinetics problem in the next chapter.

Chapter 3

DIFFUSION METHODS FOR REACTOR KINETICS

Reactor kinetics is an area of reactor physics that deals with time behaviour of the neutron population in a nuclear reactor. Although neutron transport theory provides the most exact description of the neutron behaviour in a reactor, modeling the neutron kinetics in the framework of the transport theory [Chen (1989)] would be prohibitively expensive. The group diffusion theory, an approximation to the neutron transport process, has been found to be adequate for many reactor analysis problems of practical interest. However, solution of the multigroup diffusion equations, though simpler than the transport equation, is not easy either.

It is commonly agreed that the few group diffusion equations (of 2-4 energy groups for thermal reactors and 15-20 for fast reactors) with six precursor equations is an adequate model of the neutron kinetics in a nuclear reactor [Henry (1975)]. In order to solve for the neutron group fluxes in space and time, the system of partial differential equations for the group fluxes and precursor concentrations must be discretized in space and time by using either discrete ordinate or function expansion method (as done in derivation of the group diffusion equations from the transport theory in Chapter 2).

In this chapter, a review of the most popular methods used in practice for the numerical solution of neutron kinetics equations will be given. It is customary to treat the space problem of the kinetics equations first, and then the resulting set of spatially-coupled time-dependent equations is integrated over the time domain.

3.1 Spatial Treatment of Neutron Kinetics Equations

3.1.1 The Point Kinetics Model

The simplest model of reactor kinetics is the *point kinetics*, in which both energy and spatial effects of the neutron population are neglected during a transient [Ash (1979)]. By assuming *a single energy group* of all neutrons and *a fixed shape* of the flux within a reactor core, one can easily derive the *point kinetics equations* from an exact equation of neutron balance (either the continuity equation or even the transport equation).

The *one-speed assumption* reduces the neutron balance equation to:

$$\frac{1}{v}\frac{\partial \phi}{\partial t} = -\nabla\cdot\vec{J}(\vec{r},t) - \Sigma_a\phi(\vec{r},t) + (1-\beta)\nu\Sigma_f\phi(\vec{r},t) + \sum_{i=1}^{N}\lambda_i C_i(\vec{r},t) + S_0(\vec{r},t) \qquad (3.1.1a)$$

where $\Sigma_a \equiv \Sigma_t - \Sigma_s$ is the *absorption* cross section. And the *precursor equations* become

$$\frac{\partial}{\partial t} C_i = -\lambda_i C_i(\vec{r}, t) + \beta_i \nu\Sigma_f \phi(\vec{r}, t), \qquad i = 1,..,N \tag{3.1.1b}$$

The delayed neutron fractions β_i in the one-speed equations are not of the physical value as in the energy-dependent equations but rather of the effective value (to account for the fact that the delayed neutrons are born with lower energies than the prompt fission neutrons).

Within the *fixed shape assumption*, the flux can be split into a purely time-dependent *amplitude function* p(t) and a fixed *shape function* $\psi(\vec{r})$

$$\phi(\vec{r}, t) = p(t)\psi(\vec{r}) \tag{3.1.2}$$

Here, the *shape* $\psi(\vec{r})$ is set equal to the initial flux assuming a steady state for $t \le 0$:

$$\psi(\vec{r}) = \phi(\vec{r}, 0) = \phi_0(\vec{r})$$

The initial condition for the *amplitude* p(t) then simply is

$$p(0) = 1$$

Substituting the flux expression (3.1.2) into the one-speed equations and integrating them with respect to space over the whole core volume V_{core}, one arrives at a set of the so called *point kinetics equations*

$$\frac{dp(t)}{dt} = \frac{\rho - \beta}{\Lambda} p(t) + \sum_{i=1}^{N} \lambda_i c_i(t) + s_o(t) \tag{3.1.3a}$$

$$\frac{dc_i(t)}{dt} = -\lambda_i c_i(t) + \frac{\beta_i}{\Lambda} p(t), \quad i = 1,..,N \tag{3.1.3b}$$

where

$$\rho \equiv 1 - \frac{\iiint_{Vcore} \nu\Sigma_f \phi(\vec{r}, t) d^3r}{\iiint_{Vcore} [\nabla \cdot \vec{J}(\vec{r}, t) + \Sigma_a \phi(\vec{r}, t)] d^3r}$$ - referred to as the *reactivity*

$$\Lambda \equiv \left[\frac{\iiint_{Vcore} \nu\Sigma_f \psi(\vec{r},t) d^3r}{\frac{1}{v} \iiint_{Vcore} \psi(\vec{r}) d^3r} \right]^{-1}$$ - the *neutron generation time*

$$c_i(t) \equiv \frac{\iiint_{Vcore} C_i(\vec{r},t) d^3r}{\frac{1}{v} \iiint_{Vcore} \psi(\vec{r}) d^3r}$$ - the *weighted precursor concentration* in group i

$$s_o(t) \equiv \frac{\iiint_{Vcore} S_{ext}(\vec{r},t) d^3r}{\frac{1}{v} \iiint_{Vcore} \psi(\vec{r}) d^3r}$$ - the *weighted extra source* of neutrons

Solution of this set of (N+1) *ordinary differential equations* yields the neutron population amplitude p(t) and the weighted precursor concentrations $c_i(t)$ in the reactor.

The fact that the global parameters ρ, β, and Λ are generally functions of time makes it difficult to solve the point kinetics equations *analytically* (except for only a few special cases, such as a step or ramp insertion of reactivity with one-group representation of all delayed neutrons). In practice, these point kinetics parameters are usually determined experimentally rather than by using the above integral expressions. As it is usually adequate to have only six groups (N = 6) to represent the delayed neutrons, a set of the seven point kinetics equations can be solved *numerically* at the least cost compared with any other spatial method [Blanchon et al. (1983), Sanchez (1989), Planchard (1991), Kinard & Allen (2004)].

As a rule, the point kinetics model can only be used for small tightly-coupled reactors or very slow transients in a reactor near its criticality. In fact, the point kinetics results are not only *inaccurate* in important cases but also generally *non-conservative* [Ott & Neuhold (1985)]. The reason is that the neglect of the flux shape variation during a transient will lead to an underestimation of the value of positive reactivity insertion, which may cause or intensify accidents, and, on contrary, to an overestimation of the value of negative counteracting reactivity. The fixed flux shape assumption, which the point kinetics model is based on, will obviously be invalid in many cases of interest. For example, in case a fuel assembly is dropped into a near critical core or a control rod of the highest reactivity worth is ejected (as usually postulated for reactor accident analysis), a strong localized perturbation in the core composition would certainly cause a considerable deviation from the spatial shape and would invalidate the point kinetics

model. Furthermore power reactor cores are quite large so that the neutron behaviour in such cores tends to be quite loosely coupled from point to point. This means that a change in the flux or power at one point in the core will not be felt at other points after an appreciable time delay. This implies a break down in the point kinetics model that characterizes every point in the reactor by the same time amplitude [Duderstadt & Hamilton (1976)].

The analyses of many reactor dynamics problems actually require a solution utilizing *space-energy dependent neutron kinetics*. As a general rule, the point kinetics model is incapable of predicting the detailed behaviour of reactor transients either initiated by rapid local changes in reactivity or occurring in large power reactors. In these instances, one must take explicit account of the space-energy dependence of the neutron flux [Weston & Stacey (1970), Buckner & Stewart (1976), Asahi & Okumura (2001)].

3.1.2 The Flux Factorization Approach

The *flux factorization* approach [Ott & Meneley (1969)] is formally closest to the point kinetics method since it factorizes the space-time dependent flux into a purely time-dependent amplitude p(t) and a weakly time-dependent shape [Ψ]

$$[\Phi] = p(t)[\Psi], \qquad [\Psi] \equiv \{\psi_g(\vec{r},t) \mid g = 1,..,G\} \tag{3.1.4.}$$

Unlike the point kinetics, the flux factorization allows for the shape function to vary with time; therefore, it is formally an exact method. Since the shape function normally varies much more slowly with time than the amplitude, the overall computation time can be significantly reduced by choosing much larger time intervals between shape calculations than between amplitude calculations. Although the amplitude must be recomputed more frequently, its computation is relatively inexpensive.

To derive an equation for the amplitude p(t), an arbitrary set of time-independent weighting functions $[W] \equiv \{w_g(\vec{r})\}$ is chosen and multiplied with the original kinetics equations before integration over space. To make the factorization unique, and to facilitate the derivation as well, the constraint condition for the variation of the flux shape is imposed such that

$$\langle [W],[V^{-1}][\Psi] \rangle = \text{const} \tag{3.1.5}$$

Here, $\langle [W],[V^{-1}][\Psi] \rangle$ is the *inner product* of two vectors [W] and $[V^{-1}][\Psi]$, which may be explicitly expressed as

$$\langle [W],[V^{-1}][\Psi] \rangle \equiv \sum_{g=1}^{G} \iiint_{Vcore} w_g(\vec{r}) \frac{\psi_g(\vec{r},t)}{v_g} d^3r$$

The weighting functions are often chosen to be the solution $[\Phi_0^*]$ of the *static adjoint system* of the multigroup diffusion equations corresponding to the initial steady state.

Substituting the flux expansion (3.1.4) into the original kinetics equations (2.2.21) and taking the inner product with the weighting vector [W], one obtains a set of ordinary differential equations, quite similar to the point kinetics system (3.1.3), for the flux amplitude

$$\frac{d}{dt}p(t) = \frac{\rho(t) - \bar{\beta}(t)}{\Lambda(t)}p(t) + \sum_{i=1}^{N}\lambda_i\zeta_i(t) + \xi(t) \qquad (3.1.6a)$$

$$\frac{d}{dt}\zeta_i(t) = -\lambda_i\zeta_i(t) + \frac{\bar{\beta}_i(t)}{\Lambda(t)}p(t), \qquad i = 1,..,N \qquad (3.1.6b)$$

where

$$\zeta_i(t) \equiv \frac{\langle [W], C_i(\vec{r},t)[\chi_i^d]\rangle}{\langle [W],[V^{-1}][\Psi]\rangle},$$

$$\xi(t) \equiv \frac{\langle [W],[S_{ext}]\rangle}{\langle [W],[V^{-1}][\Psi]\rangle}$$

$$\Lambda(t) \equiv \frac{\langle [W],[V^{-1}][\Psi]\rangle}{\langle [W],[F_P + F_d][\Psi]\rangle}, \qquad \text{with} \quad [F_d] \equiv \sum_{i=1}^{N}\beta_i[\chi_i^d][F]$$

$$\rho(t) \equiv 1 - \frac{\langle [W],[M][\Psi]\rangle}{\langle [W],[F_P + F_d][\Psi]\rangle}$$

$$\bar{\beta}_i(t) \equiv \frac{\langle [W],\beta_i[\chi_i^d][F][\Psi]\rangle}{\langle [W],[F_P + F_d][\Psi]\rangle}, \qquad \bar{\beta}(t) \equiv \sum_{i=1}^{N}\bar{\beta}_i(t)$$

To obtain an equation for the shape function, one simply substitutes the flux expansion (3.1.4) into the original flux equation (2.2.21a) and divides both sides by p(t)

$$[V^{-1}]\frac{\partial}{\partial t}[\Psi] + [V^{-1}][\Psi]\frac{\dot{p}(t)}{p(t)} =$$
$$[F_p - M][\Psi] + \frac{1}{p(t)}\sum_{i=1}^{N}\lambda_i C_i(\vec{r},t)[\chi_i^d] + \frac{1}{p(t)}[S_0] \qquad (3.1.7)$$

All the equations derived so far are still quite exact since no approximations have been introduced. A sequence of further approximations will lead to a sequence of approximate methods for finding the shape function. The first approximation is to replace the time derivative of the shape function in equation (3.1.7) with the backward difference

$$\frac{\partial}{\partial t}[\Psi] \cong \frac{1}{\Delta t_s}\{[\Psi(\vec{r}, t_j)] - [\Psi(\vec{r}, t_{j-1})]\} \tag{3.1.8}$$

resulting in the so-called *Improved Quasistatic* (IQS) method. The IQS shape equation has the form

$$\left(F_p - M - \frac{[V^{-1}]}{\Delta t_s} - [V^{-1}]\frac{\dot{p}}{p}\right)[\Psi(\vec{r}, t_j)] = - \frac{1}{p(t')}\sum_{i=1}^{N}\lambda_i C_i(\vec{r}, t')[\chi_i^d] + \frac{[V^{-1}]}{\Delta t_s}[\Psi(\vec{r}, t_{j-1})] - \frac{1}{p(t')}[S_0] \tag{3.1.9}$$

where $\Delta t_s = t_j - t_{j-1}$ is the time step for shape calculations and t' is the last time when the amplitude is computed. Since the amplitude varies rapidly, the set of amplitude equations (3.1.7) is solved using a much smaller time step, i.e. $t_{j-1} << t' < t_j$. However, the shape function $[\Psi]$ present in the integral parameters at an intermediate time between shape calculations must be somewhat estimated, mostly by extrapolation.

In the next approximation, leading to the *quasistatic method*, the time derivative of the shape function is neglected (i.e. $\partial[\Psi]/\partial t \cong 0$) to simplify the shape equation to

$$\left(F_p - M - [V^{-1}]\frac{\dot{p}}{p}\right)[\Psi(\vec{r}, t)] = -\frac{1}{p(t')}\sum_{i=1}^{N}\lambda_i C_i(\vec{r}, t')[\chi_i^d] - \frac{1}{p(t')}[S_{ext}] \tag{3.1.10}$$

The quasistatic problem is formally a static problem with cross sections depending parametrically on time. Further, in equation (3.1.10), if the delayed neutron source is approximated as a fraction of the prompt source, and if the $[V^{-1}]\dot{p}/p$ term is neglected, the specific kinetics features in the shape equation are completely eliminated, resulting in the *adiabatic method*. Finally, neglect of the time dependence of the operators [M] and [F] in equation (3.1.10) reduces the problem to the simple point kinetics model.

As the degree of sophistication is reduced from the IQS to the quasistatic and then to the adiabatic and finally to the point kinetics method, the accuracy of the solution diminishes. For solving problems involving feedback, the amount of complexity and computational effort required for all but the point kinetics method is roughly the same. Thus, it would seem that there is no advantage to the use of any factorization method

other than the IQS method for spatial kinetics calculations. The flux factorization approach is useful when implemented in conjunction with other spatial methods in order to lengthen computational time steps [Koclas et al. (1996), Ikeda & Takeda (2001)].

3.1.3 The Modal Approach

The conceptual idea of the modal approach [Stacey (1969)] is to discretize the kinetics equations by using a function expansion technique. That is, the space and time-dependent group fluxes are expanded in terms of the pre-computed functions (modes) $\{\psi_k^g(\vec{r})\}$ and the unknown expansion coefficients $\{p_k^g(t)\}$ of the form

$$[\Phi] = \sum_{k=1}^{K} [\Psi_k][P_k] \tag{3.1.11}$$

where $[\Psi_k] \equiv \text{Diag}\{\psi_k^g(\vec{r})\}$ and $[P_k] \equiv \text{Col}\{p_k^g(t)\}$, g = 1,..,G; k = 1,..,K. To save computation time, only a very few modes are employed. Moreover, these expansion modes can be computed with simple methods, for example, as eigenfunctions of some form of the multigroup diffusion functions [Wachspress (1966), Stacey (1969)] or solutions of static problems of normally lower dimensionality [Kaplan et al. (1964), Hetrick (1971)] (the latter is usually referred to as the *synthesis method*). It is obvious that the use of only one term in the above expansion (i.e. K = 1) corresponds to the point reactor model if the flux mode is chosen as the initial one-speed flux.

The modal equation system for the expansion coefficients can be obtained by using a weighted residual procedure. For this, a set of time-independent weighting functions $\{w_m(\vec{r}) \mid m = 1,..,M\}$ is chosen. By substituting the flux expansion expression (3.1.11) into the original kinetics equations (2.2.21), pre-multiplying every equation in the system one-by-one with each of the weighting functions, and integrating all over the core volume, one obtains the following system for expansion coefficients

$$\sum_{k=1}^{K} [\Gamma_k^m] \frac{d}{dt}[P_k] = \sum_{k=1}^{K} [H_k^m][P_k] + \sum_{i=1}^{N} \lambda_i \zeta_i^m(t)[\chi_i^d] + [S_0^m], \qquad m = 1,..,M \tag{3.1.12a}$$

$$\frac{d}{dt}\zeta_i^m(t) = -\lambda_i \zeta_i^m(t) + \beta_i \sum_{k=1}^{K} [\Pi_k^m][P_k], \qquad i = 1,..,N; \quad m = 1,..,M \tag{3.1.12b}$$

where

$$[\Gamma_k^m] \equiv \iiint_{Vcore} w_m(\vec{r})[V^{-1}][\Psi_k]d^3r$$

$$[H_k^m] \equiv \iiint_{Vcore} w_m(\vec{r})[F_p - M][\Psi_k]d^3r$$

$$[\Pi_k^m] \equiv \iiint_{Vcore} w_m(\vec{r})[F][\Psi_k]d^3r$$

$$\zeta_i^m(t) \equiv \iiint_{Vcore} w_m(\vec{r})C_i(\vec{r},t)d^3r \text{ - the integral weighted concentrations}$$

$$[S_{ext}^m] \equiv \iiint_{Vcore} w_m(\vec{r})[S_{ext}(\vec{r},t)]d^3r \text{ - the integral weighted extra sources.}$$

Solution of these ordinary differential equations would yield the expansion coefficients $[P_k]$. The group fluxes are then constructed using expression (3.1.11).

The modal kinetics model is usually more accurate than the point kinetics model because it accounts for the flux variation in both space and energy during a transient. Modal kinetics computation is fast if compared with other spatial kinetics methods. However, the lack of systematic criteria for the error analysis has made the modal methods almost vanish from practical reactor kinetics calculations. That is, while the use of a greater number of mesh points in finite difference discretization will actually improve accuracy of the solution, the use of a greater number of expansion modes does not guarantee the same result. In addition, it is difficult to formulate a suitable algorithm for solving the modal discretized equations on a digital computer [Ott & Neuhold (1985)].

3.1.4 Finite Difference Methods

Finite difference methods (FDM) are the simplest and most direct approach to the solution of any space-time problems. The FDM basically consists of replacing the spatial derivative in the neutron kinetics equations by the corresponding finite difference approximation. For this, the geometric domain - the reactor core volume extended to its boundary at an extrapolated length - is partitioned into a number of subregions. In each of such regions (or *cells*), the material properties are spatially averaged and hence are assumed to be uniform. For simplicity, consider the discretization in 3D Cartesian geometry $\vec{r} = \{x,y,z\}$ with the cells in form of rectangular boxes defined by a series of planes parallel with each of the three coordinate directions. However, it is also possible to use other cell forms (such as quadrilaterals, triangles, and hexagons [Abu-Shumays & Hageman (1975)]). Two types of discretization can be used: the cell-centred and the vertex-centred. In the cell-centred discretization, the unknowns (group fluxes and precursor

concentrations) are defined at the cell centre (Fig.3.1.1), while in the vertex-centred discretization they are defined at each cell corner [Mitchell & Griffiths (1980)]. As a result, a grid of mesh points at which a set of discrete values of fluxes and concentrations is to be computed is formed.

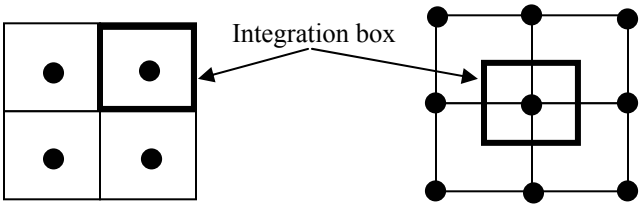

Figure 3.1.1. Cell-centered and vertex-centered discretization grids

One now performs the integration of the kinetics equations with respect to space over volume V_p of the box surrounding each mesh point P (Fig.3.1.1). In the cell-centered discretization, the integration box is the same as the cell that contains the given mesh point, while in the vertex-centered case, the integration box is formed by six planes perpendicular to the three coordinate directions at the midpoints between point P and its six neighbouring points which are left (L) or right (R) of the node in each direction $u \in (x,y,z)$ (Fig.3.1.2).

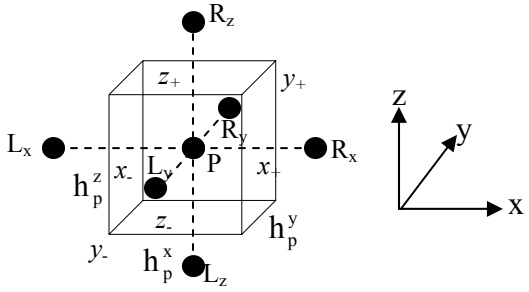

Figure 3.1.2. An integration box

The fluxes and precursor concentrations within each integration box are assumed approximately equal to their values at point P, i.e.

$$\phi_g(\vec{r},t) \cong \phi_g^p(t)$$

and

$$C_i(\vec{r},t) \cong C_i^p(t),$$

for $\vec{r}$ belongs in the box containing point P.

The result of the box integration is given by

$$\frac{1}{v_g}\frac{d}{dt}\phi_g^P(t) = -\sum_{u=x,y,z}\frac{J_{u+}^p(t)-J_{u-}^p(t)}{h_u^p} - \Sigma_{tg}^p\phi_g^p(t) + \sum_{g'=1}^{G}\left[\Sigma_{sg'g}^p + (1-\beta)\chi_g\nu\Sigma_{fg'}^p\right]\phi_{g'}^p(t) + \sum_{i=1}^{N}\lambda_i C_i^p(t)\chi_{i,g}^d + S_{0g}^p(t) \qquad (3.1.13a)$$

$$\frac{d}{dt}C_i^p(t) = -\lambda_i C_i^p(t) + \beta_i\sum_{g'=1}^{G}\nu\Sigma_{fg'}^p\phi_{g'}^p(t), \quad i = 1,..,N \qquad (3.1.13b)$$

where

$\Sigma_{\alpha,g}^p \equiv \frac{1}{V_p}\iiint_{V_P}\Sigma_{\alpha,g}(\vec{r})d^3r$ - the averaged cross section of type α within volume $V_p \equiv h_x^p\, h_y^p\, h_z^p$ of the integration box

$J_{u\pm}^p(t) \equiv J_u(\vec{r},t)\big|_{u_\pm^p}$ - the components the net current in the u-direction at the box surfaces, $u_\pm^p = u^p \pm \tfrac{1}{2}h_u^p$, $u \in \{x,y,z\}$

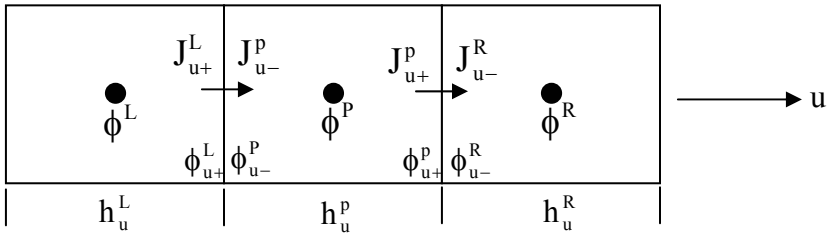

Figure 3.1.3. Fluxes and current components at integration box interfaces

The components of the net current at the faces of the integration box are determined by a finite difference approximation to the Fick's law expression as

$$J_{u\pm}^p(t) = \mp D_g^p\frac{\partial}{\partial u}\phi(x,y,z,t)\big|_{u_\pm^p} \approx \mp D_g^p\frac{\phi_{gu\pm}^p(t)-\phi_g^p(t)}{h_u^p/2}$$

where

$D_g^p \equiv \frac{1}{V_p} \iiint_{V_P} D_g(\vec{r}) d^3r$ - the diffusion coefficient averaged over box volume V_p

$\phi_{u\pm}^p(t) \equiv \phi(x,y,z,t)\big|_{u_\pm^p}$ - the flux defined at surfaces $u_\pm^p$ of the integration box

It is possible to eliminate the surface fluxes by using the continuity conditions of the flux and current at the interface between two adjacent integration boxes (Fig.3.1.3):

$$\phi_{u+}^p(t) = \phi_{u-}^R(t)$$

and

$$J_{u+}^p(t) = J_{u-}^R(t)$$

or

$$-D_g^p \frac{\phi_{gu+}^p(t) - \phi_g^p(t)}{h_u^p/2} = D_g^R \frac{\phi_{gu-}^R(t) - \phi_g^R(t)}{h_u^R/2}$$

$$\therefore \qquad J_{u+}^p(t) = \frac{-2}{h_u^p/D_g^p + h_u^R/D_g^R}\left(\phi_g^R(t) - \phi_g^p(t)\right) \tag{3.1.14a}$$

Similarly,

$$J_{u-}^p(t) = \frac{-2}{h_u^p/D_g^p + h_u^L/D_g^L}\left(\phi_g^p(t) - \phi_g^L(t)\right) \tag{3.1.14b}$$

Elimination of the currents from equation (3.1.13a) yields

$$\frac{1}{v_g}\frac{d}{dt}\phi_g^P(t) = \sum_{nb\subset P} a_g^{nb}\phi_g^{nb}(t) - \left(\sum_{nb\subset P} a_g^{nb} + \Sigma_{tg}^p\right)\phi_g^p(t) + \sum_{g'=1}^{G}\left[\Sigma_{sg'g}^p + (1-\beta)\chi_g \nu\Sigma_{fg'}^p\right]\phi_{g'}^P(t) + \sum_{i=1}^{N}\lambda_i C_i^p(t)\chi_{i,g}^d + S_{0g}^P(t) \tag{3.1.15}$$

where the algebraic coefficient coupling point P with its neighbour 'nb', i.e. the point either right or left of P in each direction, is given by

$$a_g^{nb} = 2\left(\frac{h_u^p}{D_g^p} + \frac{h_u^{nb}}{D_g^{nb}}\right)^{-1} \frac{1}{h_u^p} \tag{3.1.16}$$

Equation (3.1.15) derived above is for any inner point of the grid, i.e. the point not on the computational boundary. The flux equations for boundary points are derived directly from the extrapolated boundary condition, simply

$$\phi_g^B(t) = 0, \qquad P \equiv B \text{ on the boundary}$$

The finite difference technique described above in deriving equation (3.1.15) is second order accurate [Waschpress (1966)]. By using a Taylor expansion for accuracy analysis, one can show that the truncation error in (3.1.14) is proportional to quadratic spacing, i.e. $O(h^2)$. To obtain acceptable accuracy of the approximation (3.1.14), the mesh spacing h_u^p must be of order of the smallest group diffusion length (which is less than 2cm in a light water reactor - LWR). For most reactor problems of practical interest, this fine mesh requirement results in the use of an extremely large number of mesh points and hence equations to solve for. Moreover, an implicit scheme of time integration is likely to be used in order to overcome the problem stiffness, giving rise to an extremely large number of algebraic equations that must be simultaneously solved at each time step. In addition to extensive computer resources required, there is a greater problem of inverting such a large algebraic system.

Although being the most computationally expensive, the finite difference methods have found a number of applications in the spatial kinetics calculation thanks to their many desirable features. A great advantage of the FDM is that the discretization error is guaranteed to vanish in the fine mesh limit. Also, the algebraic systems of finite difference equations are well studied by mathematicians and there exist many sophisticated, fast methods for solving them. As a result, the FDM is still used for benchmarking of other spatial methods such as coarse mesh or nodal methods.

3.1.5 Coarse Mesh Methods

The inefficiency of finite difference methods in dealing with the spatial kinetics problem has led to the development of various methods that utilize a much larger mesh size for spatial discretization. Coarse mesh methods take advantage of the fact that in many practical reactor calculations it is possible to define equivalent homogenized group constants that allow large regions - such as entire assemblies - to be treated as if they were spatially homogenized. By using relatively large mesh spacing for discretization, these methods increase the computational efficiency by reducing the number of discretized equations that must be solved. To achieve acceptable accuracy even with mesh spacing larger than the diffusion length, these methods must employ a higher order

spatial treatment within a mesh box than does the finite difference method. Although the computational work per mesh box is typically greater than that for the finite difference method, the reduction in the number of mesh boxes often results in a substantial reduction in computational time and storage requirements.

The *flux expansion method* is a coarse mesh method implemented by Langenbuch et al. (1977). This method divides the core volume into a number of non-overlapped mesh boxes V_p, such that $V = \cup V^p$. The group constants are assumed to be uniform throughout the box volume. Within a mesh box the kinetics equations can be written as

$$\frac{1}{v_g}\frac{\partial}{\partial t}\phi_g(\vec{r},t) = D_g^p \nabla^2 \phi_g(\vec{r},t) - \Sigma_{tg}^p \phi_g(\vec{r},t) + S_g(\vec{r},t) \tag{3.1.17a}$$

$$\frac{\partial}{\partial t} C_i(\vec{r},t) = -\lambda_i C_i(\vec{r},t) + \beta_i \sum_{g'=1}^{G} \nu\Sigma_{fg'}^p \phi_{g'}(\vec{r},t) \tag{3.1.17b}$$

The flux expansion method solves equations (3.1.17) using a symmetric weighted residual method with polynomial basis functions defined for each mesh box. The basis functions, quadratic or cubic polynomials, have points of support at the box centre and at the centres of six faces, and are identically zero everywhere outside the box. The shape of the flux within a mesh box is approximated in terms of the basis functions and the flux values - as expansion coefficients - at the points of support. Algebraic equations are obtained by substituting this approximation into the kinetics equations and using the weighting residual procedure.

Finite element methods (FEM) may also be considered as coarse mesh methods. In the FEM employed by Kang & Hansen (1973), the group flux is approximated as the sum of multi-dimensional polynomials that are identical zero everywhere outside the volume element surrounding the mesh point. Linear basis functions are usually chosen, but higher order functions, e.g. cubic polynomials, are also possible. Regardless of the degree of the polynomials used, a system of first order differential equations in time is obtained.

In general, coarse mesh methods are more accurate than the FDM for large mesh spacing; for example, the use of linear basis functions in FEM allows mesh spacing about twice that of the FDM for the same degree of accuracy. The FEM has the additional advantage of not limiting to a regular mesh. This allows great flexibility in adjusting the mesh size, allowing it to be reduced in regions that require more resolution and expanded in regions where the flux has little spatial variation. The disadvantage is that this would result in a more complicated structure of the algebraic coefficient matrix. Nevertheless, in terms of accuracy and efficiency, the coarse mesh methods are only slightly better than the FDM but cannot compete with the family of nodal methods presented in the next section.

3.1.6 Nodal Methods

Very much resembling coarse-mesh methods, nodal methods [Lawrence (1986)] utilize relatively large mesh discretization in order to significantly reduce computational resources. In the nodal method, reactor volume V is partitioned into a number of relatively large (e.g. assembly-size) non-overlapped regions (or *nodes*) V^p, so that $V=\cup V^p$. In the Cartesian geometry formulation, the grid of nodes are formed by parallel planes perpendicular to each of the three coordinate directions x, y, and z. Other geometries (e.g. hexagonal-z geometry [Zimin & Baturin (2002)]) are also possible and easily accommodated. The material properties (i.e. the group constants) in each node are spatially averaged (i.e. homogenized), and hence are assumed to be uniform throughout node volume. By integrating kinetics equations with respect to space over node volume V^p, the nodal balance equations are obtained as

$$\frac{1}{v_g}\frac{d}{dt}\bar{\phi}_g^p(t) = -\sum_{u=x,y,z}\frac{\bar{J}_{gu+}^p(t)-\bar{J}_{gu-}^p(t)}{h_u^p} - \Sigma_{tg}^p(t)\bar{\phi}_g^p(t) + \bar{S}_g^p(t) \qquad (3.1.18a)$$

$$\bar{C}_i^P(t) = -\lambda_i\bar{C}_i^P(t) + \beta_i\sum_{g'=1}^{G}\nu\Sigma_{fg'}^P\bar{\phi}_{g'}^P(t), \qquad i = 1,..,N \qquad (3.1.18b)$$

where

$\bar{\phi}_g^p(t) \equiv \frac{1}{V^P}\iiint_{V^P}\phi_g(\vec{r},t)d^3r$ - the node-averaged group flux

$\bar{C}_i^P(t) \equiv \frac{1}{V^P}\iiint_{V^P}C_i(\vec{r},t)d^3r$ - the node-averaged group precursor concentration

$\bar{S}_g^p(t) \equiv \sum_{g'=1}^{G}\left[\Sigma_{sg'g}^p(t)+(1-\beta)\chi_g\nu\Sigma_{fg'}^p(t)\right]\bar{\phi}_{g'}^p(t) + \sum_{g'=1}^{G}\lambda_i\bar{C}_i^p(t)\chi_{i,g}^d + \bar{S}_{0g}^p(t)$

- the node-averaged group source

$\bar{J}_{gx\pm}^p(t) \equiv \frac{1}{h_y^P h_z^P}\int_{y_-^P}^{y_+^P}dy\int_{z_-^P}^{z_+^P}dzJ_{gx}(x_\pm^p,y,z,t)$, and similarly for $\bar{J}_{gy\pm}^p(t)$ and $\bar{J}_{gz\pm}^p(t)$

- the node face-averaged net currents in the corresponding direction

Here, $u_\pm^p = u^p \pm \frac{1}{2}h_u^p$, $u\in\{x, y, z\}$, denote the coordinates of the right (+) and left (-) faces, respectively, of the node in the u-direction. Further, it is convenient to express the

face-averaged *net* currents in terms of the face-averaged *partial* currents $\bar{j}^{\pm p}_{gu\pm}(t)$ as (Fig.3.1.4):

$$\bar{J}^{p}_{gu\pm}(t) = \bar{j}^{+p}_{gu\pm}(t) - \bar{j}^{-p}_{gu\pm}(t)$$

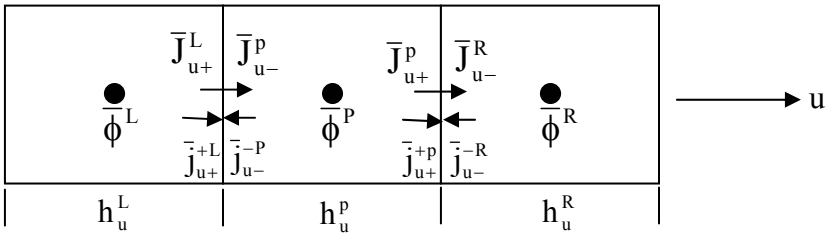

Figure 3.1.4. Node-averaged flux and face-averaged currents

The idea behind all nodal methods is to establish the spatial coupling relationships between the node face-averaged net or partial currents and the node-averaged fluxes by applying high-order approximations to the spatial variation of the flux within a node. Different nodal methods employ different procedures to derive such relationships, making them distinguished from each other. These coupling relationships are then used to eliminate the currents in the nodal equation (3.1.18a), thus yielding a first-order differential system of purely time-dependent nodal equations (still coupled in space and energy). It can be seen that, a nodal scheme would be identical to the cell-centred finite difference method had one assumed a linear shape of the flux within a node. Such a linear approximation of the flux would require very fine mesh discretization for an acceptable accuracy. Thus, in a nodal method, a somewhat higher order spatial treatment of the flux within a node is employed to reduce discretization error.

Early Nodal Methods

The *early nodal methods*, developed in 1960s, employ some sort of coupling parameters to relate the face-averaged partial currents to the node-averaged flux as

$$\bar{j}^{\pm p}_{gu\pm}(t) = \alpha^{\pm p}_{gu}\, h^{p}_{u}\, \bar{\phi}^{p}_{g}(t) \qquad (3.1.19)$$

where $\alpha^{\pm p}_{gu}$ are referred to as the *coupling coefficients* at the interface between node P and the node right (R) or left (L) of it in the u-direction. These relationships can be used to eliminate the currents in equation (3.1.18a), yielding the semi-discretized nodal system

$$\frac{1}{v_g}\frac{d}{dt}\bar{\phi}_g^p(t) = -\left[\sum_{u=x,y,z}\left(\alpha_{gu}^{+p}+\alpha_{gu}^{-p}\right)+\Sigma_{tg}^p\right]\bar{\phi}_g^p(t) + \sum_{u=x,y,z}\left[\alpha_{gu}^{+L}\bar{\phi}_g^{L_u}(t)+\alpha_{gu}^{-R}\bar{\phi}_g^{R_u}(t)\right] + \bar{S}_g^p(t) \qquad (3.1.20)$$

The coupling coefficients $\alpha_{gu}^{\pm p}$ are usually obtained from a more accurate fine-mesh calculation of a steady-state diffusion or transport problem as reference. Nodal methods of this type may work well when the problem being analyzed is sufficiently *close* to the reference problem at which the coupling coefficients were computed; otherwise, they often produce disastrous results. This appears to be a particularly serious problem in kinetics calculations, where the flux shape within a node - and hence the nodal coupling - can vary considerably during the course of calculation. Despite some improvements in determining the coupling parameters without matching any particular reference fine-mesh calculation, the nodal methods of this type have not escaped the criticism of being *inconsistent methods* which rely on 'tuned' parameters [Moulton (1996)].

Modern Nodal Methods

In searching for consistent discretization methods for solution of the nodal equations (3.1.18), Finnemann et al. (1977) developed a *transverse integration* procedure, which has found the greatest acceptance among the reactor physics community for reactor calculations. Starting from the original multigroup diffusion equations (2.2.21), the transverse integration procedure transforms each 3D equation in the system into three 1D *transverse-integrated equations* by integrating over two of the three directions. The transverse-integrated balance equations in any direction $u \in \{x,y,z\}$ are given by

$$\frac{1}{v_g}\frac{\partial}{\partial t}\bar{\phi}_{gu}^P(u,t) = -\frac{\partial}{\partial u}\bar{J}_{gu}^P(u,t) - \Sigma_{tg}^P(t)\bar{\phi}_{gu}^p(u,t) + \bar{S}_{gu}^P(u,t) - \bar{L}_{gu}^P(u,t) \qquad (3.1.21a)$$

$$\bar{J}_{gu}^P(u,t) = -D_g^P(t)\frac{\partial}{\partial u}\bar{\phi}_{gu}^P(u,t), \qquad g = 1,..,G \qquad (3.1.21b)$$

$$\frac{\partial}{\partial t}\bar{C}_{iu}^P(u,t) = -\lambda_i\bar{C}_{iu}^P(u,t) + \beta_i\sum_{g'=1}^{G}\nu\Sigma_f^P(t)\bar{\phi}_{g'}^P(u,t), \; i = 1,..,N \qquad (3.1.21c)$$

where

$$\overline{(\cdot)}_{gx}^p(x,t) \equiv \frac{1}{h_y^p h_z^p}\int_{y_-^P}^{y_+^P}dy\int_{z_-^P}^{z_+^P}dz(\cdot)(x,y,z,t), \qquad [(\cdot) \text{ stands for } \phi_g, J_g \text{ or } C_i]$$

as well as the similar expressions in y- and z-directions, denote the node *transverse-averaged* quantities of fluxes $\overline{\phi}^{P}_{gu}(u,t)$, net currents $\overline{J}^{P}_{gu}(u,t)$ or precursor concentrations $\overline{C}^{P}_{iu}(u,t)$

$$\overline{S}^{P}_{gu}(u,t) = \sum_{g'=1}^{G}\left[\Sigma^{p}_{sg'g}(t) + (1-\beta)\chi_g \nu\Sigma^{p}_{fg'}(t)\right]\overline{\phi}^{p}_{g'u}(u,t) + \sum_{g'=1}^{G}\lambda_i \overline{C}^{p}_{iu}(u,t)\chi^{d}_{i,g} + \overline{S}^{p}_{0gu}(u,t)$$

- the transverse-averaged source

$$\overline{L}^{P}_{gx}(x,t) \equiv \frac{1}{h^{p}_{y}}\int_{y^{P}_{-}}^{y^{P}_{+}} dy \frac{J_{gz}(x,y,z^{p}_{+},t) - J_{gz}(x,y,z^{p}_{-},t)}{h^{p}_{z}} + \frac{1}{h^{p}_{z}}\int_{z^{P}_{-}}^{z^{P}_{+}} dz \frac{J_{gy}(x,y^{p}_{+},z,t) - J_{gy}(x,y^{p}_{-},z,t)}{h^{p}_{y}}$$

and the similar expressions for $\overline{L}^{P}_{gy}(y,t)$ and $\overline{L}^{P}_{gz}(z,t)$ as well are the *transverse leakages*

The transverse leakage $\overline{L}^{P}_{gu}(u,t)$ in equation (3.1.21) is usually approximated in some way. Early implementations of the transverse-integrated nodal method used a *flat approximation* in which the transverse leakage is taken constant in the longitudinal direction and equal to

$$\overline{L}^{P}_{gx}(x,t) \cong \frac{\overline{J}^{p}_{gy+}(t) - \overline{J}^{p}_{gy-}(t)}{h^{p}_{y}} + \frac{\overline{J}^{p}_{gz+}(t) - \overline{J}^{p}_{gz-}(t)}{h^{p}_{z}}$$

It has been found, however, that this approximation does not adequately represent the true shape of the transverse leakage and may lead to a substantial error. To reduce the error, most current transverse integrated nodal methods employ a more accurate approximation of the transverse leakage in a *quadratic polynomial expansion*

$$\overline{L}^{P}_{gu}(u,t) \cong \widetilde{L}^{P}_{gu}(u,t) = \overline{L}^{P}_{gu0}(t) + \overline{L}^{P}_{gu1}(t)P_1\left(\frac{u-u^{p}}{h^{P}_{u}}\right) + \overline{L}^{P}_{gu2}(t)P_2\left(\frac{u-u^{p}}{h^{P}_{u}}\right)$$

where $P_k(\xi)$ is a polynomial of order k (k = 0, 1, 2), and the expansion coefficients $\overline{L}_{guk}(t)$ are chosen so as to preserve the average transverse leakage of the node as well as that of its neighbouring nodes in the u-direction [Sutton & Aviles (1996)].

In addition to the transverse leakage term, the time derivatives in the transverse integrated equations are also approximated such that

$$\frac{\partial}{\partial t}\,\bar{\phi}^{P}_{gu}(u,t) \cong \omega^{p}_{gu}(t)\,\bar{\phi}^{P}_{gu}(u,t)$$

and

$$\frac{\partial}{\partial t}\,\bar{C}^{P}_{iu}(u,t) \cong \vartheta^{p}_{iu}(t)\,\bar{C}^{P}_{iu}(u,t)$$

where $\omega^{p}_{gu}(t)$ and $\vartheta^{p}_{iu}(t)$ are referred to as the *dynamic frequencies*. With these dynamics frequencies introduced, it is possible to solve (3.1.21c) for the precursor concentrations

$$\bar{C}^{P}_{iu}(u,t) = \frac{\beta_i \sum_{g=1}^{G} \nu\Sigma^{p}_{fg}(t)\bar{\phi}^{p}_{gu}(u,t)}{\vartheta^{p}_{iu}(t)+\lambda_i} \tag{3.1.22}$$

Substituting (3.1.22) into (3.1.21a), one arrives at the *transverse-integrated equations*

$$\frac{\partial}{\partial u}\,\bar{J}^{P}_{gu}(u,t) + \left[\Sigma^{P}_{tg}(t) + \frac{\omega^{p}_{gu}(t)}{v_g}\right]\bar{\phi}^{p}_{gu}(u,t) - \sum_{g'=1}^{G}\left\{\Sigma^{P}_{sg'g}(t) + \left[(1-\beta)\chi_g + \sum_{i=1}^{N}\frac{\beta_i\lambda_i\chi^{d}_{ig}}{\vartheta^{p}_{iu}(t)+\lambda_i}\right]\nu\Sigma^{p}_{fg'}\right\}\bar{\phi}^{p}_{g'u}(u,t) = \tilde{L}^{P}_{gu}(u,t) \tag{3.1.23}$$

From this point, there are two distinct applications of the transverse-integrated equations (3.1.23) in constructing a nodal scheme. The first is the *Nodal Expansion Method* that expands the transverse-integrated flux in a set of polynomials. The second is *Analytic Nodal Method* that solves transverse-integrated equations analytically.

The Nodal Expansion Method

The Nodal Expansion Method (NEM) [Finnemann et al. (1977)] was originally developed for the static neutron diffusion calculation and has been easily extended to reactor kinetics problems [Lawrence (1986), Sutton & Aviles (1996)]. The NEM is based on a polynomial expansion of the transverse integrated flux

$$\bar{\phi}^{P}_{gu}(u,t) = \sum_{k=1}^{K} a^{p}_{guk}(t)P_k\left(\frac{u-u^{p}}{h^{P}_{u}}\right) \tag{3.1.24}$$

where a set of polynomial functions $P_k(\xi)$ are chosen as

$$P_0(\xi) = 1,$$

$$P_1(\xi) = \xi,$$

$$P_2(\xi) = 3\xi^2 - 1/4,$$

$$P_3(\xi) = \xi(\xi - 1/2)(\xi + 1/2),$$

$$P_4(\xi) = (\xi^2 - 1/20)(\xi - 1/2)(\xi + 1/2), \qquad \text{etc.}$$

In NEM, the expansions of degree lower than 2 are not considered, and more than 4 are also unlikely due to great complexity in finding the expansion coefficients $a^p_{guk}(t)$. Using the given set of polynomials, the low order expansion coefficients can be found in terms of the partial currents and the node-averaged flux as

$$a^p_{gu0}(t) = \bar{\phi}^p_g(t)$$

$$a^p_{gu1}(t) = 2[\bar{j}^{+p}_{gu+}(t) + \bar{j}^{-p}_{gu+}(t)] - 2[\bar{j}^{+p}_{gu-}(t) + \bar{j}^{-p}_{gu-}(t)]$$

$$a^p_{gu2}(t) = 2[\bar{j}^{+p}_{gu+}(t) + \bar{j}^{-p}_{gu+}(t) + \bar{j}^{+p}_{gu-}(t) + \bar{j}^{-p}_{gu-}(t) - \bar{\phi}^P_g(t)]$$

The higher order ($k \geq 3$) expansion coefficients may be obtained by using a weighting residual procedure. For this, the transverse-integrated equation (3.1.21a), with the substitution of flux expansion (3.1.24), is multiplied by an arbitrary weighting function, each per coefficient being determined, and then integrated from u^p_- to u^p_+, resulting the desired relationships.

Once all expansion coefficients are found in terms of the nodal quantities, the flux expansion (3.1.24) is substituted into the transverse-integrated Fick's law equation (3.1.21b) as

$$\bar{J}^p_{gu\pm}(t) \equiv \bar{j}^{+p}_{gu\pm}(t) - \bar{j}^{-p}_{gu\pm}(t) = \mp \frac{D^p_g(t)}{h^p_u} \sum_{k=0}^{4} a^p_{guk}(t) \frac{d}{d\xi} P_k(\xi)\Big|_{\xi=\pm 1/2}$$

It is possible to express the outgoing partial currents in terms of the incoming partial currents and the node-averaged flux as

$$\bar{j}^{+p}_{gu+}(t) = \alpha^{p}_{gu0}(t)\bar{\phi}^{p}_{g}(t) + \alpha^{p}_{gu1}(t)\bar{j}^{-p}_{gu+}(t) + \alpha^{p}_{gu2}(t)\bar{j}^{+p}_{gu-}(t) + \alpha^{p}_{gu3}(t) + \alpha^{p}_{gu4}(t)$$

and

$$\bar{j}^{-p}_{gu-}(t) = \alpha^{p}_{gu0}(t)\bar{\phi}^{p}_{g}(t) + \alpha^{p}_{gu1}(t)\bar{j}^{+p}_{gu-}(t) + \alpha^{p}_{gu2}(t)\bar{j}^{-p}_{gu+}(t) - \alpha^{p}_{gu3}(t) + \alpha^{p}_{gu4}(t)$$

where $\alpha^{\pm p}_{guk}(t)$ are the algebraic coefficients that depend only on the ratio $D^{P}_{g}(t)/h^{P}_{u}$. These relationships are now used to eliminate the outgoing partial currents from the nodal equation (3.1.18), yielding the semi-discretized nodal equation of the form

$$\frac{1}{v_g}\frac{\partial}{\partial t}\bar{\phi}^{p}_{g}(t) = -\left[\Sigma^{p}_{tg} + \sum_{u=x,y,z}\frac{2\alpha^{p}_{gu0}}{h^{p}_{u}}\right]\bar{\phi}^{p}_{g}(t) + \sum_{u=x,y,z}\frac{1}{h^{p}_{u}}\left[\left(1-\alpha^{p}_{gu1}-\alpha^{p}_{gu2}\right)\left(j^{+p}_{gu-}(t)+j^{-p}_{gu+}(t)\right)-2\alpha^{p}_{gu4}\right] + \bar{q}^{p}_{g}(t) \qquad (3.1.25)$$

In equation (3.1.25), spatial coupling between node P and its neighbouring nodes is via the incoming partial currents since

$$\bar{j}^{+p}_{gu-}(t) = \bar{j}^{-R}_{gu-}(t) \qquad \text{and} \qquad \bar{j}^{-p}_{gu+}(t) = \bar{j}^{+L}_{gu+}(t)$$

The Analytic Nodal Method

Analytic transverse-integrated nodal methods (ANM) [Fu & Cho (2002)] are characterized by the use of the analytic solution of the transverse-integrated diffusion equation in constructing the nodal scheme. The ANM begins with combining the transverse-integrated equations (3.1.23) and (3.1.21b), written in matrix notation as

$$-\frac{d^2}{du^2}[\phi(u)] + [A][\phi(u)] = [\ell(u)] \qquad (3.1.26)$$

where

$[\phi(u)] = \text{Col}\{\bar{\phi}^{p}_{gu}(u)\}$ - the vector of the unknown fluxes

$[\ell(u)] = \text{Col}\{\bar{L}^{p}_{gu}(u)/D^{p}_{g}\}$ - the vector of the approximate leakage shapes

$[A]$ - the G×G buckling matrix.

Although all quantities in (3.1.26) are time-dependent, this equation can be considered as purely one-dimensional in space, so the time-dependence variable is omitted for simplicity. The analytic solution of this inhomogeneous equation consists of the homogenous solution and one particular solution of the inhomogeneous equation. The homogeneous solution is dictated by the eigenvalues $\{\lambda_g^p\}$ and eigenvectors $\{r_g^p\}$ of the matrix [A]. For each eigenvalue λ_g^p and its corresponding eigenvector r_g^p, there are two fundamental solutions

$$r_g^p \cosh\left(\sqrt{\lambda_g^p}\,u\right) \qquad \text{and} \qquad r_g^p \sinh\left(\sqrt{\lambda_g^p}\,u\right).$$

The homogenous solution is the linear combination of these 2G fundamental solutions. For general multigroup problem, it is possible that the eigenvalue is complex. Thus the nodal formulation may involve complex arithmetic, which is undesirable as long as the efficiency is considered. That is why ANM are usually restricted to no more than two neutron energy groups only (i.e. $G \le 2$).

Table 3.1.1. Basis solution functions X(u) and Y(u) [Fu (2002)]

X(u)	Y(u)	Eigenvalue
$\cosh\left(\sqrt{\lambda}u\right)$	$\sinh\left(\sqrt{\lambda}u\right)$	$\lambda > 0$
1	u	$\lambda = 0$
$\cos\left(\sqrt{\lvert\lambda\rvert}u\right)$	$\sin\left(\sqrt{\lvert\lambda\rvert}u\right)$	$\lambda < 0$
$\cosh^2(\alpha u)\cos^2(\beta u)$ $+ \sinh^2(\alpha u)\sin^2(\beta u)$	$\sinh^2(\alpha u)\cos^2(\beta u)$ $+ \cosh^2(\alpha u)\sin^2(\beta u)$	$\lambda = \alpha + i\beta$

Let $[\phi^{part}(u)]$ be one inhomogeneous particular solution of equation (3.1.26), then its general analytic solution is given by

$$[\phi(u)] = [R]\{[X(u)][c_1] + [Y(u)][c_2]\} + [\phi^{part}(u)] \qquad (3.1.27)$$

where [R] is a column of eigenvectors; $[c_1]$, $[c_2]$ are two unknown constant vectors; and [X(u)], [Y(u)] are two block diagonal matrixes of the functions depending on whether the eigenvalue is positive, zero, negative or complex (see Table 3.1.1).

The particular solution $[\phi^{part}(u)]$ can be determined by substituting (3.1.27) into equation (3.1.26) and solving for the unknown functions $[c_1(u)]$ and $[c_2(u)]$. The two unknown constant vectors $[c_1]$ and $[c_2]$ are subject to the boundary conditions. If the

analytic solution (3.1.27) is integrated over h_u^p, the odd constants $[c_1]$ can be determined in terms of the node-averaged fluxes $[\bar{\phi}] = \text{Col}\{\bar{\phi}_g^p\}$

$$[c_1] = \{[R][\bar{X}]\}^{-1}\{[\bar{\phi}] - [\bar{\phi}^{part}]\} \tag{3.1.28}$$

where $[\bar{X}]$ and $[\bar{\phi}^{part}]$ are the averages of $[X(u)]$ and $[\phi^{part}(u)]$ over the nodal width. The even constants $[c_2]$ can be determined in terms of the averaged fluxes and currents at node interfaces

$$[\phi_\pm] \equiv [\phi(u_\pm^p)] = [R]\{[X_\pm][c_1] + [Y_\pm][c_2]\} + [\phi_\pm^{part}] \tag{3.1.29}$$

$$[J_\pm] = -[D]\frac{d}{du}[\phi(u)|_{u_\pm^p}] = -[D]\{[\dot{X}_\pm][c_1] + [\dot{Y}_\pm][c_2]\} + [J_\pm^{part}] \tag{3.1.30}$$

Inserting (3.1.28) into (3.1.29) and (3.1.30) and eliminating $[c_2]$ in favour of the currents result in

$$[\phi_\pm] = \mp[\Gamma]\{[J_\pm] - [\hat{J}_\pm]\} + [\hat{\phi}_\pm] \tag{3.1.31}$$

where

$$[\Gamma] = [R]\{[Y_\pm][\dot{Y}_\pm]\}^{-1}[R]^{-1}[D]^{-1}$$

$$[\hat{J}_\pm] = \mp[D][R]\{[X_\pm][\dot{X}_\pm]\}^{-1}[R]^{-1}\{[\bar{\phi}] - [\bar{\phi}^{part}]\} \pm [J_\pm^{part}]$$

$$[\hat{\phi}_\pm] = [R]\{[X_\pm][\dot{X}_\pm]\}^{-1}[R]^{-1}\{[\bar{\phi}] - [\bar{\phi}^{part}]\} \pm [\phi_\pm^{part}]$$

The relationships between the face-averaged quantities and node-averaged flux are thus established. For two-group calculations, the 2×2 matrix eigenvalue problem can be solved without any difficulty, but for G > 2, its solution requires a sophisticated similarity transformation.

Compared to other methods for dealing with the spatial kinetics problem, the nodal methods described above have demonstrated the best performance in terms of discretization accuracy and computational efficiency. As a consequence, applications of other spatial methods for reactor kinetics calculations have significantly diminished, except for very special cases. However, the nodal methods are not free of difficulties and criticisms. As mentioned above, the early nodal methods are inconsistent, while the modern consistent nodal methods add more complexity and limitations. Also, lack of the mathematical foundation for nodal methods has made them rarely be applied beyond the

neutronics calculation. It is generally very difficult to analyze the nodal discretization error, and hence the benchmarking of nodal codes still relies on a finite difference counterpart. Also, the unusual choice of the nodal unknowns (i.e. node-averaged fluxes and faced-averaged currents) results in a complex nodal algebraic system that is incompatible with modern sophisticated, fast algebraic solvers. It is usually very difficult to accelerate the convergence of solution of a nodal discretized system of tens of thousands of equations (though this number is much smaller than that in a finite difference system for the same reactor problem). In addition, unlike the finite difference solution in which the detailed flux and hence reaction rate distributions in space are readily obtained, the direct nodal solution is actually for homogenized (node-averaged) quantities which require dehomogenization to obtain the reaction rate distribution within the node. In fact, the homogenization and dehomogenization procedures always rely on the use of fine mesh discretization [Sutton & Aviles (1996)].

3.2 Time Integration

After spatial treatment, the original kinetics system of space-time partial differential equations is reduced to a set of coupled ordinary differential equations of the only time dependence

$$\frac{d}{dt}[y] = [L][y], \qquad \text{for } t > 0 \tag{3.2.1a}$$

with the initial condition

$$[y(t = 0)] = [y_0] \tag{3.2.1b}$$

where [y] is the vector of the time-dependent unknowns - the values of group fluxes and delayed precursor concentrations at a discrete set of grid points in space, as well as of group partial or net currents at the faces of each node surrounding such a grid point if a nodal discretization technique is used; [L] is the matrix consisting of algebraic coefficients which depend only on the material and geometric properties of the grid (i.e. the group constants and the mesh size). Except for the point kinetics model, the unknowns in vector [y] are coupled in both space and energy, giving rise to a rather complicated structure of the matrix [L].

Since the group constants are generally functions of the neutron flux (through various feedback mechanisms), the matrix coefficients are also time-dependent and the system (3.2.1) is generally nonlinear. Analytical solution of such a system is impossible for any practical reactor problems. Even numerical solution is not easy either due to the stiffness of the neutron kinetics problem. Thus, the time constants characterizing the nuclear processes represented by the equations in the kinetics system range all the way from the lifetime of the neutrons in the fastest energy group (i.e. on order of their inverse

speed, ~10^{-8} sec) to the lifetime of the longest lived precursor (i.e. ~80 sec). Many standard numerical schemes are quite inefficient for integrating such a strongly stiff system since the time step size allowed for acceptable accuracy and numerical stability of the solution is determined by the smallest time constant [Devooght & Mund (1985)].

3.2.1 Integration Schemes

There exist many numerical schemes for integrating (3.2.1) over the time domain to obtain the spatial group fluxes [Sutton & Aviles (1996)]. Finite difference methods are the straightforward approach to temporal discretization of the time-dependent equations in the kinetics system (3.2.1) (or, indeed, of any parabolic differential equations). The most popular time integration schemes are the *explicit* (forward finite difference), *implicit* (backward finite difference), and *Crank-Nicholson* (central finite difference) schemes, which can be expressed in the following general form

$$\frac{[y(t+\Delta t)]-[y(t)]}{\Delta t} = \theta[L(t+\Delta t)][y(t+\Delta t)] + [1-\theta][L(t)][y(t)] + O(\Delta t^n) \qquad (3.2.2)$$

where Δt is the time step size; [y] and [L] are assumed known at time t but unknown at $t+\Delta t$; $O(\Delta t^n)$ is the discretization error and n is order of accuracy of the scheme. The *explicit*, *Crank-Nicholson* and the *fully implicit* schemes are obtained by setting the value of θ equal to 0, ½, and 1, respectively [Bru et al. (2002)].

In the explicit scheme ($\theta = 0$), since the right hand side of (3.2.2) is completely known at time t, the advance to the next time step is straightforward

$$[y(t+\Delta t)] = [1 + \theta\Delta t L(t)][y(t)] \qquad (3.2.3)$$

Unfortunately, this explicit scheme requires too restrictive time step size ($\Delta t < 10^{-8}$ sec) to avoid numerical instability of the solution. The explicit scheme, though simple, is not only inefficient (a great number of time steps must be used) but also eventually inaccurate (due to the large error accumulated after so many computing steps).

If an implicit scheme (θ = ½ or 1) is used, the matrix inversion is required to advance to the next time step

$$[y(t+\Delta t)] = \{[I] - \theta\Delta t[L(t+\Delta t)]\}^{-1}\{[I] + \theta\Delta t[L(t)]\}[y(t)] \qquad (3.2.4)$$

Although more computational effort per time step is spent on solving the implicit system (3.2.4), the implicit schemes are preferred for solving complex stiff parabolic equations since they are usually unconditionally stable for any time step size. In such an implicit scheme the time step size is restricted only by accuracy of the solution. It can be

mathematically proved that the explicit and fully implicit schemes are both first order accurate (n = 1), while the Crank-Nicholson scheme is second order (n = 2).

Generally, it is difficult to solve the non-linear system (3.2.4) for reactor kinetics because the group constants (i.e. the matrix [L(t+Δt)]) is indirectly dependent on the group fluxes (i.e. the unknown [y(t+Δt)]) that has yet to be solved, moreover, in coupling with other dynamics processes (e.g. thermalhydraulics) whose solution is also of somewhat the same computational effort. In practice, the coupled neutron kinetics and thermahydraulics problem can be solved during each of a sequence of small time intervals by a thermalhydraulics computation, for which the input is the local neutron fluxes at the beginning of the interval, followed by a neutron kinetics computation, for which the group constants are evaluated based on local temperatures and densities obtained from the just-completed thermalhydraulics calculation. In this case, the kinetics system (3.2.4) becomes linear, i.e. [L] is assumed constant over the interval [t, t+Δt].

Applying the implicit scheme (3.2.4) to the spatially-discretized kinetics system of finite difference equation (3.1.15) and equation (3.1.13b), one can first compute the precursor concentrations at point P

$$C_i^P = \frac{\left[1+(1-\theta)\lambda_i\Delta t\right]C_i^{p,t} + (1-\theta)\Delta t\beta_i\sum_{g'=1}^{G}\nu\Sigma_{fg'}^{p}\phi_{g'}^{p,t} + \theta\Delta t\beta_i\sum_{g'=1}^{G}\nu\Sigma_{fg'}^{p}\phi_{g'}^{p}}{1+\theta\lambda_i\Delta t} \qquad (3.2.5)$$

$$i = 1,..,N$$

where

$$C_i^P \equiv C_i^P(t+\Delta t) \quad \text{and} \quad \phi_g^P \equiv \phi_g^P(t+\Delta t)$$

$$C_i^{p,t} \equiv C_i^p(t) \quad \text{and} \quad \phi_g^{P,t} \equiv \phi_g^P(t)$$

Then, utilizing the expression (3.2.5), one obtains the group fluxes at point P, given in matrix notation, as

$$[A_P][\Phi_P] = \sum_{nb\subset P}[A_{nb}][\Phi_{nb}] + [B_p] \qquad (3.2.6)$$

where the vectors and matrixes are given by (cf. 3.1.15, 3.1.16)

$$[\Phi_p] = \begin{bmatrix} \phi_1^p \\ \phi_2^p \\ \dots \\ \phi_G^p \end{bmatrix}, \qquad [B_p] = \begin{bmatrix} b_1^P \\ b_2^P \\ \dots \\ b_G^P \end{bmatrix} \quad (\text{P is an inner grid point}),$$

$$[A_P] = \begin{bmatrix} a_1^P - a_{11}^P & -a_{21}^P & \dots & -a_{G1}^P \\ -a_{12}^P & a_2^P - a_{22}^P & \dots & -a_{G2}^P \\ \dots & \dots & \dots & \dots \\ -a_{1G}^P & -a_{2G}^P & \dots & a_G^P - a_{GG}^P \end{bmatrix}, \qquad [A_{nb}] = \theta \begin{bmatrix} a_1^{nb} & & & \\ & a_2^{nb} & & \\ & & \dots & \\ & & & a_G^{nb} \end{bmatrix}$$

with the algebraic coefficients

$$a_g^p = \frac{1}{v_g \Delta t} + \theta \left(\sum_{nb \subset P} a_g^{nb} + \Sigma_{tg}^P \right)$$

$$a_{g'g}^p = \theta \left[\Sigma_{sg'g}^P + \left((1-\beta)\chi_g + \sum_{i=1}^{N} \frac{\theta \Delta t \lambda_i \beta_i \chi_{i,g}^d}{1 + \theta \Delta t \lambda_i} \right) \nu \Sigma_{fg'}^P \right]$$

$$\begin{aligned} b_g^p = & \left[\frac{1}{v_g \Delta t} - (1-\theta) \left(\sum_{nb \subset P} a_g^{nb} + \Sigma_{tg}^P \right) \right] \phi_g^{P,t} + (1-\theta) \sum_{nb \subset P} a_g^{nb} \phi_g^{nb,t} \\ & + (1-\theta) \sum_{g'=1}^{G} \left[\Sigma_{sg'g}^P + \left((1-\beta)\chi_g^p + \sum_{i=1}^{N} \frac{\theta \Delta t \lambda_i \beta_i \chi_{i,g}^d}{1 + \theta \Delta t \lambda_i} \right) \nu \Sigma_{fg'}^P \right] \phi_{g'}^{P,t} \\ & + \sum_{i=1}^{N} \left[(1-\theta) + \frac{\theta - (1-\theta)\theta \Delta t \lambda_i}{1 + \theta \Delta t \lambda_i} \right] \lambda_i C_i^{P,t} \chi_{i,g}^d + (1-\theta) S_{0g}^{P,t} + \theta S_{0g}^P \end{aligned}$$

The initial and boundary conditions are

$$\phi_g^{P,t=0} = \phi_{0g}^P, \quad \forall g, P;$$

$$\phi_g^{P=B} = 0, \qquad \forall g, t > 0;$$

$$C_i^{p,t=0} = \frac{\beta_i}{\lambda_i} \sum_{g'=1}^{G} \nu \Sigma_{fg'}^p \phi_{0g'}^p, \qquad \forall g, i \tag{3.2.7}$$

3.2.2 Time Step Adjustment

If the implicit scheme is used for time integration of the kinetics equations, the time step size is restricted only to accuracy of the numerical solution. In reactor dynamic simulation, the constant time step size would be desirable but not necessary. In fact, one always tries to lengthen the time step size as long as the accuracy of the solution is acceptable in order to save computation time. Time step adjustment should not be based on the rate of change of the flux but rather on its temporal truncation error

$$Err = \max|\hat{\phi}_g^p(t) - \phi_g^p(t)| \tag{3.2.8}$$

where $\hat{\phi}_g^p(t)$ is the exact solution of (3.1.15) and $\phi_g^p(t)$ is the solution of the algebraic system (3.2.6), excluding the spatial discretization error [Crouzet & Turinsky (1996)]. The problem is that the exact solution is not known (and never known) so the truncation error can only be estimated.

The step-doubling adaptive time step technique by Taiwo et al. (1993) is a straightforward algorithm in which the solution at the end of the time step is computed twice; once with a step size Δt and once with two half steps $\frac{1}{2}\Delta t$. The temporal truncation error is estimated by

$$Err = \max|\phi_{\Delta t/2} - \phi_{\Delta t}| \tag{3.2.9}$$

If this error is small enough, for example, less than some specified tolerance ε, then the solution in the current time step is acceptable and the next time step size is estimated as

$$\Delta t_{next} \le \Delta t_{next}(\varepsilon/Err)^{1/n}, \qquad n \text{ - accuracy order} \tag{3.2.10}$$

Otherwise, the current step solution is rejected and the time step is repeated with the step size computed from (3.2.10). The step-doubling technique requires 300% as much computational effort per time step as a constant time step method.

If the accuracy order of the time integration scheme is known (or can be estimated by some way), then the more exact solution may be extracted from the above step-doubling procedure. Assume that the truncation error is proportional, with some constant C, to the time step size power of accuracy order as

$$\phi_{exact} - \phi_{\Delta t} \approx C\Delta t^n$$

$$\phi_{exact} - \phi_{\Delta t/2} \approx 2C(\Delta t/2)^n$$

Eliminating the constant C, one computes the 'exact' solution as

$$\phi_{exact} \approx \phi_{\Delta t} - (\phi_{\Delta t} - \phi_{\Delta t/2})2^{n-1}\Delta t^n \tag{3.2.11}$$

*
* *

No matter what method we use for reactor kinetics calculations, it is the problem space that will require most of our effort for the treatment. Although nodal methods have been preferred by many (thanks to their ability to reduce the number of discretized equations to be solved), applications of nodal methods give rise to such complexity and limitation that any other methods, *simpler in derivation but less (or just comparable) in computation cost*, would be worthwhile to investigate. The research direction that we are focusing on in this thesis is to employ a finite difference method for simplicity ***plus*** a fast algebraic solver for efficiency. But before proceeding to the development of such an efficient solver, we will first, in the following chapter, study usual numerical methods for solving algebraic systems that would arise from discretization of the neutron kinetics equations with any one of the spatial methods described above.

Chapter 4

SOLUTION OF ALGEBRAIC SYSTEMS

During the process of numerical solution of the space-time neutron group diffusion equations - or indeed of any partial differential equations in general - sooner or later one will encounter a *coupled set of algebraic equations* arising from discretization of the original equation system. Such an algebraic system must be solved simultaneously for either discrete values of the unknown functions (i.e. the neutron flux and delayed neutron precursor distributions) or their expansion coefficients, depending on whether a discrete ordinate or function expansion technique is used for the discretization.

In this chapter, we are concerned with the utilization of numerical methods for solving *linear* algebraic systems. Non-linear systems can always be linearized so that solving non-linear systems is reduced to a succession of solving linear systems. There are two general schemes for solving an algebraic system: *direct* and *iterative methods* [Young & Gregory (1973), Hageman & Young (1981), Axelsson (1994), Saad (1996)]. The direct methods are often preferred for low to medium sized systems (usually hundreds of equations or less). For large systems, iterative methods are almost always used. This switch is required from *accuracy* considerations (related to round-off errors), from memory limitations for *storage* of the algebraic coefficients and other computing values, and from overall *efficiency* concerns.

4.1 Direct Algebraic Solvers

Consider a system of N linear algebraic equations in N unknowns in a general form

$$\begin{bmatrix} a_{11} & a_{12} & \dots & a_{1N} \\ a_{21} & a_{22} & \dots & a_{2N} \\ \dots & \dots & \dots & \dots \\ a_{N1} & a_{N2} & \dots & a_{NN} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \dots \\ x_N \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ \dots \\ b_N \end{bmatrix} \qquad \text{or} \qquad Ax = b \tag{4.1.1}$$

where A is the N×N matrix of algebraic coefficients $\{a_{ij} \mid i = 1,..,N;\ j = 1,..,N\}$; x and b are two N-size vectors of unknowns $\{x_i \mid i = 1,..,N\}$ and constants $\{b_i \mid i = 1,..,N\}$, respectively. For such a system to have a unique solution, it is necessary and sufficient that the matrix A be *nonsingular*, i.e. it has nonzero determinant, $\det(A) \neq 0$, or, equivalently, there exists the matrix A^{-1} (the *inverse matrix* of A) so that the product $A^{-1}A$ is the *identity matrix* I (the identity matrix I has all coefficients equal zero except those on the main diagonal equal 1). Then, the solution of the system is formally given by

$$\bar{x} = A^{-1}b \tag{4.1.2}$$

Solution (4.1.2) is algebraically equivalent to the one that is provided by *Cramer's rule*, as a ratio of two N×N determinants

$$\overline{x}_i = \frac{\det(A_i)}{\det(A)}, \qquad i = 1,..,N \tag{4.1.3}$$

where A_i is the same as A except its i-th column being replaced by the vector b. However, computation of matrix determinants is almost impossible in practice because it generally involves more than 2(N+1)! arithmetic operations [Young & Gregory (1973)]. Such an excessive number of operations are too costly to be executed even on modern computers. In fact, with just only N = 4, any method based on determinant computation can no longer compete with the *elimination method* discussed in the following.

The simplest of all direct methods, from the computational point of view, for solving a general system of linear algebraic equations is the *Gaussian elimination method*, which systematically applies to row operations to transform the original system (4.1.1) into a form that is easier to solve. The Gaussian elimination scheme consists of two stages: a forward elimination and a backward substitution.

In the first stage of this scheme, the *forward elimination* replaces the original system by a series of successive *equivalent* systems (i.e. those with the same solution):

$$\begin{aligned} Ax = b \qquad \text{or} \qquad & [A \mid b] \equiv [A^{(1)} \mid b^{(1)}] \\ \therefore \quad & [A^{(2)} \mid b^{(2)}] \\ & \cdots \\ \therefore \quad & [A^{(N-1)} \mid b^{(N-1)}] \\ \therefore \quad & [A^{(N)} \mid b^{(N)}] \end{aligned}$$

where

$$[A^{(k)} \mid b^{(k)}] \equiv \left(\begin{array}{ccccccc|c} a_{11}^{(1)} & a_{12}^{(1)} & a_{13}^{(1)} & & & & a_{1N}^{(1)} & b_1^{(1)} \\ & a_{22}^{(2)} & a_{23}^{(2)} & & & & a_{2N}^{(2)} & b_2^{(2)} \\ & & \cdots & \cdots & & \cdots & \cdots & \cdots \\ & & a_{kk}^{(k)} & a_{k,k+1}^{(k)} & & & a_{kN}^{(k)} & b_k^{(k)} \\ & & \cdots & \cdots & & \cdots & \cdots & \cdots \\ & & a_{Nk}^{(k)} & a_{N,k+1}^{(k)} & & & a_{NN}^{(k)} & b_N^{(k)} \end{array} \right), \qquad k \geq 1$$

At the end of the forward stage, one obtains

$$\begin{bmatrix} a_{11}^{(1)} & a_{12}^{(1)} & \dots & a_{1N}^{(1)} \\ & a_{22}^{(2)} & \dots & a_{22}^{(2)} \\ \dots & \dots & \dots & \dots \\ & & & a_{NN}^{(N)} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \dots \\ x_N \end{bmatrix} = \begin{bmatrix} b_1^{(1)} \\ b_2^{(2)} \\ \dots \\ b_N^{(N)} \end{bmatrix} \tag{4.1.4}$$

The matrix and vector coefficients are computed as

$$a_{ij}^{(1)} = a_{ij} \quad \text{and} \quad b_i^{(1)} = b_i$$

$$a_{ij}^{(k+1)} = a_{ij}^{(k)} - \frac{a_{ik}^{(k)}}{a_{kk}^{(k)}} a_{kj}^{(k)} \quad \text{and} \quad b_i^{(k+1)} = b_i^{(k)} - \frac{a_{kj}^{(k)}}{a_{kk}^{(k)}} b_k^{(k)}, \quad \text{for} \quad k \geq 1$$

$$i = k,.., N; \quad j = i,.., N$$

In the next stage of the Gaussian elimination scheme, the *backward substitution* solves the triangular system (4.1.4) from the last equation to the first for x_i as

$$\bar{x}_N = \frac{b_N^{(N)}}{a_{NN}^{(N)}}$$

$$\bar{x}_i = \frac{1}{a_{ii}^{(i)}} \left(b_i^{(i)} - \sum_{j=i+1}^{N} a_{ij}^{(i)} x_j \right), \qquad i = N\text{-}1,..,1$$

The above described Gaussian elimination method belongs in a more general family of methods that are based on factorization of the matrix A into a product of a lower L and upper U triangular matrix

$$A = LU \tag{4.1.5}$$

First, a forward elimination sweep is performed to construct and invert L

$$Ux = L^{-1}b = y$$

Then, a backward substitution is performed to invert U and solve for x

$$x = U^{-1}L^{-1}b = U^{-1}y$$

It should be noted here that, the Gaussian elimination scheme would fail at some step in the forward elimination if the coefficient $a_{kk}^{(k)}$ (called the *pivot*) is zero. However, as it is always possible to find at least one row in the rest (say, j > k) that has the nonzero k-th coefficient (i.e. $a_{j,k}^{(k)} \neq 0$), one can simply replace these two rows by each other and proceed with the elimination procedure. Moreover, to minimize the round-off error, which may be large as a result from dividing by a small number, it is desirable that one find the pivotal row with the largest (in absolute value) coefficient for pivot.

In practice, except for tridiagonal matrix systems, the direct methods are not used for solving large algebraic systems (of a hundred or more equations) due to several reasons. First, the direct method requires a rather large number of arithmetic operations compared with iterative methods. Second, the coefficient matrix A, usually of sparse and banded structure, is not preserved during the process, and, hence large computer memory is required to store all algebraic coefficients. Next, it is difficult to program such a successive procedure in parallel. Finally, the most serious limitation of the direct methods is associated with the round-off error accumulation when a large number of arithmetic operations are executed consecutively.

However, most of the limitations of the direct algebraic solvers are not encountered when applied to algebraic systems that have tridiagonal matrix structure. The matrix of such a system, often arising from 3-point discretization of a 1D problem [Duderstad & Hamilton (1976)], has three diagonals as shown below

$$A = \begin{bmatrix} a_{11} & a_{12} & & & & \\ a_{21} & a_{22} & a_{23} & & & \\ & \dots & \dots & \dots & & \\ & & a_{N-1,N-2} & a_{N-1,N-1} & a_{N-1,N} & \\ & & & a_{N,N-1} & a_{N,N} & \end{bmatrix} \tag{4.1.6}$$

The *tridiagonal matrix algorithm* (TDMA), a special case of the Gaussian elimination scheme, also consists of a forward elimination and a backward substitution. The forward elimination sweep reduces the original tridiagonal matrix (4.1.6) to a form that has only two diagonals

$$\begin{bmatrix} \alpha_{11} & \alpha_{12} & & & & \\ & \alpha_{22} & \alpha_{23} & & & \\ & & \dots & \dots & & \\ & & & \alpha_{N-1,N-1} & \alpha_{N-1,N} & \\ & & & & \alpha_{N,N} & \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \dots \\ x_{N-1} \\ x_N \end{bmatrix} = \begin{bmatrix} \beta_1 \\ \beta_2 \\ \dots \\ \beta_{N-1} \\ \beta_N \end{bmatrix} \tag{4.1.7}$$

where

$$\alpha_{11} = a_{11}; \qquad \alpha_{12} = a_{12}; \qquad \beta_1 = b_1$$

$$\alpha_{ii} = a_{ii} - a_{i,i-1}\frac{\alpha_{i-1,i}}{\alpha_{i-1,i-1}}; \qquad \alpha_{i,i+1} = a_{i,i+1}; \qquad \beta_i = b_i - a_{i,i-1}\frac{\beta_{i-1}}{\alpha_{i-1,i-1}}, \qquad i \geq 2$$

The backward substitution solves for x as

$$\overline{x}_N = \frac{\beta_N}{\alpha_{N,N}}$$

$$\overline{x}_i = \frac{\beta_i - \alpha_{i,i+1}x_{i+1}}{\alpha_{i,i}}, \qquad i = N\text{-}1,..,1$$

In the TDMA, the number of arithmetic operations is only 8N and the storage is minimal (since there is no need for storing the bands of zero coefficients). It is also the easiest to program. Because it is very efficient, the TDMA is always preferred for solving 3-point difference equation systems. Moreover, the TDMA frequently appears as an integral part of the iterative methods used in 2D and 3D problems (i.e. 5-point and 7-point difference equation systems, respectively) when all equations in one row (or column) is inverted simultaneously.

4.2 Iterative Solvers for Algebraic Systems

The practical use of direct methods is only possible for solving algebraic systems of small size (less than 100 equations). For large and complex algebraic systems, iterative methods are almost always used. An iterative method is any one of a wide range of techniques that use successive approximations to obtain more accurate solutions to a linear system at each step [Saad & Vorst (2000)]. There are two types of iterative methods in use: *stationary* and *non-stationary*. Stationary or basic iterative methods (e.g. Jacobi, Gauss-Seidel or Successive Over-Relaxation (SOR) methods) perform in each iteration the same operations on the current iteration vectors [Hageman & Young (1981), Saad (1996)]. They are simple to understand and implement, but usually not as efficient as non-stationary iterative methods (e.g. Conjugate Gradient or Krylov subspace methods [Vorst (2000), Xu (2001)]) which are based on the idea of sequences of orthogonal vectors or polynomials.

4.2.1 Stationary Iterative Methods

The stationary iterative methods considered here for solving a linear algebraic system (4.1.1) can be expressed in the simple form [Hageman & Young (1981)]

$$x^{(m+1)} = Gx^{(m)} + k \tag{4.2.1}$$

where G is the real N×N iteration matrix for the method and k is an associated known vector. Neither G nor k depends on the iteration count m. The iterative scheme (4.2.1) is derived from (4.1.1) so that the solution to the system

$$(I-G)x = k \tag{4.2.2}$$

is also the unique solution to (4.1.1). In this case, an iterative method (4.2.1) is said to be *completely consistent*.

A *basic iterative method* (such as the Jacobi, Gauss-Seidel or SOR method) begins with some arbitrary initial guess $x^{(0)}$ and improves the current iteration vector $x^{(m)}$ until it *converges*. The method (4.2.1) is said to be *convergent* if for any $x^{(0)}$ the sequence $x^{(1)}$, $x^{(2)}$, … defined by (4.2.1) converges to the solution (4.1.2). To analyze the convergence property of an iterative method, one defines the error vector $e^{(m)}$

$$e^{(m)} = \bar{x} - x^{(m)} \tag{4.2.3}$$

It is clear that, if the iterative scheme (4.2.1) converges, then the error $e^{(m)}$ approaches zero as $m \to \infty$. Substituting (4.2.3) into (4.2.1) and utilizing (4.2.2), one has

$$e^{(m)} = Ge^{(m-1)} = \ldots = G^m e^{(0)} \tag{4.2.4}$$

It follows that

$$\|e^{(m)}\| \le \|G^m\| \cdot \|e^{(0)}\| \tag{4.2.5}$$

where $\|\cdot\|$ is some norm of a vector or matrix. Thus, the matrix norm $\|G^m\|$ gives a measure by which the norm of the error has been reduced after m iterations. Convergence of the iterative scheme (4.2.1) is guaranteed if and only if the largest eigenvalue of the iteration matrix is less that unity

$$\begin{aligned} \lim_{m\to\infty} \| G^m \|^{1/m} &= \rho(G) \equiv \textit{spectral radius} \text{ of } G \\ &= \max_{k=I,\ldots,N} | \lambda_k | < 1 \end{aligned} \tag{4.2.6}$$

The spectral radius ρ can be used as a measure of the rapidity of convergence of an iterative scheme. If $\rho << 1$, the iterative scheme will converge rapidly. If $\rho \approx 1$ but less than 1, the scheme will be slowly converging. The iterative scheme will diverge if $\rho > 1$.

The matrix eigenvalue/eigenvector concept and convergence of an iterative method are closely related as described in the following. For some algebraic matrix such as G, any vector $v = \{v_1,..,v_N\}$ that satisfies

$$Gv = \lambda v$$

is called the *eigenvector* of matrix G. The number λ corresponding to an eigenvector v is called an *eigenvalue*, which may be real or complex. In general, an N by N matrix has N eigenvectors $v_k = \{ v_1^{(k)},.., v_N^{(k)} \}$, k=1,..,N, and N corresponding eigenvalues λ_k, k=1,..,N. Since a set of all the eigenvectors of an algebraic matrix is complete, any arbitrary vector such as the initial error vector $e^{(0)}$ can be expressed by a linear combination of these eigenvectors as

$$e^{(0)} = \sum_{k=1}^{N} c_k v_k$$

where $\{c_k\}$ is a set of constants. If we repeatedly apply G on vector $e^{(0)}$, we get

$$e^{(1)} = Ge^{(0)} = \sum_{k=1}^{N} c_k Gv_k = \sum_{k=1}^{N} c_k \lambda_k v_k ,$$

…

$$e^{(m)} = G^m e^{(0)} = \sum_{k=1}^{N} c_k \lambda_k^m v_k$$

It follows that $e^{(m)} \to 0$ as $m \to \infty$ if and only if all eigenvalues in absolute value are less than unity, i.e. $|\lambda_k| < 1$. In addition, the smaller the eigenvalue $|\lambda_k|$ is, the faster the term $c_k \lambda_k^m v_k$ diminishes. Therefore, it is the largest eigenvalue $\max_{k=1,...,N} |\lambda_k|$ which determines the overall rate at which the iterative method converges. Different iterative schemes for an algebraic system (4.1.1) distinguish between each other by different ways to construct the iteration matrix G with different spectral radius $\rho(G) \equiv \max_{k=1,...,N} |\lambda_k|$.

Although the eigenvalue/eigenvector concept is quite important for understanding the performance of an iterative method, we hardly attempt to compute eigenvalues for a general matrix G, except when we have to do so, since this would be as costly as the

solution of the algebraic system itself. However, certain structures of algebraic matrices have been studied and identified as applicable to a convergent iterative scheme. A class of such matrices is branded as *symmetric positive definite* (SPD), which typically arises from finite difference discretization of partial differential equations. The eigenvalues of a SPD matrix are all positive and evenly distributed between 0 and 1.

In practice, the convergence is tested during the iteration process by using some criterion for judging solution accuracy. When this criterion is satisfied, the iteration process is terminated and the current iteration vector is accepted as the solution. The ideal criterion to terminate the iteration process would be when the error norm $||e^{(m)}||$ is within some desired accuracy ε, i.e.

$$||e^{(m)}|| \leq \varepsilon \tag{4.2.7}$$

However, it is hard to estimate the error $e^{(m)}$ directly since the exact solution $\bar{x}$ is not known. Instead, one may compute the relative change from one iteration to the next and compare its norm with some desired tolerance, such that

$$||x^{(m)} - x^{(m-1)}|| \leq \varepsilon \tag{4.2.8}$$

The problem associated with the use of this stopping criterion is that, if an iterative scheme is very slowly converging, one may encounter the so called *false convergence*, for which the condition (4.2.7) is actually not satisfied.

Another approach to bound the error $e^{(m)}$ is to define the *residual* vector $r^{(m)}$

$$r^{(m)} \equiv b - Ax^{(m)} = Ae^{(m)} \tag{4.2.9}$$

which implies

$$||e^{(m)}|| \leq ||A^{-1}|| \cdot ||r^{(m)}||$$

Therefore, the stopping criterion

$$||r^{(m)}|| \leq \delta \tag{4.2.10}$$

also yields the upper bound on the error as

$$||e^{(m)}|| \leq \delta ||A^{-1}|| = \varepsilon.$$

This latter approach is more preferred in practice.

The Jacobi Method

Perhaps the simplest iterative method is the Jacobi method. In order that the method can be formally applied, it is required that no diagonal element of A should vanish, i.e. $a_{ii} \neq 0$. The Jacobi method is defined by

$$x_i^{(m+1)} = \frac{b_i - \sum_{k \neq i}^{N} a_{ik} x_k^{(m)}}{a_{ii}}, \qquad i = 1,..,N \tag{4.2.11}$$

or in matrix notation of (4.2.1) with

$$G_J \equiv -D^{-1}(L+U) = D^{-1}A - I$$

$$k_J \equiv D^{-1}b$$

where L, U and D are respectively the lower triangular, upper triangular and diagonal matrices constituting A such that

$$A = L + D + U$$

It can be shown that if matrix A has weak diagonal dominance, i.e.

$$|a_{ii}| \geq \sum_{k \neq i}^{N} | a_{ik} |$$

then $\rho(G) < 1$ or the Jacobi method is convergent [Young & Gregory (1973)].

A rigorous analysis of the convergence of the Jacobi method shows that the spectral radius of the method is

$$\rho_J = 1 - O(1/N^2)$$

Therefore, if N is large, $\rho \approx 1$, so the Jacobi scheme converges very slowly.

In addition to slow convergence, the Jacobi iterative scheme requires the storage of two vectors $x^{(m+1)}$ and $x^{(m)}$ at each iteration. The only feature of the Jacobi scheme which is most favourable for parallel computation is that the order in which the values of the current iteration vector are evaluated is irrelevant, that is, they could be updated simultaneously.

The Gauss-Seidel Method

The Gauss-Seidel method is very similar to the Jacobi method except that it uses the updated values of the iteration vector as soon as they are available. Thus, for this iterative scheme, only one iteration vector needs to be stored. The Gauss-Seidel method is defined by

$$x_i^{(m+1)} = \frac{b_i - \sum_{k=i+1}^{N} a_{ik} x_k^{(m)} - \sum_{k=1}^{i-1} a_{ik} x_k^{(m+1)}}{a_{ii}}, \qquad i = 1,..,N \tag{4.2.12a}$$

or in matrix notation

$$x^{(m+1)} = -D^{-1}Lx^{(m+1)} - D^{-1}Ux^{(m)} + D^{-1}b \tag{4.2.12b}$$

Using the form (4.2.1), one finds the Gauss-Seidel iteration matrix as

$$G_S = -(L+D)^{-1}U = -(I + D^{-1}L)^{-1}(D^{-1}U)$$

$$k_S = (L+D)^{-1}D^{-1}b$$

The convergence of the Gauss-Seidel method is assured if the matrix A is strictly diagonally dominant, i.e.

$$|a_{ii}| > \sum_{k \neq i}^{N} | a_{ik} |$$

The spectral radius for the Gauss-Seidel method, as of the Jacobi method, is found to be

$$\rho_S = 1 - O(1/N^2)$$

but the Gauss-Seidel method converges twice faster than the Jacobi method. Unlike the Jacobi method, the order in which the values of the iteration vector are updated may sensitively influence on the convergence rate of the Gauss-Seidel scheme. Nevertheless, it is possible to re-order the equations in the system, resulting in several schemes (for example, the Red-Black Gauss-Seidel scheme) that are also suited for the parallel computation.

The Successive Over-Relaxation Method

The *successive over-relaxation* (SOR) method is only a slight modification of the Gauss-Seidel method but it may converge faster than the latter by an order of magnitude. The SOR method is defined by

$$x_i^{(m+1)} = \omega \frac{b_i - \sum_{k=i+1}^{N} a_{ik} x_k^{(m)} - \sum_{k=1}^{i-1} a_{ik} x_k^{(m+1)}}{a_{ii}} + (1-\omega)\, x_i^{(m)}, \qquad i = 1,..,N \tag{4.2.13}$$

where ω is a real number known as the *relaxation factor*. If $\omega = 1$, the SOR simplifies to the Gauss-Seidel method. SOR fails to converge if ω is outside the interval (0, 2). The choice of $\omega < 1$ would result in *under-relaxation*, and hence it is of little interest. To accelerate the convergence, the value of ω should be between 1 and 2. In matrix notation of the form (4.2.1), the SOR iteration matrix is found as

$$G_\omega = -(I+\omega D^{-1}L)^{-1}[\omega D^{-1}U - (1-\omega)I]$$

$$k_\omega = (I+\omega D^{-1}L)^{-1}\omega D^{-1}b$$

Theoretically, it is possible to choose an optimum value for ω so that the SOR converges the most rapidly. In principle, such an optimum value for ω may be given by [Hageman & Young (1981)]

$$\omega_b = \frac{2}{1+\sqrt{1-\rho_J^2}} \tag{4.2.14}$$

where ρ_J is the spectral radius of the Jacobi iteration matrix. This is, however, seldom done, since calculating the spectral radius of the Jacobi matrix requires an impractical amount of computation. Frequently, ω_b is estimated during the iteration process. The corresponding spectral radius for the SOR method with optimum relaxation can be found as

$$\rho_\omega = \omega_b - 1 = 1 - O(1/N)$$

The Alternating Direction Implicit Method

The *alternating direction implicit* (ADI) method is a very powerful method, especially for solving algebraic systems arising from discretization of differential equations on rectangular domain [Hageman & Young (1981)]. Generally, ADI methods are based on decomposing the coefficient matrix into a small number of matrices of simpler structure

and the problem of solving the original algebraic system is reduced to a succession of solving these resulting simpler algebraic systems. By using an ADI method, an extremely rapid convergence can be achieved for certain modelled problems. However, the ADI methods still suffer greatly from lack of generality. In fact, slow convergence and difficulty in producing effective iteration parameters become increasingly more detrimental as deviation from model conditions increases.

One of the first ADI methods is the *Peaceman-Rachford method* that splits the matrix A into

$$A = H + V$$

The method is defined by

$$(H + \tau_h I)x^{(m+½)} = b - (V - \tau_h I)x^{(m)} \tag{4.2.15}$$

$$(V + \tau_v I)x^{(m+1)} = b - (H - \tau_v I)x^{(m+½)}$$

Here it is assumed that for any positive numbers τ_h and τ_v (the *iteration parameters*) the first system can be solved easily for $x^{(m+½)}$, given $x^{(m)}$, and the second can be solved easily for $x^{(m+1)}$, given $x^{(m+½)}$. In a typical case involving a linear system arising from an elliptic differential equation on a rectangular domain, H and V are tridiagonal matrices: H is the matrix corresponding to horizontal differences and V is the matrix corresponding to vertical differences. It is known that, for tridiagonal matrix systems, the TDMA has been proved to be a very simple and fast solver.

Extension of the ADI method to a 3D problem is not straightforward as it might seem to be since the resulting equation system similar to (4.2.15) will involve much more than three equations and many iteration parameters to compute [Dai & Nasaar (1998)]. Moreover, in spite of many attempts to analyze the ADI method, a solid mathematical foundation still does not exist.

4.2.2 Non-Stationary Iterative Methods

Non-stationary methods differ from stationary methods in that the computations involve the iteration parameters that change at each iteration. Typically, non-stationary methods are faster than the basic iterative methods but add more complexity and computational efforts.

The Conjugate Gradient Method

The *Conjugate Gradient* (CG) method is an effective method for solving linear algebraic systems of form (4.1.1), whose coefficient matrix A is *symmetric* and *positive definite*, i.e.

$$A = A^T \qquad \text{and} \qquad \langle v, Av\rangle > 0 \qquad \forall v \neq 0$$

where $\langle x, y\rangle$ denotes an inner (scalar) product of two N-size vectors $\{x_k\}$ and $\{y_k\}$

$$\langle x, y\rangle \equiv \sum_{k=1}^{N} x_k y_k \tag{4.2.16}$$

The CG method can be regarded as a modification of the method of *steepest descent* (SD) [Hageman & Young (1981)]. To derive the SD method, consider the quadratic function

$$F(x) = \tfrac{1}{2}\langle x, Ax\rangle - \langle b, x\rangle \tag{4.2.17}$$

It is known from calculus that, if F(x) has a minimum at some point x, then all partial derivatives of F must be zero at that point. Calculating the partial derivatives of F with respect to each x_i, and setting each partial derivative to zero results in the linear system

$$b - Ax = 0 \qquad \text{or} \qquad Ax = b$$

That is, solving Ax = b is equivalent to minimizing F(x). From (4.2.17), the gradient of F is given by

$$\nabla F(x) = b - Ax \tag{4.2.18}$$

The direction of the vector $\nabla F(x)$ is the direction for which the function F(x) at the point x has the greatest instantaneous rate of change. If $x^{(m)}$ is some approximation to $\bar{x}$, then in the SD method one obtains an improved approximation $x^{(m+1)}$ by moving in the direction of $\nabla F(x^{(m)})$ to a point where $F(u^{(m+1)})$ is minimal, i.e.

$$x^{(m+1)} = x^{(m)} + \alpha_m \nabla F(x^{(m)}) = x^{(m)} + \alpha_m r^{(m)} \tag{4.2.19}$$

since $\nabla F(u^{(m)}) = b - Ax^{(m)} \equiv r^{(m)}$. The scalar constant α_m is an iteration parameter chosen to minimize $F(x^{(m+1)})$, i.e. $\nabla F(x^{(m+1)}) = 0$. Utilizing (4.2.18) and (4.2.19), and taking an inner product of the gradient vector $\nabla F(x^{(m+1)})$ with the residual vector $r^{(m)}$, one can compute α_m by

$$\alpha_m = \frac{\left\langle r^{(m)}, r^{(m)} \right\rangle}{\left\langle r^{(m)}, Ar^{(m)} \right\rangle} \tag{4.2.20}$$

The SD method is easy to program, but it often converges very slowly. The main reason for its slow convergence is that the method may spend time minimizing F along parallel search directions. However, by choosing the direction vectors differently, one obtains the CG method, which gives the solution in at most N iterations in the absence of rounding errors. Let $x^{(0)}$ be arbitrary and let successive approximations to the solution $\bar{x}$ be given by

$$x^{(m+1)} = x^{(m)} + \alpha_m p^{(m)} \tag{4.2.21}$$

where $p^{(m)}$ is a direction vector given by

$$p^{(0)} = r^{(0)}$$

$$p^{(m)} = r^{(m)} + \beta_m p^{(m-1)}, \quad \text{for } m \geq 1 \tag{4.2.22}$$

The scalar constant β_m is chosen so that $p^{(m)}$ is A-conjugate to $p^{(m-1)}$, i.e.

$$\langle p^{(m)}, Ap^{(m-1)} \rangle = 0$$

Evidently,

$$\beta_m = -\frac{\left\langle r^{(m)}, Ap^{(m-1)} \right\rangle}{\left\langle p^{(m-1)}, Ap^{(m-1)} \right\rangle} \tag{4.2.23}$$

As in SD method, choosing α_m to minimize $F(x^{(m+1)})$, one obtains

$$\alpha_m = \frac{\left\langle p^{(m)}, r^{(m)} \right\rangle}{\left\langle p^{(m)}, Ap^{(m)} \right\rangle} \tag{4.2.24}$$

It can be shown that the residual vectors $r^{(0)}$, $r^{(1)}$, … are mutually orthogonal, i.e.

$$\langle r^{(m)}, r^{(n)} \rangle = 0 \qquad \text{for} \qquad m \neq n$$

It follows that $r^{(k)} = 0$ for some $k \leq N$. Thus, the CG method converges, in the absence of rounding errors, in at most N iterations.

The vector sequences in the CG method correspond to a factorization of a tridiagonal matrix similar to the coefficient matrix. Therefore, a breakdown of the algorithm can occur corresponding to a zero pivot if the matrix is indefinite. Furthermore, for indefinite matrices the minimization property of the CG method is no longer well-defined. Various schemes based on CG method have been developed for solving more general algebraic systems where non-symmetric indefinite matrices may take place.

The Chebyshev Acceleration Method

The *Cherbyshev acceleration method* is based on a general procedure for accelerating the rates of convergence of basic iterative methods. This acceleration procedure involves the formation of a new vector sequence from linear combinations of the iterates obtained from the basic method. Suppose the completely consistent basic method (4.2.1) is used to obtain approximations for the solution of the nonsingular matrix problem (4.1.1). Let sequence of iterates generated by the basic method be given by $\{\tilde{x}^{(m)}\}$, i.e. given $\tilde{x}^{(0)}$, the sequence $\{\tilde{x}^{(m)}\}$ is formed by

$$\tilde{x}^{(m)} = G\tilde{x}^{(m-1)} + k \tag{4.2.25}$$

The corresponding error vector defined by $\tilde{e}^{(m)} \equiv \bar{x} - \tilde{x}^{(m)}$ satisfies

$$\tilde{e}^{(m)} = G^m \tilde{e}^{(0)} \tag{4.2.26}$$

As a means to enhance the convergence of $\tilde{x}^{(m)}$, one considers a new vector sequence $\{x^{(m)}\}$ determined by the linear combination

$$x^{(m)} = \sum_{\ell=0}^{m} \alpha_{m,\ell} \tilde{x}^{(\ell)}, \quad m = 1, 2, \ldots \tag{4.2.27}$$

In order to ensure that $x^{(m)} = \bar{x}$ whenever $\tilde{x}^{(0)} = \bar{x}$, the only restriction imposed on the real numbers $\alpha_{m,\ell}$ is that

$$\sum_{\ell=0}^{m} \alpha_{m,\ell} = 1 \tag{4.2.28}$$

The error vector $e^{(m)} \equiv \bar{x} - x^{(m)}$ can also be expressed in terms of $\tilde{e}^{(0)}$ as

$$e^{(m)} = -\sum_{\ell=0}^{m} \alpha_{m,\ell} \tilde{x}^{(\ell)} = \sum_{\ell=0}^{m} \alpha_{m,\ell} \tilde{e}^{(m)} = \left(\sum_{\ell=0}^{m} \alpha_{m,\ell} G^{\ell} \right) \tilde{e}^{(0)}$$

It follows that $e^{(0)} = \tilde{e}^{(0)}$ and

$$e^{(m)} = Q_m(G)e^{(0)} \tag{4.2.29}$$

where $Q_m(G)$ is the *matrix polynomial* given by

$$Q_m(G) \equiv \alpha_{m,0}I + \alpha_{m,1}G + \ldots + \alpha_{m,m}G^m \tag{4.2.30}$$

One now defines an *associated algebraic polynomial* as

$$Q_m(\xi) = \alpha_{m,0} + \alpha_{m,1}\xi + \ldots + \alpha_{m,m}\xi^m \tag{4.2.31}$$

From (4.2.29), it follows that

$$\|e^{(m)}\| \le \|Q_m(G)\|\cdot\|e^{(0)}\| = \rho(Q_m(G))\|e^{(0)}\| \tag{4.2.32}$$

The *Chebyshev acceleration procedure* reduces the error norm $\|e^{(m)}\|$ by picking the polynomials $\{Q_m(\xi)\}$ such that the spectral radius $\rho(Q_m(G))$ is small. More precisely, if $\{\lambda_k \mid k = 1,..,N\}$ is a set of eigenvalues for G, then $\{Q_m(\lambda_k)\}$ is the set of eigenvalues for $Q_m(G)$. Thus,

$$\rho(Q_m(G)) = \max_{k=1,\ldots,N} |Q_m(\lambda_k)| \tag{4.2.33}$$

Since the complete eigenvalue spectrum of G is seldom known, it is more convenient to consider the *virtual spectral radius* $\bar{\rho}(Q_m(G))$ instead of $\rho(Q_m(G))$ as

$$\rho(Q_m(G)) \le \bar{\rho}(Q_m(G)) = \max_{\lambda_{min}\le\xi\le\lambda_{max}} |Q_m(\xi)| \tag{4.2.34}$$

The polynomials $\{Q_m(\xi)\}$ are chosen such that $\bar{\rho}(Q_m(G))$ is minimized. The above described procedure is called a *polynomial acceleration procedure* applied to a basic iterative method. The high arithmetic cost and the large amount of storage required in using (4.2.27) to obtain $x^{(m)}$ make it necessary to seek alternative, less costly way to compute $x^{(m)}$. A simpler computation form for $x^{(m)}$ is possible whenever the polynomials $\{Q_m(\xi)\}$ satisfy the following recurrence relation

$$Q_0(\xi) = 1; \qquad Q_1(\xi) = \gamma_1\xi - \gamma_1 + 1 \tag{4.2.35}$$

$$Q_{m+1}(\xi) = \delta_{m+1}(\gamma_{m+1}\xi - \gamma_{m+1} + 1)Q_m(\xi) + (1 - \delta_{m+1})Q_{m-1}(\xi), \qquad \text{for } m \ge 1$$

where γ_m and δ_m are real numbers. In this case, the iterates $x^{(m)}$ of (4.2.27) may be obtained as

$$x^{(1)} = \gamma_1(Gx^{(0)} + k) + (1-\gamma_1)x^{(0)}$$

$$x^{(m+1)} = \delta_{m+1}[\gamma_{m+1}(Gx^{(m)} + k) + (1-\gamma_{m+1})x^{(m)}] + (1 - \delta_{m+1})x^{(m-1)} \tag{4.2.36}$$

The matrix polynomial $Q_m(G)$ which minimizes $\bar{\rho}(Q_m(G))$ is unique and can be defined in term of Chebyshev polynomials. The Chebyshev polynomials are given by

$$T_0(z) = 1, \qquad T_1(z) = z, \tag{4.2.37}$$

$$T_{m+1}(z) = 2zT_m(z) - T_m(z)$$

One then defines the polynomials $\{Q_m(\xi)\}$ as

$$Q_m(\xi) = \frac{T_m\left(\dfrac{2\xi - |\lambda_{max}| - |\lambda_{min}|}{|\lambda_{max}| - |\lambda_{min}|}\right)}{T_m\left(\dfrac{2 - |\lambda_{max}| - |\lambda_{min}|}{|\lambda_{max}| - |\lambda_{min}|}\right)} \tag{4.2.38}$$

It is obvious that $\{Q_m(\xi)\}$ satisfy three-term recurrence relation (4.2.35), with

$$\gamma_{m+1} = \bar{\gamma} = \frac{2}{2 - |\lambda_{max}| - |\lambda_{min}|},$$

$$\delta_{m+1} = 2z(1)\frac{T_m(z(1))}{T_{m+1}(z(1))},$$

$$z(\xi) = \frac{2\xi - |\lambda_{max}| - |\lambda_{min}|}{|\lambda_{max}| - |\lambda_{min}|}.$$

This iterative scheme is referred to as the *optimal Chebyshev acceleration procedure* for the basic iterative method (4.2.1). The convergence rate of this iterative scheme is estimated by the virtual spectral radius of $Q_m(G)$ as

$$\bar{\rho}(Q_m(G)) = [T(z(1))]^{-1} = 1 - O(1/N)$$

The Chebyshev acceleration procedure described above avoids the computation of inner products as required in CG method. For parallel computation, these inner products may be a bottleneck with respect to efficiency. The price one pays for avoiding inner products is that the method requires enough knowledge about the spectrum of the coefficient matrix. Since it is impractical to obtain the eigenvalue spectrum of the iteration matrix, the largest and smallest eigenvalues are only estimated, leading to non-optimal acceleration procedures.

*
* *

Iterative methods are almost always used for solving large or complex algebraic systems, such as the ones which typically arise from discretization of the reactor kinetics equations. Unfortunately, convergence of a basic iterative method sensitively deteriorates with an increasing number of equations in the solving algebraic system. The simpler the method is, the stronger its convergence depends on the size of the solving algebraic system. Even the fastest, and the most complicated, too, methods would require at least the number of iterations as same as the number of equations for solution.

Fortunately, there is a class of iterative methods whose convergence is independent of the size of a solving algebraic system and which we will discuss in the rest of this thesis.

Chapter 5

MULTIGRID METHODS

Discretization of the spatial neutron kinetics equations - or, indeed, of any multidimensional partial differential equations in general - results in a large, sparse set of coupled algebraic equations. It is not practical or even possible to use direct methods for inverting such a large system due to both excessive cost and inaccuracy in computation. In such cases, iterative methods can always be used but they are not so efficient if the size of algebraic systems is very large, from a few thousand to a few million. The reason is that the convergence rate of most basic iterative methods deteriorates with an increasing size of the algebraic system. Implementation of parallel computation on modern high-speed multiprocessor computer systems can, in principle, reduce the 'wide' problem (associated with memory storage and computation work as an algebraic system swept through) but it does nothing with the 'deep' problem (associated with the total number of such sweeps required for the solution to converge, i.e. the convergence rate).

It is obvious that, the overall efficiency of numerical solution of a partial differential equation is mostly affected by solution of its discretized system. Currently in reactor calculations, a traditional approach to increase efficiency of neutronics computation is to use a coarse-mesh discretization technique, such as a nodal method, to reduce the number of discretized equations that must be solved. Although being much smaller in size than the difference equation system (which results from fine-mesh discretization of the same problem with a finite difference technique), the nodal discretized system is still relatively large (typically, of tens of thousands of equations) and not easy to solve for, from the viewpoint of numerical analysis. Due to its complex and nonlinear nature, an algebraic system of nodal discretized equations usually requires expensive *preconditioning* (involving multiplication of matrices to transform an original system to its approximation that is easier to solve for) in order for its iterative solution to converge. Moreover, because nodal discretized systems appear to be incompatible with sophisticated iterative schemes, it is difficult to accelerate convergence of the iterative solution [Moulton (1996)].

Another quite different approach that we wish to introduce in this work is to increase efficiency of solution of finite difference discretized systems, though very large but rather simple, by accelerating convergence of the iterative solution. In this approach, *optimal iterative methods* (that have the convergence rate independent of the size of the algebraic system) are sought and *multigrid methods* - the fastest iterative methods known today - are among them. With an optimal multigrid, the 'deep' problem of solving large algebraic systems can be resolved.

Multigrid methods have been developed only recently but become quite a standard iterative method finding more and more applications in many fields of science

and engineering [Douglas (1997)]. The first multigrid algorithm was formulated by Fedorenko (1961) for solving the Poisson equation descretized with the standard 5-point finite difference technique on a square domain. This work was then generalized to the central difference discretization of the general linear elliptic partial differential equation on a square domain by Bachvalov (1966). However, the first practical results were reported by Brant (1977), clearly outlining principles and the practical utility of the multigrid, which drew wide attention and marked the beginning of rapid development.

In this chapter, the *essential principle* of the multigrid will be given and analyzed.

5.1 Error Smoothing

5.1.1 One-Dimensional Model Problem

The essential principle of the multigrid methods for PDE will be explained by studying a 1D model problem. Although 1D problems do not require application of iterative methods, since direct solution is efficient for solving tridiagonal matrix systems resulting from their discretization, but in one dimension the iterative methods, including the multigrid, can be analyzed by elementary methods, and their essential principle is easily demonstrated.

Consider the following model diffusion problem in one space dimension x

$$-\frac{d^2}{dx^2}\phi(x) = s(x), \quad x \in [0, 1], \qquad \phi(0) = \phi(1) = 0 \tag{5.1.1}$$

A computational grid Ω is defined by

$$\Omega = \{x_j = jh;\ j = 0,1,..,N;\ h = 1/N\}$$

Equation (5.1.1) is discretized with either vertex-centered or cell-centered finite differences [Wesseling (1992)], resulting in the following 3-point difference equation system

$$\begin{aligned} &2\phi_1 - \phi_2 = h^2 s_1 = b_1 \\ &-\phi_{j-1} + 2\phi_j - \phi_{j+1} = h^2 s_j = b_j, \qquad j = 2,..,N-2 \\ &\phi_{N-2} - 2\phi_{N-1} = h^2 s_{N-1} = b_{N-1} \end{aligned} \tag{5.1.2a}$$

or, in matrix notation,

$$A\phi = b \tag{5.1.2b}$$

where s(x) is the given source function that is represented by a discrete set of the values $s_j \equiv s(x_j)$; ϕ_j is intended to approximate the unknown function $\phi(x_j)$; A is a tridiagonal matrix and b is a column vector.

5.1.2 Iterative Solution

Of course, the tridiagonal system (5.1.2) can be solved efficiently by using a direct method such as the TDMA (see in Chapter 4). However, for large algebraic systems arising from discretization of multidimensional problems, direct solvers are no longer either efficient or accurate; therefore, iterative methods are used instead. When system (5.1.2) is solved by using a basic iterative method with an arbitrary initial guess $\phi^{(0)}$, the m-th iterate vector $\phi^{(m)}$ is given by

$$\phi_j^{(m)} = \frac{\omega}{2}[\phi_{j-1}^{(m-\kappa)} + \phi_{j+1}^{(m-1)} + b_j] + (1-\omega)\phi_j^{(m-1)} \tag{5.1.3}$$

where $\kappa = 1$ for the Jacobi scheme and $\kappa = 0$ for the Gauss-Seidel scheme. Both schemes can be used without relaxation ($\omega = 1$) or with relaxation $\omega \neq 1$. Note that the SOR scheme is a case of the Gauss-Seidel scheme with an over-relaxation, $1 < \omega < 2$.

5.1.3 Error Smoothing Analysis

If $\bar{\phi}$ is the exact solution of (5.1.2), then the error $e^{(m)} \equiv \bar{\phi} - \phi^{(m)}$ satisfies

$$e_j^{(m)} = \frac{\omega}{2}[e_{j-1}^{(m-\kappa)} + e_{j+1}^{(m-1)}] + (1-\omega)e_j^{(m-1)} \tag{5.1.4}$$

Such a grid error function can be represented by the following Fourier series [Wesseling (1992)]

$$e_j^{(m)} = \sum_{k=1}^{N-1} \alpha_k^{(m)} e^{ij\theta_k}, \qquad i = \sqrt{-1}, \qquad \theta_k = \frac{k\pi}{N}$$

Equation (5.1.4) can be rewritten in terms of the Fourier modes as

$$\sum_{k=1}^{N-1} \alpha_k^{(m)} e^{ij\theta_k} = \frac{\omega}{2}\left[\sum_{k=1}^{N-1} \alpha_k^{(m-\kappa)} e^{i(j-1)\theta_k} + \sum_{k=1}^{N-1} \alpha_k^{(m-1)} e^{i(j+1)\theta_k}\right] + (1-\omega)\sum_{k=1}^{N-1} \alpha_k^{(m-1)} e^{ij\theta_k}$$

From the orthogonality of $\{e^{ij\theta_k}\}$, i.e.

$$\sum_{j=1}^{N-1} e^{ij\theta_k} e^{-ij\theta_\ell} = \sum_{j=1}^{N-1} e^{ij(\theta_k-\theta_\ell)} = \begin{cases} N & \text{if} \quad k = \ell \\ 0 & \text{if} \quad k \neq \ell \end{cases}$$

it follows that, for any mode k,

$$\alpha_k^{(m)} = \frac{\omega}{2}[\,\alpha_k^{(m-\kappa)} e^{-i\theta_k} + \alpha_k^{(m-1)} e^{i\theta_k}\,] + (1-\omega)\,\alpha_k^{(m-1)}$$

The growth or decay of a Fourier mode of the error during iteration is measured by the *amplification factor*, defined by

$$g(\theta_k) = \left| \frac{\alpha_k^{(m)}}{\alpha_k^{(m-1)}} \right| \tag{5.1.5}$$

The amplification factor in the Jacobi scheme can be computed as

$$g_J(\theta_k) = |1 - \omega(1 - \cos\theta_k)| \tag{5.1.5a}$$

and, in the Gauss-Seidel scheme, as

$$g_S(\theta_k) = \sqrt{\frac{4(1-\omega)^2 + 4\omega(1-\omega)\cos\theta_k + \omega^2}{4 - 4\omega\cos\theta_k + \omega^2}} \tag{5.1.5b}$$

For any iterative scheme to converge, it is necessary and sufficient that

$$g(\theta_k) < 1 \qquad \text{for all } k = 1,..,N\text{-}1$$

which means that all Fourier modes of the error must decay with iteration. However, different Fourier modes decay at different rates, since $g(\theta_k)$ is varying with the *wave number* k. For example, the Gauss-Seidel scheme without relaxation ($\omega = 1$) has

$$g_S(\theta_k) = \frac{1}{\sqrt{5 - 4\cos\theta_k}} < 1 \qquad \text{for all } k$$

Hence, the Gauss-Seidel scheme is convergent. Moreover, since

$$g_S(\theta_k) \to 1 \qquad \text{if} \qquad k \to 0$$

and

$$g_S(\theta_k) \to 1/3 \quad \text{if} \qquad k \to N,$$

the long-wave modes (with small k) decay very slowly while the short-wave modes (with large k) decay more rapidly. It is the largest amplification factor, $\max\{g(\theta_k)\}$, which determines the overall convergence of the iterative scheme. We recall from Chapter 4 that the convergence rate of an iterative scheme is bounded by the spectral radius of the iterative matrix $\rho(G)$. Thus, the convergence rate of the Gauss-Seidel method for solving (5.1.2) is given by

$$\rho_S \equiv \max\{g(\theta_k)\} = g(\theta_1) = \frac{1}{\sqrt{5 - 4\cos\dfrac{\pi}{N}}} \approx 1 - \frac{10}{N^2}$$

corresponding to the rate at which the largest wavelength mode (k = 1) decays. It is obvious that convergence of the Gauss-Seidel scheme strongly depends on the size of the algebraic system, or the number of the mesh points used for discretization of the original PDE. The total number of Gauss-Seidel iterations required for the solution of (5.1.2) to converge to within a given accuracy is proportional to the square of the system size. Although the Gauss-Seidel is not an efficient iterative method for solving large systems because $g(\theta_1)$ is very close to 1, it is rather effective for damping down the short-wave modes with $N > k \geq N/2$, whose half-wavelengths are equal or less than doubled mesh spacing ($\leq 2h$).

Iterative methods which are capable of damping down the short-wave Fourier modes more rapidly than the long-wave modes are called *smoothers*. A good smoother should have the short-mode damping factor to be small and independent of the system size. For $N > k \geq N/2$, the damping factor of the Gauss-Seidel scheme is bounded by

$$g_S(\theta_{k=N/2}) = 1/\sqrt{5} = 0.4472$$

Hence, the Gauss-Seidel method is a good smoother. As an illustration of a smoother, we use the Gauss-Seidel method to solve the system (5.1.2). We set b = 0 so that to solve $A\phi=0$. The exact solution of the system is $\bar{\phi} = 0$. Let $\phi^{(0)}$ be an initial guess, consisting of the following three modes with different wave numbers k = 1, k = N/4 and k = N/2

$$\phi_j^{(0)} = \sin(jh) + \sin(¼Njh) + \sin(½Njh)$$

The error $e^{(m)}$ will be the same as $\phi^{(m)}$ but of the opposite sign, i.e. it also consists of those three modes. The total error and its Fourier modes decay during the iteration process as shown in Fig.5.1.1.

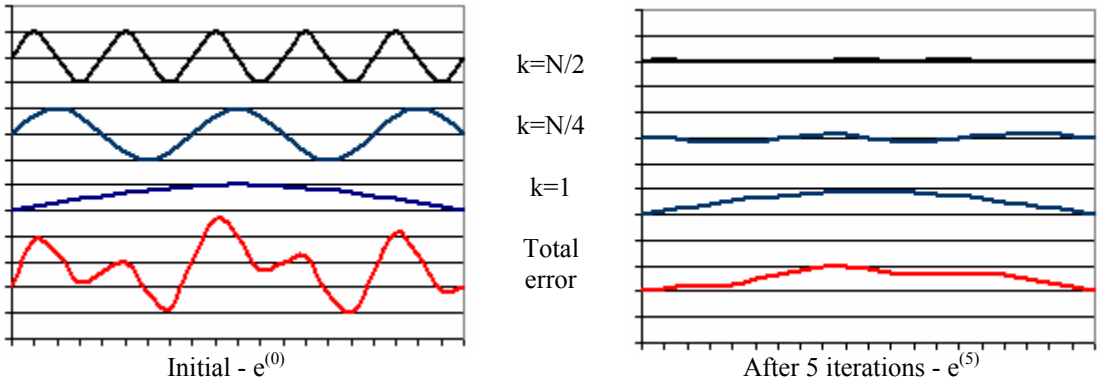

Figure 5.1.1. Error smoothing effect of the Gauss-Seidel method

From Fig.5.1.1 it is seen that, after only several Gauss-Seidel sweeps, the short mode with k = N/2 has been effectively eliminated, but the longer modes still remain there. In fact, for N = 20, it would require as much as 96 iterations for the longest mode (k = 1) to decrease by just an order of magnitude, i.e. $||e^{(96)}|/||e^{(0)}|| \approx 0.1$. If N doubles to 40, this number of iterations will quadruple to about 380.

The Jacobi method without relaxation (ω = 1), though convergent, is not a smoother, because $g_J(\theta_k) \to 1$ as $k \to N$. However, if the relaxation parameter ω is chosen less than 1, it may become a smoother, and it best reduces the short-wave modes at ω=2/3 (see in Fig.5.1.2). With ω > 1, the method is divergent because the amplification factors for some long-wave modes are more than 1.

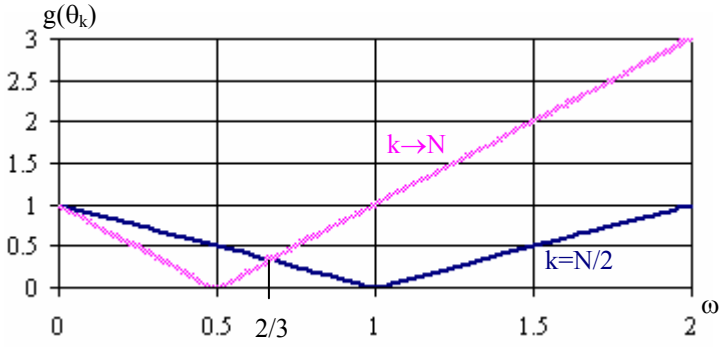

Figure 5.1.2. Short-wave mode amplification factor by the Jacobi method

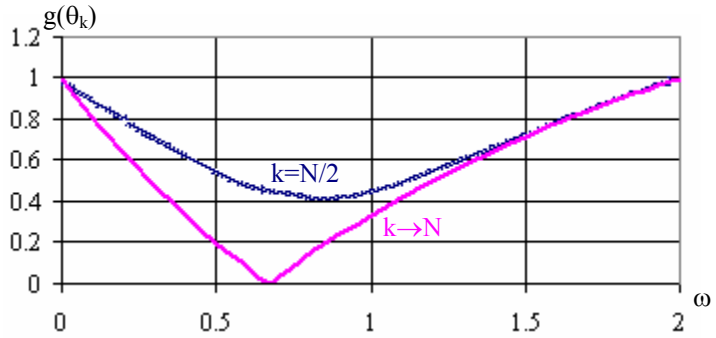

Figure 5.1.3. Gauss-Seidel amplification factor for short wave modes

It is interesting to note that, while the SOR - a Gauss-Seidel method with over-relaxation ($1 < \omega < 2$) - is generally much faster than the Gauss-Seidel method in solving an algebraic system, it is not as a good smoother as the Gauss-Seidel method and, therefore, usually not used as a smoother. However, the Gauss-Seidel method with little under-relaxation may become a slightly better smoother. That is, at $\omega = 0.8285$ the short-wave damping factor is minimal and equal 0.4142 (cf. 0.4472 at $\omega = 1$) (Fig.5.1.3).

5.2 The Two-Grid Algorithm

As seen in the preceding section, a basic iterative method such as the Gauss-Seidel is capable of smoothing the error by removing its non-smooth part (i.e. the short-wave Fourier modes) in a fixed, small number of iterations, but leaving the smooth part little changed on a given computational grid. We immediately realize that if the long-wave modes can be eliminated in some ways before iteration, then we can solve the algebraic system efficiently by using a simple smoothing method. In order to eliminate the long-wave Fourier modes of the error, we transfer them to a coarser grid and solve for them there. This procedure is referred to as the *two-grid algorithm*.

5.2.1 Coarse Grid Approximation

Consider the algebraic system (5.1.2) resulting from discretization of the model problem (5.1.1) on grid Ω^h (or, simply, Ω) with the mesh size h. Grid Ω is further referred to as the *fine grid*. The *coarse grid* Ω^H is formed by either doubling the mesh-size of the fine grid Ω, i.e. $H = 2h$, as in *vertex-centered coarsening*, or by agglomerating every two inner consecutive mesh intervals on Ω, as in *cell-centered coarsening* (Fig.5.2.1).

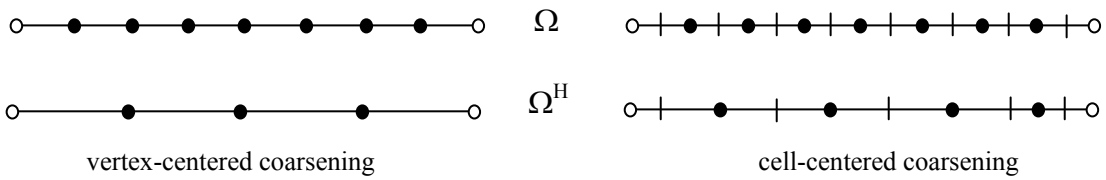

Figure 5.2.1. Grid coarsening in one dimension

Let $\tilde{\phi}$ be an approximation to the solution of (5.1.2) on the fine grid. It is obvious that solving the original system

$$A\phi = b$$

for ϕ with the initial guess $\tilde{\phi}$ by using an iterative method is equivalent to solving the *residual system*

$$Ae = r \equiv b - A\tilde{\phi}$$

for the error $e \equiv \bar{\phi} - \tilde{\phi}$ and then correcting the solution ϕ as

$$\bar{\phi} \leftarrow \tilde{\phi} + e$$

In this context, the solution to the error e is regarded as the *correction* which we wish to approximate on the coarse grid by solving the *coarse grid equation system*

$$A^H\phi^H = b^H \tag{5.2.1}$$

We define a *prolongation operator* $P: \Omega^H \rightarrow \Omega^h$ which takes the solution ϕ^H on the coarse grid as the correction on the fine grid, as

$$e = \mathbf{P}\phi^H \tag{5.2.2}$$

A *restriction operator* $R: \Omega^h \rightarrow \Omega^H$ is defined to send the residual on the fine grid to the coarse grid, as

$$b^H = \mathbf{R}r \tag{5.2.3}$$

The coarse grid system (5.2.1) must be an approximation to the fine grid system (5.1.2). Like the fine grid matrix A, the coarse grid matrix A^H may be obtained by discretizing the original PDE on the coarse grid Ω^H. This is called *discretization coarse grid*

approximation (DCA). An alternative way to obtain A^H is to apply the restriction to the residual system

$$RAe = Rr \quad \propto \quad RAP\phi^H = b^H$$

From (5.2.1), it follows that

$$A^H = RAP \tag{5.2.4}$$

This is called *Galerkin coarse grid approximation* (GCA) or RAP.

Now, with R, P and A^H provided, the two-grid algorithm for linear problems is defined as

The Two-Grid Algorithm

- *Guess* $\phi^{(0)}$
- *Smooth* m_1 *times on* Ω: $\phi^{(1/3)} = S(\phi^{(0)}, A, b, m_1)$
- *Compute the residual*: $r = b - A\phi^{(1/3)}$
- *Restrict the residual to* Ω^H: $b^H = Rr$
- *Solve the coarse grid equation*: $\phi^H = A_H^{-1} b^H$
- *Correct the solution on* Ω: $\phi^{(2/3)} = \phi^{(1/3)} + P\phi^H$
- *Smooth* m_2 *times on* Ω: $\phi^{(1)} = S(\phi^{(2/3)}, A, b, m_2)$

In the two-grid algorithm, $S(\phi^*,A,b,m)$ denotes m smoothing iterations with a basic iterative method (e.g. the Gauss-Seidel method) applied to $A\phi = b$, starting with ϕ^*. The first application of S is called *pre-smoothing* (with m_1 iterations), the second is *post-smoothing* (with m_2 iterations). The indexes $\frac{1}{3}$ and $\frac{2}{3}$ denote the values of an iterate vector just before and after the coarse grid correction, respectively.

5.2.2 Two-Grid Convergence Analysis

We now analyze the convergence of the two-grid algorithm for the model problem. Consider the case of vertex-centered coarsening with $N = 2N^H$, in which each coarse grid point j coincides with the fine grid point 2j (vertex-centered coarsening simply removes the odd points on the fine grid to form the coarse grid). The prolongation operator is chosen such that

$$\text{P:} \qquad e_{2j} = \phi_j^H$$

$$e_{2j+1} = \tfrac{1}{2}\phi_j^H + \tfrac{1}{2}\phi_{j+1}^H \qquad j = 0,1,..,N^H$$

This is called *linear interpolation*. The restriction operator is

$$\text{R:} \qquad b_j^H = r_{2j-1} + 2r_{2j} + r_{2j+1}$$

In this particular model problem, either DCA or GCA will give the coarse grid equations of the same form as the fine grid equations, as

$$2\phi_1^H - \phi_2^H = b_1^H$$

$$-\phi_{j-1}^H + 2\phi_j^H - \phi_{j+1}^H = b_j^H, \qquad j = 2,..,N^H\text{-}2$$

$$-\phi_{N^H-2}^H + 2\phi_{N^H-1}^H = b_{N^H-1}^H$$

Suppose that we do not apply the pre-smoothing, i.e. $m_1 = 0$, so $\phi^{(1/3)} \equiv \phi^{(0)}$ and $e^{(1/3)} \equiv e^{(0)}$. After coarse grid correction, the error is

$$e^{(2/3)} \equiv \bar{\phi} - \phi^{(2/3)} = \bar{\phi} - [\phi^{(0)} + P\phi^H] = e^{(0)} - P A_H^{-1} RAe^{(0)} = Ee^{(0)} \tag{5.2.5}$$

where $E \equiv [I - P A_H^{-1} RA]$ is called an *error amplification matrix*.

For the given model problem, $e^{(2/3)}$ can be explicitly expressed in terms of $e^{(0)}$. Because the solution ϕ^H on the coarse grid is exact and it corresponds to the exact correction of the even points on the fine grid, it follows that

$$e_{2j}^{(2/3)} = e_{2j}^{(0)} + \bar{\phi}_j^H = 0$$

$$\therefore \quad e_{2j}^{(0)} = -\bar{\phi}_j^H$$

$$e_{2j+1}^{(2/3)} = e_{2j+1}^{(0)} + \tfrac{1}{2}\phi_j^H + \tfrac{1}{2}\phi_{j+1}^H = -\tfrac{1}{2}e_{2j}^{(0)} + e_{2j+1}^{(0)} - \tfrac{1}{2}e_{2j+2}^{(0)}$$

Since coarse grid correction is to reduce the error, it follows that

$$\|e^{(2/3)}\| \le \|e^{(0)}\| \tag{5.2.6}$$

The equality may hold if restriction of the residual is identically zero, i.e. $Rr = b^H = 0$. It is obvious that, when $Rr = 0$, coarse grid correction is not necessary.

Consider the effect of post-smoothing by one Gauss-Seidel iteration (m_2=1). From (5.1.4), we have

$$e_1^{(1)} = \tfrac{1}{2} e_2^{(2/3)} = 0$$

$$e_{2j}^{(1)} = \tfrac{1}{2} e_{2j-1}^{(1)} + \tfrac{1}{2} e_{2j+1}^{(2/3)} = \frac{e_{2j+1}^{(2/3)}}{2} + \frac{e_{2j-1}^{(2/3)}}{8} + \frac{e_{2j-3}^{(2/3)}}{32} + \ldots$$

$$e_{2j+1}^{(1)} = \tfrac{1}{2} e_{2j}^{(1)}$$

By induction, we find that

$$\|e^{(1)}\|_\infty < \tfrac{2}{3} \|e^{(2/3)}\|_\infty \le \tfrac{2}{3} \|e^{(0)}\|_\infty \qquad (\text{where, } \|\cdot\|_\infty = \max|\cdot|) \tag{5.2.7}$$

We see that the Gauss-Seidel method reduces the maximum norm of the error by a factor bounded by $\tfrac{2}{3}$. It shows that the rate of convergence of the two-grid method is *independent of the mesh size* h. It is also noted that, equation (5.2.3) when applied to the model problem will be

$$RAe^{(2/3)} = 0 = r_{2j-1}^{(2/3)} + 2\, r_{2j}^{(2/3)} + r_{2j+1}^{(2/3)}$$

The zero of the sum of local weighted residuals with positive weighting factors implies that the residual $r^{(2/3)}$ has many sign changes, and therefore, is *non-smooth* (or *rough*), so is the error after coarse grid correction. The smoother is efficient in removing this rough part further, which explains the h-independent rate of convergence of the Gauss-Seidel method.

For a numerical illustration of two-grid application, we solve (5.1.2) by using the two-grid algorithm with vertex-centered coarsening. Again, let b = 0 and the initial guess $\phi^{(0)}$ consist of three modes with k = N/2, N/4 and 1, as in section (5.1.3). Without pre-smoothing, the short-wave mode (k = N/2) of the error remains unchanged after the coarse grid correction. The smooth part (k > N/2) is effectively removed by exact solving the coarse grid system (Fig.5.2.2).

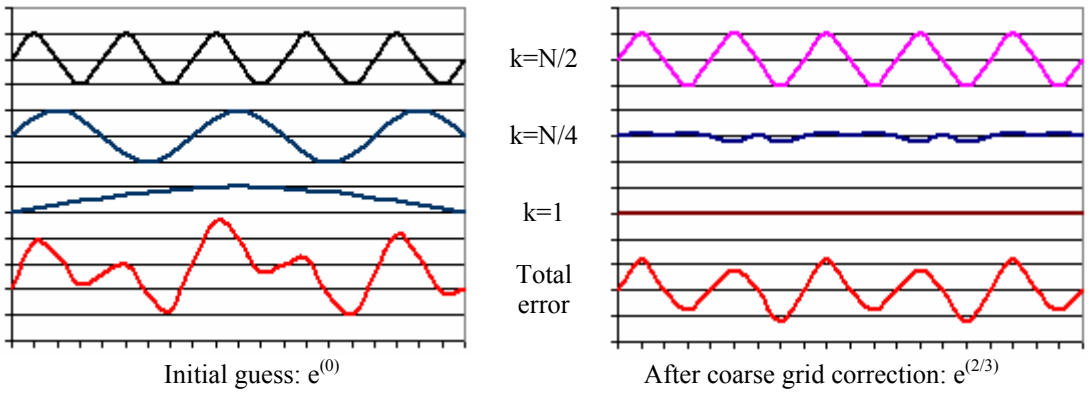

Figure 5.2.2. Coarse grid correction

However, the prolongation has introduced some sort of error, fortunately, of short waves. As a result, the error after coarse grid correction contains mostly short Fourier modes that can be efficiently damped out by applying the smoother. That is, applying only three Gauss-Seidel iterations will reduce the error norm to below 0.1, and eight iterations will reduce it to below 0.01. Moreover, these iteration numbers are fixed, i.e. independent of N. To compare with the Gauss-Seidel iteration on a single fine grid, for N = 20, it would take, respectively, 71 and 164 iterations to reduce the error norm to the similar levels. If N now doubles, these numbers will quadruple.

It is seen that, the Fourier modes with the half wavelength less than or equal to H cannot be represented on the coarse grid and hence coarse grid correction has no effect on them. That is why the coarse grid should have the mesh size $H \leq 2h$, otherwise all the modes with half wavelengths between 2h and H, though regarded as smooth on the fine grid, will be left unchanged after coarse grid correction.

5.3 Multigrid Methods

5.3.1 The Essential Multigrid Principle

As seen in the two-grid algorithm, the coarse grid has fewer mesh points than that on the fine grid and, hence, there are fewer algebraic equations to be solved on the coarse grid. Thus, two-grid application may somewhat reduce computation work for solving the original algebraic system. The extra work required for forming of the coarse grid system and transferring operations between the grids, is easily offset by solving much fewer coarse grid equations, roughly $1/2^d$ (d is the number of dimensions) the number of equations on the fine grid. However, exact solution of a multidimensional coarse grid system still suffers inefficiency if the coarse grid still has many mesh points. This can be easily remedied by *recursive* applying the same two-grid algorithm to the coarse grid equation system until the *coarsest grid* is reached. The coarsest grid should have only a

very few mesh points (it is possible to have only one non-trivial mesh point on that the coarsest grid) and, hence, the algebraic system on it is small enough to be solved exactly at a negligible cost. As a result, an efficient algorithm - the *multigrid* - for solving large algebraic systems is formulated.

At first glance, multigrid application reduces computational work thanks to solving fewer and fewer equations on coarser and coarser grids. But this is not the only reason for the multigrid to be used as a PDE solver. The principal motivation behind the multigrid is that long-wave modes of the error on one grid can be viewed as short waves on a coarser related grid and, therefore, can be damped down effectively there by a cheap, simple smoother. That is, the *essential principle of multgrid* is to approximate the smooth (long wave) part of the error on coarser grids.

As an iterative method, the multigrid can be regarded as a technique to accelerate the convergence of basic iterative methods which are the smoothers in the multigrid context. But while the convergence of usual iterative methods strongly deteriorates with increasing number of algebraic equations in the discretized system, the convergence rate of the multigrid method is independent of it.

5.3.2 The Multigrid Algorithm

Let $\{\Omega_\ell \mid \ell=1,..,L\}$ be a set of coarse grids formed from the original fine grid $\Omega_0 \equiv \Omega$, on which the original PDE is discretized to result in an algebraic system

$$A\phi = b \qquad \text{on} \qquad \Omega \tag{5.3.1}$$

The coarse grids will be labelled with an increasing index ℓ, i.e. $\Omega_{\ell+1}$ is a coarser grid next to Ω_ℓ. The coarsest grid Ω_L must have at least one non-trivial mesh point (which does not belong to the boundary). Let an algebraic equation system on grid Ω_ℓ be

$$A_\ell\phi_\ell = b_\ell \qquad \text{on} \qquad \Omega_\ell \tag{5.3.2}$$

Given restriction and prolongation operators R and P, and a smoothing method $S(\phi,A,b,m)$ as in the two-grid algorithm, the multigrid algorithm for linear algebraic systems is defined as

The Multigrid Algorithm

- *Guess* $\phi^{(0)}$
- *Iterate until* $\phi^{(m)}$ *converges*: $\phi^{(m+1)} = \mathbf{MG}(\ell=0, \phi^{(m)}, m_1, m_2, m_c)$
- *Accept* $\phi^{(m+1)}$ *as the solution to* $A\phi = b$

Multigrid Solver

$\mathbf{MG}(\ell, \phi^*_\ell, m_1, m_2, m_c)$
{
if ($\ell = L$){
Solve $\phi_L = A_L^{-1} b_L$
}
else{
Smooth m_1 *times* $\phi_\ell^{(1/3)} = \mathbf{S}(\phi^*_\ell, A_\ell, b_\ell, m_1)$
Restrict the residual $b_{\ell+1} = \mathbf{R}(b_\ell - A_\ell \phi_\ell^{(1/3)})$
Call m_c *times* $\phi_{\ell+1} = \mathbf{MG}(\ell+1, \phi^*_{\ell+1}=0, m_1, m_2, m_c)$
Correct the solution $\phi_\ell^{(2/3)} = \phi_\ell^{(1/3)} + \mathbf{P}\phi_{\ell+1}$
Smooth m_2 *times* $\phi_\ell^{(1)} = \mathbf{S}(\phi_\ell^{(2/3)}, A_\ell, b_\ell, m_2)$
}
return $\phi_\ell^{(1)}$
}

The only parameter that is new in the above multigrid algorithm is the *cycling strategy number* m_c which may be $m_c = 1$ or $m_c = 2$, corresponding to two frequently used multigrid schemes: V-cycle ($m_c = 1$) and W-cycle ($m_c = 2$) (Fig.5.3.1). A scheme with $m_c \geq 3$ is so cumbersome that it is practically never used.

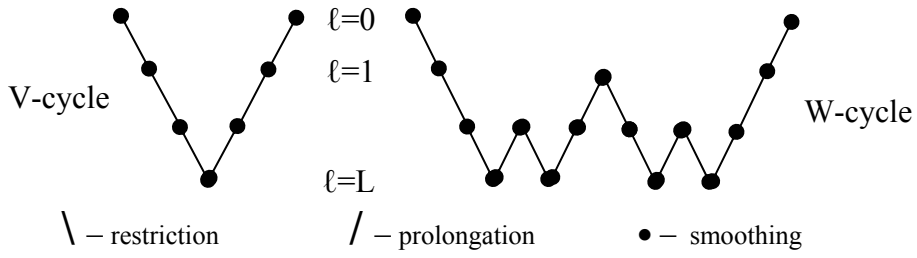

Figure 5.3.1. Multigrid V-cycle and W-cycle

It is also possible to slightly modify the above *'fixed'* MG cycles (i.e. they have the fixed numbers of pre- and post-smoothing steps, m_1 and m_2, on every grid) by setting a certain condition on smoother's performance to obtain the so-called *'flexible'* MG cycles [Lightstone (2002)]. That is, instead of doing a fixed number of smoothing steps, one keeps sweeping on any grid level until either the smoother stalls (e.g. when the iteration contraction number of the residual norm increases considerably, say, by 0.1), or the solution converges, which ever comes first. If the smoother is stalls, one goes down to a next coarser grid. If the solution converges, one goes up to a finer grid. The price to pay is one always has to compute the residual when entering any grid level plus as many times as the number of smoothing steps on that grid (in a fixed cycle one only computes the residual once when exiting any grid level).

5.3.3 Multigrid Components

Grids

The *fine grid* Ω in which the PDE is to be solved is assumed to be the d-dimensional unit cube. This greatly simplifies the construction of coarse grids and transfer operators between grids. In practice, multigrid for finite discretization can in principle be applied to more general geometry, but the description of the method would become complicated [Wesseling (1992)]. This is not a serious limitation because the current main trend in grid generation consists of decomposition of the physical domain in subdomains, each of which is mapped onto a cubic computational domain.

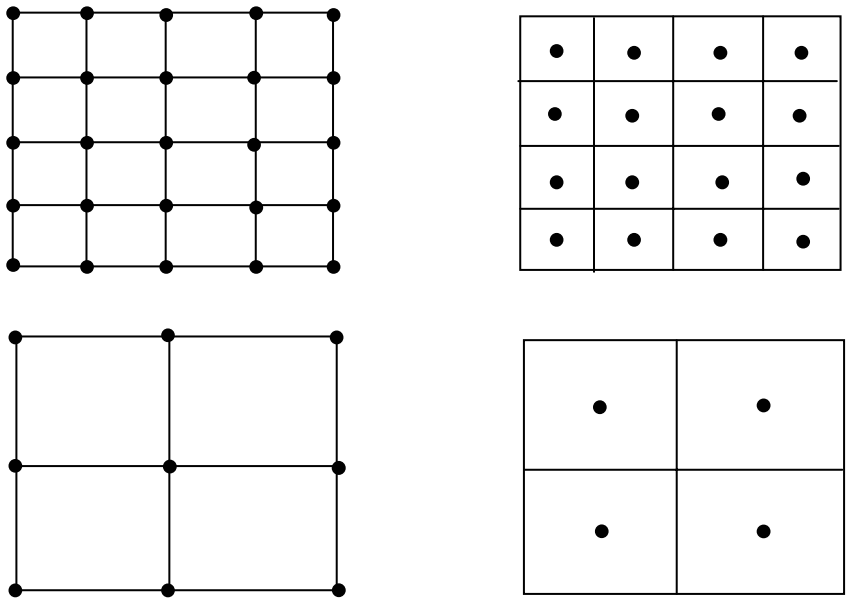

Figure 5.3.2. Vertex-centered and cell-centered coarsening in 2D

In the case of *vertex-centered discretization*, the fine grid Ω is the union of a set of regular cells, whose vertices are the grid points $u \in \Omega$ defined by

$$\Omega = \{u = jh;\ j = (j_1,..,j_d);\ h = (h_1,..,h_d);\ j_\alpha = 0,1,..,N_\alpha;\ h_\alpha = 1/N_\alpha;\ \alpha = 1,..,d\} \quad (5.3.3a)$$

In the case of *cell-centered discretization*, the fine grid Ω is also divided in cells as in the vertex-centered case, but now the grid points are the centres of the cells. The computational cell-centered grid is defined by

$$\Omega = \{u = (j-\tfrac{1}{2})h;\ j = (j_1,..,j_d);\ h = (h_1,..,h_d);\ j_\alpha = 1,..,N_\alpha;\ h_\alpha = 1/N_\alpha;\ \alpha = 1,..,d\} \quad (5.3.3b)$$

A *coarse grid* $\Omega_{\ell+1}$ is derived from Ω_ℓ, where $\Omega_0 \equiv \Omega$ and $\ell=1,..,L-1$, by either vertex-centered or cell-centered *coarsening*, depending on whether Ω is the vertex-centered or cell-centered grid. It is also possible to apply cell-centered coarsening to vertex-centered grids, and vice versa, but this will not be considered here because new methods or insights are not obtained. *Vertex-centered coarsening* consists of deleting every other vertex in each direction. Cell-centered coarsening consists of taking union of fine grid cells to obtain coarse grid cells (Fig.5.3.2) [Khalil & Wesseling (1991)].

It is assumed that N_α is even in the vertex-centered grid. This is to facilitate vertex-centered coarsening. Cell-centered coarsening can be with any N_α, even or odd. Since the coarsest grid must have at least one non-trivial grid point, the maximum level L_{max} that the coarsest grid can be, i.e. $L \le L_{max}$, is

$$L_{max} = [\log_2 N_\alpha] - 1 \qquad \text{for vertex-centered coarsening}$$

and

$$L_{max} = [\log_2 (N_\alpha - 1)] + 1 \qquad \text{for cell-centered coarsening}$$

Restriction and Prolongation

The transfer operators are denoted by P for *prolongation* and R for *restriction*

$$P: \quad \Omega_{\ell+1} \rightarrow \Omega_\ell$$

$$R: \quad \Omega_\ell \rightarrow \Omega_{\ell+1}$$

Normally, prolongation P is based on linear interpolation and restriction R may be simply taken as [Wessenling (1992)]

$$R = cP^T \quad (5.3.4)$$

where P^T is the transpose of P and c is a suitable scaling factor. But this property is not essential. Although R and P can be chosen rather arbitrarily, they should satisfy certain conditions in order for the multigrid method to have a mesh-size independent rate of convergence. Hackbusch (1985) gave the following simple condition and Hemker (1990) proved its necessity

$$n_P + n_R > n_E \tag{5.3.5}$$

Here *orders* n_P, n_R of operators P and R are defined as the highest degree plus one of polynomials that are interpolated exactly by P or R^T, respectively, and n_E is the order of the partial differential equation (PDE) to be solved. Since higher order prolongations and restrictions require more work on transferring operations, the transfer operators R and P are chosen as simple as possible but must satisfy condition (5.3.5). The linear prolongation and restriction given in section (5.2) for the second order diffusion problem ($n_E = 2$) have $m_P = m_R = 2$, so (5.3.5) is satisfied.

It is convenient to write the transfer operators R and P in *stencil* notation. For example, the stencils of *vertex-centered* prolongation and restriction based on *linear interpolation* in 1D and 2D are given as

In 1D: $$P = [\tfrac{1}{2} \quad 1 \quad \tfrac{1}{2}] \qquad R = 2P^T = \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix} \tag{5.3.6a}$$

In 2D: $$P = \begin{bmatrix} \tfrac{1}{4} & \tfrac{1}{2} & \tfrac{1}{4} \\ \tfrac{1}{2} & 1 & \tfrac{1}{2} \\ \tfrac{1}{4} & \tfrac{1}{2} & \tfrac{1}{4} \end{bmatrix} \qquad R = 2^2P^T = \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix} \tag{5.3.6b}$$

The use of a stencil consists of moving it across the first grid to generate elements on the second grid so as the stencil centre is mapped with each point of the coarser grid. The prolongation stencil P will operate on each coarse grid point as follows: A fraction of the value held at the coarse grid point is transferred to the fine grid points that enclose the coarse grid point. The effect is cumulative so at any fine grid point there is a sum of fractional parts from nearby coarse grid point values. The restriction stencil R sums up the weighted values held at the fine grid points that surround the coarse grid point and assign the sum to the coarse grid value. The weighting factors are elements of the stencil.

The simplest *cell-centered* prolongation and restriction are based on *piecewise constant interpolation*:

$$\text{In 1D:}\quad e_\ell = \mathbf{P}\phi_{\ell+1} \qquad e_{2i}^{(\ell)} = e_{2i+1}^{(\ell)} = \phi_i^{(\ell+1)} \qquad \text{or,} \qquad P = [1 \;\; 1]$$

$$b_{\ell+1} = \mathbf{R}r_\ell \qquad b_i^{(\ell+1)} = r_{2i}^{(\ell)} + r_{2i+1}^{(\ell)} \qquad \text{or,} \qquad R = P^T = \begin{bmatrix} 1 \\ 1 \end{bmatrix} \tag{5.3.7a}$$

$$\text{In 2D:} \qquad R = P = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} \tag{5.3.7b}$$

This gives $m_P = m_R = 1$. The higher order cell-centered prolongation and restriction are based on linear interpolation and have the following stencils

$$\text{In 1D:} \qquad P = [\,\tfrac{1}{4} \quad \tfrac{3}{4} \quad \tfrac{3}{4} \quad \tfrac{1}{4}\,] \qquad\qquad R = P^T \tag{5.3.8a}$$

$$\text{In 2D:} \qquad R = P = \begin{bmatrix} \tfrac{1}{16} & \tfrac{3}{16} & \tfrac{3}{16} & \tfrac{1}{16} \\ \tfrac{3}{16} & \tfrac{9}{16} & \tfrac{9}{16} & \tfrac{3}{16} \\ \tfrac{3}{16} & \tfrac{9}{16} & \tfrac{9}{16} & \tfrac{3}{16} \\ \tfrac{1}{16} & \tfrac{3}{16} & \tfrac{3}{16} & \tfrac{1}{16} \end{bmatrix} \tag{5.3.8b}$$

It is noted that, in case the coefficients in PDE are discontinuous across interfaces between subdomains (or cells) of different physical properties, linear interpolation across discontinuities is inaccurate and operator-dependent prolongation has to be used instead [Alcouffe et al. (1981), Behie & Forsyth (1983)]. It is required only in vertex-centered multigrid, but not in cell-centered multigrid.

Coarse Grid Equations

There are basically two ways to form the coarse grid matrix A_ℓ, as already mentioned in section (5.2):

(i) *discretization coarse grid approximation* (DCA), in which A_ℓ is obtained by discretization of the original PDE on grid Ω_ℓ; and

(ii) *Galerkin coarse grid approximation* (GCA) or RAP, in which A_ℓ is derived from the finer grid matrix $A_{\ell-1}$ by

$$A_\ell = RA_{\ell-1}P \tag{5.3.9}$$

Although DCA seems more straightforward, it may be unreliable if the coefficients are variable on very coarse grids, because these coefficients are sampled in very few points. Multigrid may fail because of this effect. For this reason, GCA is to be

used for interface problems (discontinuous coefficients) [Dendy (1987), Ersland & Teigland (1993), Engquist & Luo (1997), Schaffer S. (1998), Chan & Wan (2000)]. Another advantage of GCA is that it is purely *algebraic* in nature; that is, we can solve a general set of algebraic equations without knowing the original PDE or its geometry. However, when geometric multigrid is possible, it is simpler and faster than algebraic multigrid applied to the same problem. Further more, since the GCA involves matrix multiplication, it may become quite expensive if the fine grid matrix has a complicated structure. In addition, matrix structure of the fine grid problem may not preserve on coarse grids as a result of GCA.

The problems associated with using GCA can be remedied by applying the so called *additive correction multigrid* (ACM) [Hutchison & Raithby (1986), Hutchison et al. (1988)]. In fact, the ACM is equivalent to GCA, with piecewise constant prolongation and restriction (5.3.7), applied to a difference equation system in cell-centered discretization [Gjesdal (1996), Elias et al. (1997), Teigland (1998)]. We will demonstrate the ACM procedure for generating coarse grid equations in 1D.

In one dimension, an algebraic system of three-point difference equations on a cell-centered grid Ω can generally given by

$$a_{pi}\,\phi_i = a_{wi}\,\phi_{i-1} + a_{ei}\,\phi_{i+1} + b_i, \qquad i = 1,..,N\text{-}1 \tag{5.3.10}$$

$$\phi_0 = \phi_N = 0$$

The corresponding residual equation system is

$$a_{pi}\,e_i = a_{wi}\,e_{i-1} + a_{ei}\,e_{i+1} + r_i, \qquad i = 1,..,N\text{-}1 \tag{5.3.11}$$

$$e_0 = e_N \equiv 0$$

where e_i is the error (or the correction, in the mutigrid context) and r_i is the residual. The coarse grid Ω_1 is formed by cell-centered coarsening. The coarse grid points are labelled with index $j = 1,.., N_1$, which are related to fine grid points by

$$j = [(i+1)/2] \qquad N_1 = [(N+1)/2] \tag{5.3.12}$$

That is, each coarse grid point j is formed by the union of two fine grid points, i = 2j-1 and i+1=2j. If the number of fine grid points is odd, then the last coarse grid point $j = N_1$ is formed by only the last fine grid point i = N. Summing up the residual equations (5.3.11) of the fine grid points i and i+1 that form the coarse grid point j, we have

$$(a_{pi} - a_{wi+1})\, e_i + (a_{pi+1} - a_{ei})\, e_{i+1} = a_{wi}\, e_{i-1} + a_{ei+1}\, e_{i+2} + r_i + r_{i+1} \tag{5.3.13}$$

We now have to approximate the correction e on the coarse grid. If we assume that

$$e_i \approx e_{i+1} = \phi_j^{(1)}$$

where $\phi^{(1)}$ is intended to be an approximate correction on the coarse grid, then we have the *coarse grid equation* as

$$a_{pj}^{(1)}\, \phi_j^{(1)} = a_{wj}^{(1)}\, \phi_{j-1}^{(1)} + a_{ej}^{(1)}\, \phi_{j+1}^{(1)} + b_j^{(1)} \tag{5.3.14}$$

where

$$a_{pj}^{(1)} = (a_{pi} + a_{pi+1} - a_{ei} - a_{wi+1}), \qquad a_{wj}^{(1)} = a_{wi}, \qquad a_{ej}^{(1)} = a_{ei+1}, \qquad b_j^{(1)} = r_i + r_{i+1}$$

The coarse grid equation (5.3.14) is exactly the same as the fine grid equation (5.3.10) in form, i.e. the original matrix structure is preserved. Although not explicitly involved, restriction and prolongation in the ACM are equivalent to the piecewise constant operators (5.3.6a). Similarly, we can easily form equations on coarse grid Ω_2 based on Ω_1 which now plays a role of the fine grid, and so on, until the coarsest grid Ω_L is reached.

For 2D or 3D problems, the above 1D ACM procedure is applied to each direction separately to generate the coarse grid equations. It is proved to be very flexible since we may not want to coarsen a grid direction that has only a few points.

Smoothers

A smoother used in multigrid can be any of iterative methods that have smoothing property, i.e. it is capable of damping down rough part (long-wave Fourier modes) of the error in a fixed, small number of iterations [Thole & Trottenberg (1986), Adams et al. (2003)]. The *point* Jacobi with under-relaxation and *point* Gauss-Seidel methods are the cheapest smoothers. The Jacobi scheme is suitable for parallel computation but it requires storage of two iterate vectors at the same time. The Gauss-Seidel scheme needs only one iterate vector and some of its modifications, such as the Red-Black Gauss-Seidel scheme, is also favourable for parallel computation [Kuo & Levy (1989), Yavneh (1995,1996)]. ADI methods can also be used for smoothing in 2D multigrid applications [Phillips (1987)]. Recently, CG-typed methods (as known as *Krylov subspace methods*) have also been used as multigrid smoothers [Scheilchl (2000), Elman et al. (2001)] despite these methods are actually the rougher, i.e. they reduce the long-wave Fourier modes faster than the short-wave modes. In fact, these methods are only used when a simple point relaxation method (e.g.

the Jacobi or Gauss-Seidel method) fails to smooth for certain types of algebraic systems such as of a non-symmetric non-positive definite matrix structure.

In 2D problems, the *line* Jacobi or *line* Gauss-Seidel methods can be used as smoothers if the matrix structure is regular. The line relaxation method utilizes the TDMA to invert a whole line at a time and is usually faster than the point methods [Alcouffe et al. (1981)]. However, when applied to 3D problems, an equivalent *plane* relaxation method is not easy to implement. In general, point relaxation methods appear to be the first choice for multgrid smoothing since the point methods are the simplest and cheapest. Therefore, multigrid methods are sometimes constructed so as to favour smoothing with a point relaxation method.

5.3.4 Multigrid Convergence

It suffices to use two-grid method for analysis of multigrid convergence. The purpose of two-grid analysis is to show that the rate of convergence of the two-grid method is independent of mesh size of the computational grid. For a simple 1D model problem, a convergence analysis of two-grid method with linear interpolation and a Gauss-Seidel smoother has been given in section (5.2). Here we will examine the multigrid components in more general. The two-grid algorithm is a special case of the multigrid algorithm, with L = 1.

Let the smoothing method $S(\phi,A,b,m)$ in the two-grid algorithm with m = 1 be defined such that

$$\phi = S\phi + k \tag{5.3.15}$$

The normal notation G of an iterative scheme as given in Chapter 4 is replaced with S to emphasize that the iterative method used here is a smoother. Applying pre-smoothing m_1 times, we will have the error as

$$e^{(1/3)} = S^{m_1} e^{(0)} \tag{5.3.16}$$

After coarse grid correction, the error becomes

$$e^{(2/3)} = [I - P A_H^{-1} RA]e^{(1/3)} = Ee^{(1/3)} \tag{5.3.17}$$

where $E \equiv [I - P A_H^{-1} RA]$ is the coarse grid correction matrix. Further applying post-smoothing m_2 times, we will have the error after *one* two-grid iteration as

$$e^{(1)} = S^{m_2} e^{(2/3)} = Qe^{(0)}, \qquad Q \equiv S^{m_2} E S^{m_1} \tag{5.3.18}$$

The convergence of two-grid method is governed by its contraction number $\|Q\|$ ($\|\cdot\|$ is some matrix norm). Assuming $m_2 = 0$ for simplicity, we may write

$$Q = [A^{-1} - P A_H^{-1} R][A S^{m_1}] \tag{5.3.19}$$

So that

$$\|Q\| \le \|A^{-1} - P A_H^{-1} R\| \, \|A S^{m_1}\| \tag{5.3.20}$$

The separate study of the two factors present in (5.3.20) leads to the following definitions [Hachbusch (1985)]:

- ***Smoothing property***: S has the smoothing property if there is exist constant K_S and a function $\eta(m)$ independent of h such that

$$\|AS^m\| \le K_S h^{-n_E} \eta(m) \qquad \eta(m) \to 0 \qquad \text{for } m \to 0 \tag{5.3.21}$$

  where n_E is the order of the PDE to be solved.

- ***Approximation property***: The approximation property holds if there exists a constant K_A independent of h such that

$$\|A^{-1} - P A_H^{-1} R\| \le K_A h^{n_E} \tag{5.3.22}$$

Now, let the smoothing property and the approximation property hold. Then there exists a number μ independent of h such that

$$\|Q\| \le K_S K_A \eta(m) < 1 \qquad \forall m > \mu \qquad \square \tag{5.3.23}$$

The *smoothing property* implies that the smoothing method is a convergent iteration method

$$\|S^m\| \le \|A^{-1}\| \cdot \|AS^m\| \le \|A^{-1}\| K_S h^{-n_E} \eta(m) \to 0 \qquad \text{as } m \to \infty \tag{5.3.24}$$

In general the rate of convergence of the smoothing method depends on h. The *approximation property* implies that that P and R satisfy (5.3.5) and that A and A_H are sufficiently accurate discretizations.

From the two-grid algorithm, it follows that if A_H = RAP (by using GCA), then $\mathbf{R}r^{(2/3)}=0$. Since restriction R is weighted average of neighbouring grid function values with positive weights, this implies that $r^{(2/3)}$ has many sign changes. In other words $r^{(2/3)}$ is

non-smooth, or rough, therefore, it can be further reduced effectively by smoothing. Every grid function can be decomposed into smooth and rough part. Let the error before coarse grid correction be split as

$$r^{(1/3)} = r_s^{(1/3)} + r_r^{(1/3)}$$

where the residuals with subscripts 's' and 'r' stand for smooth part and rough part, respectively. Using (5.3.17), we can write

$$r^{(2/3)} = Ae^{(2/3)} = \hat{E}\, r^{(1/3)}, \qquad \text{with } \hat{E} \equiv AEA^{-1}.$$

$$r_s^{(2/3)} = 0 \qquad r_r^{(2/3)} = r_r^{(1/3)} + \hat{E}\, r_s^{(1/3)} \tag{5.3.25}$$

It is seen once more that coarse grid correction reduces the smooth part of the residual, but there is also possibility that the rough part is amplified. If this amplification is too great, multigrid will not work properly. To avoid this, P and R must satisfy condition (5.3.5).

*
* *

For many problems in science and engineering, the multigrid has been shown to be an efficient way to handle big problems, using a principle "divide and conquer": different error modes are sieved through and eliminated on different grids. The greatest property of the multigrid that it tends to be an optimal method for algebraic solution suggests that we should try the multigrid to deal with the space problem in reactor physics, and this will be the topic of our discussion in the next chapter.

Chapter 6

MULTIGRID APPLICATION TO REACTOR PHYSICS

Having described how multgrid methods work in general for solving partial differential equations, we now explore the possibility of using these methods to deal with reactor physics problems. At first glance, it seems that the multigrid could be straightforwardly applied for solving the neutron kinetics equations as for any partial differential equations. However, we will see shortly that it is not easy to render both *principal properties* of the multigrid - *coarse grid approximation* and *smoothing* - in an application to reactor physics. Perhaps, this is the main reason why we have not found many multigrid applications to reactor physics in the literature. On the contrary, a quite different picture is observed in many areas of computation in science and engineering [Douglas (1997, 2003)]; for example, in computational fluid dynamics and heat transfer the multigrid methods have flourished with success over the past two decades [Wesseling (1992), Wesseling & Oosterlee (2001)].

In this chapter, we first try to identify the particular traits of reactor physics problems that make it difficult to apply the multigrid to their numerical solution, and then to find the way to overcome these difficulties.

6.1 Difficulties in Application of Multigrid to Reactor Physics Problems

As mentioned in Chapter 2, the primary objective of reactor physics is to determine the neutron flux distribution throughout a nuclear reactor core at any time. In practical reactor calculations, it is sufficient to solve the group diffusion theory equations - a set of *elliptic* partial differential equations - for the neutron group fluxes in space. In case of reactor kinetics, we need to solve the system of *parabolic* partial differential equations for these group fluxes both in space and time. In Chapter 3, we have seen that the most popular reactor static methods can be extended to spatial reactor kinetics. As a result, a reactor kinetics problem is solved at each time step as a boundary value problem.

Discretization of the neutron diffusion equations or, indeed, of any partial differential equations, leads to algebraic equation systems that we have to solve using computers. Among the spatial discretization methods used in nuclear reactor calculations, the family of nodal methods is currently predominant because these nodal methods allow a considerable reduction in the number of discretized equations to be solved (compared to an extremely larger number of difference equations if a finite difference method is used instead). However, due to structure complexity and non-linearity, the nodal discretized systems not only are not easy to solve but also appear incompatible with modern fast numerical methods, including the multigrid [Moulton (1996)].

Finite difference discretization of the neutron diffusion equations results in linear algebraic systems that, though extremely large in size, are most favourable to multigrid handling. In principle, the basic iterative methods as those given in Chapter 4 can be used to solve such large finite-difference systems but they are too slow to converge. Multigrid algorithms, as shown in Chapter 5, may help to accelerate the convergence of such iterative methods to an optimal level.

In a multigrid application, besides the given fine grid (on which we wish to obtain the discrete values of an unknown function), a set of coarse grids with fewer and fewer mesh points is created and an iterative method is used to eliminate the error components of different wave-lengths on different grids. These two main tasks of multigrid - *coarse grid approximation* (or *grid coarsening*) and *smoothing* - face difficulties when applied to reactor problems.

We will restrict ourselves to the *spatial multigrid*, as opposed to the *temporal multigrid*; that is, only *geometric grids* of spatial mesh points are considered in our multigrid application. For certain parabolic problems such as of low dimension, it is possible to coarsen grids of both spatial and temporal mesh points [Hackbusch (1984), Horton (1995), Larsson et al. (1995)], but this possibility should be excluded from reactor kinetics computations. The reason is that reactor neutron kinetics (neutronics) is usually coupled with other reactor dynamics processes such as thermalhydraulics. If a multiple time-step method for neutron kinetics were used for temporal grid to be coarsened, one would have to perform more or less expensive themalhydraulic calculations during the neutronics iteration process. This would not only require much larger memory storage but also deteriorate the convergence rate of either process solution.

6.1.1 Difficulty in Coarse Grid Approximation

As for coarse grid approximation, we first generate a set of coarse grids $\{\Omega_\ell \mid \ell=1,..,L\}$ from a given fine grid Ω, and, then, approximate the fine grid equation system

$$A\phi = b \tag{6.1.1}$$

on each of these coarse grids

$$A_\ell\phi_\ell = b_\ell\,, \qquad \ell = 1,..,L \tag{6.1.2}$$

Coarse grids Ω_ℓ will have fewer mesh points with an increasing index ℓ. Ω_L is the coarsest grid that should have only a very few (usually one) non-trivial mesh points.

The fine grid equation system (6.1.1) is obtained simply by discretizing the neutron diffusion equations on the fine grid Ω of mesh size $h = (h_x,h_y,h_z)$. The coarse grid equation system (6.1.2), as mentioned in Chapter 5, can be obtained either

i) by discretizing the original diffusion equations with the coarse grid mesh size h_ℓ (as known as discretization coarse grid approximation - DCA), or

ii) as a matrix product RAP (where R and P are, respectively, the restriction and prolongation operator matrices).

The first approach is commonly referred to as the *geometric multigrid* as we must know the detailed properties not only of the fine grid but also of every coarse grid as well. In the second approach, referred to as the *algebraic multigrid*, we do not need to know the coarse grid properties since we only use the coarse grid indexes for storing the coarse grid equation coefficients and unknowns. In reactor physics, neither of these approaches is easy.

As for the geometric multigrid, a coarse grid Ω_ℓ is formed from its finer-related grid $\Omega_{\ell-1}$ simply by increasing the grid mesh size $h_\ell = \sigma h_{\ell-1}$ (typically, σ is about 2 or less). In practical reactor calculations, the material properties of a heterogeneous core (i.e. the diffusion coefficients and the neutron-nuclear macroscopic cross sections - the *group constants*) are usually averaged (or homogenized) over volume of a core region (e.g. a fuel assembly or subassembly). Thus, the coefficients present in the neutron diffusion equations are piece-wise constant in each core region but discontinuous across the interface between the regions. The computational grid (which is the fine grid in our multigrid application) is usually chosen such that each integration box (or *node*) is completely bounded within such a homogenized core region and, therefore, has its group constants already defined. If a grid is so coarse that a grid node overlaps two or more different homogenized regions, we have to re-homogenize the group constants over this coarse node volume. Homogenization of a very large core region is costly and may introduce considerable errors (since it depends on the flux distribution which has yet to be computed). For this reason, Scheichl (2000) could not coarsen the grid but rather refined it, as did Kaveh et al. (2000). However, Kaveh et al. (2000) did coarsen the grids but could attain one or two coarse grids only, leaving so many grid points on the coarsest grid. It is suggested that the coarsest grid should have only a few grid points (possibly, only one non-trivial grid point) so that the algebraic system there would be small enough in size and hence could be solved *exactly* at negligible cost (this would get rid of all remaining error components) [Hacbucsh (1985)]. Apparently, it is not necessary to coarsen the grids to a very small size because there should exist a size (about ten points per grid dimension) of the computational grid at which both multigrid and single-grid solution costs are comparable. But a multigrid solver with hundred or more grid points remaining on the coarsest grid would be obviously not an efficient one.

Another issue arising in geometric multigrid is the validity of the multigrid convergence proof (5.3.23) as given in Chapter 5. We have seen that this proof is based on the relation (5.2.4),

$$A^H = RAP$$

which is essentially the other method of grid coarsening. While A^H in geometric multigrid is formed simply by discretization on the coarse grid H, only a few specific choices of R and P that can make the geometric coarse grid matrix A^H approximate to the product RAP. Therefore, it is doubtful that the convergence of a general geometric multigrid method is mesh-independent.

To avoid the need for determining the coarse grid group constants, the algebraic multigrid may be used for grid coarsening. In algebraic multigrid, a set of coarse-grid points is a subset of the fine-grid points. Formally, the coarse grid equation system is determined from its finer grid system by matrix multiplication

$$A_\ell = RA_{\ell-1}P \tag{6.1.3}$$

Except for some special cases when R, P and A are very simple, it is quite costly to compute (6.1.3). Also, as a result of algebraic coarsening (6.1.3), the coarse grid matrix may have a structure other than the fine grid matrix, e.g. it may become non-symmetric or even no longer diagonal-dominant. Consequently, basic iterative methods may fail to converge on coarse grids. Although algebraic multigrid methods appear to have great potential for practical applications, so far there have been no rigorous theoretical justifications of these methods and more investigations in this approach are still required [Xu (2001)].

6.1.2 Difficulty in Smoothing

Even when grid coarsening of reactor physics problems is possible (e.g. when one is able to homogenize the group constants over any arbitrary core volume, or alternatively employs an algebraic multigrid method), we may find it difficult to smooth the error on coarse grids, i.e. to use an iterative method for reducing the error components of different wavelengths on different grids. As for *smoothers* in multigrid application, one tends to use basic relaxation methods, such as the point Jacobi or Gauss-Seidel method. These point relaxation methods are simple and cheap but, unfortunately, they do not always work in solving the algebraic system of discretized neutron diffusion equations.

To see the limitation of the point relaxation methods in solving the neutron diffusion equation, let us consider the one-group one-dimensional equation with constant coefficients:

$$\frac{1}{v}\frac{\partial\phi}{\partial t} = D\frac{\partial^2\phi}{\partial x^2} - \Sigma_a\phi(x,t) + \nu\Sigma_f\phi(x,t) \qquad x \subset [0, \tilde{a}\,],\ t \geq 0 \tag{6.1.4}$$

with the initial condition: $\phi(x,0) = \phi_0(x)$

and the Dirichlet boundary condition: $\phi(0,t) = \phi(\tilde{a},t) = 0$

In equation (6.1.4), we neglect any source of delayed or independent neutrons since their inclusion does not change much the degree of difficulty in solving this equation.

Discretization of equation (6.1.4) with finite differences and a fully implicit time scheme gives the following algebraic system

$$\left[2 - h^2\frac{\nu\Sigma_f - \Sigma_a}{D} + \frac{h^2}{Dv\Delta t}\right]\phi_j = \phi_{j-1} + \phi_{j+1} + \frac{h^2}{Dv\Delta t}\phi_j^t, \qquad (6.1.5)$$

$$j = 1,..,N-1, \quad h = \tilde{a}/N$$

$$\phi_0 = \phi_N = 0$$

Although we can solve the tridiagonal system (6.1.5) directly (e.g. by using the Gaussian elimination method), we will use the point Jacobi and Gauss-Seidel methods to examine the behaviour of the error components of different wavelengths on a given grid during the iteration process. In case of multidimensional problems, the discretized systems have to be solved by iterations and the point relaxation method will affect the error components invariantly in each dimensional direction.

It is seen that, if $\Sigma_a \geq \nu\Sigma_f$ as in the case of neutron diffusion in an *absorbing medium*, the above system is strongly diagonal-dominant and there always exists a unique solution. The point Jacobi or Gauss-Seidel method is perfectly suited for smoothing the error as demonstrated in Chapter 5 for a pure diffusion problem. Let us consider the case when $\Sigma_a < \nu\Sigma_f$ as of neutron diffusion in a *multiplying medium* of a reactor core. The term $\frac{\nu\Sigma_f - \Sigma_a}{D}$ is now positive and it may cause the system matrix to lose its diagonal-dominant structure. It is customary to denote this term as the *reactor material buckling*

$$B^2 = \frac{\nu\Sigma_f - \Sigma_a}{D} \qquad (6.1.6)$$

For simplicity of analysis, suppose $\frac{h^2}{Dv\Delta t}$ is negligible in comparison with other terms in the brackets. Then the system (6.1.5) becomes

$$(2 - h^2B^2)\phi_j = \phi_{j-1} + \phi_{j+1} + b_j \qquad (6.1.7)$$

In order for the algebraic coefficients in the left hand side of (6.1.7) to be positive (so that the solution cannot be negative to be physically meaningful), we must set

$$hB < \sqrt{2}$$

More restrictive conditions have been suggested to avoid instability of the numerical solution of (6.1.7), for example,

$hB < 1$ [Alcouffe et al. (1981)]

$hB \le \frac{\pi}{5}$ [Elman et al. (2001)]

These conditions are actually the particular cases of a more general condition that we will derive shortly for the point relaxation solution of (6.1.7) to converge.

Similarly as in Chapter 5, the iterative solution of (6.1.7) with the point Jacobi or Gauss-Seidel relaxation method is given by

$$\phi_j^{(m)} = \frac{\omega}{2 - h^2 B^2}\left[\phi_{j-1}^{(m-\kappa)} + \phi_{j+1}^{(m-1)} + b_j\right] + (1-\omega)\,\phi_j^{(m-1)}, \qquad 0 < \omega < 2 \qquad (6.1.8)$$

where $\kappa = 1$ for the Jacobi method, and $\kappa = 0$ for the Gauss-Seidel method. Again, by using the Fourier mode analysis, we can obtain the amplification factor for each error mode, with the Jacobi relaxation,

$$g_J(\theta_k) = \left|1 - \omega\left(1 - \frac{2\cos\theta_k}{2 - h^2 B^2}\right)\right|, \qquad \theta_k = \frac{k\pi h}{\tilde{a}} = \frac{k\pi}{N}, \qquad k = 1,..,N\text{-}1$$

and with the Gauss-Seidel relaxation,

$$g_{GS}(\theta_k) = \sqrt{\frac{[(1-\omega)(2 - h^2 B^2)]^2 + 2\omega(1-\omega)(2 - h^2 B^2)\cos\theta_k + \omega^2}{(2 - h^2 B^2)^2 - 2\omega(2 - h^2 B^2)\cos\theta_k + \omega^2}}$$

It can be seen from Fig. 6.1.1 and Fig. 6.1.2 that, both Jacobi and Gauss-Seidel relaxation methods are capable of smoothing the error, i.e. they reduce the Fourier modes of short wavelengths ($k \ge N/2$). But while the Gauss-Seidel method smoothes the error with any relaxation parameter ω between 0 and 2, only the underrelaxation or "damped" Jacobi method with an appropriately chosen relaxation parameter can do so. This proper parameter ω for the damped Jacobi method depends on both mesh size and material

buckling B but we can always choose $\omega \le \omega_0 = 2/3$ to ensure that the damped Jacobi method remains as a smoother (see in Fig.6.1.1).

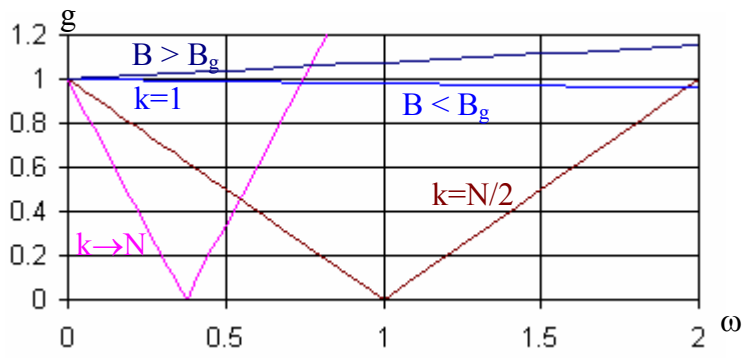

Figure 6.1.1. Amplification factor, g_J, for the Jacobi relaxation method applied to definite and indefinite problems

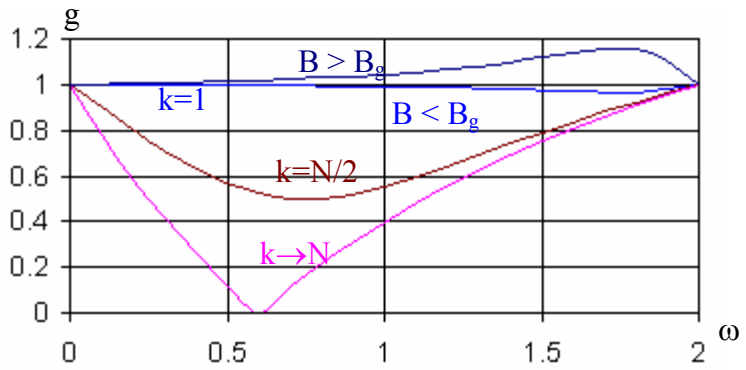

Figure 6.1.2. Amplification factor, g_{GS}, for the Gauss-Seidel relaxation method applied to definite and indefinite problems

A serious problem actually arises with the smooth part of Fourier modes (k<N/2), because both damped Jacobi and Gauss-Seidel methods may amplify some long-wave error modes under certain circumstances. As for the longest Fourier mode (k = 1), if h is small enough (or N is large) we can approximate

$$\cos\theta_1 = 1 - 2\sin^2\frac{\pi h}{2\tilde{a}} \approx 1 - \frac{h^2 B_g^2}{2},$$

where $B_g^2 = \left(\frac{\pi}{\tilde{a}}\right)^2$ is called the *geometric buckling*. The Jacobi amplification factor for the longest Fourier mode is

$$g_J = 1 + \omega \frac{h^2B^2 - 2 - 2\cos\theta_1}{2 - h^2B^2} \approx 1 + \omega \frac{h^2(B^2 - B_g^2)}{2 - h^2B^2}$$

and the plain Gauss-Seidel ($\omega = 1$) amplification factor is

$$g_{GS} \approx \frac{1}{\sqrt{1 + (2 - h^2B^2)h^2(B_g^2 - B^2)}}.$$

Obviously, these point relaxation methods amplify (i.e. $g > 1$) the longest Fourier mode when the material buckling is larger than the geometric buckling as in case of a supercritical reactor

$$B > B_g \tag{6.1.9}$$

We recall from reactor analysis that B_g^2 is the smallest eigenvalue of the steady-state one-group neutron diffusion equation for a bare homogenous reactor [Duderstadt & Hamilton (1976)]. It is known that when $B^2 = B_g^2$, the reactor is critical, and system (6.1.7) is singular and has an indefinite number of solutions - the eigenvectors. When $B^2 > B_g^2$, system (6.1.7) becomes *indefinite* [Elman et al. (2001)]. It is this indefiniteness of the Helmholtz equation that makes the point relaxation methods fail to converge. But even when we can solve the indefinite system (6.1.7) by some way, its solution is not everywhere non-negative and thus contradicts the physical meaning of the neutron flux.

On a very coarse grid where h is large, we have a slightly stricter condition than (6.1.9) as

$$hB_g > 2\sin\frac{\theta_1}{2} > hB$$

For N = 2, $hB < 2\sin\frac{\pi}{4} = \sqrt{2}$ or $B < 0.90B_g$

For N = 3, $hB < 2\sin\frac{\pi}{6} = 1$ or $B < 0.95B_g$

For N = 5, $hB < 2\sin\frac{\pi}{10} < \frac{\pi}{5}$ or $B < 0.98B_g$

In solving multidimensional diffusion equations, the point relaxation methods work well only when the problem is isotropic. In case of anisotropic problems, they only smooth the error in the direction of strong coupling. The semi-coarsening strategy [Schaffer (1998)], i.e. when grids are coarsened only in one direction, is one way to achieve the fast convergence of the multigrid algorithm but is applicable only to very specific problems such as fluid flows in a narrow channel. Another way is to use *block relaxation methods* instead of point relaxation for smoothing. *Line relaxation* has been proven to be very efficient in two dimensions [Alcouffe et al. (1981)] but *plane relaxation*, which is equivalent to line relaxation in three dimensions, become very complicated to implement [Stuben (2001)].

When the algebraic multigrid method is applied for grid coarsening, the coarse grid equation systems may become no longer symmetric positive definite. In such cases, the basic iterative methods are not suitable for use as smoothers. Zalavsky (1993, 1995) employed the so called *Kaczmarz iteration* to smooth the error components on every grid. The Kaczmarz iteration is the Gauss-Seidel iteration applied to the following equivalent symmetric system

$$A^T A\phi = A^T b \tag{6.1.10}$$

The Kazmacz iteration has an advantage of not amplifying any modes but it is very slow at reducing the short modes and hence is not a good smoother [Elman et al. (2001)].

Recently, a class of *Krylov subspace methods* [Vorst (2000)] has been tried for smoothing in a number of multigrid applications, especially when the algebraic systems are not symmetric positive definite. In these methods, a general linear system (6.1.1) is solved by constructing the best approximate solution in the Krylov subspace of dimension m

$$K_m(A,r^{(0)}) = \text{span}\{ r^{(0)}, Ar^{(0)}, A^2r^{(0)},\ldots, A^{m-1}r^{(0)}\} \tag{6.1.11}$$

where $r^{(0)} \equiv b - Ax^{(0)}$ is the initial residual vector. The idea is to keep all approximants computed so far in the iteration process and to recombine them to a better solution. Since the Krylov subspace is of dimension N (N is the number of equations in the linear system), after N-1 steps, the iteration process must terminate with an exact solution.

The simplest of the Krylov subspace methods is the Conjugate Gradient method, given in Chapter 4, but it can only be used for symmetric positive definite problems. For problems when the matrix is not symmetric positive definite, the *General Minimal Residual* (GMRES) method [Saad & Schultz (1986), Kwak (1997)] is used instead. In the GMRES, one has to store all basis vectors for the Krylov subspace, which means the more iterations the more basis vectors to be stored. Also, the work per iteration increases linearly with the iteration number. Krylov subspace methods are much more expensive

than any basic iterative method and are also largely problem-dependent. In general, a Krylov subspace method requires *preconditioning* to achieve the efficient performance, and, sometimes, even the convergence. The preconditioning transforms the original linear system into a new one

$$K^{-1}Ax = K^{-1}b \tag{6.1.12}$$

where K is the *preconditioner* which approximates the matrix A in some way but its inversion is easy. There is no general approach to construct an appropriate preconditioner. Moreover, it is still unclear how a Krylov subspace method smoothes the error when applied as a multigrid smoother. The Krylov subspace methods could converge after a number of iterations but they do not necessarily reduce the short-wave error modes in several first iterations as we must require for smoothers in multigrid.

The difficulties described above are just for the solution of the one-group neutron diffusion equation in which only spatial coupling is taking place. For multigroup problems, in addition to the spatial coupling between the neutron fluxes of the same energy group, there is also the energy coupling between the group fluxes at a grid point. It is the group coupling of the fluxes that makes the structure of the algebraic coefficient matrix A complicated and non-symmetric. To avoid this complexity, it is most common to use the *source iteration technique* [Waschpress (1966)] consisting of an *outer loop* to update the source term and an *inner loop* to solve for spatial distribution of the group fluxes at a fixed source. With this source iteration procedure, the multigrid could only be applied to the inner iteration to accelerate the solution of the fluxes in space at a given source. However, this approach appears not very efficient because there is no need to compute *exactly* the spatial fluxes per inner iteration while the source is still roughly approximated. Also, except for several first outer iterations where the multigrid is really effective at eliminating the error modes of all wavelengths, the use of multigrid is wasted when the source is close to convergence because at this stage of the iteration process only a couple of inner iterations on a single fine grid is quite enough.

6.2 Additive Correction Multigrid

Multigrid methods are intended to reduce the computational work of solving large linear algebraic systems to such an extent that solution cost will be linearly proportional to the number of equations in the system. But the proportional factor also depends on performance of multigrid operations, i.e. grid coarsening, smoothing and intergrid transferring. If these multigrid operations are expensive, the proportional factor is so large that efficiency of the multigrid method may not be proven. It is the more usual practice to minimize the multigrid work by losing some of the proportional linearity in return for reducing the proportional factor. Therefore, in our development of a multigrid solver for the spatial neutron kinetics problem, we always try to keep the grid operations as cheap as possible.

As already mentioned, both geometric and algebraic methods described above for grid coarsening in solving the neutron diffusion equations are quite expensive, and also may lead to coarse grid equation systems that are difficult to solve. It is desirable that grid coarsening is simple but still produces an easy coarse grid system to be solved. The *Additive Correction Multigrid* (ACM) method is the one which, in our opinion, satisfies the above mentioned requirements. The ACM was first introduced by Hutchison & Raithby (1986) to a heat diffusion problem, and was implemented in computational fluid dynamics as well [Hutchison et al. (1988)]. Gjesdal (1996) pointed out that the ACM can be regarded as a basic algebraic multigrid method where restriction and prolongation are based on piecewise constant interpolation. The only concern was that a multigrid method with these lowest order restriction and prolongation might not provide a solution that is linearly proportional to the grid size for the second order differential equations. But it is the simplicity of the method that reduces the proportional factor to a minimum.

6.2.1 Discretization

As for the ACM, the cell-centered grid is the most suitable and a finite volume discretization technique [Patankar (1980)] is particularly useful. Thus, the reactor core with the boundary extended to include the extrapolated length is divided into a number of control volumes in rectangular form. Each control volume or cell is bounded entirely within a homogenized core region with the average material properties initially defined. The balance equations of neutron fluxes, and of delayed neutron precursors as well in case of reactor kinetics calculations, are then integrated over the control volume with respect to the spatial variable [Waschpress (1966)]. In fact, the common discretization techniques in reactor physics, including finite difference, finite element and nodal methods, all utilize this control volume integration method. These reactor physics methods differ only in the way the neutron flux shape is defined within the control volume for computing the neutron current at faces of this control volume. The finite difference method approximates a linear shape of the flux and thus requires a very small size of the control volume to assure an acceptable accuracy of discretization. Coarse-mesh methods, e.g. nodal or finite element methods, utilize a higher order approximation of the flux shape and, consequently, can have a quite larger control volume but still provide the same accuracy. In term of computational performance, the finite difference method has an advantage over the other coarse-mesh methods of producing a symmetric, usually positive definite, linear system of finite difference equations whose numerical solution is well studied by mathematicians [Sutton & Aviles (1996)]. That is, by analyzing the matrix of algebraic coefficients, we, in many cases, can predict the convergence behaviour and computational work of an iterative scheme applied to the system.

As for time discretization, the reactor kinetics equations are temporally integrated over a rather short time interval Δt, during which the material properties (i.e. the group constants) are assumed to be unchanged (see in section 3.2.1).

In Chapter 3, we have obtained the algebraic system for group fluxes by using the cell-centered finite difference discretization in a 3D Cartesian regular grid and with a general implicit scheme of time integration as

$$a_g^p \phi_g^p = \sum_{nb \subset P} a_g^{nb} \phi_g^{nb} + \sum_{g'=1}^{G} a_{g'g}^p \phi_{g'}^p + b_g^p, \qquad g = 1,..,G \tag{6.2.1}$$

$$\phi_g^B = 0 \text{ (for boundary nodes)}$$

where g denotes the energy group of neutrons; 'nb' denotes a neighbouring node, left or right in any of three directions x, y, or z, of an inner node P (which is not on the grid boundary). The algebraic coefficients present in (6.2.1) are given in Chapter 3.

6.2.2 Algebraic Solution

For multidimensional reactor kinetics problems, we have to resort to an iterative solution of the discretized system (6.2.1). As already mentioned, this discretized kinetics system can be solved by using the *source iteration* technique, which consists of an outer loop of source iteration and an inner loop of flux iteration. In general, at the m-th outer iteration, the source term is approximated (guessed) as

$$S_g^{p(m)} = \sum_{g'=1}^{G} a_{g'g}^p \phi_{g'}^{p(m)} + b_g^P \tag{6.2.2}$$

and the group-coupled system (6.2.1) is transformed into G purely spatially-coupled systems of the form

$$a_g^p \phi_g^p = \sum_{nb \subset P} a_g^{nb} \phi_g^{nb} + S_g^{p(m)}, \qquad g = 1,..,G \tag{6.2.3}$$

In the inner iteration, these G uncoupled systems (6.2.3) are now solved by using the Gauss-Seidel or SOR method. Since each of systems (6.2.3) is symmetric and positive definite, the Gauss-Seidel relaxation is guaranteed to converge. The solution by such an inner iteration is then used to update the source term as in (6.2.2). This outer-inner iteration process is repeated until the solution converges to within a given accuracy. In order to accelerate convergence of the source term, it is common to use the Chebyshev acceleration technique [Duderstad & Hamilton (1976)]. However, utilization of the Chebyshev method requires sufficient knowledge about the eigenvalues of the multigroup equation system (6.2.1) which are difficult to estimate in practice. Within this source iteration approach, multigrid methods can be easily applied to systems (6.2.3) to accelerate

convergence of the inner iteration. But the multigrid apparently is not very efficient if implemented this way because it is unnecessary to solve systems (6.2.3) exactly while their source terms are still not exact.

It appears that we can combine the above inner and outer iterations in such a way that multigrid application for solving the multigroup diffusion equations becomes more efficient. Such an iteration procedure is to iterate (6.2.1) from grid point to grid point by using a *block* Gauss-Seidel method. For this purpose, let us rewrite system (6.2.1) in matrix notation as

$$[A_p][\Phi_p] = \sum_{nb \subset P} [A_{nb}][\Phi_{nb}] + [B_p], \tag{6.2.4}$$

where all vectors and matrices are given by

$$[\Phi_p] = \begin{bmatrix} \phi_1^p \\ \phi_2^p \\ \dots \\ \phi_G^p \end{bmatrix}; \qquad [B_p] = \begin{bmatrix} b_1^P \\ b_2^P \\ \dots \\ b_G^P \end{bmatrix}$$

$$[A_P] = \begin{bmatrix} a_1^P - a_{11}^P & -a_{21}^P & \dots & -a_{G1}^P \\ -a_{12}^P & a_2^P - a_{22}^P & \dots & -a_{G2}^P \\ \dots & \dots & \dots & \dots \\ -a_{1G}^P & -a_{2G}^P & \dots & a_G^P - a_{GG}^P \end{bmatrix}; \qquad [A_{nb}] = \begin{bmatrix} a_1^{nb} & & & \\ & a_2^{nb} & & \\ & & \dots & \\ & & & a_G^{nb} \end{bmatrix}.$$

The price to pay is we have to invert the matrix $[A_p]$ at each grid point for determining the group fluxes as

$$[\Phi_p]^{(m+1)} = [A_p]^{-1} \{ \sum_{nb \subset P} [A_{nb}][\Phi_{nb}] + [B_p] \} \tag{6.2.5}$$

where the values of $[\Phi_{nb}]$ are the most recent available. In terms of computational cost on a single grid this iteration procedure does not much differ from the source iteration procedure (we only change the order of iteration from a group-then-point order as in the source iteration with a point-then-group order) but it much better accommodates the multigrid implementation. Also, since the number of neutron groups G is usually relatively small (up to 4 groups for thermal reactor calculations and 20 groups for fast reactor calculations), it is possible to solve (6.2.5) with a direct elimination method at low cost.

In practice, we can check the convergence of the relaxation method by examining any of the following residual norms

$$||r^{(m)}||_\infty = \max_{p,g}\{|\, r_g^{p(m)}\,|\} \tag{6.2.6a}$$

$$||r^{(m)}||_1 = \sum_p \sum_{g'=1}^{G} |\, r_g^{p(m)}\,| \tag{6.2.6b}$$

$$||r^{(m)}||_2 = \left(\sum_p \sum_{g'=1}^{G} |\, r_g^{p(m)}\,|^2 \right)^{1/2} \tag{6.2.6c}$$

where $r_g^{p(m)}$ is the residual after m iterations computed as

$$r_g^{p(m)} = \sum_{nb\subset P} a_g^{nb}\phi_g^{nb(m)} + \sum_{g'=1}^{G} a_{g'g}^{p}\phi_{g'}^{p(m)} + b_g^p - a_g^p\,\phi_g^{p(m)} \tag{6.2.7a}$$

or

$$[R_p]^{(m)} \equiv \mathrm{Col}\{\, r_g^{p(m)} \mid g=1,..,G\} = [A_{nb}][\Phi_{nb}]^{(m)} + [B_p] - [A_p][\,\Phi_p]^{(m)} \tag{6.2.7b}$$

The iteration process is stopped when the residual norm is reduced to a desirable extent as

$$||R^{(m)}|| \,/\, ||R^{(0)}|| \le \varepsilon \tag{6.2.8}$$

where ε is a given positive number.

6.2.3 Coarse Grid Approximation

The ACM does not require the coarse grids to be geometrically created because only the indexes of coarse grid points are needed to store the coarse grid equation coefficients and unknowns. But for a better view, we will form a set of coarse grids as if they were geometric ones [Nguyen & Garland (2004)].

Let I, J, K be the inner sizes, i.e. excluding the boundary points, of the fine grid Ω where the algebraic system (6.2.1) is obtained from discretization of the original partial differential equations. The cell-centered grid Ω is composed of a total of I×J×K non-overlapped nodes of rectangular form. Each grid node contains a grid point in its centre and is denoted by a three-integer index (i,j,k),

$$i = 1,..,I; \qquad j = 1,..,J; \qquad k = 1,..,K.$$

The boundary nodes that contain the points on the grid boundary are chosen such that they all have a zero width in the direction to its nearby inner node. It is often that a boundary condition is absorbed into a nearby inner node equation; therefore, practically we only deal with the inner nodes of the grid.

We now form the coarse grid Ω_1 by agglomerating successively two (or one if it is the last one) layers (or planes) of fine grid nodes in one direction, say, the i-direction, into one layer of coarse grid nodes. Then, the process is repeated in the second, and, finally, in the third direction. As a result, one coarse grid node is formed from a block of 8 fine grid nodes or less, namely, 4, 2 or 1. The index of coarse grid nodes is denoted by $(i,j,k)_1$,

$$i_1 = 1,..,I_1; \qquad j_1 = 1,..,J_1; \qquad k_1 = 1,..,K_1.$$

Similarly, each of coarser grids Ω_ℓ, $\ell = 2,..,L$, is formed from grid $\Omega_{\ell-1}$ in the same way until the coarsest grid Ω_L that has only one inner grid node ($I_L = J_L = K_L = 1$) is reached. It is convenient to assign index $\ell = 0$ to the fine grid so that we have the following set of grids

Ω_0 (or Ω) $\qquad I_0 = I; \quad J_0 = J; \quad K_0 = K$

Ω_ℓ ($\ell \geq 1$) $\qquad I_\ell = [(I_{\ell-1}+1)/2]; \qquad J_\ell = [(J_{\ell-1}+1)/2]; \qquad K_\ell = [(K_{\ell-1}+1)/2]$

Ω_L $\qquad I_L = J_L = K_L = 1$

with grid node indexes $(i,j,k)_\ell$,

$$i_\ell = 1,..,I_\ell; \qquad j_\ell = 1,..,J_\ell; \qquad k_\ell = 1,..,K_\ell; \qquad \ell = 0,1,..,L.$$

Although system (6.2.1) or (6.2.5) can be solved by iterations on the fine grid, it is costly to do so. The reason, as we have learned in Chapter 5, is that the basic iterative method only smoothes the error, i.e. while the rough part of the error (the short-wave Fourier modes) is effectively damped down by this iteration, the smooth part (the long-wave modes) is little affected. The idea of the multigrid is that we send the long-wave modes left on a given grid to coarser grids and eliminate them there. The long-wave Fourier modes on one grid turn out to be short waves on a coarser related grid and hence can be damped down effectively there.

Suppose that after a number of iterations applied to the system (6.2.5) on the fine grid, the exact solution can be calculated by

$$[\Phi_{ijk}] = [\Phi_{ijk}]^{(m)} + [E_{ijk}] \qquad i = 1,..,I;\ j = 1,..,J;\ k = 1,..,K \tag{6.2.9}$$

where $[E_{ijk}] = \mathrm{Col}\{ e_g^{ijk} \mid g=1,..,G\}$ is the vector of errors at a node $P \equiv (i,j,k)$ and m is the number of iterations. It is obvious that if the solution converges then $[E_{ijk}] \to 0$. Substituting (6.2.9) into (6.2.5), we will obtain the so called residual equation

$$\begin{aligned}[A_P^{ijk}][E_{ijk}] = {} & [A_W^{ijk}][E_{i-1,j,k}] + [A_E^{ijk}][E_{i+1,j,k}] + [A_S^{ijk}][E_{i,j-1,k}] + [A_N^{ijk}][E_{i,j+1,k}] \\ & + [A_U^{ijk}][E_{i,j,k-1}] + [A_L^{ijk}][E_{i,j,k+1}] + [R_{ijk}]\end{aligned} \tag{6.2.10}$$

Here we have explicitly written down the indexes of the nodes in neighbourhood of the (i,j,k)-node. The algebraic coefficients with a subscript W, E, S, N, U or L are for coupling between the node (i,j,k) with the node west, east, south, north, upper and lower of it, respectively. It can be also noted that, solving the original system (6.2.5) for $[\Phi_{ijk}]$ is equivalent to solving the residual system (6.2.10) for the error $[E_{ijk}]$ and then using (6.2.9) to correct the solution. *Multigrid coarse correction* is an algorithm that transfers the residual system (6.2.10) to a coarser-related grid and solves for the *correction* $[E_{ijk}]$ there. In the ACM, the algebraic equation for a given coarse grid node is obtained by summing up all residual equations in the block of fine grid nodes that forms the coarse grid node, assuming all the fine nodes to have the same error. Let a coarse grid node $(i,j,k)_l$ be formed by a block of 8 fine grid nodes $\{(i,i+1),(j,j+1),(k,k+1)\}$. Also, we denote

$$[\Phi_{ijk}^l] = [E_{i+a,j+b,k+c}] \quad \forall a,b,c;\ \ a = 0,1;\ \ b = 0,1;\ \ c = 0,1 \tag{6.2.11}$$

Adding all residual equations in the block and rearranging the resulting equation with (6.2.11) taken into account, we get

$$\begin{aligned}[A_P^{(ijk)_l}][\Phi_{ijk}^l] = {} & [A_W^{(ijk)_l}][\Phi_{i-1,j,k}^l] + [A_E^{(ijk)_l}][\Phi_{i+1,j,k}^l] \\ & + [A_S^{(ijk)_l}][\Phi_{i,j-1,k}^l] + [A_N^{(ijk)_l}][\Phi_{i,j+1,k}^l] \\ & + [A_U^{(ijk)_l}][\Phi_{i,j,k-1}^l] + [A_L^{(ijk)_l}][\Phi_{i,j,k+1}^l] + [B_{ijk}^l]\end{aligned} \tag{6.2.12a}$$

or, in a short form,

$$[A_P^{(ijk)_l}][\Phi_{ijk}^l] = \sum_{nb \subset (ijk)_l} [A_{nb}^{(ijk)_l}][\Phi_{nb}^l] + [B_{ijk}^l] \tag{6.2.12b}$$

where

$$[B^{l}_{ijk}] = \sum_{abc=0}^{1} [R^{(m)}_{i+a,j+b,k+c}] \equiv \text{the sum of all residuals in the block}$$

$$[A^{(ijk)_l}_{W}] = \sum_{bc=0}^{1} [A^{i-1,j+b,k+c}_{W}] \equiv \text{the sum of } [A_W] \text{ in the } \textit{west} \text{ nodes of the block,}$$

and, similarly for the other $[A_{nb}]$, i.e.

$[A^{(ijk)_l}_{nb}] \equiv$ the sum of $[A_{nb}]$ in the nodes on the 'nb' face of the block

$$[A^{(ijk)_l}_{p}] = \sum_{abc=0}^{1} [A^{i+a,j+b,k+c}_{P}] - \sum_{bc=0}^{1} [A^{i,j+b,k+c}_{W} + A^{i+1,j+b,k+c}_{E}]$$

$$- \sum_{ac=0}^{1} [A^{i+a,j,k+c}_{S} + A^{i+a,j+1,k+c}_{N}] - \sum_{ab=0}^{1} [A^{i+a,j+b,k}_{U} + A^{i+a,j+b,k+1}_{L}]$$

$\equiv$ the sum of all $[A_P]$ in the block minus the sum of all $[A_{nb}]$ in the nodes not on the 'nb' faces of the block

In case a coarse grid node is formed from fewer fine grid nodes, the above expressions can still be used, simply by setting to zero the algebraic coefficient of the fine grid equation whenever any of its indexes is beyond its bound.

The great feature of the ACM is that the coarse grid equation (6.2.12) has exactly the same form as the fine grid equation. Therefore, it is possible to use the same smoothing method on the fine and coarse grid.

Since the solution of (6.2.12) on the coarse grid Ω_1 is still expensive, we again apply coarser grid correction to its solution. This procedure is repeated in a recursive manner until we reach the coarsest grid, on which the algebraic system is solved exactly to get rid of all remaining error modes. In an analogous way we can form the coarse grid equation on any coarse grid, for example, $\Omega_{\ell+1}$, from the equation system already obtained on the finer-related grid Ω_ℓ. As a result, the coarse grid equation systems on all coarse grids have the same form as of the fine grid equation (6.2.5).

It is clearly seen that the formation of coarse grid equations does not require any knowledge about the coarse grid properties. Thus the ACM can be regarded as an algebraic multigrid method. But unlike the other algebraic multigrid methods, it does not require expensive multiplication of matrices. Moreover, it preserves the matrix structure of the algebraic system on all grid levels. With the formal definition of the coarse grid matrix $A_{\ell+1} = RA_\ell P$, it has been shown that R and P are piecewise constant operators [Gjesdal (1996)]. In fact, the prolongation operator P takes the solution of the coarse grid equation $\phi_{\ell+1}$ as the correction for the fine grid solution ϕ_ℓ. The restriction operator R sums up the fine grid residuals of the unity weighting and transfers it to the coarse grid.

Another remarkable feature of the ACM is that even when the fine grid system is close to singular or indefinite, the coarse grid systems may be not. For example, consider again the 1D problem (6.1.4), where the algebraic coefficients are obtained as

- on the fine grid: $a_p^i = 2 - h^2B^2$; $a_W^i = a_E^i = 1$; $i = 1,..,N$; $h = \tilde{a}/N$

- on a coarse grid:

by DCA: $a_p^{i_1} = 2 - h_1^2B^2 = 2 - 4h^2B^2$; $i_1 = 1,..,[N/2]$; $h_1 = 2h$

by ACM: $a_p^{i_1} = a_p^i + a_p^{i+1} - a_E^i - a_W^{i+1} = 2 - 2h^2B^2 > 2 - h_1^2B^2$

In the case of DCA, any singularity or indefiniteness of the algebraic system on the fine grid is preserved on the coarse grids. But with the ACM, the diagonal dominance of the coarse grid equations would be regained. Consequently, when point relaxations are slow or even fail for DCA multigrid they still work for ACM.

6.2.4 Avoiding Indefiniteness

As mentioned above, the indefiniteness of the discretized system will prevent the use of a point relaxation method for smoothing. For the neutron diffusion equation, the indefiniteness of the discretized system is possible only when the material buckling is larger than the geometric buckling (cf. (6.1.9)). Even when we are able to solve an indefinite discretized system, for example, by using a direct method, part of its numerical solution obtained may be negative so that this solution is physically meaningless (because the neutron flux must be always non-negative). Therefore, it is important that the indefiniteness of the discretized system be avoided.

In case of *reactor criticality calculations*, we have either to choose a right composition of the reactor core or to introduce a fudging coefficient k (which turns out to be the *effective multiplication factor* of a given core composition [Duderstadt & Hamilton (1976)]) so that

$$B^2 = \frac{\frac{1}{k}\nu\Sigma_f - \Sigma_a}{D} \approx B_g^2 \tag{6.2.13}$$

If such a core composition is chosen for the reactor to be critical, then $B = B_g$ and $k = 1$. Alternatively, $k > 1$ if $B > B_g$ and $k < 1$ if $B < B_g$. In either case, to avoid the

indefiniteness of the discretized system, one has to adjust the material buckling B or the fudging coefficient k such that B^2 or $\frac{\frac{1}{k}\nu\Sigma_f - \Sigma_a}{D}$ approaches but is less than B_g^2.

For *reactor kinetics* problems, when $B \neq B_g$ we will expect the neutron flux to change with time. If $B < B_g$ the flux decreases to the level that is determined by an extra source. More importantly, the discretized system in this case is definite and we do not have any problem with solving the algebraic system (6.1.5). Now, when $B > B_g$, the discretized system may become indefinite so that not only cannot it be solved by point relaxations but its algebraic solution has no physical meaning. It follows from (6.1.5) that, in order to avoid this indefiniteness, we must choose a time step Δt such that

$$B^2 - \frac{1}{Dv\Delta t} < B_g^2$$

Supposing that the reactor is critical with a fudging factor k as of (6.2.13), we have

$$\frac{1}{Dv\Delta t} > \frac{\nu\Sigma_f - \Sigma_a}{D} - \frac{\frac{1}{k}\nu\Sigma_f - \Sigma_a}{D} = \frac{k-1}{k}\frac{\nu\Sigma_f}{D},$$

or

$$\Delta t < \frac{1}{\rho v \nu\Sigma_f}, \tag{6.2.14}$$

where $\rho \equiv \frac{k-1}{k}$ is the *reactivity*. Although we have used an implicit scheme for time integration of the time-dependent neutron diffusion equation, there is still a restriction to the time step as in case of the explicit scheme. But unlike the explicit time step, which is too small to use in practice because it depends not only on the neutron dynamic characteristics but, more importantly, on the squared spatial mesh size h^2, the implicit time step (6.2.14) appears to depend only on rapidity of the neutron population growth, i.e. a value of positive reactivity. If this positive reactivity is small, we can use a large time step Δt. But when the reactivity is large we must reduce the time step accordingly, and this is quite consistent because when a computed function changes rapidly we must require a small time step, at least, for accuracy.

We can estimate values of the time step given in (6.2.14). When $\rho \sim 0.01$ (this value of reactivity would make a reactor critical on the prompt neutrons alone), for

thermal neutron group ($v \sim 2.2\times10^5$cm/s, $\nu\Sigma_f \sim 0.1$cm^{-1}), $\Delta t \sim 5\times10^{-3}$sec and for the fastest group ($v \sim 10^9$cm/s but $\nu\Sigma_f \sim 0.001$cm^{-1}), $\Delta t \sim 10^{-4}$-10^{-5}sec (cf. the explicit time step is less than 10^{-8}sec).

*
* *

It is not easy to apply a multigrid method to reactor physics problems in general, due to difficulties in both grid coarsening and error smoothing, with an exception of the additive correction multigrid (ACM) method. The only doubt about the ACM is that its convergence may be mesh-dependent when used for numerical solution of the reactor kinetics problem. But it is the simplicity of the method that encourages us to use it in the hope that the ACM convergence does not too strongly depend on the size of the computational grid. We will investigate into this hypothesis in numerical examples in the next chapter.

Chapter 7

NUMERICAL EXPERIMENTS

This Chapter is intended to show how efficient the Additive Correction Multigrid (ACM) method would be when applied to such difficult problems of spatial reactor kinetics, through a number of numerical examples, from the simplest one-dimensional one-group problem to the most general multidimensional multigroup problem.

7.1 Mesh-Dependence of Multigrid Convergence

The Additive Correction Multigrid (ACM) is the simplest and, of course, cheapest method in the multigrid family. However, as mentioned in Chapter 5, since the ACM employs the lowest order prolongation and restriction based on the piecewise constant interpolation, its convergence for solving second order differential equations may depend on the mesh size of a grid used for spatial discretization of the differential equations. The fact that other multigrid methods based on higher order transfer operations are too difficult to implement for solving reactor physics problems has given us no choice but to use the ACM. However, we expect that its convergence would not sensitively depend on the grid size as well as its solution cost is small compared with usual iterative methods.

7.1.1 A One-Dimensional One-Group Example

The original equation

In order to verify our expectation of the ACM convergence, let us consider a one-group time-dependent slab reactor problem, neglecting any delayed or extra neutron sources

$$\frac{1}{v}\frac{\partial}{\partial t}\phi(x,t) = \frac{\partial}{\partial x}D(x)\frac{\partial}{\partial x}\phi(x,t) + [\nu\Sigma_f(x) - \Sigma_a(x)]\phi(x,t), \qquad (7.1.1)$$

$$x \subset [0, \tilde{a}\,], \qquad t > 0$$

Initial condition $\phi(x, 0) = \sin\frac{\pi x}{\tilde{a}}$

Boundary conditions $\phi(0, t) = \phi(\tilde{a}\,, t) = 0$

where $\tilde{a}$ is the thickness of the slab reactor, including the extrapolated length. All notations in equation (7.1.1) are standard. Assume that the slab core consists of an array of fuel and moderator cells (Fig.7.1.1) each of which has their own diffusion coefficient and cross sections homogenized over the cell volume. That is, D, $\nu\Sigma_f$ and Σ_a are piecewise constant but may vary from cell to cell. In general, numerical solution of

equation (7.1.1) does not pose any great difficulty, except when we have to use so many mesh points for discretization. We will shortly see that basic numerical methods (such as Jacobi, Gauss-Seidel or SOR) cannot compete with the ACM in terms of computational efficiency, and it is only the ACM that can handle a very large number of mesh points effectively.

Discretization

For spatial discretization of equation (7.1.1), we will use a cell-centered grid (Fig.7.1.1) defined as

$$\Omega = \{x_i \mid i = 0,1,..,N+1\} \tag{7.1.2}$$

where $x_0 = 0$; $x_{N+1} = \tilde{a}$; $x_i = x_{i-1} + \dfrac{h_i + h_{i-1}}{2}$ (i = 1,..,N); $h_0 = h_{N+1}=0$.

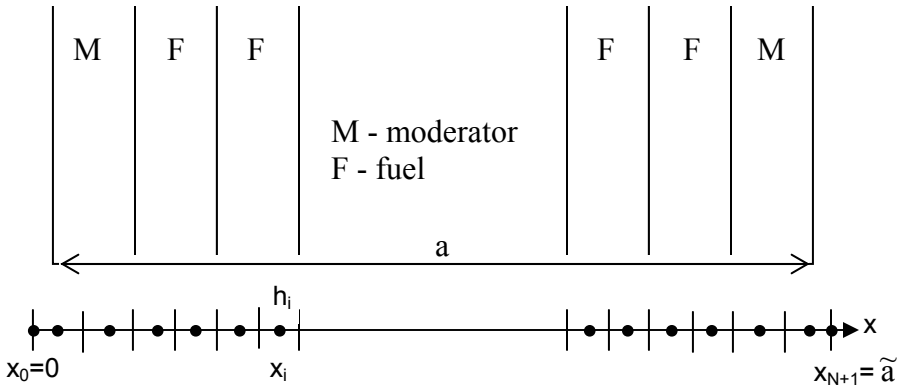

Figure 7.1.1. Cell-centered grid for discretization of a 1D diffusion problem

Denoting

$\phi_i^t \equiv \phi(x_i, t)$ - the known flux already computed at a previous time step, and

$\phi_i \equiv \phi_i^{t+\Delta t} \equiv \phi(x_i, t+\Delta t)$ - the unknown flux at a current time step

and by using a fully implicit scheme for time integration, we obtain the following algebraic system for the unknown fluxes at any time step as

$$a_{pi}\phi_i = a_{wi}\phi_{i-1} + a_{ei}\phi_{i+1} + b_i, \qquad i = 1,..,N \tag{7.1.3}$$
$$\phi_0 = \phi_{N+1} = 0,$$

where the algebraic coefficients are calculated as

$$a_{pi} = a_{wi} + a_{ei} - \left[\nu\Sigma_{fi} - \Sigma_{ai} - \frac{1}{\nu\Delta t} \right] \frac{h_i^2}{D_i};$$

$$a_{wi} = \frac{2}{\left(1 + \frac{D_i}{D_{i-1}} \frac{h_{i-1}}{h_i} \right)}, \qquad i \geq 2; \qquad a_{w1} = 2;$$

$$a_{ei} = \frac{2}{\left(1 + \frac{D_i}{D_{i+1}} \frac{h_{i+1}}{h_i} \right)}, \qquad i \leq N\text{-}1; \qquad a_{eN} = 2;$$

$$b_i = \frac{1}{\nu\Delta t} \frac{h_i^2}{D_i} \phi_i^t, \qquad i = 1,..,N.$$

7.1.2 Numerical Solution

The tridiagonal system (7.1.3) can be solved directly by using a direct elimination method such as the TriDiagonal Matrix Algorithm (TDMA, see in Chapter 4) and thus we, in principle, are able to obtain the *exact* solution to the algebraic system (7.1.3) (which is still not an exact solution to the original differential equation (7.1.1) but would approach it as $h \to 0$ and $\Delta t \to 0$). It is not necessary to use iterative methods for solving a tridiagonal system as long as the number of algebraic equations in the system is not so large that computer round-off errors do not spoil the direct solution. However, the use of a 1D problem as an example here would be advantageous. First of all, the number of unknowns in a 1D discretized system is not so large (for any practical reactor, one dimension of the core is likely to be divided into several hundred mesh intervals) that we would not experience a problem with memory storage of unknowns and coefficients even on a single-processor computer. Moreover, the trend of error behaviour during the iteration process for a 1D problem can be well extended to a 3D problem since in a multidimensional domain the error will change invariantly in each direction. Finally, the 1D exact solution can be found directly by using TDMA (to the computer and software decimal precision) so that we can evaluate the error norms for analysis.

It should be noted here that the algebraic system (7.1.3) may be singular or indefinite depending on values of the algebraic coefficients. To avoid singularity and indefiniteness of the system, the time step Δt should be chosen properly as addressed in Chapter 6. Since we will use a basic iterative method to solve this system, it is desirable

that the algebraic system to be solved is *diagonal dominant*. For this to hold, we must require

$$a_{pi} \geq a_{wi} + a_{ei}, \tag{7.1.4a}$$

or

$$v\Delta t \leq \min_{i=1..N}\left[\frac{1}{\max(0, v\Sigma_{fi} - \Sigma_{ai})}\right] \tag{7.1.4b}$$

The time step calculated by using (7.1.4b) is still restricted but only by the material properties and it is mesh independent.

The measure of convergence

Since we are able to obtain the exact solution to this model problem by using the TDMA as aforementioned, we can compute one of the following error norms $||e||_q$ (q is ∞, 1 or 2):

$$||e^{(m)}||_\infty = \max_{i=1..N}\left\{\left|e_i^{(m)}\right|\right\} \tag{7.1.5a}$$

$$||e^{(m)}||_1 = \frac{1}{N}\sum_{i=1}^{N}\left|e_i^{(m)}\right| \tag{7.1.5b}$$

$$||e^{(m)}||_2 = \sqrt{\frac{1}{N}\sum_{i=1}^{N}\left[e_i^{(m)}\right]^2} \tag{7.1.5c}$$

where $e^{(m)} \equiv x^{(exact)} - x^{(m)}$ is the error vector after the m-th iteration, $x^{(exact)}$ is the exact solution obtained by using the direct TDMA and $x^{(m)}$ is the m-th iterate vector obtained by using an iterative method. In practice, the exact solution is not known in general and we must use the residual norms $||r||_q$ instead of the error norms. As shown in Fig.7.1.2, both *error* norms and *residual* norms behave quite the same for a convergent basic iterative method.

As we have learned from Chapter 4, an iterative method is convergent if and only if an error norm (or residual norm) decreases with iterations. The iteration process is often terminated when the norm reduces to a given fraction ε of its initial value as

$$\frac{||\cdot||^{(m)}}{||\cdot||^{(0)}} \leq \varepsilon, \qquad 0 < \varepsilon << 1 \tag{7.1.6}$$

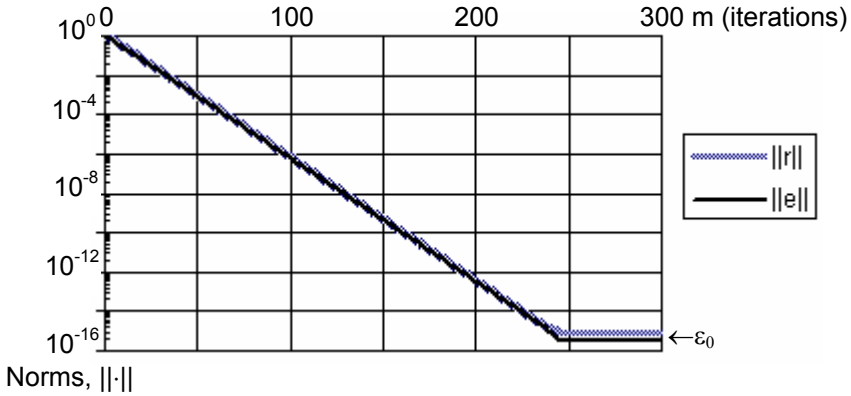

Figure 7.1.2. Norm behaviour for a convergent iterative solution

In (7.1.6), the required number of iterations (m) can be used to measure how fast the iterative method converges. Recall from Chapter 5 that it is also possible to use a contraction number by which the error (or residual) norm is reduced after each iteration step as a measure of the convergence

$$\rho_m = \frac{\|\cdot\|^{(m)}}{\|\cdot\|^{(m-1)}} \to \rho \qquad \text{as} \qquad m \to \infty \tag{7.1.7}$$

where ρ is the spectral radius of the iteration matrix. Since ρ_m is usually not constant, especially, during the beginning of the iteration process or when the total number of iterations is small as in case of a multigrid application, it is more convenient to use its average value over a number of iterations which is derived as follows

$$\overline{\rho}^m = \rho_1\rho_2\ldots\rho_m = \frac{\|\cdot\|^{(1)}}{\|\cdot\|^{(0)}}\frac{\|\cdot\|^{(2)}}{\|\cdot\|^{(1)}}\cdots\frac{\|\cdot\|^{(m)}}{\|\cdot\|^{(m-1)}} = \frac{\|\cdot\|^{(m)}}{\|\cdot\|^{(0)}} = \varepsilon$$

$$\therefore \qquad \overline{\rho} = \varepsilon^{1/m} \tag{7.1.8}$$

It can be noticed that while the pointwise norm $\|\cdot\|_\infty$ is the cheapest to compute, it, unlike the other summation norms ($\|\cdot\|_1$ or $\|\cdot\|_2$), may show an irregular trend of its change during the beginning of the iteration process. Thus, any of the summation norms that behave monotonously is more preferred. Theoretically, as $m\to\infty$, we should get $\varepsilon\to 0$. In practice, due to the computing decimal precision, ε cannot be reduced to absolute zero (0) but to some non-zero constant $\varepsilon_0 > 0$ (see in Fig. 7.1.2). The 2-norm ($\|\cdot\|_2$) is not only more expensive to compute than the 1-norm ($\|\cdot\|_1$) (as it requires additional N multiplications and taking a square root), but also its value of ε_0 somewhat increases with the number of unknowns. For example, if the value of computing precision is of order 10^{-16}, then the

value of $\|\cdot\|_2$ cannot be reduced below the square root of this value, i.e. $\varepsilon_0 > 10^{-8}$. Therefore, we will further use the 1-norm ($\|\cdot\|_1$) throughout our analysis and simply denote it as $\|\cdot\|$.

Basic iterative solution

If we use the Gauss-Seidel (GS) or successive overrelaxation (SOR) method to iteratively solve the system (7.1.3) on a computational grid with different sizes (provided the time step is properly chosen for its solution to converge), we will see from Fig.7.1.3 that the number of iterations required to reduce the error (or residual) norm of the iterate solution by $1/\varepsilon$, say 10^5, times is proportional to the grid size raised to the power of some constant $\alpha > 1$, as

$$m_\varepsilon = cN^\alpha \tag{7.1.9}$$

where c is a constant of proportionality; N is the number of algebraic equations (or the grid size); $\alpha \approx 2$ for the GS and $\alpha \approx 1$ for the SOR. That is, the convergence of basic iterative methods, as expected, is strongly mesh-dependent. Although SOR, being a GS with an optimum relaxation ω_{op}, is faster than GS by nearly an order of magnitude, it is impossible to obtain the optimum value ω_{op} in practice or it is too costly to do so. Obviously, when the grid size is relatively large, i.e. several hundreds of mesh points in 1D or tens of millions in 3D, these iterative methods are too slow to be used.

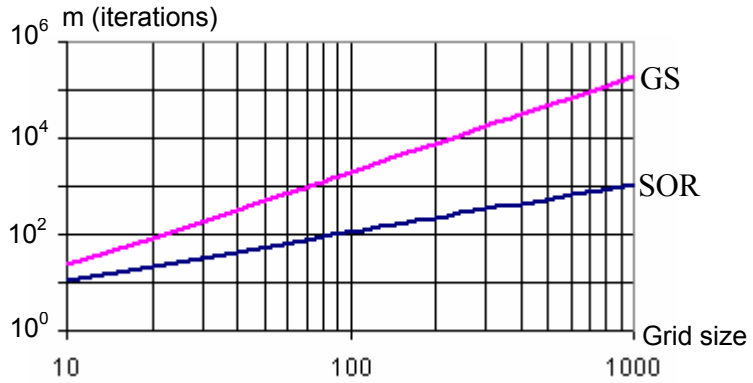

Figure 7.1.3. Required number of basic iterations to reduce the error norm by 10^5 times

Multigrid solution

We now apply a multigrid method to solve the system (7.1.3). In Fig.7.1.4 we show the results obtained by using the ACM algorithm with the simplest V(1,1)-cycle (i.e. one pre-smoothing and one post-smoothing step) and with the plain GS method as a smoother. We will allow for the multigrid solver to reach the coarsest grid that has only one non-

trivial grid node. The accuracy ε at which the multigrid iteration is terminated is set to a small value, say, 10^{-5} in one case and to the computer's precision ε_0 in the other case (i.e. the iteration is stopped as soon as the error norm no longer decreases).

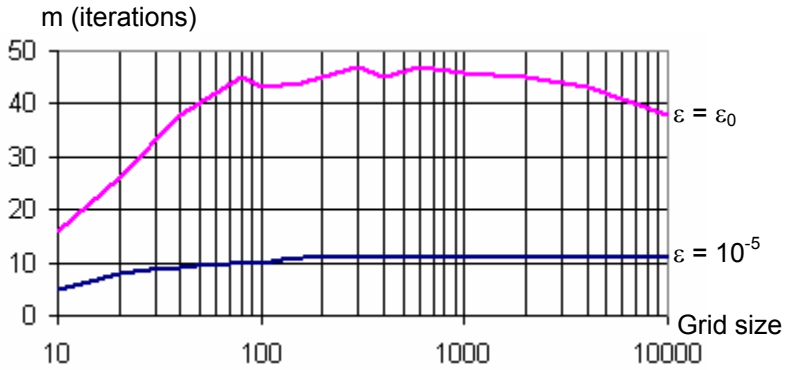

Figure 7.1.4. Required number of ACM iterations for solution to converge to within ε

At first glance, one is able to observe the spectacular performance of the multigrid in terms of convergence. That is, it requires much fewer iteration steps than GS or SOR to reduce the error norm to the same level (cf. Fig.7.1.3). Moreover and most importantly, the required number of multigrid iterations depends very weakly on the grid size and tends to be *mesh-independent at a larger grid size*. Even if we allow for the most accurate solution to be obtained, $\varepsilon = \varepsilon_0$ (i.e. when the error norm could no longer decrease with further iteration), the needed number of multigrid iterations is bounded. It is interesting to note that, when the grid size is extremely large, this iteration number tends to decrease (see Fig.7.1.4). This, however, does not mean that fewer iterations are needed for a larger grid size, but simply it is due to deterioration of precision in computing so many small values of the error norm at a very large grid size. As we can see from the ACM application (Fig.7.1.4), at small grid sizes (below 100, which is not typical for the finite-difference disretization of a practical reactor problem) its convergence is still mesh-dependent but weakly. Particularly, it is clearly seen that, regardless of the grid size when it is about several hundreds or more (that size is typically used in practical reactor calculations), the number of ACM iterations required to solve a reactor kinetics problem at each time step is *small* and *fixed*. That is, in terms of grid-size dependency for the multigrid convergence as in relationship (7.1.9), we should have

$$\alpha \le \tfrac{1}{2} \quad \text{if } N < 100$$

$$\alpha \approx 0 \quad \text{if } N \ge 100$$

(cf. $\alpha \ge 1$ for the other known unigrid iterative methods).

The fact that the ACM convergence is mesh-independent when the grid size is large can be explained as in the following. In Chapter 5 we know that a multigrid method with high order transfer operators (based on linear or more accurate interpolation) could provide the mesh independence of convergence for solving a second-order partial differential equations. With higher order restriction and prolongation, the coarse grid correction is closer to the error that must be solved for on the fine grid. For example, linear interpolation for a 1D diffusion problem discretized on a vertex-centered grid provides an exact equality between the correction at a coarse grid point and the error at the fine grid point that is projected by that coarse grid point onto the fine grid. Although the errors at the fine grid points that are not projected from any coarse grid points are not exactly corrected, the vector of newly corrected errors on the fine grid becomes so oscillated (with a half-wavelength about the fine grid mesh size) that can be removed effectively by a fixed number of basic iterations. On the other hand, the best coarse correction that the ACM, which has the lowest order of restriction and prolongation, could ever provide is only an average value of the errors in the fine grid points that form the corresponding coarse grid point. However, if the grid is fine enough, the values in any two neighbouring grid points tend to be closer to each other so that there is little difference between their average and their own values. This is the case when the fine grid has such a large number of mesh points that the ACM is able to correct the fine grid error most accurately.

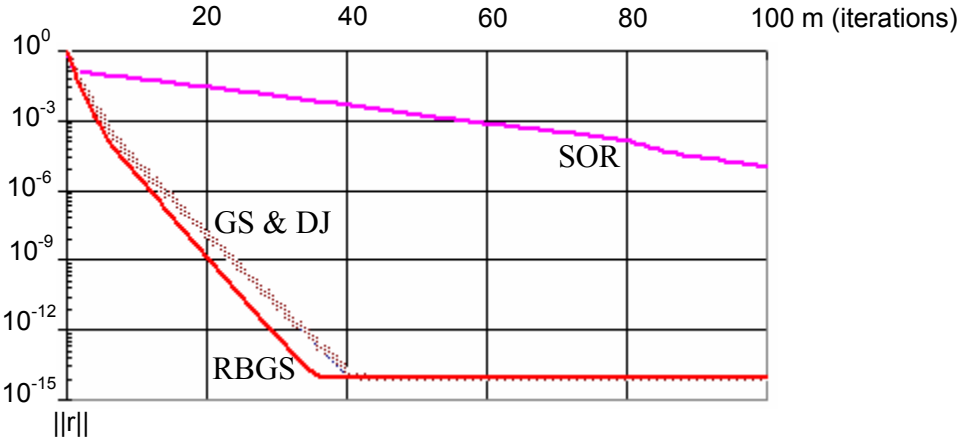

Figure 7.1.5. ACM convergence with different smoothing methods

To end this section, we show the results obtained with different smoothing methods in our ACM application. Although the SOR is the fastest method among the known basic iterative methods on a single grid, it is a rather poor smoother for multigrid application, as seen in Fig.7.1.5. On the other hand, the plain Jacobi method is also a poor smoother, but the 'damped' Jacobi (DJ) method with a properly chosen underrelaxation parameter is as good as the GS. Application of DJ, though requiring storage of one vector of unknowns more than GS or SOR, is favourable to parallel computation since it does

not require an orderly sweep through the equations in the algebraic system as GS or SOR does. It is possible, however, to reorder the equations in an algebraic system so that GS and SOR could be used to sweep though the system regardless keeping any order. The Red-Black Gauss-Seidel (RBGS) is such a method of reordering by colouring the grid mesh points red and black such that there are no two adjacent points of the same colour. The RBGS will sweep through red points first and then through black points. The RBGS is not only suitable for parallel computation but also saves the storage of one unknown vector less than DJ, as well as it is as a good smoother as DJ or GS (see in Fig. 7.1.5).

7.2 The Multigroup Kinetics Problem

In practical reactor calculations, we usually have to deal with more than one group of neutron energy (typically, 2-4 groups for thermal reactors and 15-20 groups for fast reactors). That is, besides the *spatial coupling* due to diffusion of neutrons in the same energy group, the reactor kinetics equations are also coupled through the scattering and the fission neutron sources - the *group coupling*. It is this group coupling that makes the algebraic matrix structure of the reactor kinetics discretized system difficult to manipulate if matrix multiplication or inversion is required.

7.2.1 A Two-Group Problem Example

The original equations

To examine the effect of group coupling of neutrons in solving the reactor kinetics equations, we will again consider a modelled 1D reactor problem as in the previous section but with two neutron groups, one fast group (g = 1) and one thermal group (g = 2)

$$\frac{1}{v_1}\frac{\partial}{\partial t}\phi_1(x,t) = \frac{\partial}{\partial x}D_1\frac{\partial}{\partial x}\phi_1 - \Sigma_{R1}\phi_1 + \nu\Sigma_{f2}\phi_2,$$

$$\frac{1}{v_2}\frac{\partial}{\partial t}\phi_2(x,t) = \frac{\partial}{\partial x}D_2\frac{\partial}{\partial x}\phi_2 - \Sigma_{R2}\phi_2 + \Sigma_{s12}\phi_1, \qquad (7.2.1)$$

$$x \subset [0, \tilde{a}\,], \qquad t > 0$$

Initial condition $\phi_g(x, 0) = \phi_{g0}(x)$

Boundary condition $\phi_g(0, t) = \phi_g(\tilde{a}\,, t) = 0; \quad g = 1, 2$

where the unknown functions ϕ_1 and ϕ_2 are respectively the fast and thermal neutron fluxes. In this example, we assume that the source term in the fast group equation is the fission neutrons produced by thermal fissions ($\nu\Sigma_{f2}\phi_2$), and the source term in the thermal

group equation is the neutrons scattering (i.e. slowing down) from the first group ($\Sigma_{s12}\phi_2$). All notations in (7.2.1) are standard.

Discretization

Again, we utilize the cell-centered grid (7.1.2) as in the previous one-group example for spatial discretization of (7.2.1) and the fully implicit scheme for time integration to obtain the following two-group algebraic system:

$$a_{pi}^{(1)}\phi_i^{(1)} = a_{wi}^{(1)}\phi_{i-1}^{(1)} + a_{ei}^{(1)}\phi_{i+1}^{(1)} + a_{si}^{(1)}\phi_i^{(2)} + b_i^{(1)},$$

$$a_{pi}^{(2)}\phi_i^{(2)} = a_{wi}^{(2)}\phi_{i-1}^{(2)} + a_{ei}^{(2)}\phi_{i+1}^{(2)} + a_{si}^{(2)}\phi_i^{(1)} + b_i^{(2)}, \qquad i = 1,.., N \tag{7.2.2}$$

$$\phi_0^{(1)} = \phi_{N+1}^{(1)} = 0;\ \phi_0^{(2)} = \phi_{N+1}^{(2)} = 0$$

where the algebraic coefficients are calculated as

$$a_{pi}^{g} = a_{wi}^{g} + a_{ei}^{g} - \left[\Sigma_{Ri}^{g} - \frac{1}{v_g \Delta t}\right]\frac{h_i^2}{D_i^g};$$

$$a_{wi}^{g} = \frac{2}{\left(1 + \dfrac{D_i^g}{D_{i-1}^g}\dfrac{h_{i-1}}{h_i}\right)}, \qquad i \geq 2; \qquad a_{w1}^{g} = 2;$$

$$a_{ei}^{g} = \frac{2}{\left(1 + \dfrac{D_i^g}{D_{i+1}^g}\dfrac{h_{i+1}}{h_i}\right)}, \qquad i \leq N\text{-}1; \qquad a_{eN}^{g} = 2;$$

$$a_{si}^{(1)} = \nu\Sigma_{f2}\frac{h_i^2}{D_i^{(1)}};$$

$$a_{si}^{(2)} = \nu\Sigma_{s12}\frac{h_i^2}{D_i^{(2)}};$$

$$b_i^{g} = \frac{1}{v_g \Delta t}\frac{h_i^2}{D_i^g}\phi_i^{gt}.$$

7.2.2 Numerical Solution

As noted in Chapter 6, a traditional approach to solve the group-coupled discretized system like (7.2.2) is to use the *source iteration* (or *power*) method [Wachspress (1966)], which consists of two loops of iteration: an outer loop and an inner loop. The outer (or source) iteration begins with guessing the source term in every algebraic equation

$$b_{si}^{(1)} = a_{si}^{(1)}\phi_i^{(2)} + b_i^{(1)} \quad \text{and} \quad b_{si}^{(2)} = a_{si}^{(2)}\phi_i^{(1)} + b_i^{(2)} \tag{7.2.3}$$

and reduces the group-coupled system (7.2.2) into two single-group systems

$$a_{pi}^{g}\phi_i^{g} = a_{wi}^{g}\phi_{i-1}^{g} + a_{ei}^{g}\phi_{i+1}^{g} + b_{si}^{g}, \qquad g = 1,2 \tag{7.2.4}$$

The inner iteration will solve each of the single-group systems (7.2.4) for spatial flux distribution in the group with that guessed source b_s^g. The outer iteration then improves the source terms (7.2.3) for a next inner iteration. This tandem outer-inner iteration process is repeated until the flux solution, and the source as well, converges to within some acceptable accuracy. An increase in accuracy of inner iteration would obviously reduce the required number of outer iterations for the overall solution to converge (Fig.7.2.1) but it does not necessarily reduce the computational cost as we will see shortly.

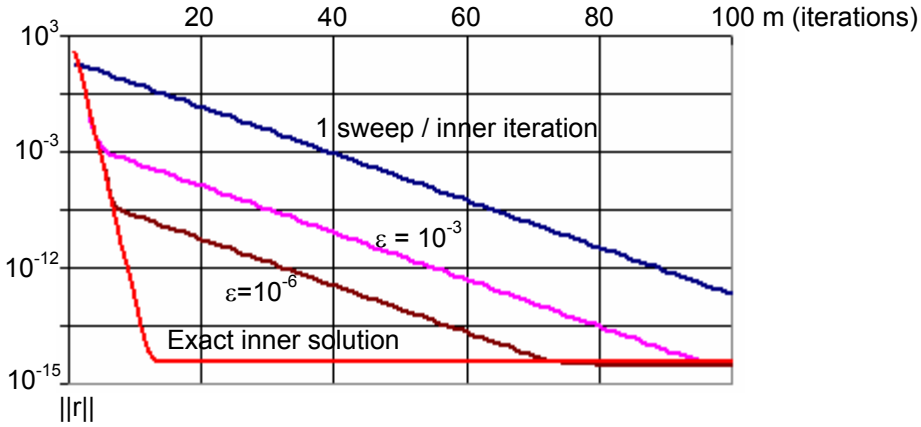

Figure 7.2.1. Source iteration at different accuracies of inner solution

With the source iteration method, a multigrid method can be straightforwardly applied to iteratively solve each single-group algebraic system (7.2.4) with the purpose of accelerating convergence of the inner iteration. Although multigrid application of this type would somewhat reduce the overall work of computation for solving the system (7.2.4), it is not the most efficient way. The reason is that to solve for spatial fluxes accurately is not only expensive but also unnecessary while the source term is still an

approximate. One could save a few steps of the outer iteration but at the greater cost spent on a more accurate solution of the algebraic systems at the beginning of the iteration process. In fact, as seen in Fig.7.2.2, although the number of outer iterations decreases, the total number of inner iterations (and so the total computational work) increases significantly with increasing accuracy of the inner solution. It is expected that during a first few outer iterations the multigrid actually works well, but it quickly becomes wasteful because after these first iterations only a couple or even one inner iteration is quite enough to have the same effect as a full multigrid cycle.

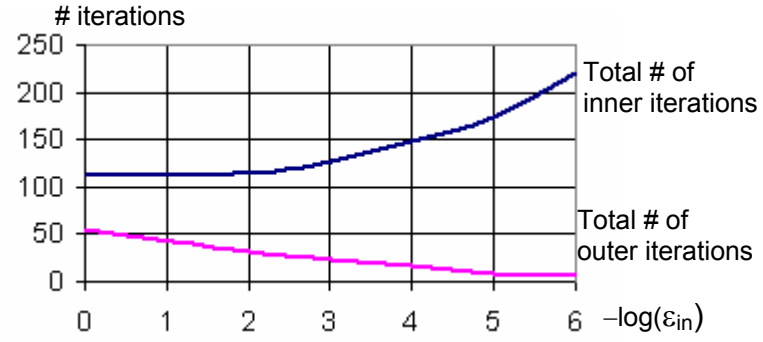

Figure 7.2.2. Iteration number vs. accuracy of the inner solution

In order to apply the multigrid more efficiently for solving the coupled system (7.2.2), we will make a clever change in the order of the iteration procedure, by moving from the source iteration (which is in the group-then-space iteration order) to the space-then-group iteration order. The idea of switching the iteration order comes from the fact that if we use the source iteration method but with a single sweep for the inner iteration, the final effect is the same as if we move from a grid point to a grid point and solve the group equations at each grid point [Garland (2001)]. This change has no gain on single-grid (*unigrid*) application but it greatly facilitates multigrid implementation. We no longer need to guess the source terms and, more importantly, the matrix structure of the algebraic equation systems is preserved on all grids, both fine and coarse. In addition, since the number of neutron groups in a practical reactor problem is small, especially for thermal reactor calculations, the direct solution of the group equation system at a grid point is not so expensive that one could even improve the overall efficiency. With this idea we proceed to a multidimensional multigroup reactor kinetics problem in the next section.

7.3 Multidimensional Multigroup Problem

We have seen from above that multigrid methods are greatly faster than any basic iterative method in reducing the solution error in the 1D reactor problem as they tend to have mesh-independent convergence with an increasing grid size. In a multidimensional problem, the error is reduced independently in each direction during the iterative solution

process. Hence, multigrid methods are expected to work well in solving the 3D reactor kinetics problem as they do in 1D. Moreover, multigrid application in 3D is even more efficient than in 1D because the number of mesh points in a 3D coarse grid is roughly 8 times fewer than in its finer related grid (it is only twice in 1D), and so is the computational work spent on sweeping through an algebraic equation system on a coarse grid. We have also observed that the ACM do not experience difficulties in handling the group coupling of the 1D kinetics equations and this is extended to the 3D case as well.

7.3.1 A 3D Multigroup Example

The starting equation system

The general reactor kinetics system of a multigroup diffusion model in 3D Cartesian geometry is given by (see in Chapter 3)

$$\frac{1}{v_g}\frac{\partial}{\partial t}\phi_g = \nabla\cdot D_g\nabla\phi_g - \Sigma_{tg}\phi_g(x,y,z,t) + \sum_{g'=1}^{G}\left[\Sigma_{sg'g} + (1-\beta)\chi_g\nu\Sigma_{fg'}\right]\phi_{g'} + \sum_{i=1}^{N}\lambda_i C_i\chi_{i,g} + S_{0g}, \qquad g = 1,..,G \tag{7.3.1a}$$

$$\frac{\partial}{\partial t}C_i = -\lambda_i C_i(x,y,z,t) + \beta_i\sum_{g'=1}^{G}\nu\Sigma_{fg'}\phi_{g'}, \qquad i = 1,..,N \tag{7.3.1b}$$

$$x\subset[0,\tilde{a}_x]; \quad y\subset[0,\tilde{a}_y]; \quad z\subset[0,\tilde{a}_z]; \quad t\geq 0$$

I.C. $\phi_g(x,y,z,0) = \phi_{0g}(x,y,z)$

B.C. $\phi_g(x_s,y,z,t) = \phi_g(x,y_s,z) = \phi_g(x,y,z_s) = 0; \quad x_s = 0|\tilde{a}_x ; y = 0|\tilde{a}_y ; z = 0|\tilde{a}_z$

Not only the group neutron fluxes ϕ_g and the delayed precursor concentrations C_i, but also the group constants D_g and $\Sigma_{(\cdot)g}$ are functions of space and time variables (x,y,z,t). The extrapolated lengths, where the diffusion model treats the neutron flux as equal to zero, are added to the physical geometry and, as a result, the computational domain is formed and its dimensions are denoted as $\tilde{a}_x, \tilde{a}_y, \tilde{a}_z$.

Discretization

By discretizing the system (7.3.1) with finite differences in a cell-centered spatial grid and with a general theta-method for time integration, and eliminating the precursors in the flux equations, we obtained the algebraic system for the neutron group fluxes at any given time step as (see Chapter 3 for details)

$$[A_P][\Phi_P] = \sum_{nb\subset P}[A_{nb}][\Phi_{nb}] + [B_p], \quad \forall P \equiv \text{an inner grid point} \tag{7.3.2}$$

where

$$[\Phi_p] = \begin{bmatrix} \phi_1^p \\ \phi_2^p \\ \dots \\ \phi_G^p \end{bmatrix}; \qquad [B_p] = \begin{bmatrix} b_1^P \\ b_2^P \\ \dots \\ b_G^P \end{bmatrix};$$

$$[A_P] = \begin{bmatrix} a_1^P - a_{11}^P & -a_{21}^P & \dots & -a_{G1}^P \\ -a_{12}^P & a_2^P - a_{22}^P & \dots & -a_{G2}^P \\ \dots & \dots & \dots & \dots \\ -a_{1G}^P & -a_{2G}^P & \dots & a_G^P - a_{GG}^P \end{bmatrix}; \qquad [A_{nb}] = \theta \begin{bmatrix} a_1^{nb} & & & \\ & a_2^{nb} & & \\ & & \dots & \\ & & & a_G^{nb} \end{bmatrix}$$

with the algebraic coefficients

$$a_g^p = \frac{1}{v_g \Delta t} + \theta\left(\sum_{nb\subset P} a_g^{nb} + \Sigma_{tg}^P\right), \quad \theta \subset [0,1];$$

$$a_{g'g}^p = \theta\left[\Sigma_{sg'g}^P + \left((1-\beta)\chi_g + \sum_{i=1}^N \frac{\theta\Delta t\lambda_i\beta_i\chi_{i,g}^d}{1+\theta\Delta t\lambda_i}\right)\nu\Sigma_{fg'}^P\right];$$

$$a_g^{nb} = 2\left(\frac{h_u^p}{D_g^p} + \frac{h_u^{nb}}{D_g^{nb}}\right)^{-1}\frac{1}{h_u^p}, \qquad u = (x, y, z);$$

$$\begin{aligned} b_g^p = {} & \left[\frac{1}{v_g\Delta t} - (1-\theta)\left(\sum_{nb\subset P} a_g^{nb} + \Sigma_{tg}^P\right)\right]\phi_g^{P,t} + (1-\theta)\sum_{nb\subset P} a_g^{nb}\phi_g^{nb,t} \\ & + (1-\theta)\sum_{g'=1}^G\left[\Sigma_{sg'g}^P + \left((1-\beta)\chi_g^p + \sum_{i=1}^N \frac{\theta\Delta t\lambda_i\beta_i\chi_{i,g}^d}{1+\theta\Delta t\lambda_i}\right)\nu\Sigma_{fg'}^P\right]\phi_{g'}^{P,t} \\ & + \sum_{i=1}^N\left[(1-\theta) + \frac{\theta - (1-\theta)\theta\Delta t\lambda_i}{1+\theta\Delta t\lambda_i}\right]\lambda_i C_i^{P,t}\chi_{i,g}^d + (1-\theta)S_{0g}^{P,t} + \theta S_{0g}^P \end{aligned}$$

The initial and boundary conditions become

$$\phi_g^{P,t=0} = \phi_{0g}^{P}, \quad \forall g, P; \qquad \phi_g^{P=B} = 0, \qquad \forall g, t > 0$$

$$C_i^{p,t=0} = \frac{\beta_i}{\lambda_i} \sum_{g'=1}^{G} \nu\Sigma_{fg'}^{p} \phi_{0g'}^{p}, \qquad \forall g, i \tag{7.3.3}$$

As we are only interested in cases where an implicit scheme (with $\theta > 0$) is used, namely, when $\theta = ½$ in the Crank-Nicholson scheme or $\theta = 1$ in the fully implicit scheme, the algebraic system (7.3.2) is required to be solved iteratively for the group fluxes at every grid point.

McMaster Nuclear Reactor

As for our numerical experiments, we will solve the kinetics equations applied for the McMaster Nuclear Reactor (MNR) core (Fig.7.3.1).

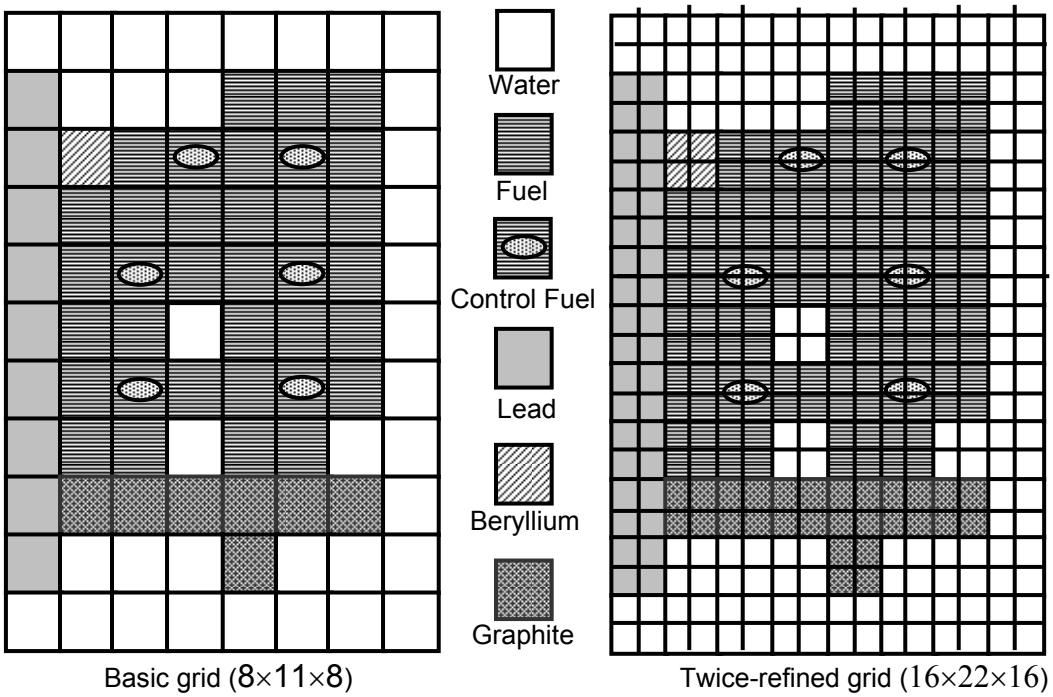

Figure 7.3.1. MNR core: horizontal plane view

Although the MNR core is small compared to any practical power reactor, its composition is quite complex as it is composed of various types of core assemblies. On the other hand, we can in principle refine the grid, i.e. use as many mesh points as we want, to form a grid of practically any size. We will refer to the grid where the upper (or lower) face of a grid node is the horizontal cross section of a core assembly as the *basic*

grid. Note that the north-south pitch and the west-east pitch of the grid can be different. The height of each grid node (or the vertical pitch of the grid) should be about either of the horizontal pitches. As a result, the basic grid is composed of a number of rectangular nodes of the same dimension.

Usually, the group constants in a core assembly are homogenized over each of its sections (e.g. a header, a body and a tail of a fuel assembly or a control-in section and control-out section of a control rod, etc.. Calculations of homogenized group constants for a specific core composition are beyond the scope of our work, and here we only use the provided results from these calculations [Day (2002)]. From the basic grid, we can form a grid of a larger size simply by dividing each dimension of the basic grid node into 2 or $4,..,2^n$ equal intervals.

7.3.2 Numerical Solution

Unigrid solution

The algebraic system (7.3.2) can be solved by using a basic iterative method on a given single grid. Usually, the GS method would be the first choice before any other methods because of its simplicity. However, for an extremely large grid, as usually used in neutron diffusion calculations, the GS would be too slow to converge since doubling the grid size would reduce the GS convergence rate by quadruple. Of course, on a single grid, any overrelaxation, i.e. $1<\omega<2$, for the GS scheme would accelerate the solution convergence to some extent, up to an order of magnitude. Not only can overrelaxation accelerate the *spatial solution* (similarly to the inner iteration) of the neutron kinetics system, it also speeds up the overall solution thanks to accelerating the *group solution* (similarly to the source iteration). The maximal acceleration could be achieved at an optimum value of the relaxation parameter ω_{op}, which, unfortunately, is impossible to compute in advance, especially, in cases where algebraic coefficients of the solving system vary from one time step to another time step.

In Fig.7.3.2 there is shown convergence behaviour of the GS and SOR methods for solving the 3D few-group neutron diffusion equations on a single computational grid - the unigrid methods (UG), in comparison with that of the multigrid methods (MG) for which these iterative methods are used as smoothers.

We have mentioned that within each grid node we can solve directly the group-coupled system as we have done for the two-group problem in the previous section. If there are only two or three neutron groups, it is possible to use Cramer's rule to directly solve such a small system at the lowest cost. But if the neutron group number is four or more, the computational cost of Cramer's direct solution would be no less than that of iterative solution. In addition and the most importantly, we can see from Fig.7.3.2 that only a single sweep through the algebraic equations within a node has exactly the same

effect as if a direct solution by the Gaussian elimination were performed. This is, again, to demonstrate our simple principle that it *is unnecessary to solve exactly an algebraic system while the source term is still approximate*.

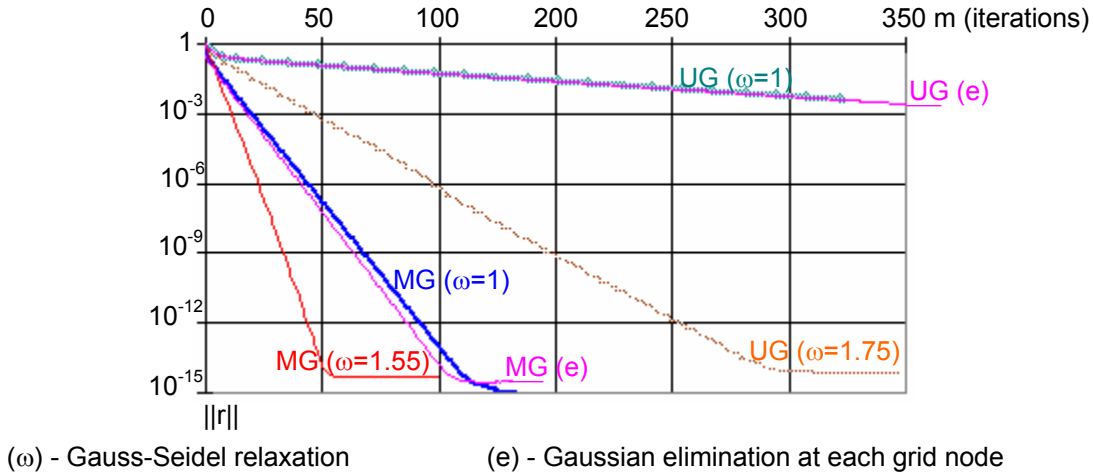

Figure 7.3.2. Unigrid and multigrid convergence behaviour on grid 32×44×32

Multigrid solution

The ACM can be easily applied to solving system (7.3.2). As for a smoothing method, we use the plain GS method or its modification, the Red-Black Gauss-Seidel method. Both GS and RBGS provide exactly the same convergence in solving our 3D few-group problem (Fig.7.3.1). In the Table 7.1 below we compare the convergence rate (which can be either the number of iterations (m) required to reduce the residual norm ||r|| from its original value by 1/ε times or the average contraction number $\bar{\rho}$) for both unigrid and multigrid schemes on the computational grid of different discretization sizes.

Table 7.3.1. Unigrid (UG) and multigrid (MG) convergence rates ($\varepsilon=10^{-5}$)

Grid		Number of nodes	UG-GS		MG-V(1,1)	
			m(ε)	$\bar{\rho}$	m(ε)	$\bar{\rho}$
Basic grid	8×11×8	704	60	0.825	13	0.410
2x-refined grid	16×22×16	5,632	183	0.939	20	0.555
4x-refined grid	32×44×32	45,056	699	0.984	36	0.774
8x-refined grid	64×88×64	360,448	~2800	0.996	55	0.810

Although the computational work per multigrid iteration step is several times greater than that per unigrid iteration step, the total cost of the multigrid iterative solution is only a fraction of the unigrid solution cost. If the same iterative method used in the

unigrid application is also used as a smoother in the multigrid application, and if the work per iteration of the former is denoted by U (this UG work unit is proportional to the number of the algebraic equations in the discretized system), then the work per iteration of the multigrid V(1,1) cycle can be estimated about $\frac{24}{7}$U, added up of

i) pre-smoothing and post-smoothing work on all grids

$$2U(1 + 2^{-d} + 2^{-2d} + \ldots + 2^{-Ld}) \approx \frac{2U}{1-2^{-d}} = \frac{16}{7}U$$

where d = 3 for 3D grid and L is the number of course grids used.

ii) restriction and prolongation work (these operations, though required for all grid nodes, consist only of simple arithmetic additions and are over-estimated as half the smoothing work) $\approx \frac{8}{7}$U.

We also have to account for the work spent on the calculation of coarse-grid algebraic coefficients, i.e. grid coarsening, which is done once at the beginning of the iteration process. With the ACM, grid coarsening consists of only arithmetic additions and is estimated as an extra iteration step. If other algebraic multigrid methods were applied, the grid coarsening work would be significantly great as complex matrix multiplications are required. These calculated multigrid costs are, in fact, overestimated, as our numerical experiments have shown that the work for multigrid coarsening is only about 0.55U and for a V(1,1) cycle is (2.2-2.6)U.

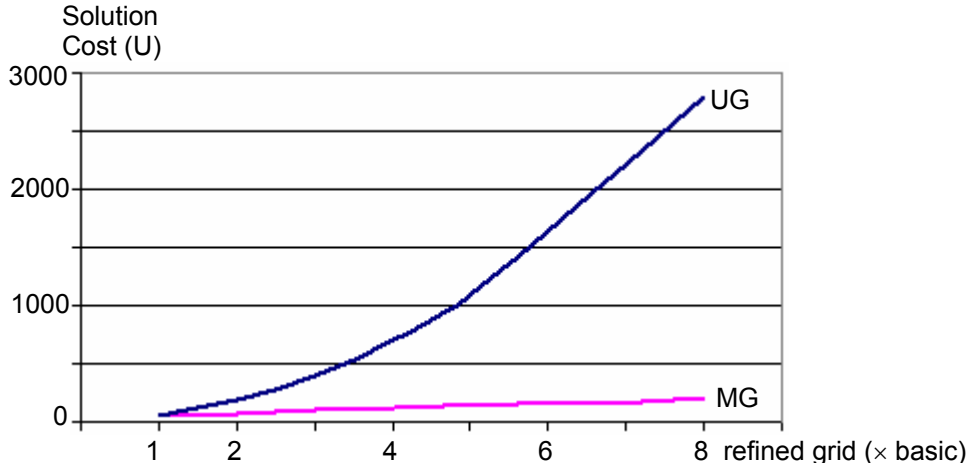

Figure 7.3.3. Comparison of unigrid and multigrid solution costs

It is clearly seen in Fig.7.3.3 that while the computational work per grid node for solving the algebraic system (7.3.2) by using the GS method on a single grid almost quadruples if the grid size doubles, such work done by the multigrid method (the ACM in our case) tends to level up at some small fraction of the former. It should be noted here that, the grid size in 3D is not the total number of grid nodes as in a 1D grid but rather the number of intervals used to divide either one of the 3D grid dimensions whichever is the largest.

It is of interest to compare the solution costs of the multigrid V and W cycles, implemented with different numbers of pre- and post-smoothing steps (Table 7.3.2).

Table 7.3.2. Solution by V and W cycles in comparison to V(1,1)

MG Cycle	(1,1)	(2,1)	(1,2)	(1,3) or (2,2)
V	1	0.96	0.95	0.97
W	0.58	0.60	0.57	0.62

Note: The solution work by the V(1,1) cycle is set to unity (1)
In parentheses are the numbers of pre- and post-smoothing steps, respectively.

As seen in Table 7.3.2, W-cycle solution is much faster than V-cycle solution, roughly twice faster than the latter. Since implementation of a W-cycle does not require any additional efforts in coding other than a V-cycle, it is always advantageous to switch to a W-cycle in any multigrid application. Also, adding either one pre- or post-smoothing step will slightly reduce the computation cost of a multigrid cycle, but the effect of post-smoothing is a little better than pre-smoothing. However, to continue adding more smoothing steps would degrade the multigrid efficiency and, therefore, should be avoided.

Remarks on smoothing methods

We further notice that, a properly chosen overrelaxation parameter $\omega > 1$ could well accelerate the multigrid convergence (Fig.7.3.2). However, the optimal value of relaxation in a multigrid application is quite different than that of SOR in a unigrid application. If the SOR ($\omega = \omega_{op}$) were used for smoothing, the final results would be worse than using the plain GS ($\omega = 1$) as already known in the above 1D one-group example, and a further increase in the relaxation parameter ($\omega > \omega_{op}$) would be likely to cause non-convergence or even divergence of the solution. On the other hand, for a multigroup neutronic problem, the overrelaxation iteration will accelerate convergence of the source terms in a similar manner as provided by the Chebyshev acceleration method [Dederstadt & Hamilton (1976)]. Therefore, an appropriate value of the multigrid overrelaxation parameter should be between 1 and ω_{op}. Because there is no general way to predict the optimal value for multigrid overrelaxation, it is safer to use the plain GS as

a smoother. A quite similar reason, but only in the opposite direction, is for the damped Jacobi not to be used as a smoother. An underrelaxation for the Jacobi scheme would accelerate smoothing of the spatial solution but, at the same time, slow down the source convergence. As a result, the damped Jacobi fails to smooth the multigrid solution of the multigroup neutronic problem.

In principle, it is possible to use any known iterative method for smoothing in multigrid application but we need to consider the overall efficiency. In multidimensional multigrid the short-wave and long-wave modes are not completely separated as in the 1D case, because the error modes are generally changed at different rates in different directions, and thus they may be short in one direction (dimension) but still long in other direction(s). It is important not to allow any modes to amplify; otherwise, the iterative method used for smoothing may not reduce the error as expected and even ruins the solution eventually (see in Chapter 5). If the solving system is indefinite, singular or nearly singular, the basic iterative methods are likely to fail as smoothers because while they reduce the short-wave modes in one direction they may amplify the long-wave modes in another. It is also very questionable to use the Conjugate Gradient (CG) method or any one of its kind in the family of Krylov subspace methods as a smoother. Such a CG-like method is known as a rougher rather than a smoother because it reduces the long-wave modes of the error faster than the short-wave modes.

Time step restriction

Up to this point we have not mentioned about how to choose time steps for temporal integration of the multidimensional multigroup kinetics equations. We have seen in the 1D one-group example that, the time step, though mesh independent, has to be restricted by the grid material properties in order to avoid singularity or indefiniteness of the discretized equation system. In the multidimensional multigroup case, such a restriction must also be imposed on the time step, but, unfortunately, we do not have a direct relation to predict a proper value of time steps. In most cases, we would rely on the fact that a value of the time step required to avoid numerical instability is usually smaller than that required for acceptable accuracy. In the worst case, by checking if the residual norm either decreases too slowly or increases, we could halt the iteration process, return to its beginning, and then repeat it with a newly reduced time step.

Flux shape

It is interesting to show how different the flux solution is for the basic grid and for one of its refined grids. In Fig.7.3.4 are shown the thermal neutron fluxes at the midplane and in several vertical positions of the MNR core, with all control rods 2/3 out. The maximum flux location is well predicted in both low and high resolution grids, but the high-resolution value is greater than the low-resolution value by 15-20%, implying the low-resolution solution is less conservative with respect to reactor safety analysis.

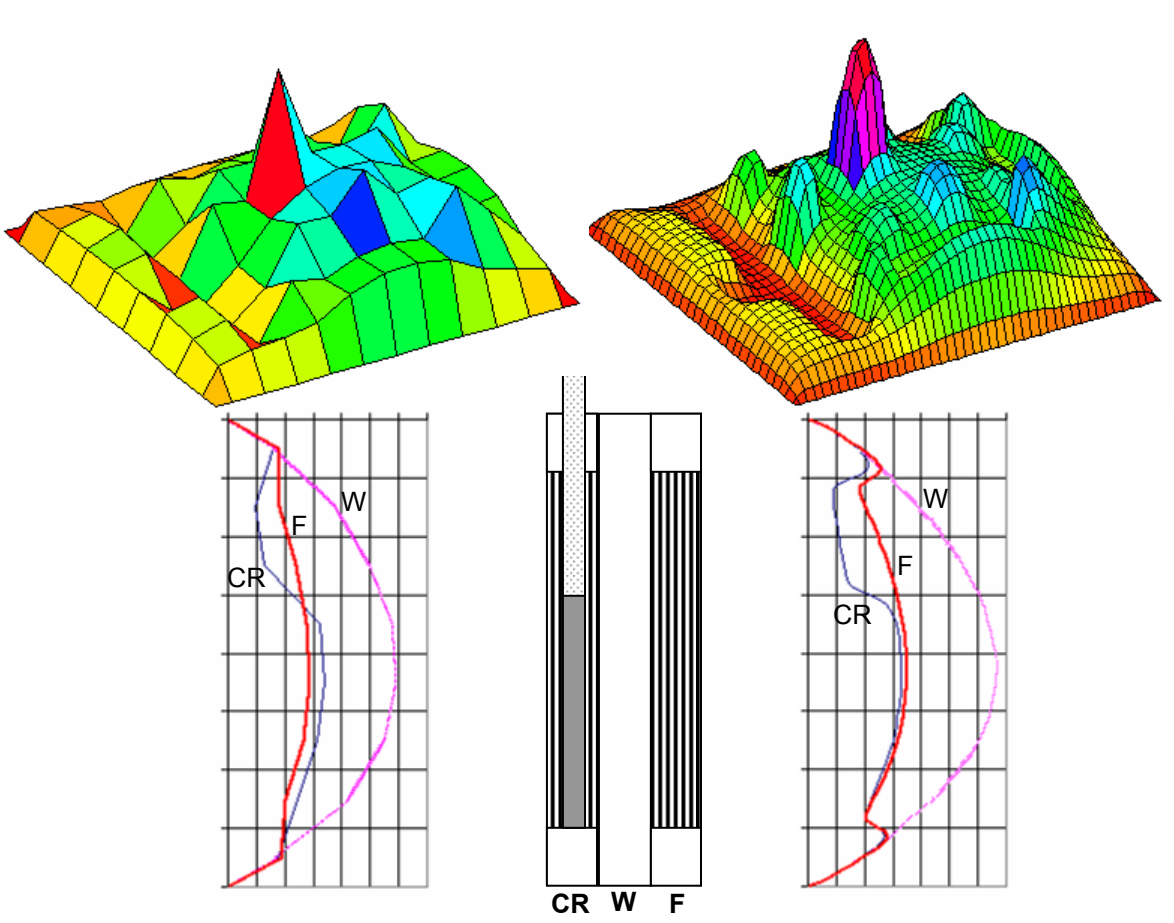

Figure 7.3.4. Thermal neutron flux in the MNR core

*
* *

The multigrid solver developed in this research approaches an optimal solution of spatial neutron kinetics problems in a nuclear reactor of practical interest, for the convergence of the solver tends to be almost *independent* of the typical mesh size of the grid used for problem discretization. Moreover, the solver is certainly the simplest method, since it utilizes the simplest transfer operators as well as the cheapest smoothing method. All of these features essentially make it a very efficient method for reactor physics calculations.

We have finally reached our goal of research and will summarize our work in the next, last chapter, together with some concluding remarks and recommendations.

Chapter 8

CONCLUSIONS

8.1 Research Summary

In this research we have explored the possibility of using a *multigrid method* for numerical solution of *spatial neutron kinetics* in nuclear reactors. We have addressed the fundamental issues that have restricted the practical use of multigrid methods to reactor physics computation, ranging from physics of reactor kinetics to numerical solution of algebraic systems, as summarized below. We have reached our goal of developing an *efficient multigrid solver* which is *computationally cheaper* but *faster* than any other spatial methods known in reactor kinetics calculations.

Numerical simulation of reactor dynamics has been recognized to be vital for control and safety analysis of nuclear reactors, in which the neutronics computation plays the most important role and usually consumes the largest portion of computational resources. Although the neutron transport theory provides the most exact and fundamental description of behaviour of neutrons in a nuclear reactor, it is extremely difficult to solve the neutron transport equation for any nuclear reactor system of practical interest. Until today and, perhaps, in the near future, the *group diffusion theory* - an approximation to the transport theory assuming the neutron as a continuum gas diffusing within a reactor core - has been and will be the most practical model to be used in reactor physics calculations, for the diffusion model is more computationally manageable while it still provides sufficiently reliable results (Chapter 2).

It is commonly agreed that kinetics of neutron population in a nuclear reactor is adequately represented by the time-dependent few-group neutron diffusion equations and the group delayed neutron precursor equations. Various numerical methods have been developed for solving this kinetics system of partial differential equations of second order in space and first order in time (Chapter 3). The point kinetics model is the simplest and fastest, but its results tend to be not only inaccurate but also non-conservative for many cases of reactor transients important to reactor safety analysis. Therefore, the use of a full 3D spatial kinetics model is necessary to obtain satisfactory results for numerical simulation of reactor dynamics.

In general, finite difference methods are the simplest and most direct approach to numerical solution of the spatial neutron kinetics equations, but they usually require such large computational resources that hardly find practical application in their straightforward manner. A finite difference method must utilize a very fine mesh grid for spatial discretization of the neutron diffusion equations to obtain acceptable accuracy, giving rise to an extremely large algebraic system of discretized equations to be solved. Most usual numerical methods are inefficient for solving such a large algebraic system.

To overcome the inefficiency of finite difference methods, a plethora of spatial methods for reactor kinetics problems have been developed, among which the family of nodal methods have found the greatest acceptance within the reactor physics community. Nodal methods utilize a relatively large mesh size in discretization, hence, allow a significant reduction in the number of equations in the nodal discretized system and, consequently, in the computational cost to an affordable degree. Nevertheless, the nodal methods are not free of difficulties and, most importantly, the nodal model computation is still very time-consuming.

Either method for discretization of reactor kinetics equations leads to an algebraic system, usually large, that must be solved by using digital computers. Direct numerical methods are normally used for solving those algebraic systems either of a small size or of a specific form such as a tridiagonal matrix system. For large algebraic systems, an iterative method must be used in favour of computational efficiency and accuracy. However, most iterative methods have a property that their convergence strongly deteriorates with an increasing size of the algebraic system being solved; therefore, these usual iterative methods are quite inefficient for solving the discretized neutron kinetics system (Chapter 4).

Multigrid methods (Chapter 5) are among the fastest iterative methods known today for solving large algebraic systems arising from disretization of partial differential equations. The essential principle of multigrid methods is to eliminate different error components on different grids of corresponding scales. A number of basic iterative methods, though simple and cheap, are so effective at damping down the short-wave components that they could be used for smoothing errors of the solution in multigrid application. Long-wave components, which are not effectively reduced by a smoother on a given grid, turn out to be short on some coarser-related grids and hence could be effectively damped down there by using the same smoothing method. The greatest advantage of multigrid applications is that the multigrid method, if properly constructed, could provide an optimal convergence that is independent of the size of the solving algebraic system.

It is possible to develop a multigrid method for numerical solution of the spatial neutron kinetics equation system (Chapter 6) provided a couple problems associated with grid coarsening and error smoothing are identified and resolved. By using an Additive Correction Multigrid (ACM) technique, whose transferring operations are based on piece-wise constant interpolation, we are able to achieve the minimal cost for grid coarsening and inter-grid transferring. Also, by adjusting the time step for temporal integration, we could avoid another problem with singularity and/or indefiniteness of the discretized system so as to use a cheap iterative method (such as the point Gauss-Seidel relaxation method) for multigrid smoothing of the error.

Numerical experiments (Chapter 7) have shown that our multigrid solver could efficiently handle the spatial multigroup neutron kinetics problem. The ACM convergence tends to be mesh-independent with an increasing size of the computational grid and approaches an *optimal iterative method* at a large grid size that is typically used in neutron diffusion calculations of a practical reactor. Even on a relatively coarse grid, although the ACM convergence is mesh-dependent, this dependence is so weak that the multigrid solver could still well compete with other spatial methods in terms of efficiency for solving the reactor kinetics problem.

8.2 Concluding Remarks

In this work we have demonstrated that multigrid methods can be well applied to reactor physics problems. In particular, we have developed a multigrid solver that is capable of solving the time-dependent few-group neutron diffusion equations in a full 3D Cartesian geometry. Our multigrid solver is shown to be an *accurate* and *efficient method* for spatial neutron kinetics, owing to *finite difference discretization* of the original kinetics equations and *multigrid iterative solution* of the discretized equation system. Some concluding remarks on the multigrid solver can be made as in the following.

1) Finite difference methods are capable of providing *accurate discretization* of the multigroup neutron diffusion equations.

A finite difference method is, no doubt, the simplest and most direct way to discretize any space-time partial differential equations. With a finite difference method, we are able to achieve any desirable accuracy of the solution simply by changing the discretization length. The discretized system resulting from finite difference discretzation on a required fine-mesh grid, though extremely large, can be effectively solved by using a multigrid method. In addition, fine-mesh utilization would allow the *direct use* of the algebraic solution of the discretized system for other related calculations, as well as it would not require the homogenization of large core regions.

2) Multigrid application provides *efficient solution* of an algebraic equation system arising from finite difference discretization.

The multigrid solver developed in this work is based on the Additive Correction Multigrid (ACM), which is the simplest and cheapest multigrid method. Unlike traditional multigrid methods, the ACM does not require knowledge of coarse grid properties (such requirements probably prevent a traditional multigrid method from being applied in reactor physics), hence, it is an algebraic multigrid method. But unlike other algebraic multigrid methods, the ACM does not involve expensive operations on algebraic matrices. The ACM restriction and prolongation are based on piece-wise constant interpolation, making it easy to handle neutron group couplings within the

reactor kinetics system. As a result, the ACM solver minimizes computational work on grid coarsening and transferring (since these are simple arithmetic additions).

An important modification to the source iteration procedure in solving the multigroup system of neutronics equations has been made, by switching from the source-then-point iteration order to the point-then-source iteration order, enabling the multigrid solver to step in both group and space at the same time and, thus, maximizing its performance.

3) The ACM solver tends to be *an optimal method* on a computational grid with an increasing size.

An optimal multigrid solver must have a convergence rate that is *independent* of computational grid size. Convergence of the ACM, due to its lowest order transferring operators, still depends on the grid size, but, fortunately, this dependence is rather weak on a coarse grid and becomes practically mesh-free on a very fine grid. Numerical experiments have shown that for a grid size that is typically used for reactor kinetics calculations the ACM convergence approaches an optimum.

Even on a very coarse computational grid, multigrid solution is still better than any unigrid method (e.g. SOR) in terms of computational efficiency, while it is only a fraction of the latter once the multigrid solver becomes optimal on very fine grids.

4) The point Gauss-Seidel method is a good choice for multigrid smoothing.

In order for a cheap basic iterative method, such as the point Gauss-Seidel relaxation method, to be used as a multigrid smoother, it is necessary to avoid singularity or indefiniteness of the discretized equation system. This can be achieved by choosing an appropriate time step for temporal integration of neutron kinetics equations. Although there exists an overrelaxation parameter at which the multigrid convergence is maximal for multigroup multidimensional problems, it is difficult or too costly to obtain such an optimal relaxation value in practice; therefore, we would rather use the *plain Gauss-Seidel method* for multigrid smoothing. It is even better to use the Red Black Gauss-Seidel method in order to facilitate parallel computation without loss of convergence speed.

5) Multigrid solution can be improved by choosing only a few pre- and post-smoothing steps as well as by switching to W-cycle implementation.

Although V(1,1)-cycle multigrid solver is the simplest, its solution is far from the best. It is possible to improve performance of the multigrid solver by a simple adjustment of multigrid parameters. That is, in our numerical experiments, a combination of one pre-

smoothing step and two post-smoothing steps, i.e. V(1,2) or W(1,2), has given the best results. Also, W-cycle multigrid solution can be twice faster than its V-cycle counterpart.

6) With our multigrid solver, we could avoid many serious problems which would arise should other spatial methods be utilized in reactor physics.

Taking, for example, the nodal methods, which are known as the most efficient methods today for reactor physics calculations, such problems could be:

- the great analytical efforts required for deriving the nodal discretized system;
- the need for homogenization of large core regions (nodes);
- the need for dehomogenization of the algebraic solution to obtain the flux distribution within a node to be used in related calculations;
- the difficulty in numerical solution of a nodal discretized system since the nodal algebraic system is not linear and diagonal-dominant;
- the need for validation of nodal results with some finite difference benchmark results.

8.3 Recommendations for Future Research

Although multigrid methods, one of which we have developed in this work, prove to be efficient for solving a diffusion modelled problem of spatial neutron kinetics in nuclear reactors, there are some issues requiring further investigation in order to maximize multigrid capability.

1) Estimation of appropriate time steps

We have identified that the time step for our multigrid solver are mesh-independent but unable to relate the time step to the reactor group constants in a general multigroup problem (we still rely on a suggestion that this time step should be larger than that required for acceptable accuracy of the solution). Further research on this issue should be directed to examining the algebraic coefficients in the discretized system for whether it is an "equivalently" diagonal-dominant system. Since a multigroup neutron model represents discretization of the energy variable, it is logical to treat the fission and scattering terms as another coupling direction *in addition* to the three spatial coupling directions. Therefore, it would be suggested that for a discretized multigroup system to be diagonal-dominant (so that its solution error could be smoothed with a point relaxation method), the algebraic coefficients of group coupling terms should satisfy certain conditions yet to be found.

2) Flux factorization to lengthen time steps

We have applied multigrid only to spatial grids because it is difficult to coarsen the temporal grid. In principle, a flux factorization technique could be used to split the group neutron flux into a flux amplitude and a flux shape. Then, the multigrid can now be applied only to the shape, while the amplitude is found by solving a much simpler system of ordinary differential equations of first order in time. This procedure does not relax our burden in solving for spatial distribution of the flux, but it is useful for lengthening the time interval required for re-computing the flux shape.

Nevertheless, it is unclear whether it is worth lengthening the time step for neutronics computation while that for the thermalhydraulics computation, which is carried out in parallel with the former, may be more restrictive.

3) Implementation of parallel computation

It can be imagined that any numerical solution of an algebraic system would have two sides, a wide and a deep. The wide side of the solution is associated with computational work spent on computing equation by equation in the algebraic system per iteration, hence, this work is directly proportional to the number of equations in the system or the system size. The deep side of the solution is associated with the total number of iterations required for solution convergence. The multigrid application actually deals with the deep side, by minimizing the required number of iterations regardless the size of the solving algebraic system.

To deal with the wide side, it is rational and necessary to implement parallel computation, by dividing the wide side work load over a number of processors at the same time. Although this aspect is beyond the scope of our work, we have carefully chosen a smoothing method for our solver such as the Red Black Gauss-Seidel method that is suited to parallel computation. We have recognized that only when both wide and deep sides of the problem solution are resolved, the real efficient method is fully developed.

The scheme of parallel computation described above would face no problem in a single grid application. However, in a multigrid application, since coarse grids have smaller and smaller sizes, there would be an issue with non-even distribution of work load over the processors on a very coarse grid (one or two processors works but many other are idle).

4) Extension to transport computation

There is nothing serious that could prevent the multigrid from being applied to the transport problem, at least for the spatial variables as in the case of multigroup diffusion

problems. The S_N method for discretization of the angular variable, along with the group treatment of the energy variable, will result in a very large algebraic system having the angular, energy and spatial couplings (again, as in the diffusion problem case, it is not reasonable to coarsen the temporal grid). If the number of elementary solid angles and the number of energy groups are both small, then multigrid methods should be as efficient for the transport problem as for the diffusion problem, in which only spatial grids are subject to coarsening. Otherwise, it would be required that an appropriate coarsening strategy be developed for the angle and energy grids as well.

REFERENCES

1. Abu-Shumays I.K. and Hageman L.A. (1975), "Development and comparison of practical discretization methods for neutron diffusion equation over general quadrilateral partitions", *Proc. Conf. Computational Methods in Nuclear Engineering*, Charleston, USA, April 1975.
2. Adams M.L. and Larsen E.W. (2002), "Fast iterative methods for discrete-ordinates particle transport calculations", *Prog. Nucl. Energy* **40**, 3-159.
3. Adams M., Brezina M., Hu J. and Tuminaro R. (2003), "Parallel multigrid smoothing: polynomial versus Gauss–Seidel", *J. Comput. Phys.* **188**, 593-610.
4. Ackroyd R.T. (1981), "A finite element method for diffusion theory embracing nodal and difference methods", *Prog. Nucl. Energy* **18**, 7-20.
5. Al-Chalabi R.M. and Turinsky P.J. (1994), "Application of multigrid method to solving the NEM form of multigroup neutron difusion equation", *Trans. Amer. Nucl. Soc.* **71**, 259-261.
6. Alcouffe R.E. and Albrecht R.W. (1970), "A generalization of the finite difference approximation method with an application to space-time nuclear reactor kinetics", *Nucl. Sci. Eng.* **39**, 1-13.
7. Alcouffe R.E., Brandt A., Dendy J.E. and Painter J.W. (1981), "The multigrid methods for the diffusion equation with strongly discontinuous coefficients, *SIAM J. Sci. Stat. Comput.* **2**, 430-454.
8. Arnone A. and Sestini A. (1991), "Multigrid heat transfer calculations using different iterative schemes", *Numer. Heat Transf. - B: Fundamental* **19**, 1-11.
9. Asahi Y. and Okumura K. (2001), "A spatial kinetics method ensuring neutron balance with thermalhydraulic feedback and its application to a main steam line break", *Nucl. Sci. Eng.* **139**, 78-95.
10. Ash M. (1979), "Nuclear reactor kinetics", McGraw-Hill, New York.
11. Axelsson O. (1994), "Iterative solution methods", Cambridge University.
12. Bakhvalov N.S. (1966), "On the convergence of a relaxation method with natural constraint on the elliptic operator", *USSR Comput. Math. Math. Phys.* **6**, 101-135.
13. Bank R.E. and Douglas C.C. (1985), "Sharp estimates for multigrid rates of convergence with general smoothing and acceleration", *SIAM J. Numer. Anal.* **22**, 617-633.
14. Behie A. and Forsyth P.A. Jr (1983), "Multi-grid solution of three dimensional problems with discontinuous coefficients", *Appl. Math. Comput.* **13**, 229-240.
15. Blanchon F., Ha-Duong T. and Planchard J. (1988), "Numerical methods for solving the reactor kinetics equations", *Prog. Nucl. Energy* **22**, 173-180.

16. Braess D. (1984), "The convergence rate of a multigrid method with Gauss-Seidel relaxation for the Poisson equation", *Math. Comput.* **42**, 505-519.

17. Bramble J., Ewing R., Pasciak J. and Shen J. (1996), "The analysis of multigrid algorithms for cell centered finite difference methods", *Adv. Comput. Math.* **5**, 15–29.

18. Brandt A. (1977), "Multilevel adaptive solution to boundary-value problems", *Math. Comput.* **31**, 333-390.

19. Brandt A. and Greenwald J. (1992), "Parabolic multigrid revisited", In: *Multigrid Methods* **III** (Hackbusch W. and Trottenbeerg U., eds.), *International Series of Numerical Mathematics* **98**, Birkhauser Verlag, Basel, 143-154.

20. Brandt A. (1994), "Rigorous quantitative analysis of multigrid - I: Constant coefficients two-level cycle with L2-norm", *SIAM J. Numer. Anal.* **31**, 1695-1730.

21. Brenner S.C. (2002), "Smoothers, mesh dependent norms, interpolation and multigrid", *Appl. Numer. Math.* **43**, 45-56.

22. Bru R., Ginestar D., Marin J., Verdu G., Mas J. and Manteuffel T. (2002), "Iterative schemes for the neutron diffusion equation", *Comput. Math. Appl.* **44**, 1307-1323.

23. Buckner M.R. and Stewart J.W. (1976), "Multidimensional space-time nuclear reactor kinetics studies. Part I: Theoretical", *Nucl. Sci. Eng.* **59**, 289-297.

24. Cavdar S. and Ozgener H.A. (2004), "A finite element/boundary element hybrid method for 2-D neutron diffusion calculations", *Ann. Nucl. Energy* **31**, 1555-1582.

25. Chan T. and Wan W. (2000), "Robust multigrid methods for nonsmooth coefficient elliptic linear systems", *J. Comput. Appl. Math.* **123**, 323-352.

26. Chang Q. and Huang Z. (2002), "Efficient algebraic multigrid algorithms and their convergence", *SIAM J. Sci. Comput.* **24**, 597-618.

27. Chen G. (1989), "The time-dependent multigroup transport equations in reactor kinetics", *Ann. Nucl. Energy* **16**, 279-291.

28. Crouzet N. and Turinsky P. (1996), "A second-derivative-base adaptive time-step method for spatial kinetics calculations", *Nucl. Sci. Eng.* **123**, 206-214.

29. Dahmani M., Baudron A.M., Lautard J.J. and Erradi L. (2001), "A 3D nodal mixed dual method for nuclear reactor kinetics with improved quasistatic model and a semi-implicit scheme to solve the precursor equations", *Ann. Nucl. Energy* **28**, 805-824.

30. Dai W. and Nassar R. (1998), "A second-order ADI scheme for three-dimensional parabolic differential equations", *Numer. Meth. PDE* **14**, 159-168.

31. Das S. (1994), "The importance of delayed neutrons in nuclear research - A review", *Prog. Nucl. Energy* **28**, 209-264.

32. Day S. (2002), Personal Communication.

33. Dendy J.E. Jr (1986), "Black box multigrid for systems", *Appl. Math. Comput.* **19**, 57-74.

34. Dendy J.E. Jr (1987), "Two multigrid methods for three-dimensional equations with highly discontinuous coefficients", *SIAM J. Sci. Sta. Comput.* **8**, 673–685.

35. Deng S., Ito K. and Li Z. (2003), "Three-dimensional elliptic solvers for interface problems and applications", *J. Comput. Phys.* **184**, 215-243.

36. Devooght J. and Mund E. (1985), "Numerical solution of neutron kinetics equations using A-stable algorithms", *Prog. Nucl. Energy* **16**, 97-126.

37. Douglas C.C. (1984), "Multigrid algorithms with applications to elliptic boundary-value problems", *SIAM J. Numer. Anal.* **21**, 236-254.

38. Douglas C.C. (1997), "Multigrid methods in science and engineering", *IEEE Comput. Sci. Eng.* **3**, 55-68.

39. Douglas C.C. (2003), "MGNet Bibliography", at *http://www.mgnet.org*.

40. Dubois G. (1975), "The importance of flux shape changes in space-time kinetics calculations", *Proc. The Joint NEACRP/CSNI Specialists' Meeting on New Development in Three-Dimensional Neutron Kinetics and Review of Kinetics Benchmark Calculations,* Garching, Germany, April 1975.

41. Duderstadt J.J. and Hamilton L.J. (1976), "Nuclear reactor analysis", John Wiley & Sons, New York.

42. Elias S.R., Stubley G.D. and Raithby G.D. (1997), "An adaptive agglomeration method for additive correction multigrid", *Int. J. Numer. Meth. Eng.* **40**, 887-903.

43. Elman H.C., Ernst O.G. and O'Leary D.P. (2001), "A multigrid method enhanced by Krylov subspace iteration for discrete Helmholtz equations", *SIAM J. Sci. Comput.* **23**, 1291-1315.

44. Engquist B. and Luo E. (1997), "Convergence of a multigrid method for elliptic equations with highly oscillatory coefficients", *SIAM J. Numer. Anal.* **34**, 2254-2273.

45. Ersland B.G. and Teigland R. (1993), "Comparison of two cell-centered multigrid schemes for problems with discontinuous coefficients", *Numer. Meth. PDE* **9**, 265-283.

46. Fedorenko R.P. (1961), "A relaxation method for solving elliptic difference equations", *USSR Comput. Math. Math. Phys.* **1**, 1092-1096.

47. Finnemann H., Bennewitz F. and Wagner M.R. (1977), "Interface current techniques for multidimensional reactor calculations", *Atomkernenergie* **30**, 123-128.

48. Finnemann H., Brehm J., Michel E. and Volkert J. (1988), "Solution of the neutron diffusion equation through multigrid methods implemented on a memory-coupled 25-processor system", *Nucl. Sci. Eng.* **100**, 226-236.

49. Fu X.D. and Cho N.Z. (2002), "Nonlinear analytic and semi-analytic nodal methods for multigroup neutron diffusion calculations", *J. Nucl. Sci. Tech.* **39**, 1015-1025.

50. Garland W.J. (2001), Personal Communication.

51. Gerard P. (1997), "MACSIM: Simulating the McMaster nuclear reactor using a distributed approach", M.Eng. Thesis, McMaster University.

52. Ginestar D., Marin J. and Verdu G. (2001), "Multilevel methods to solve the neutron diffusion equation", *Appl. Math. Model.* **25**, 463-467.

53. Gjesdal T. (1996), "A note on the additive correction multigrid method", *Int. Commun. Heat and Mass Transf.* **23**, 293-298.

54. Greenbaum A. (1984), "Analysis of a multigrid method as an iterative technique for solving linear systems", *SIAM J. Numer. Anal.* **21**, 473-485.

55. Guessous N. and Akhmouch M. (2002), "Higher order analytical nodal methods in response-matrix formulation for the multigroup neutron diffusion equations", *Ann. Nucl. Energy* **29**, 1765-1778.

56. Gupta N.K. (1981), "Nodal methods for three-dimensional simulators", *Prog. Nucl. Energy* **7**, 127-149.

57. Hackbusch W. (1982), "Multigrid convergence theory", In: *Multigrid Methods* (Hackbusch W. and Trottenbeerg U., eds.), *Lecture Notes in Mathematics* **960**, Springer, Berlin, 177-219.

58. Hackbusch W. (1984), "Parabolic multigrid methods", In: *Computing Methods in Applied Sciences and Engineering* **VI** (Glowinski R. and Lions J.L., eds.), North-Holland, Amsterdam, 189-197.

59. Hackbusch W. (1985), "Multigrid methods and applications", Springer, Berlin.

60. Hageman L.A. and Yasinsky J.B. (1969), "Comparison of alternating-direction time-differencing methods and other implicit methods for the solution of the neutron group diffusion equations", *Nucl. Sci. Eng.* **38**, 8-32.

61. Hageman L.A. and Young D.M. (1981), "Applied iterative methods", Academic Press, New York.

62. Hanna B.N. (1997), "CATHENA: A thermalhydraulic code for CANDU analysis", *Nucl. Eng. Des.* **180**, 113-31.

63. Hansen K.F. (1972), "Finite-difference solution for space-dependent kinetics equations", In: *Dynamics of Nuclear System* (Hetrick D.L., ed.), University of Arizona Press.

64. Hebert A. (1987), "Development of the nodal collocation method for solving the neutron diffusion equation", *Ann. Nucl. Energy* **14**, 527-541.

65. Hemker P.W. (1990), "On order of prolongations and restrictions in multigrid procedures", *J. Comput. Appl. Math.* **32**, 423-429.

66. Hennart J.P. (1986), "A general family of nodal schemes", *SIAM J. Sci. Stat. Comput.* **7**, 264-287.

67. Hennart J.P. (1988), "On the numerical analysis of analytical nodal methods", *Numer. Meth. PDE* **4**, 233-254.

68. Henry A.F. (1975), "Approximations that make space dependent kinetics computations more efficient", *Proc. The Joint NEACRP/CSNI Specialists' Meeting on New Development in Three-Dimensional Neutron Kinetics and Review of Kinetics Benchmark Calculations*, Garching, Germany, April 1975.

69. Henry A.F. (1980), "Nuclear reactor analysis", Massachusetts Institute of Technology.

70. Hetrick D.L. (1971), "Dynamics of nuclear reactors", University of Chicago Press.

71. Horton G. and Vandewalle S. (1995), "A space time multigrid method for parabolic partial differential equations", *SIAM J. Sci. Comput.* **16**, 848-864.

72. Hutchinson B.R. and Raithby G.D. (1986), "A multigrid method based on the additive correction strategy", *Numer. Heat Transf.* **9**, 511-537.

73. Hutchinson B.R., Galpin P.F. and Raithby G.D. (1988), "Application of additive correction multigrid to the coupled fluid flow equations", *Numer. Heat Transf.* **13**, 133-147.

74. Ikeda H. and Takeda T. (2001), "Development and verification of an efficient spatial neutron kinetics methods for reactivity-initiated event analyses", *J. Nucl. Sci. Tech.* **38**, 492-502.

75. Jackson C.J., Cacuci D.G. and Finnemann H.B. (1999), "Dimensionally adaptive neutron kinetics for multidimensional reactor safety transients - I: New features of RELAP5/PANBOX", *Nucl. Sci. Eng.* **131**, 143-163.

76. Jatuff F.E. and Gho C.J. (1999), "Time-dependent boundary source term for advanced nodal diffusion theory", *Ann. Nucl. Energy* **26**, 1065-1082.

77. Kang C.M. and Hansen K.F. (1973), "Finite element methods for reactor analysis", *Nucl. Sci. Eng.* **51**, 456-495.

78. Kang K.S. and Kwak D.Y. (1997), "Convergence estimates for multigrid algorithms", *Comput. Math. Appl.* **34**, 15-22.

79. Kaplan S., Marlowe O.J. and Bewick J. (1964), "Application of synthesis techniques to problems involving time dependence", *Nucl. Sci. Eng.* **18**, 163-176.

80. Kaveh S., Koclas J. and Roy R. (2000), "Space-time kinetics calculations using coarse-grid acceleration technique", Proc. to PHYSOS 2000, Pittsbourgh, May 2000.

81. Kavenoky A. and Lautard J.J. (1986), "New development in finite element reactor calculations", *Prog. Nucl. Energy* **18**, 3-6.

82. Kettler R. and Wesseling P. (1986), "Aspects of multigrid methods for problems in three dimensions", *Appl. Math. Comput.* **19**, 159-168.

83. Khalil M. and Wesseling P. (1991), "Vertex-centered and cell-centered multigrid for interface problems", *J. Comput. Phys.* **98**, 1-10.

84. Kinard M. and Allen E.J. (2004), "Efficient numerical solution of the point kinetics equations in nuclear reactor dynamics", *Ann. Nucl. Energy* **31**, 1039-1051.

85. Koclas J., Sissaoui M.T. and Hebert A. (1996), "Solution of the improved and generalized quasi-static methods using an analytical calculation or a semi-implicit scheme to compute the precursor equations", *Ann. Nucl. Energy* **23**, 901-907.

86. Koclas J. (1998), "Comparison of the different approximations leading to mesh centered finite differences starting from analytic nodal methods", *Ann. Nucl. Energy* **25**, 821-838.

87. Kollas J.G. and Henry A.F. (1976), "The determination of homogenized group diffusion theory parameters", *Nucl. Sci. Eng.* **60**, 464-471.

88. Kuo C.J. and Levy B.C. (1989), "Two-color Fourier analysis of the multigrid method with red-black Gauss-Seidel smoothing", *Appl. Math. Comput.* **29**, 69-87.

89. Kwak D.Y. (1997), "A preconditioned GMRES method", *Appl. Math. Comput.* **85**, 201-208.

90. Kwak D.Y. (1999), "V-cycle multigrid for cell-centered finite differences", *SIAM J. Sci. Comput.* **21**, 552-564.

91. Kwak D.Y. and Lee J.S. (2004), "Multigrid algorithm for the cell-centered finite difference method - II. Discontinuous coefficient case", *Numer. Meth. PDE* **20**, 742-764.

92. Langenbuch S., Maurer W. and Werner W. (1977), "Coarse-mesh flux-expansion method for the analysis of space-time effects in large light water reactor cores", *Nucl. Sci. Eng.* **63**, 437-456.

93. Larsson S., Thomee V. and Zhou S.Z. (1995), "On multigrid methods for parabolic problems", *J. Comput. Math.* **13**, 193-205.

94. Lawrence R.D. (1986), "Progress in nodal methods for the solution of the neutron diffusion and transport equations", *Prog. Nucl. Energy* **17**, 271-301.

95. Lewis E.E. and Miller W.F. Jr (1993), "Computational methods of neutron transport", American Nuclear Society, La Grange Park.

96. Lightstone M.F. (2002), Course notes on Computational fluid dynamics, McMaster University (Winter 2002).

97. McCormick S.F. (1982), "An algebraic interpretation of multigrid methods", *SIAM J. Numer. Anal.* **19**, 548-560.

98. Miro R., Ginestar D., Verdu G. and Hennig D. (2002), "A nodal modal method for the neutron diffusion equation. Application to BWR instabilities analysis", *Ann. Nucl. Energy* **29**, 1171-1194.

99. Mitchell A.R. and Griffiths D.F. (1980), "The finite difference method in partial differential equations", Wiley, Chichester.

100. Montagnini B., Raffaelli P., Sumini M. and Zardini D.M. (1996), "A 3D coarse-mesh time dependent code for nuclear reactor kinetic calculations", *Ann. Nucl. Energy* **23**, 517-532.

101. Moulton J.D. (1996), "Nodal methods: analysis, performance and fast iterative solvers", PhD Thesis, University of British Columbia.

102. Nguyen T.S. and Garland W.J. (2004), "A multigrid method for spatial neutron kinetics", Proc. the 25th Canadian Nuclear Society Conference, Toronto, June 2004.

103. Noh J.M. and Cho N.Z. (1994), "A new approach of analytic basis function expansion to neutron diffusion nodal calculation", *Nucl. Sci. Eng.* **116**, 165-180.

104. Nukamura S. (1997), "Computational methods in engineering and science", Wiley & Sons.

105. Oosterlee C.W. and Wienands R. (2002), "A genetic search for optimal multigrid components within a Fourier analysis setting", *SIAM J. Sci. Comput.* **24**, 924-944.

106. Ott K.O. and Meneley D.A. (1969), "Accuracy of quasistatic treatment of spatial reactor kinetics", *Nucl. Sci. Eng.* **36**, 402-411.

107. Ott K.O. and Neuhold R.J. (1985), "Nuclear reactor dynamics", American Nuclear Society, La Grange Park.

108. Ougouag A.M. and Rajic H.L. (1988), "ILLICO-HO: a self-consistent higher order coarse-mesh nodal method", *Nucl. Sci. Eng.* **100**, 332.

109. Patankar S.V. (1980), "Numerical heat transfer and fluid flow", Hemisphere, Washington D.C.

110. Penland R.C., Azmy Y.Y and Turinsky P.J. (1997), "Error analysis of the nodal expansion method for solving the neutron diffusion equation", *Nucl. Sci. Eng.* **125**, 284-299.

111. Phillips R.E. and Schmidt F.W. (1984), "Multigrid techniques for numerical solution of the diffusion equation", *Numer. Heat Transf.* **7**, 251-268.

112. Phillips T.N. (1987), "The smoothing properties of the alternating direction implicit method in multigrid iterations", *Appl. Numer. Math.* **3**, 513-522.

113. Planchard J. (1991), "On the point-reactor kinetics approximation", *Prog. Nucl. Energy* **26**, 207-216.

114. RELAP5 Code Development Team (1995), "RELAP5/MODE3 Code manual", NUREG/CR-5535.

115. Reusken A. (1994), "On maximum norm convergence of multigrid methods for elliptic boundary value problems", *SIAM J. Numer. Anal.* **31**, 378-392.

116. Saad Y. and Schultz M.H. (1986), "GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems", *SIAM J. Sci. Stat. Comput.* **7**, 856-869

117. Saad Y. (1996), "Iterative methods for sparse linear systems", *The Pws Series in Computer Science*.

118. Saad Y. and van der Vorst H.A. (2000), "Iterative solution of linear systems in the 20th century", *J. Comput. Appl. Math.* **123**, 1-33**.**

119. Sanchez J. (1989), "On the numerical solution of the point kinetics equations by generalized Runge-Kutta methods", *Nucl. Sci. Eng.* **103**, 94–99.

120. Schaffer S. (1998), "A semi-coarsening multigrid method for elliptic partial differential equations with highly discontinuous and anisotropic coefficients", *SIAM J. Sci. Comput.* **20**, 228-242.

121. Scheilchl R. (2000), "Parallel solvers for transient multigroup neutron diffusion equations", *Int. J. Numer. Methods Eng.* **47**, 1751-1771.

122. Schmidt F. and Fremd R. (1981), "Experience in using the finite element method for reactor calculations", *Ann. Nucl. Energy* **8**, 567-580.

123. Shapira Y. (1988), "Multigrid methods for 3-D definite and indefinite problems", *Appl. Numer. Math.* **26**, 377-398.

124. Shapira Y. (1997), "Note on the multigrid W-cycle", *J. Comput. Appl. Math.* **85**, 351-353.

125. Shober R.A., Sims R.N. and Henry A.F. (1977), "Two nodal methods for solving time-dependent two-dimensional diffusion problems", *Nucl. Sci. Eng.* **64**, 582-592.

126. Smith K.S. (1986), "Assembly homogenization techniques for LWR analysis", *Prog. Nucl. Energy* **17**, 303-335.

127. Song J.W. and Kim J.K. (1993), "An efficient nodal method for transient calculations in light water reactors", *Nucl. Tech.* **103**, 157–167.

128. Stacey W.M. Jr. (1969), "Space-time nuclear reactor kinetics", Academic Press, New York.

129. Stacey W.M. (1971), "Space- and energy-dependent neutronics in reactor transient analysis", *Reactor Technology* **14**, 169-197.

130. Stuben K. (2001), "A review of algebraic multigrid", *J. Comput. Appl. Math.* **128**, 281-309.

131. Sutton T.M. and Aviles B.N. (1996), "Diffusion theory methods for spatial kinetics calculations", *Prog. Nucl. Energy* **30**, 119-182.

132. Taiwo T.A., Khalil H.S., Cahalan J.E. and Morris E.E. (1993), "Time-step selection considerations in analysis of reactor transients with DIF3D-K", *Trans. Am. Nucl. Soc.* **68**, 429-430.

133. Takeda T. and Saji E. (1980), "Application of improved coarse mesh method to BWR core calculations", *J. Nucl. Sci. Tech.* **18**, 150.

134. Teigland R. (1998), "On some variational acceleration techniques and related methods for local refinement", *Int. J. Numer. Meth. Fluids* **28**, 945 - 960.

135. Thole C. and Trottenberg U. (1986), "Basic smoothing procedures for the multigrid treatment of elliptic 3D-operators", *Appl. Math. Comput.* **19**, 333-345.

136. van der Vorst H.A. (2000), "Krylov subspace iteration", *IEEE Comput. Sci. Eng.* **2**, 32-37.

137. Verdu G., Ginestar D., Vidal V. and Munoz-Cobo J.L. (1995), "A consistent multidimensional nodal method for transient calculations", *Ann. Nucl. Energy* **22**, 395–410.

138. Wachspress E.L. (1966), "Iterative solution of elliptic systems and applications to the neutron diffusion equations of reactor physics", Prentice-Hall, Eaglewood Cliffs.

139. Walters W.F. (1986), "The relation between finite element methods and nodal methods in transport theory", *Prog. Nucl. Energy* **18**, 21-26.

140. Wesseling P. (1992), "An introduction to multigrid methods", John Wiley & Sons, Chichester.

141. Wesseling P. and Oosterlee C.W. (2001), "Geometric multigrid with applications to computational fluid dynamics", *J. Comput. Appl. Math.* **128**, 311-334.

142. Weston J.R. and Stacey M. (1970), "Space-time nuclear reactor theory", Academic Press, New York.

143. Wienands R. and Oosterlee C.W. (2001), "On three-grid Fourier analysis for multigrid", *SIAM J. Sci. Comput.* **23**, 651-671.

144. Whitlock J. (1991), "Comparison of nodal and finite-difference methods for solving the steady-state multigroup neutron diffusion equation", M.Eng. Thesis, McMaster University.

145. Xu J. (2001), "The method of subspace corrections", *J. Comput. Appl. Math.* **128**, 335-362.

146. Yasinsky J.B. and Henry A.F. (1965), "Some numerical experiments concerning space-time reactor kinetics behaviour", *Nucl. Sci. Eng.* **22**, 171-181.

147. Yavneh I. (1995), "Multigrid smoothing factors for red-black Gauss-Seidel applied to a class of elliptic operators", *SIAM J. Numer. Anal.* **32**, 1126–1138.

148. Yavneh I. (1996), "On red-black SOR smoothing in multigrid", *SIAM J. Sci. Comput.* **17**,180–192.

149. Young D.M. and Gregory R.T. (1973), "A survey of numerical mathematics", Volume II, Addison-Wesley.

150. Zhang J. (1998), "Two-grid analysis of minimal residual smoothing as a multigrid acceleration technique", *Appl. Math. Comput.* **96**, 27-45.

151. Zhang J. (2000), "Preconditioned iterative methods and finite difference schemes for convection–diffusion", *Appl. Math. Comput.* **109**, 11-30.

152. Zhang H., Rizwan-Udin and Dorning J.J. (1995), "Systematic homogenization and self-consistent flux and pin power reconstruction for nodal diffusion methods - I: Diffusion equation-based theory", *Nucl. Sci. Eng.* **121**, 226-244.

153. Zaslavsky L.Y. (1993), "An adaptive algebraic multigrid for multigroup neutron diffusion reactor core calculations", *Appl. Math. Comput.* **53**, 13-26.

154. Zaslavsky L.Y. (1995), "An adaptive algebraic multigrid for reactor criticality calculations", *SIAM J. Sci. Comput.* **16**, 840-847.

155. Zeng S. and Wesseling P. (1994), "Multigrid solution of the compressible Navier-Stokes equations in general coordinates", SIAM J. Numer. Anal. 31, 1764-1784.

156. Zimin V.G. and Ninokata H. (1996), "Acceleration of the outer iterations of the space-dependent neutron kinetics equations solution", *Ann. Nucl. Energy* **23**, 1407–1420.

157. Zimin V.G. and Ninokata H. (1998), "Nodal neutron kinetics model based on nonlinear iteration procedure for LWR analysis", *Ann. Nucl. Energy* **25**, 507-528.

158. Zimin V.G. and Baturin D.M. (2002), "Polynomial nodal method for solving neutron diffusion equations in hexagonal-z geometry", *Ann. Nucl. Energy* **29**, 1105-1117.

Appendix

CODE LISTINGS

A1. Example of Input File for MNR Kinetics Simulation

```
File:       INPUT.DAT
[Delayed group constants]
N           6
betta       0.0069
lambda      0.0129  0.0311  0.134  0.331  1.26   3.21
gamma       0.038   0.213   0.188  0.407  0.128  0.026
[Material Group Constants]
G           4
inv         3.78E-10    7.60E-09    6.36E-07    4.55E-06
kappa       0.872261    0.127738297 1.18177E-08 0
kappad      0.089156    0.909340    0.001504    0
Materials   6
0     Water
SIGMAa      0.000356541 0.000159863 0.002433038 0.018564954
nuSIGMAf    3.27382E-10 2.86001E-09 9.06383E-09 1.15812E-07
SIGMAt      0.183534328 0.538618188 0.644639481 2.295133858
SIGMAs      0.082247006 0.10092893  1.84402E-06 0
            0           0.435985711 0.096540588 0.005932247
            0           0           0.223166261 0.431198693
            0           0           8.79444E-05 2.27647925
D           1.869083523 0.622768342 0.512394651 0.148033356
1     Fuel
SIGMAa      0.000910669 0.004386477 0.01559947  0.072668247
nuSIGMAf    0.001560095 0.003867626 0.01299741  0.119888635
SIGMAt      0.173860801 0.35794341  0.41469881  1.33177145
SIGMAs      0.102579408 0.070369703 1.24218E-06 8.1352E-14
            0           0.294979139 0.055193996 0.003383923
            0           0           0.126701404 0.272397617
            0           0           0.000348934 1.258752323
D           1.942543661 0.942493601 0.804173988 0.279613014
2     CR-in
SSIGMAa     0.000851411 0.009847043 0.08019748  0.100738256
nuSIGMAf    0.00102181  0.002077098 0.007410718 0.083439786
SIGMAt      0.137540725 0.344930327 0.42699934  1.232308882
SIGMAsg     0.072580467 0.064108023 1.11916E-06 8.06297E-12
            0           0.277684483 0.054084575 0.003314541
            0           0           0.09678535  0.25001615
            0           0           0.000433534 1.131138887
D           2.512030834 0.984157622 0.78101611  0.303849184
3     CR-out
SIGMAa      0.000695568 0.002444313 0.009230591 0.04531265
nuSIGMAf    0.001018259 0.002013301 0.006544246 0.060412812
SIGMAt      0.1673375   0.361138845 0.426232532 1.59150838
SIGMAs      0.095620875 0.071019738 1.25759E-06 5.56254E-14
            0           0.294591503 0.060398905 0.003703777
            0           0           0.127404708 0.289596717
            0           0           0.000214083 1.545980679
D           2.02321137  0.937097341 0.782493463 0.22942734
4     Lead
SIGMAa      0.000111899 0.000347981 0.000593734 0.002480705
nuSIGMAf    2.9732E-10  2.31383E-09 9.02311E-09 5.37142E-08
SIGMAt      0.155943603 0.329770693 0.367788196 0.370410927
```

```
SIGMAs      0.151876825 0.003954283 8.54707E-10 5.22851E-11
            0           0.329230487 0.000191896 0
            0           0           0.365230583 0.001969883
            0           0           0.002172028 0.365758379
D           2.138079992 1.025304164 0.906303466 0.899917141
5     Graphite
SIGMAa      0.000109509 0.000633526 0.00163672  0.235402784
nuSIGMAf    3.09693E-10 8.38279E-10 8.31551E-09 1.79773E-08
SIGMAt      0.047145073 0.044605511 0.035227811 0.259437079
SIGMAs      0.042328875 0.004706692 3.27388E-08 1.13886E-10
            0           0.043970495 1.46175E-06 0
            0           0           0.029033424 2.12311E-07
            0           0           0.001179971 0.022854296
D           7.136363436 7.664834066 10.8868435  1.289991584
100 [End of material constants]
[Assembly/Cell Structure]
Dimensions  7.7   8.1   80
Assemblies  5
0 W 1 80 0
1 F 3 10 0 60 1 10 0
2 C 4 10 0 20 2 40 3 10 0
3 L 1 80 4
4 G 1 80 5
100 [End of assembly structure]
[Core composition]
Sizes 8 11 8 4
   0 1 2 3 4 5 6 7
 0 W W W W W W W W
 1 L W W F F F W W
 2 L G F C F C W W
 3 L F F F F F F W
 4 L F C F F C F W
 5 L F F W F F F W
 6 L F C F F C F W
 7 L W F W F F W W
 8 L G G G G G G W
 9 L W W W G W W W
10 W W W W W W W W
100 [End of core composition]
[Initial conditions]
node# 0     0     0
s     0     0     0     0
phi   0     0     0     1
c     0     0     0     0     0     0
[End of ititial conditions]
```

A2. MNR Kinetics Simulation

File: MNRKIN.CPP

```
#include <math>
#include <fstream>
#include <conio>

#include "param.h"
#include "grid.h"

main()
{
Grid f,*cg,**CG;              //declare fine and coarse grids

ifstream fin("input.dat");    //open a file for input
f.Input(fin);                 //reading input

ofstream fout("output.dat");  //open a file for output

int I,J,K; f.Get(I,J,K);      //fine grid size
int Imax=I; if(Imax<J) Imax=J; if(Imax<K) Imax=K;     //max dimension
int Lmax= ceil(log(Imax)/log(2));   //max# of grid levels
CG=new Grid*[Lmax+1];         //pointers to grids
CG[0]=&f;                     //first coarse grid = fine grid

//Set up coarse grids
if(Lmax){
      cg=new Grid[Lmax];
      for(int lvl=0;lvl<Lmax;lvl++){
            I=(I+1)/2; J=(J+1)/2; K=(K+1)/2; //grid sizes
            cg[lvl].Set(I,J,K);
            CG[lvl+1]=&cg[lvl];
      }
}
f.Get(I,J,K);                 //restore I,J,K of the fine grid

Parameter Rsd, Rsd1;
double t=0, tend=3600, dt=1e-4;     //times in sec
double dphi, epso=1e-5, omega=1.;   //iteration parameters
int m1=1, m2=1, mc=1;               //MG-cycle parameters
int itos=MAXINT; if(epso>=1) itos=int(epso);//#iterations

//Time advancement
while(t<=tend){t+=dt;   //time increment

      {
      //Group constant adjustment
      //based on changes in temperature, composition or control
      }

      f.StorePhi();     //store old fluxes
      f.UpdateAB(dt);   //update fine grid equations
```

```
      for(int lvl=0;lvl<Lmax;lvl++){//generate coarse grid matrices
            CG[lvl]->TransferA(*CG[lvl+1]);
      }

      Rsd1=f.Residual();      //compute initial residual

      for(int ito=1;ito<=itos;ito++){//iterative solution for fluxes

            MG(CG, Lmax, 0, m1, m2, mc, omega); //call MG solver

            Rsd=f.Residual();       //compute residual
            if(epso<1 && Rsd(1)/Rsd1(1)<=epso) break; //stop iteration
      }

      f.UpdateC(dt);    //compute precursors

      {
      //Send flux values as input for computation of other processes
      //i.e. thermalhydraulics, poisoning, burnup, etc.
      }

}//t>tend

}
```

A3. The Kinetics Module: Classes and Methods

File: GRID.H

```
#ifndef GRID_H
#define GRID_H

#include <math>
#include <values>
#include "param.h"
#include "node.h"

//MG solver
double MG(Grid **C, int Lmax, int lvl,
            int m1, int m2, int mc, double omega);//declare

double MG(Grid **C, int Lmax, int lvl,
            int m1, int m2, int mc, double omega)
{
double dphi;
if(lvl==Lmax) dphi=C[lvl]->Gauss();//solve on coarsest grid
else{
      if(m1>0) dphi=C[lvl]->PGS(m1,omega);//pre-smoothing
      C[lvl]->UpdateRsd();                //calculate residuals
      C[lvl]->Restriction(*C[lvl+1]);     //transfer to coarse grid
      for(int i=1; i<=n; i++)
            MG(C, Lmax, lvl+1, m1, m2, n,omega);      //recursive call
      C[lvl]->CorrectPhi(*C[lvl+1]);      //correct solution
      if(m2>0) dphi=C[lvl]->PGS(m2,omega);//post-smoothing
}
return dphi;
}

//Grid structure
class Grid{
      int   I,    //#west-east columns
            J,    //#north-south rows
            K;    //#top-bottom layers
      int div;                //#cell divisor
      Material *material;     //material group constants
      Cell **pos;             //core composition

      public:
      Node *nnode;            //array of grid nodes
      Node null;              //Null node (boundary)

      //Methods
      Grid(int I=1, int J=1, int K=1);    //constructor
      ~Grid(){delete[]nnode;}             //destructor
      void Set(int I, int J, int K);      //set size
      void Get(int&I, int&J, int&K);      //get size
      Node& node(int n);
      Node& node(int i, int j, int k);
```

```
      double PGS(double epsi=1, double omega=1);
      double RBGS(double epsi=1, double omega=1);
      double Gauss();
      int TransferA(Grid& cg);
      int TransferRsd(Grid& cg);
      int Restriction(Grid& cg, int mode=0);
      int CorrectPhi(Grid& cg, int mode=0);
      void StorePhi();
      void UpdateAB(double dt);
      void UpdateC(double dt);
      Parameter Residual();
      double MaxRsd();
      void UpdateRsd();
      int Init(Node& n,
                  int i1, int i2, int j1, int j2, int k1, int k2);
      void Input(istream& is);
};

//Set up a grid with indexed nodes
Grid::Grid(int II, int JJ, int KK)
{
      I=II; J=JJ; K=KK;
      nnode = new Node[I*J*K];
      if(!nnode){cerr<<"Grid allocation error";}

      int i,j,k,nn;
      for(int n=1; n<=I*J*K; n++){
            nn=n-1; k=nn/I/J+1;
            nn-=(k-1)*I*J; j=nn/I+1;
            i=nn-(j-1)*I+1;
            node(n).SetIdx(i,j,k);
      }
      null.SetIdx(0,0,0);
      null.SetDim(0,0,0);
      Node *p=&null;
      for(k=1;k<=K;k++)
      for(j=1;j<=J;j++)
            for(i=1;i<=I;i++){
                  node(i,j,k).SetNb(west,i>1? node(i-1,j,k):*p);
                  node(i,j,k).SetNb(east,i<I? node(i+1,j,k):*p);
                  node(i,j,k).SetNb(north,j>1? node(i,j-1,k):*p);
                  node(i,j,k).SetNb(south,j<J? node(i,j+1,k):*p);
                  node(i,j,k).SetNb(up,k>1? node(i,j,k-1):*p);
                  node(i,j,k).SetNb(down,k<K? node(i,j,k+1):*p);
            }
}

void Grid::Set(int II, int JJ, int KK)
{
      if(I==II&&J==JJ&&K==KK) return;

      if(I*J*K!=II*JJ*KK){
```

```
            delete[]nnode;
            nnode=0;
            nnode=new Node[II*JJ*KK];
            if(!nnode){cerr<<"Grid allocation error";}
      }
      I=II; J=JJ; K=KK;
      int i,j,k,nn;
      for(int n=1; n<=I*J*K; n++){
            nn=n-1; k=nn/I/J+1;
            nn-=(k-1)*I*J; j=nn/I+1;
            i=nn-(j-1)*I+1;
            node(n).SetIdx(i,j,k);
      }
      null.SetIdx(0,0,0);
      null.SetDim(0,0,0);
      Node *p=&null;
      for(k=1;k<=K;k++)
      for(j=1;j<=J;j++)
            for(i=1;i<=I;i++){
                  node(i,j,k).SetNb(west,i>1? node(i-1,j,k):*p);
                  node(i,j,k).SetNb(east,i<I? node(i+1,j,k):*p);
                  node(i,j,k).SetNb(north,j>1? node(i,j-1,k):*p);
                  node(i,j,k).SetNb(south,j<J? node(i,j+1,k):*p);
                  node(i,j,k).SetNb(up,k>1? node(i,j,k-1): *p);
                  node(i,j,k).SetNb(down,k<K? node(i,j,k+1):*p);
            }
}

void Grid::Get(int& II, int& JJ, int& KK)
{
      II=I; JJ=J; KK=K;
}

//Node reference by single or triple index
Node& Grid::node(int n)
{
      if(n<1||n>I*J*K) return null;
      return nnode[n-1];
}

Node& Grid::node(int i, int j, int k)
{
      if(i<=0||j<=0||k<=0||i>I||j>J||k>k) return null;
      int n=(k-1)*I*J+(j-1)*I+i;
      return node(n);
}

//Point Gauss-Seidel smoothing
double Grid::PGS(double epsi, double omega)
{
      int itis=MAXINT; if(epsi>=1) itis=int(epsi);
      if(omega<0||omega>2) itis=1;
```

```
	double dif,dphi;
	int IJK=I*J*K;
	for(int iti=1;iti<=itis;iti++){	//iteration
		dphi=0;
		for(int n=1;n<=IJK;n++){	//sweep
			dif=node(n).UpdatePhi(omega);
			if(fabs(dif)>fabs(dphi)) dphi=dif;
		}
	if(epsi<1 && fabs(dphi)<=epsi) break;//stopping check
	}
	return dphi;
}

//Red-Black Gauss-Seidel smoothing
double Grid::RBGS(double epsi, double omega)
{
	int itis=MAXINT; if(epsi>=1) itis=int(epsi);
	if(omega<0||omega>2) itis=1;
	double dif,dphi;
	int IJK=I*J*K;
	for(int iti=1;iti<=itis;iti++){
	//Red nodes swept through
		for(int n=1;n<=IJK;n++){
			if(node(n).IsRed()) node(n).UpdatePhi(omega);
		}
	//Black nodes swept through
		dphi=0;
		for(int n=1;n<=IJK;n++){
			if(!node(n).IsRed()){
				dif=node(n).UpdatePhi(omega);
				if(fabs(dif)>fabs(dphi)) dphi=dif;
			}
		}
	if(epsi<1 && fabs(dphi)<=epsi) break;	//stopping check
	}
	return dphi;
}

//Direct Gauss elimination
double Grid::Gauss()
{
	double dif,dphi=0;
	int IJK=I*J*K;
	for(int n=1;n<=IJK;n++){
		dif=node(n).GUpdatePhi();
		if(fabs(dif)>fabs(dphi)) dphi=dif;
	}
	return dphi;
}

//Grid coarsening
int Grid::TransferA(Grid& cg)
```

```
{
      if(I==1&&J==1&&K==1) return 0;
      int I1,J1,K1; cg.Get(I1,J1,K1);
      if(I1<(I+1)/2||J1<(J+1)/2||K1<(K+1)/2) return 0;

      //Set coarse grid coefficients to zeros
      int i1,j1,k1;
      int i,j,k;
      for(k1=1;k1<=K1;k1++)
            for(j1=1;j1<=J1;j1++)
            for(i1=1;i1<=I1;i1++){
                  cg.node(i1,j1,k1).ap=node(1).ap*0;
                  cg.node(i1,j1,k1).anb=node(1).anb*0;
                  cg.node(i1,j1,k1).bp=node(1).bp*0;
                  cg.node(i1,j1,k1).as=node(1).as*0;
                  cg.node(i1,j1,k1).phi=node(1).phi*0;
            }
            cg.null.phi=null.phi*0;

      //Coarsening matrix A
      int isw, isn, isu;
      for(k=1;k<=K;k++){ k1=(k+1)/2; isu=k%2;
      for(j=1;j<=J;j++){ j1=(j+1)/2; isn=j%2;
      for(i=1;i<=I;i++){ 1=(i+1)/2; isw=i%2;

      for(int g=1;g<=Group::G;g++){

      //ap
      cg.node(i1,j1,k1).ap(g)+=node(i,j,k).ap(g)
        -(isw? (i<I? node(i,j,k).anb(e,g):0):node(i,j,k).anb(w,g))
        -(isn? (j<J? node(i,j,k).anb(s,g):0):node(i,j,k).anb(n,g))
        -(isu? (k<K? node(i,j,k).anb(d,g):0):node(i,j,k).anb(u,g));
      //anb
      if(isw&&i>1) cg.node(i1,j1,k1).anb(w,g)+=node(i,j,k).anb(w,g);
      if(!isw&&i<I)cg.node(i1,j1,k1).anb(e,g)+=node(i,j,k).anb(e,g);
      if(isn&&j>1) cg.node(i1,j1,k1).anb(n,g)+=node(i,j,k).anb(n,g);
      if(!isn&&j<J)cg.node(i1,j1,k1).anb(s,g)+=node(i,j,k).anb(s,g);
      if(isu&&k>1) cg.node(i1,j1,k1).anb(u,g)+=node(i,j,k).anb(u,g);
      if(!isu&&k<K)cg.node(i1,j1,k1).anb(d,g)+=node(i,j,k).anb(d,g);
      //as
      for(int gp=1;gp<=Group::G;gp++)
      if(gp-g) cg.node(i1,j1,k1).as(gp,g)+=node(i,j,k).as(gp,g);

      }//g-loop end

      }//i-loop end
      }//j-loop end
      }//k-loop end
      return 1;
}

//Restriction
```

```
int Grid::TransferRsd(Grid& cg)
{
      int i1,j1,k1;
      int i,j,k;

      for(k=1;k<=K;k++){ k1=(k+1)/2;
      for(j=1;j<=J;j++){ j1=(j+1)/2;
      for(i=1;i<=I;i++){ i1=(i+1)/2;
      cg.node(i1,j1,k1).bp=cg.node(i1,j1,k1).bp+node(i,j,k).rsd;
      }
      }
      }
      return 1;
}

int Grid::Restriction(Grid& cg, int mode)
{
      int I1,J1,K1; cg.Get(I1,J1,K1);//get size of the coarse grid

      //Set bp and phi to zeros
      int i1,j1,k1;
      int i,j,k;
      for(k1=1;k1<=K1;k1++)
            for(j1=1;j1<=J1;j1++)
            for(i1=1;i1<=I1;i1++){
                  cg.node(i1,j1,k1).bp=0;
                  cg.node(i1,j1,k1).phi=0;
                  cg.node(i1,j1,k1).phit=cg.node(i1,j1,k1).phi;
            }

      TransferRsd(cg);        //Transfer residual and store in cg.bp
      if(mode==0) return 1;   //Correction scheme
      else;                   //Full approximation scheme
      for(k=1;k<=K;k++){ k1=(k+1)/2;
      for(j=1;j<=J;j++){ j1=(j+1)/2;
      for(i=1;i<=I;i++){ i1=(i+1)/2;
      cg.node(i1,j1,k1).phi=cg.node(i1,j1,k1).phi+node(i,j,k).phi;
      cg.node(i1,j1,k1).phit=cg.node(i1,j1,k1).phit+1;
      }
      }
      }

      for(k1=1;k1<=K1;k1++){
      for(j1=1;j1<=J1;j1++){
      for(i1=1;i1<=I1;i1++){
            cg.node(i1,j1,k1).phi=
                        cg.node(i1,j1,k1).phi/cg.node(i1,j1,k1).phit;
            cg.node(i1,j1,k1).phit=cg.node(i1,j1,k1).phi;
      }
      }
      }
```

```
	//Calculate bp
	for(k1=1;k1<=K1;k1++){
	for(j1=1;j1<=J1;j1++){
	for(i1=1;i1<=I1;i1++){
		cg.node(i1,j1,k1).UpdateBpH();
	}
	}
	}
	return 1;
}

//Prolongation and Correction
int Grid::CorrectPhi(Grid& cg, int mode)
{
	int I1,J1,K1; cg.Get(I1,J1,K1);
	if(I1<(I+1)/2||J1<(J+1)/2||K1<(K+1)/2) return 0;

	int i1,j1,k1;
	for(int k=1;k<=K;k++){ k1=(k+1)/2;
	for(int j=1;j<=J;j++){ j1=(j+1)/2;
	for(int i=1;i<=I;i++){ i1=(i+1)/2;
		if(mode==0)
		node(i,j,k).phi=node(i,j,k).phi+cg.node(i1,j1,k1).phi;
		else
		node(i,j,k).phi=node(i,j,k).phi+cg.node(i1,j1,k1).phi
                                        -cg.node(i1,j1,k1).phit;
	}
	}
	}
	return 1;
}

//Store old flux
void Grid::StorePhi()
{
	int IJK=I*J*K;
	for(int n=1;n<=IJK;n++)
		node(n).phit=node(n).phi;
}

//Update A and Bp (without scattering and fission sources)
void Grid::UpdateAB(double dt)
{
	int IJK=I*J*K;
	for(int n=1;n<=IJK;n++)
		node(n).UpdateAB(material, pos, div, dt);
}

//Update precursors
void Grid::UpdateC(double dt)
{
	int IJK=I*J*K;
```

```
	for(int n=1;n<=IJK;n++){
		double sf=Sum(node(n).nXf(material, pos, div)*node(n).phi);
		node(n).UpdateC(dt,sf);
	}
}

//Compute residuals
Parameter Grid::Residual()
{
	Parameter Rsd(Group::G); Rsd=0;
	int IJK=I*J*K;
	for(int n=1;n<=IJK;n++){
		node(n).UpdateRsd();
		Rsd=Rsd+Abs(node(n).rsd);
	}
	return Rsd/(I*J*K);
}

double Grid::MaxRsd()
{
	double Rsd=0, r;
	int IJK=I*J*K;
	for(int n=1;n<=IJK;n++){
		node(n).UpdateRsd();
		r=AbsMax(node(n).rsd);
		if(Rsd<r) Rsd=r;
	}
	return Rsd;
}

void Grid::UpdateRsd()
{
	int IJK=I*J*K;
	for(int n=1;n<=IJK;n++)
		node(n).UpdateRsd();
}

//Initialize node functions
int Grid::Init(Node& n, int i1, int i2, int j1, int j2, int k1, int k2)
{
	for(int i=i1;i<=i2;i++)
	for(int j=j1;j<=j2;j++)
	for(int k=k1;k<=k2;k++){
		node(i,j,k)=n;
		node(i,j,k).phit=n.phi;
		node(i,j,k).ct=n.c;
	}
	null.phi=n.phi*0;
	null.c=n.c*0;
	return 1;
}
```

```
//Initialize the fine grid
void Grid::Input(istream &is)
{
	char note[256]; int num;

	//Delay constants
	Delay d; d.Init(is);

	//Common group constants
	Group g; g.Init(is);

	//Material group constants
	is>>note>>num;
	material=new Material[num];
	int i,j,k;
	while(is){
		is>>i; if(i>=num) break;
		material[i].Init(is,i);
	}
	is.getline(note,255);
	for(i=0;i<num;i++)

	//Assembly structure
	is.getline(note,255); cout<<note<<endl;
	is>>note>>Cell::X>>Cell::Y>>Cell::Z;
	is>>note>>num;
	Cell *cell=new Cell[num];
	while(is){
		is>>i; if(i>=num) break;
		cell[i].Init(is);//reading assembly structure
	}
	is.getline(note,255);
	is.getline(note,255);

	//Core composition
	char ct;
	int Ic, Jc, Kc;
	is>>note>>Ic>>Jc>>Kc>>div;
	pos=new Cell*[Ic];
	for(i=0;i<Ic;i++) pos[i]=new Cell[Jc];
	cout<<"\t";
	for(i=0;i<Ic;i++) {is>>ct;cout<<ct<<" ";} cout<<endl;
	while(is){
		is>>j; if(j>=Jc) break;
		for(i=0;i<Ic;i++) {
			is>>ct;
			pos[i][j]=cell[0];
			int type=1;
			while(type<num){
				if(cell[type].ctype==ct){
					pos[i][j]=cell[type];
					break;
```

```
                        }
                  type++;
                  }
            }
            }

      int I=Ic*div;
      int J=Jc*div;
      int K=Kc*div;
      double hx=Cell::X/div;
      double hy=Cell::Y/div;
      double hz=Cell::Z/K;

      //Setting grid
      Set(I,J,K);
      for(int n=1;n<=I*J*K;n++) {
            node(n).SetDim(hx,hy,hz);//inner node dimension
      }

      //Initial conditions
      Node nod;
      int i2,j2,k2;
      is.getline(note,255);
      is.getline(note,255);
      while(is){
            is>>note; if(note[0]!='n') break;
            is>>i>>j>>k;
            is.getline(note,255);
            if(note[0]){is>>i2>>j2>>k2;}
            else{
                  if(i<=0) {i=1; i2=I;} else i2=i;
                  if(j<=0) {j=1; j2=J;} else j2=j;
                  if(k<=0) {k=1; k2=K;} else k2=k;
            }
            nod.Init(is);
            Init(nod, i, i2, j, j2, k, k2);//initialization
      }
      cout<<"End of input and initilization of the fine grid"<<endl;
}
#endif
```

File: NODE.H

```
#ifndef NODE_H
#define NODE_H

#include "param.h"
#include "group.h"
#include "delay.h"

enum face{p,w,e,n,s,u,d,point=0,west,east,north,south,up,down,
            P=0,W,E,N,S,U,D,Point=0,West,East,North,South,Up,Down};
```

```
Parameter Gauss(Parameter a, Parameter b);

//Node with flux and precursors
class Node{
      int i,j,k;                    //node index

      public:
      Parameter delta;
      Node* nnb[6];                 //pointer to neighbouring
      Parameter ap, anb, bp, as;    //algebraic coefficients
      Parameter Kappa;              //energy spectrum
      Parameter c;                  //precursor
      Parameter ct;                 //old precursor
      Parameter sx;                 //extra source
      Parameter phi;                //flux
      Parameter phit;               //old flux
      Parameter rsd;                //residual

      //Node methods
      int Init(istream& is);
      Node(int ii=0, int jj=0, int kk=0);
      void SetIdx(int ii=0, int jj=0, int kk=0);
      void SetDim(double hx, double hy, double hz);
      int IsRed();
      Node* nb(face f);
      void SetNb(face f, Node& node);
      int operator==(Node& node);
      int operator!=(Node& node);
      Node& operator=(Node& node);

      //Methods for updating algebraic coefficients
      void UpdateC(double dt, double fissource);
      void UpdateAB(Material *m, Cell **pos, int div, double dt);
      double UpdatePhi(double omega=1.);
      void UpdateRsd();
      double GUpdatePhi();
      void UpdateBpH();

      //Functions for group constants
      Parameter Dg(Material *m, Cell **pos, int div=1);
      Parameter Xa(Material *m, Cell **pos, int div=1);
      Parameter Xt(Material *m, Cell **pos, int div=1);
      Parameter Xf(Material *m, Cell **pos, int div=1);
      Parameter nXf(Material *m, Cell **pos, int div=1);
      Parameter Xs(Material *m, Cell **pos, int div=1);
};

int Node::Init(istream& is)
{
      Parameter c0(Delay::N), s0(Group::G);
      char note[256];
```

```
	is>>note>>s0; sx=s0;
	is>>note>>s0; phi=s0;
	is>>note>>c0; c=c0;
	is.getline(note,255);

	if(!is) return 0;
	return 1;
}

Node::Node(int ii, int jj, int kk)
{
	SetIdx(ii,jj,kk);
}

void Node::SetIdx(int ii, int jj, int kk)
{
	i=ii; j=jj; k=kk;
}

void Node::SetDim(double hx, double hy, double hz)
{
	delta.Set(6);
	delta(w)=delta(e)=hx/2;
	delta(n)=delta(s)=hy/2;
	delta(u)=delta(d)=hz/2;
}

Node* Node::nb(face f)
{
	if(f<1||f>6) return this;
	return nnb[f-1];
}

void Node::SetNb(face f, Node& node)
{
	if(f<1||f>6) return;
	nnb[f-1]=&node;
}

Node& Node::operator=(Node& node)
{
	phi=node.phi;
	sx=node.sx;
	c=node.c;

	return *this;
}

int Node::operator==(Node& node)
{
	if(i==node.i&&j==node.j&&k==node.k) return 1;
	else return 0;
```

```
}

int Node::operator!=(Node& node)
{
      if(i==node.i&&j==node.j&&k==node.k) return 0;
      else return 1;
}

void Node::UpdateAB(Material *m, Cell **pos, int div, double dt)
{
      //a-neighbour
      anb.Set(6,Group::G);
      Parameter Dp, Dnb(Group::G);
      Dp=Dg(m,pos,div);

      for(face l=w;l<=d;l++){ int lp=l%2? l+1:l-1;//oposite direction
      Node *p=nb(l);
      double deltaL=p->delta(lp);
      if(p->i) Dnb=p->Dg(m,pos,div); else Dnb=0;
      for(int g=1;g<=Group::G;g++){
            anb(l,g)=(Dnb(g)*deltaL + Dp(g)*delta(l))
                        /(deltaL + delta(l))/(deltaL + delta(l))
                        /(delta(l) + delta(lp));
      }
      }

      //bp
      Kappa=Group::kappa*(1.-Delay::beta)
            + Group::kappad*Delay::beta
            *Sum(Delay::lambda*Delay::gamma*dt/(Delay::lambda*dt+1.));

      bp=sx+Group::inv/dt*phit
            +Group::kappad*Sum(Delay::lambda*c/(Delay::lambda*dt+1.));

      //a-scattering+fission sources
      as.Set(Group::G,Group::G);
      Parameter Xsg=Xs(m,pos,div);
      Parameter nXfg=nXf(m,pos,div);
      for(int g=1; g<=Group::G; g++)
      for(int gp=1; gp<=Group::G; gp++)
            as(gp,g) = Xsg(gp,g) + Kappa(g)*nXfg(gp);

      //ap
      ap=Group::inv/dt+Xt(m,pos,div);;
      for(int g=1;g<=Group::G;g++){
      double sumanb=0;
      for(face l=w;l<=d;l++) sumanb+=anb(l,g);
      ap(g)+=sumanb-as(g,g);
      }

}
```

```
double Node::UpdatePhi(double omega)
{
      if(omega<0||omega>2) return GUpdatePhi();//Gaussian elimination

      double sumnb, sums, phi_old, dphi;
      for(int g=1;g<=Group::G;g++){
      sumnb=sums=0;
      for(face l=w;l<=d;l++) sumnb+=anb(l,g)*nb(l)->phi(g);
      for(int gp=1;gp<=Group::G;gp++) if(gp-g) sums+=as(gp,g)*phi(gp);
      phi_old=phi(g);
      phi(g)=omega*(bp(g)+sumnb+sums)/ap(g)+(1.-omega)*phi_old;
      dphi=phi(g)-phi_old;
      }
      return(dphi);
}

void Node::UpdateBpH()
{
      double sumnb, sums;
      for(int g=1;g<=Group::G;g++){
      sumnb=sums=0;
      for(face l=w;l<=d;l++) sumnb+=anb(l,g)*nb(l)->phi(g);
      for(int gp=1;gp<=Group::G;gp++) if(gp-g) sums+=as(gp,g)*phi(gp);
      bp(g)=bp(g)+ ap(g)*phi(g)-sumnb-sums;
      }
}

void Node::UpdateRsd()
{
      Parameter sumnb(Group::G), sums(Group::G);
      sumnb=0; sums=0;
      for(int g=1;g<=Group::G;g++){
      for(face l=w;l<=d;l++)sumnb(g)+=anb(l,g)*nb(l)->phi(g);
      for(int gp=1;gp<=Group::G;gp++)
            if(gp-g) sums(g)+=as(gp,g)*phi(gp);
      }
      rsd=sumnb+bp+sums-ap*phi;
}

int Node::IsRed()
{
      return(i+j+k)%2;
}

//Gaussian elimination
double Node::GUpdatePhi()
{
      Parameter a, b, phi_old;
      a=as*(-1.); b=bp; phi_old=phi;
      for(int g=1;g<=Group::G;g++){
            double sumnb=0;
            for(face l=w;l<=d;l++) sumnb+=anb(l,g)*nb(l)->phi(g);
```

```
            b(g)=bp(g)+sumnb;
            a(g,g)=ap(g);
      }
      phi=Gauss(Transpose(a),b);
      for(int g=1;g<=Group::G;g++){
      double sumnb=0; double sums=0;
            for(face l=w;l<=d;l++) sumnb+=anb(l,g)*nb(l)->phi(g);
            for(int gp=1;gp<=Group::G;gp++) sums+=a(gp,g)*phi(gp);
      }
      return(AbsMax(phi-phi_old));
}

Parameter Gauss(Parameter a, Parameter b)
{
      int N=a.row();
      int i,j,k;

      for(i=1; i<=N; i++){//forward elimination
            if(a(i,i)==0){//check for pivot
                  if(i==N) {cerr<<"Indefinite case"; return 0;}
                  for(k=i+1;k<=N;k++){
                        if(a(k,i)){
                              double temp;
                              for(j=1;j<=N;j++){
                              temp=a(k,j);
                              a(k,j)=a(i,j);
                              a(i,j)=temp;
                              }
                              temp=b(k);
                              b(k)=b(i);
                              b(i)=temp;
                              break;
                        }
                  }
                  if(k>N) {cerr<<"Indefinite case"; return 0;}
            }
            for(k=i+1;k<=N;k++){
                  double aki=a(k,i);
                  b(k)-=b(i)/a(i,i)*aki;
                  for(j=i+1;j<=N;j++) a(k,j)-=a(i,j)/a(i,i)*aki;
                  a(k,i)=0;
            }
      }
      Parameter x(N);
      x(N)=b(N)/a(N,N);
      for(i=N-1; i>=1; i--){//backward substitution
            double sum=b(i);
            for(j=i+1;j<=N;j++) sum-=a(i,j)*x(j);
            x(i)=sum/a(i,i);
      }
      return x;
}
```

```
void Node::UpdateC(double dt, double sf)
{
      if(dt<0) return;
      if(dt)c=(c+Delay::gamma*Delay::beta*dt*sf)/(Delay::lambda*dt+1.);
      else  c=Delay::gamma*Delay::beta*sf/Delay::lambda;
}

//Diffusion coefficient
Parameter Node::Dg(Material *m, Cell **pos, int div)
{
      Parameter one(Group::G); one=1;
      int ic=(i-1)/div; //cell i-idx
      int jc=(j-1)/div; //cell j-idx

      int t1,t2;
      double dz=delta(u)+delta(d);
      double z1=(k-1)*dz;
      double f=pos[ic][jc].fract1(z1,z1+dz,t1,t2);
      return one/((one/m[t1].D)*f + (one/m[t2].D)*(1-f));
}
//XSections
Parameter Node::Xa(Material *m, Cell **pos, int div)
{
      int ic=(i-1)/div; //cell i-idx
      int jc=(j-1)/div; //cell j-idx
      int t1,t2;
      double dz=delta(u)+delta(d);
      double z1=(k-1)*dz;
      double f=pos[ic][jc].fract1(z1,z1+dz,t1,t2);
      return m[t1].SIGMAa*f + m[t2].SIGMAa*(1-f);
}
Parameter Node::Xf(Material *m, Cell **pos, int div)
{
      int ic=(i-1)/div;//cell i-idx
      int jc=(j-1)/div;//cell j-idx
      int t1,t2;
      double dz=delta(u)+delta(d);
      double z1=(k-1)*dz;
      double f=pos[ic][jc].fract1(z1,z1+dz,t1,t2);
      return m[t1].SIGMAf*f + m[t2].SIGMAf*(1-f);
}
Parameter Node::Xt(Material *m, Cell **pos, int div)
{
      int ic=(i-1)/div; //cell i-idx
      int jc=(j-1)/div; //cell j-idx
      int t1,t2;
      double dz=delta(u)+delta(d);
      double z1=(k-1)*dz;
      double f=pos[ic][jc].fract1(z1,z1+dz,t1,t2);
      return m[t1].SIGMAt*f + m[t2].SIGMAt*(1-f);
}
```

```
Parameter Node::nXf(Material *m, Cell **pos, int div)
{
      int ic=(i-1)/div;//cell i-idx
      int jc=(j-1)/div;//cell j-idx
      int t1,t2;
      double dz=delta(u)+delta(d);
      double z1=(k-1)*dz;
      double f=pos[ic][jc].fract1(z1,z1+dz,t1,t2);
      return m[t1].nuSIGMAf*f + m[t2].nuSIGMAf*(1-f);
}
Parameter Node::Xs(Material *m, Cell **pos, int div)
{
      int ic=(i-1)/div;//cell i-idx
      int jc=(j-1)/div;//cell j-idx
      int t1,t2;
      double dz=delta(u)+delta(d);
      double z1=(k-1)*dz;
      double f=pos[ic][jc].fract1(z1,z1+dz,t1,t2);
      return m[t1].SIGMAs*f + m[t2].SIGMAs*(1-f);
}
#endif
```

File: MATERIAL.H

```
#ifndef MAT_H
#define MAT_H

#include <iostream>
#include "group.h"

//Core assembly or cell
class Cell{
      public:
      int Nz;
      double *h;
      int *t;
      static double X,Y,Z;    //dimensions
      int i,j;                //position in core
      char ctype;             //type

      Cell(){Set(1);}
      Cell(int num){Set(num);}
      int Init(istream& is);
      void Set(int num);
      void SetIdx(int ic, int jc);
      void Set(int i, double hz, int tz);
      Cell& operator =(const Cell& cell);
      double fract1(double z1, double z2, int &t1, int &t2);
      friend ostream& operator<<(ostream& os, Cell& cell);
};
double Cell::X=7.7;
double Cell::Y=8.1;
double Cell::Z=80;
```

```
double Cell::fract1(double z1, double z2, int &t1, int &t2)
{
	if(Nz==1){
		t1=t2=t[0];
		return 1;
	}
	double z=0;
	int i=0;
	while(i<Nz){
		z+=h[i];
		if(z1<=z) break;
		i++;
	}
	t1=t[i];
	if(z2>z&&z2<=Z) t2=t[i+1];
	else t2=t[i];
	if(t1==t2) return 1;
	else return((z-z1)/(z2-z1));
}

int Cell::Init(istream& is)
{
	is>>ctype;
	is>>Nz; Set(Nz);
	for(int i=0;i<Nz;i++){
		is>>h[i]>>t[i];
	}
	if(!is) return 0;
	else return 1;
}

void Cell::Set(int num)
{
	Nz=num;
	h=new double[Nz];
	t=new int[Nz];
	for(int i=0;i<Nz;i++){
		h[i]=0;
		t[i]=0;
	}
	h[0]=Z;
}

void Cell::Set(int i, double hz, int tz)
{
	if(i>=Nz) i=Nz-1;
	if(i<0) i=0;
	double z=0;
	if(i>0) for(int j=0;j<i;j++) z+=h[j];
	if(hz+z>Z) hz=Z-z;
	if(hz<0) hz=0;
```

```
      h[i]=hz; t[i]=tz;
}

void Cell::SetIdx(int ic, int jc)
{
      i=ic; j=jc;
}

Cell& Cell::operator =(const Cell& cell)
{
      if(Nz!=cell.Nz){
            Nz=cell.Nz;
            Set(Nz);
      }
      ctype=cell.ctype;
      for(int i=0;i<Nz;i++){
            h[i]=cell.h[i];
            t[i]=cell.t[i];
      }
   return *this;
}

//Material group constants
class Material: public Group{
      public:
      int type;
      char mtype[20];
      Material(){};
      int Init(istream& is, int i);
};

int Material::Init(istream& is, int i)
{
      Parameter t(Group::G), tt(Group::G,Group::G);
      char note[128];
      type=i;
      is>>mtype;

      is>>note;
      is>>t;SIGMAa=t;

      is>>note;
      is>>t;nuSIGMAf=t;

      is>>note;
      is>>t;SIGMAt=t;

      is>>note;
      is>>tt;SIGMAs=tt;

      is>>note;
      is>>t;D=t;
```

```
      if(!is) return 0;
      else return 1;
}
#endif
```

File: GROUP.H

```
#ifndef GROUP_H
#define GROUP_H

#include <iostream>
#include "param.h"

//Neutron group constants

class Group{
      public:
      static int G;                 //#neutron groups
      static Parameter  inv,        //inverse velocity
                        kappa,      //fission spectrum
                        kappad;     //delay spectrum

      Parameter         D,          //diffusion coefficient
                        nuSIGMAf,   //fission neutron emision
                        SIGMAa,     //absorption X-section
                        SIGMAt,     //total X-section
                        SIGMAf,     //fission X-section
                        SIGMAs;     //scattering X-section

      Group(){}
      Group(int num);
      ~Group(){}
      void Set(int num);
      double SIGMAr(int g);
      int Init(istream& is);
};
int Group::G=1;
Parameter   Group::inv(Group::G),
            Group::kappa(Group::G),
            Group::kappad(Group::G);

Group::Group(int num)
{
      Set(num);
}

void Group::Set(int num)
{
      if(num<1) num=1;
      G=num;
      inv.Set(G);
      kappa.Set(G);
```

```
      kappad.Set(G);

      D.Set(G);
      nuSIGMAf.Set(G);
      SIGMAa.Set(G);
      SIGMAt.Set(G);
      SIGMAf.Set(G);
      SIGMAs.Set(G,G);
}

double Group::SIGMAr(int g)
{
      return SIGMAt(g)-SIGMAs(g,g);
}

int Group::Init(istream &is)//Global constants
{
      char note[256];
      int num;
      is.getline(note,255);
      is>>note>>num;
      Set(num);
      is>>note>>inv;
      is>>note>>kappa;
      is>>note>>kappad;
      if(!is) return 0;
      else return 1;
}

#endif
```

File: DELAY.H

```
#ifndef DELAY_H
#define DELAY_H

#include <iostream>
#include "param.h"
//Delayed neutron group constants
class Delay{
      public:
      static int N;
      static Parameter  lambda,     //decay constant
                        gamma;      //relative fraction yield
      static double beta;           //total delay fraction

      Delay(){;}
      Delay(int num);
      ~Delay(){}
      void Set(int num);
      int Init(istream &is);
};
//static allocation
```

```
int Delay::N=1;
Parameter   Delay::lambda(Delay::N),
            Delay::gamma(Delay::N);
double Delay::beta=0;

Delay::Delay(int num)
{
      N=num; if(num<0) N=1;
      lambda.Set(N);
      gamma.Set(N);
}

void Delay::Set(int num)
{
      if(num<0) return;
      N=num;
      lambda.Set(N);
      gamma.Set(N);
}

int Delay::Init(istream& is)
{
      char note[256];
      int num;
      is.getline(note,255);
      is>>note>>num;
      Set(num);
      is>>note>>beta;
      is>>note>>lambda;
      is>>note>>gamma;
      is.getline(note,255);
      if(!is) return 0;
      else return 1;
}
#endif

File:       PARAM.H
#ifndef PARAM_H
#define PARAM_H

#include <iostream>
#include <math>

//Parameter
class Parameter{
      int rows, cols;
      double *data;
      public:
      Parameter(int num=1);
      Parameter(int row, int col);
      Parameter(int num, double *a);
      Parameter(int row, int col, double *a);
```

```
      Parameter(Parameter& a);
      ~Parameter(){delete[]data;}

      void Set(int num);
      void Set(int row, int col);
      void Set(int num, double *a);

      int& row(){return rows;}
      int& col(){return cols;}

      friend double Sum(Parameter& a);
      friend Parameter Transpose(Parameter& a);
      friend double AbsMax(Parameter& a);
      friend Parameter Abs(Parameter& a);

      Parameter operator-(const Parameter& a);
      Parameter operator-(const double a);
      Parameter operator+(const Parameter& a);
      Parameter operator+(const double a);
      Parameter operator*(const Parameter& a);
      Parameter operator*(const double a);
      Parameter operator/(const Parameter& a);
      Parameter operator/(const double a);
      Parameter& operator=(const Parameter& a);
      Parameter& operator=(const double a);

      double& operator()(int i)const;
      double& operator()(int i, int j)const;

      friend ostream& operator<<(ostream& os, const Parameter& a);
      friend istream& operator>>(istream& is, Parameter& a);
};

Parameter::Parameter(int num)
{
      rows=1;
      if(num<1) cols=1; else cols = num;
      int bound=rows*cols;
      data = new double[bound];
      for(int i=0;i<bound;i++) data[i]=0;
}

Parameter::Parameter(int row, int col)
{
      if(row<1) rows=1; else rows=row;
      if(col<1) cols=1; else cols=col;
      int bound=rows*cols;
      data = new double[bound];
      for(int i=0;i<bound;i++) data[i]=0;
}

Parameter::Parameter(int num, double *a)
```

```
{
	rows=1;
	if(num<1) cols=1; else cols = num;
	int bound=rows*cols;
	data = new double[bound];
	for(int i=0;i<bound;i++) data[i]=a[i];
}

Parameter::Parameter(int row, int col, double *a)
{
	if(row<1) rows=1; else rows=row;
	if(col<1) cols=1; else cols=col;
	int bound=rows*cols;
	data = new double[bound];
	for(int i=0;i<bound;i++) data[i]=a[i];
}

Parameter::Parameter(Parameter& a)
{
	rows=a.rows;
	cols=a.cols;
	int bound=rows*cols;
	data = new double[bound];
	for(int i=0;i<bound;i++) data[i]=a.data[i];
}

void Parameter::Set(int num)
{
	int bound=rows*cols;
	rows=1;
	if(num<1) cols=1; else cols=num;
	int newbound=rows*cols;
	if(bound!=newbound){
		double *a=new double[bound];
		for(int i=0;i<bound;i++) a[i]=data[i];
		delete[]data;
		data=new double[newbound];
		for(int i=0;i<newbound;i++){
			if(i<bound) data[i]=a[i]; else data[i]=0;
		}
		delete[]a;
	}
}

void Parameter::Set(int row, int col)
{
	int bound=rows*cols;
	if(row<1) rows=1; else rows=row;
	if(col<1) cols=1; else cols=col;
	int newbound=rows*cols;
	if(bound!=newbound){
		double *a=new double[bound];
```

```
            for(int i=0;i<bound;i++) a[i]=data[i];
            delete[]data;
            data=new double[newbound];
            for(int i=0;i<newbound;i++){
                  if(i<bound) data[i]=a[i]; else data[i]=0;
            }
            delete[]a;
      }
}

void Parameter::Set(int num, double *a)
{
      int bound=rows*cols;
      if(num<1) num=1;
      if(num>bound) num=bound;
      for(int i=0;i<num;i++) data[i]=a[i];
}

double& Parameter::operator()(int i)const
{
      int bound=rows*cols;
      if(i<1||i>bound){cerr<<"Beyond the bounds\n"; i=1;}
      return data[i-1];
}

double& Parameter::operator()(int i, int j)const
{
      if(i<1||i>rows||j<1||j>cols)
            {cerr<<"Beyond the bounds\n"; i=1;j=1;}
      return data[(i-1)*cols+j-1];
}

Parameter& Parameter::operator=(const Parameter& a)
{
      int bound=rows*cols;
      rows=a.rows;
      cols=a.cols;
      int newbound=rows*cols;
      if(bound!=newbound){
            bound=newbound;
            delete[]data;
            data = new double[bound];
      }
      for(int i=0;i<bound;i++) data[i]=a.data[i];
      return *this;
}

Parameter& Parameter::operator=(const double a)
{
      int bound=rows*cols;
      for(int i=0;i<bound;i++) data[i]=a;
      return *this;
```

```
}

ostream& operator<<(ostream& os, const Parameter& a)
{
      int bound=a.rows*a.cols;
      for(int i=1;i<=bound;i++){
            os<<a.data[i-1];
            if(i%a.cols) os<<'\t';
            else{
                  os<<'\n';
                  if(i<bound) os<<'\t';
            }
      }
      return os;
}

istream& operator>>(istream& is, Parameter& a)
{
      int bound=a.rows*a.cols;
      for(int i=0;i<bound;i++) is>>a.data[i];
      return is;
}

Parameter Parameter::operator-(const Parameter& a)
{
      Parameter temp(*this);
      int bound=rows*cols<=a.rows*a.cols? rows*cols: a.rows*a.cols;
      for(int i=0;i<bound;i++) temp.data[i]-=a.data[i];
      return temp;
}

Parameter Parameter::operator-(const double a)
{
      Parameter temp(*this);
      int bound=rows*cols;
      for(int i=0;i<bound;i++) temp.data[i]-=a;
      return temp;
}

Parameter Parameter::operator+(const Parameter& a)
{
      Parameter temp(*this);
      int bound=rows*cols<=a.rows*a.cols? rows*cols: a.rows*a.cols;
      for(int i=0;i<bound;i++) temp.data[i]+=a.data[i];
      return temp;
}

Parameter Parameter::operator+(const double a)
{
      Parameter temp(*this);
      int bound=rows*cols;
      for(int i=0; i<bound; i++) temp.data[i]+=a;
```

```
	return temp;
}

Parameter Parameter::operator*(const Parameter& a)
{
	Parameter temp(*this);
	int bound=rows*cols<=a.rows*a.cols? rows*cols: a.rows*a.cols;
	for(int i=0; i<bound; i++) temp.data[i]*=a.data[i];
	return temp;
}

Parameter Parameter::operator*(const double a)
{
	Parameter temp(*this);
	int bound=rows*cols;
	for(int i=0; i<bound; i++) temp.data[i]*=a;
	return temp;
}

Parameter Parameter::operator/(const Parameter& a)
{
	Parameter temp(*this);
	int bound=rows*cols<=a.rows*a.cols? rows*cols: a.rows*a.cols;
	for(int i=0; i<bound; i++) temp.data[i]/=a.data[i];
	return temp;
}

Parameter Parameter::operator/(const double a)
{
	Parameter temp(*this);
	int bound=rows*cols;
	for(int i=0; i<bound; i++) temp.data[i]/=a;
	return temp;
}

double Sum(Parameter &a)
{
	double s=0;
	for(int i=1;i<=a.rows*a.cols;i++) s+=a(i);
	return s;
}

double AbsMax(Parameter &a)
{
	double s=0;
	for(int i=1;i<=a.rows*a.cols;i++) if(fabs(a(i))>fabs(s))
	s=fabs(a(i));
	return s;
}

Parameter Transpose(Parameter& a)
{
```

```
      Parameter temp(a.cols,a.rows);
      for(int i=1;i<=a.cols;i++)
      for(int j=1;j<=a.rows;j++) temp(i,j)=a(j,i);
      return temp;
}

Parameter Abs(Parameter& a)
{
      Parameter temp(a);
      for(int i=1;i<=a.rows*a.cols;i++) temp(i)=fabs(a(i));
      return temp;
}
#endif
```