AAEC/E612

AAEC/E612

AUSTRALIAN ATOMIC ENERGY COMMISSION
RESEARCH ESTABLISHMENT

LUCAS HEIGHTS RESEARCH LABORATORIES

AUS DIFFUSION NEUTRONICS MODULE - POW3D

A MATHEMATICAL DESCRIPTION

by

J.M. BARRY

J.P. POLLARD

APRIL 1987

revised July 1996

ISBN 0 642 59891 6

ISSN 1030-7745

AUSTRALIAN ATOMIC ENERGY COMMISSION
RESEARCH ESTABLISHMENT

LUCAS HEIGHTS RESEARCH LABORATORIES

AUS DIFFUSION NEUTRONICS MODULE - POW3D

A MATHEMATICAL DESCRIPTION

by

J.M. BARRY
J.P. POLLARD

ABSTRACT

The mathematical and computational structure of POW3D, a general purpose zero, one, two and three-dimensional multigroup neutron diffusion code including feedback-free kinetics, is described. The code serves as a diffusion module for the AUS nuclear code system. During the production of this efficient nuclear code, a novel mathematical approach was developed to solve the large number of linear equations involved. A description is given of this technique, the method of implicit non-stationary iteration (MINI), and the way in which it is implemented in the code. The three-dimensional implementation of several other conventional approaches to the solution of the linear systems is also discussed.

ISBN 0 642 59891 6

ISSN 1030-7745

The following descriptors have been selected from the INIS Thesaurus to describe the subject content of this report for information retrieval purposes. For further details please refer to IAEA-INIS-12 (INIS: Manual for Indexing) and IAEA-INIS-13 (INIS: Thesaurus) published in Vienna by the International Atomic Energy Agency.

A CODES; DIFFERENTIAL EQUATIONS; DIFFUSION; FORTRAN; FISSION NEUTRONS; ITERATIVE METHODS; MULTIGROUP THEORY; NUMERICAL SOLUTION; THREE-DIMENSIONAL CALCULATIONS

EDITORIAL NOTE

From 27 April 1987, the Australian Atomic Energy Commission (AAEC) is replaced by Australian Nuclear Science and Technology Organisation (ANSTO). Serial numbers for reports with an issue date after April 1987 have the prefix ANSTO with no change of the symbol (E,M,S or C) or numbering sequence.

CONTENTS

1. INTRODUCTION

It is vital in any neutronics calculation scheme to have a versatile, fast and accurate multi-dimensional, multigroup diffusion code. The first codes used at Lucas Heights were CRAM (Hassitt [1962], rewritten in FORTRAN, but originally for the IBM7040) and GOG (Hopkins and Oakes [1968], much of which was rewritten for the IBM360/50). There were a number of other suitable nuclear codes around the world but many of them were not readily available for distribution. Local experience with CRAM and GOG indicated that CRAM was versatile but insufficiently fast, whereas GOG was fast but lacked versatility. In addition, obvious shortcomings of both were revealed for some calculations. With the development of the AAEC's AUS modular scheme [Robinson 1975] a versatile, fast and accurate two-dimensional multigroup diffusion code was required as a 'workhorse'; this led to the module POW [Pollard 1974].

In some aspects POW was a radical break from the earlier codes. One essential difference was that it made a better estimate of leakage throughout the reactor compared with other codes because it used an edge flux scheme [Wachspress 1966]. POW used successive line over relaxation (SLOR) to solve the sparse linear system, and Chebyshev extrapolation for the eigenvalue calculation. The code differed from other codes in its ability to obtain extrapolation parameters for both schemes from a detailed analysis of the neutron flux calculated, using pre-optimum estimates of the parameters. In addition, both group and region rebalance were carried out to enhance convergence.

POW was designed to function efficiently on an IBM360/50 computer with 512 kilobytes of memory. As computer speeds increased and the cost of primary memory reduced, it became possible to consider the third spatial dimension in reactor models. It is not efficient, however, simply to alter an old two-dimensional code for studies of the third dimension. It is necessary to consider advances in mathematical techniques and computer architecture.

POW3D employs the novel method of implicit non-stationary iteration [MINI; Barry and Pollard 1977, 1978] to solve the large sparse linear systems of equations that arise at the heart of any multi-dimensional neutron diffusion calculation. MINI is the standard option for POW3D; however, for comparative studies or for other special functions, POW3D also supports a three-dimensional extension of the SLOR scheme from POW and ICCG the refined conjugate gradient method of Meijerink and van der Vorst [1977].

Although, like POW, POW3D may be thought of as a 'workhorse' code it also has been used as a research tool. During implementation of this work, the three iterative techniques were evaluated on three-dimensional problems before the decision was taken to make MINI the default option. Details of these studies are reported [Barry and Pollard 1977, 1978, 1981; Barry 1982]. Normal region and group rebalance in POW3D are based on three-dimensions. Testing of extensions to the normal rebalance techniques based on two-dimensions indicated that a three-dimensional implementation was not worthwhile. The extensions have been retained as a special option in POW3D should further research seem inviting.

POW3D was developed on an IBM360/65 computer, but the eventual host was expected to be an IBM370 with a virtual operating system. Careful design of the data structure for any three-dimensional model is essential and special consideration must be given to dramatic changes in computer memory architecture. The code was required to function efficiently in both the real and the virtual worlds so a special data handler was developed [Pollard, unpublished].

The original source language of POW3D was mainly FORTRAN IV, although the direct access input/output (I/O) handlers were written in Assembler language [Cawley 1976]. Several key mathematical subroutines also were written in Assembler language [Cox 1976] but FORTRAN IV versions were available. Since 1990 FORTRAN 77 modules are used exclusively to enable porting of the code to different platforms. With improvements in FORTRAN compilers and hardware performance, the need for specially coded key routines is reduced.

The program may be run alone as well as under the AUS system in as little as 8 Megabytes of real memory; however, large three-dimensional problems require much more storage. A dynamic storage system [Cox 1971] was used to overcome the fixed array dimension weakness of FORTRAN. In porting the code to UNIX platforms this is replaced with a C routine call to 'malloc' [Robinson unpublished].

POW3D has all but replaced POW as the AAEC's neutron diffusion code, even for two-dimensional studies. For the latter, POW3D is significantly faster than POW owing to the mathematical enhancements and the reconstruction of other important computational sections of code. For zero and one-dimensional studies with many energy groups, the more general data structures of POW3D are not as effective as the simpler POW structures.

From a code user's point of view, ease of use is extremely important. POW and POW3D both employ free input routines to assist in data preparation; indeed, the actual input to POW can usually run unchanged in POW3D. As well as calculating neutron fluxes, each code can prepare its own 2-region (equivalent) resonance shielded cross sections from AUS data pools (libraries), although this feature is no longer used as the cross sections may also be obtained from libraries processed by other AUS modules [Robinson 1975]. Details are given in the POW3D User's Guide [Barry *et al.*, forthcoming]. The codes also have reasonably general editing facilities including perturbation options. In addition, one- and two-dimensional graphical plots of flux *v.* reaction rate may be produced. POW3D is designed to make the transition to the third dimension as simple as possible for the POW user.

Finally, POW3D can be used for multi-dimensional kinetic studies for systems perturbed from the steady state with a prescribed pulsed (feedback-free) variation of some usually physical parameter, for example the concentration of a reactor material.

2. NEUTRON DIFFUSION EQUATIONS

2.1 Time-dependent Multigroup Neutron Diffusion Equations

POW3D was designed to undertake static (steady-state) calculations in up to three spatial dimensions, and time-dependent (kinetic) studies with a limited amount of feedback. Until now, however, most of the work with the code has been directed to steady-state studies.

The time-dependent neutron diffusion equations are

$$\begin{aligned} -\frac{1}{v_g}\frac{\partial}{\partial t}\phi_g(\underline{r},t) = & -\nabla . D_{n,g}(\underline{r},t)\nabla\phi_g(\underline{r},t) + \sigma_{rg}(\underline{r},t)\phi_g(\underline{r},t) \\ & -\sum_{g'}\sigma_{gg'}(\underline{r},t)\phi_{g'}(\underline{r},t) \\ & -\chi_{pg}(1-\beta)\sum_{g'}\frac{\nu}{k}\sigma_{fg'}(\underline{r},t)\phi_{g'}(\underline{r},t) \\ & -\sum_{d}\chi_{dg}\lambda_d C_d(\underline{r},t) - S_g(\underline{r},t)\ ,\ (g=1,2,\ldots,G) \qquad (2.1.1) \end{aligned}$$

and those for the delayed neutron precursor concentrations are

$$\frac{\partial}{\partial t}C_d(\underline{r},t) = \beta_d\sum_{g}\frac{\nu}{k}\sigma_{fg}(\underline{r},t)\phi_g(\underline{r},t) - \lambda_d C_d(\underline{r},t)\ ,\quad (d=1,2,\ldots,D)\ , \qquad (2.1.2)$$

where k is the effective steady-state multiplication, which has been determined in a criticality calculation (left hand side of **equations 2.1.1** and **2.1.2** replaced with zero as is the source term S_g); $\phi_g(\underline{r},t)$ is the neutron flux for energy group g and is calculated in all studies; $C_d(\underline{r},t)$ is the precursor concentration density for the d^th^ delayed group, which is calculated as an intermediate stage in obtaining $\phi_g(\underline{r},t)$; v_g is the average velocity for energy group g; $D_{n,g}(\underline{r},t)$ denotes possibly (tensor) directional diffusion coefficients (for directions $\underline{n}$ or $-\underline{n}$ parallel to the chosen axes), although the isotropic value $D_g(\underline{r},t) = 1/(3\sigma_{tr,g}(\underline{r},t))$ is usually used. To avoid confusion with summation, the practice of representing all macroscopic cross sections by σ

is used throughout;

$\sigma_g(\underline{r},t)$ denotes various (macroscopic) cross sections:

$\sigma_{gg'}$	=	scattering cross section (matrix) from groups g′ to g,
σ_{rg}	=	removal cross section from group g,
	=	$\sigma_{ag} + \sum_{g' \neq g} \sigma_{g'g}$ (absorption + outscatter from group g) — the self scatter term σ_{gg} is set to zero,
σ_{fg}	=	fission cross section for group g,
$\nu\sigma_{fg}$	=	the fission emission for group g (note that σ_{rg} and $\nu\sigma_{fg}$ are not matrix quantities);

χ_{pg} is the prompt fission spectrum for neutrons emitted in energy group g (normalised to a unit group sum); χ_{dg} is the delayed fission spectrum for neutrons emitted with energy g (normalised to a unit group sum) for the delayed group d; β_d is the fraction of all fission neutrons emitted in delayed group d, note β (= $\sum_d \beta_d$); $S_g(\underline{r},t)$ is the source strength in group g of an external source located at a point $\underline{r}$ of the reactor; t is the time; and λ_d is the time constant for the delayed group decay. The summations are taken over all groups (g = 1,2,...,G; d = 1,2,...,D).

The equations are solved subject to the following conditions:

(i) *The outer boundary conditions* for

(a) reflective (zero current), frequently used when reactor symmetry is applicable —

$$\underline{n} \cdot \nabla \phi_g(\underline{r},t) = 0 \ , \qquad (2.1.3)$$

where $\underline{n}$ denotes the outward pointing normal, or

(b) zero flux at the extrapolated boundary —

$$d\,\underline{n} \cdot \nabla \phi_g(\underline{r},t) + \phi_g(\underline{r},t) = 0 \ , \qquad (2.1.4)$$

where d is the extrapolation distance, that is, the distance outside the medium at which the flux would vanish, and is sometimes taken to be $2/(3\sigma_{tr})$, where σ_{tr} is the macroscopic transport cross section, although more rigorous theory leads to $d = 0.71/\sigma_{tr}$.

(ii) *The internal boundary conditions* for the material interfaces which are assumed to lie parallel to the axes. For left and right hand boundaries these are

(a) the continuity of flux —

$$\phi_g(\underline{r},t)\,\big|_L = \phi_g(\underline{r},t)\,\big|_R \quad \text{for each group g, and} \qquad (2.1.5)$$

(b) continuity of current —

$$(D_{n,g}(\underline{r},t)\ \underline{n} \cdot \nabla\phi_g(\underline{r},t)\,\big|_L - D_{n,g}(\underline{r},t)\ \underline{n} \cdot \nabla \phi_g(\underline{r},t)\,\big|_R) = 0 \qquad (2.1.6)$$

for each group g, where $\underline{n}$ is normal to the boundary.

(iii) *The initial conditions*; the reactor may be operating at a steady-state power level or starting from a shutdown position:

(a) Steady-state conditions —

$$\frac{\partial}{\partial t}\phi_g(\underline{r},t)=0,\ \frac{\partial}{\partial t}C_d(\underline{r},t)=0,\ S_g(\underline{r},t)=0,\ t\le 0 \text{ for all groups,}$$

$$\sum_g\int_v f(\underline{r},0)\,\sigma_{fg}(\underline{r},0)\,\phi_g(\underline{r},0)\,d\underline{r}=P(0),$$

where $f(\underline{r},t)$ is the energy released from each fission of the material about $\underline{r}$ and P(0) is the required power level at time zero (the start of the study). To satisfy this condition, an eigenvalue problem (*i.e.* the steady state form of **equation 2.1.1**) is solved to obtain the group fluxes ϕ_g. *The steady-state eigenvector is normalised to produce the appropriate power level P(0).*

(b) Shutdown conditions —

$$\phi_g(\underline{r},t)=0,\ C_d(\underline{r},t)=0,\ S_g(\underline{r},t)=0,\ t\le 0 \text{ for all groups.}$$

2.2 Steady-state Neutron Diffusion Equations

As well as solving the time-dependent **equations 2.1.1** and **2.1.2**, POW3D is designed to solve other problems, including the steady-state eigenvalue (or criticality) problem either in its own right, or as the first stage of a kinetics calculation. The steady-state eigenvalue problem is

$$-\nabla\cdot D_{ng}(\underline{r})\nabla\phi_g(\underline{r})+\sigma_{rg}(\underline{r})\,\phi_g(\underline{r})-\sum_{g'}\sigma_{gg'}\phi_{g'}(\underline{r}) \qquad (2.1.7)$$
$$=\chi_g\sum_{g'}\frac{\nu}{k}\,\sigma_{fg'}(\underline{r})\,\phi_{g'}(\underline{r}),\quad (g=1,2,\ldots,G)\ ,$$

where $\chi_g=\chi_{pg}(1-\beta)+\sum_d\chi_{dg}\beta_d$. The eigenvalue (1/k) is introduced to allow non-trivial (*i.e.* non-zero) solutions. The existence of a non-trivial solution corresponds to a critical physical assembly for k=1. A non-trivial solution can be imposed by artificially adjusting the number of neutrons produced in the fission process through the introduction of a positive real number 1/k, where k corresponds to the multiplication factor of the assembly.

POW3D also handles steady-state source problems of the form

$$-\nabla\cdot D_{ng}(\underline{r})\nabla\phi_g(\underline{r})+\sigma_{rg}(\underline{r})\phi_g(\underline{r})-\sum_{g'}\sigma_{gg'}\phi_{g'}(\underline{r}) \qquad (2.1.8)$$
$$=\chi_g\sum_{g'}\nu\sigma_{fg'}(\underline{r})\,\phi_{g'}(\underline{r})+S_g(\underline{r}),\quad (g=1,2,\ldots,G),$$

where a multiplicative factor k is no longer required. The adjoint flux is of considerable assistance when undertaking perturbation studies of reactor systems. The adjoint equivalent to the eigenvalue **equation 2.1.8** is

$$-\nabla\cdot D_{ng}(\underline{r})\nabla\phi_g^*(\underline{r})+\sigma_{rg}(\underline{r})\,\phi_g^*(\underline{r})-\sum_{g'}\sigma_{g'g}\,\phi_{g'}^*(\underline{r}) \qquad (2.1.9)$$
$$=\sigma_{fg}\sum_{g'}\chi_{g'}\frac{\nu}{k}\,\phi_{g'}^*(\underline{r}),\quad (g=1,2,\ldots,G)\ ,$$

whereas that of the source problem is

$$-\nabla\cdot D_{ng}(\underline{r})\nabla\phi_g^*(\underline{r})+\sigma_{rg}(\underline{r})\,\phi_g^*(\underline{r})-\sum_{g'}\sigma_{g'g}\,\phi_{g'}^*(\underline{r}) \qquad (2.1.10)$$
$$=\sigma_{fg}\sum_{g'}\chi_{g'}\,\nu\,\phi_{g'}^*(\underline{r})+S_g(\underline{r}),\quad (g=1,2,\ldots,G).$$

The multigroup diffusion **equations 2.1.7** and **2.1.8** are not self-adjoint owing to the scattering and fission production cross sections. However, the diffusion and removal terms constitute a symmetric system. In addition, the code can perform three types of criticality search, where physical attributes are adjusted by a multiplicative parameter to achieve a required effective multiplication $k=k_{required}$, usually 1. Although three types of search facilities are supplied with POW3D, advanced users may specify their own requirements by writing special subroutines which are interfaced easily with the code.

3. DISCRETISATION OF THE DIFFERENTIAL EQUATIONS

3.1 Temporal Integration of the Time-dependent Neutron Diffusion Equation

Experience with neutron diffusion calculations has shown that discretisations performed by finite difference methods lead to sets of equations that are amenable to numerical solution. The temporal method is essentially based on a routine designed by Pollard [1973] for the two-dimensional code POW. The method (**appendix A**) uses the direct space-time integration theory proposed by Stacey [1969] and includes the precursor concentration in a natural way.

3.2 Spatial Integration of the Neutron Diffusion Equation

The finite difference discretisation for a particular energy group of the 3D diffusion equation given in **appendix B** follows the basic concept of Wachspress [1966]. POW3D handles an (x,y,z) geometry system for three spatial dimensions, however (r,z) geometry is supported in two dimensions as well. It is normal for the reactor design to be oriented so that structural variation is minimal in the (z) direction; this is accommodated in the code. To avoid I/O overhead penalties, users of the code should construct similar models. Internal material boundaries must be associated with the resulting grid, so that internal boundary conditions may be satisfied. Partial integration of the diffusion **equation 2.1.1** is carried out over integration boxes surrounding each grid point. Details are presented in **appendix B**.

With the fine details as given in **appendices A** and **B**, the discrete form of the multigroup time-dependent diffusion **equation 2.1.1** is

$$
\begin{aligned}
& {}_g a^1_{ijk;p}\ {}_g\phi_{ij+1k;p} + {}_g a^2_{ijk;p}\ {}_g\phi_{i-1\,jk;p} + {}_g a^3_{ijk;p}\ {}_g\phi_{i\,j-1\,k;p} \\
& + {}_g a^4_{ijk;p}\ {}_g\phi_{i+1\,jk;p} + {}_g a^6_{ijk;p}\ {}_g\phi_{ijk+1;p} + {}_g a^7_{ijk;p}\ {}_g\phi_{ijk-1;p} \\
& + \{ {}_g a^5_{ijk;p} + \sum_l \left[\sigma_{rg}(m_l,\bar{t}_p) + 2/(v_g\,\delta t)\right] v^l_{ijk} \}\ {}_g\phi_{ijk;p} \\
& - \sum_{g'} \sum_l \left[\sigma_{gg'}(m_l,\bar{t}_p) + \chi_{(2)g}(\delta t)\frac{\nu}{k}\sigma_{fg'}(m_l,\bar{t}_p)\right] v^l_{ijk}\ {}_{g'}\phi_{ijk;p} \\
= & - {}_g a^1_{ijk;p}\ {}_g\phi_{ij+1k;p-1} - {}_g a^2_{ijk;p}\ {}_g\phi_{i-1\,jk;p-1} - {}_g a^3_{ijk;p}\ {}_g\phi_{i\,j-1\,k;p-1} \\
& - {}_g a^4_{ijk;p}\ {}_g\phi_{i+1\,jk;p-1} - {}_g a^6_{ijk;p}\ {}_g\phi_{ijk+1;p-1} - {}_g a^7_{ijk;p}\ {}_g\phi_{ijk-1;p-1} \\
& - \{ {}_g a^5_{ijk;p} + \sum_l \left[\sigma_{rg}(m_l,\bar{t}_p) - 2/(v_g\,\delta t)\right] v^l_{ijk} \}\ {}_g\phi_{ijk;p-1} \\
& + \sum_{g'} \sum_l \left[\sigma_{gg'}(m_l,\bar{t}_p) + \chi_{(1)g}(\delta t)\frac{\nu}{k}\sigma_{fg'}(m_l,\bar{t}_p)\right] v^l_{ijk}\ {}_{g'}\phi_{ijk;p-1} \\
& + 2\sum_d \chi_{dg}\,\lambda_d\,G_0(\lambda_d\,\delta t)\ {}_dC_{ijk;p-1} + {}_gS_{ijk}(\bar{t}_p) \quad ,
\end{aligned}
$$

and for **equation 3.1.2** is

$$
\begin{aligned}
{}_dC_{ijk;p} = {}_dC_{ijk;p-1}\,e^{-\lambda_d\delta t} + \beta_d\,\delta t \sum_{g'}\sum_l \frac{\nu}{k}\sigma_{fg'}(m_l,\bar{t}_p)\,v^l_{ijk} \quad \times \\
\times\ [F_1'(-\lambda_d\,\delta t)\ {}_{g'}\phi_{ijk;p-1} + F_2(-\lambda_d\,\delta t)\ {}_{g'}\phi_{ijk;p}] \quad .
\end{aligned}
\qquad (3.2.2)
$$

4. NUMERICAL SOLUTION OF THE NEUTRON DIFFUSION EQUATIONS

4.1 Problems Solved by POW3D

Several numerical computations are possible with POW3D; they may be classified into five basic categories:

(i) solution of the steady-state eigenvalue equation;

(ii) solution of the adjoint steady-state eigenvalue equation;

(iii) solution of the time-dependent equation;

(iv) solution of the steady-state neutron source equation; and

(v) solution of the adjoint steady-state neutron source equation.

4.2 Eigenvalue Problems

Apart from being important in its own right, the eigenvalue problem is the first step in a kinetics study. Here it is necessary to perform a steady-state eigenvalue calculation to determine the multiplicative factor for the assembly. The steady-state form of the eigenvalue problem can be expressed in matrix operator terms as

$$M \underline{\phi} = \frac{1}{k} F \underline{\phi} \ , \qquad (4.2.1)$$

where $M \underline{\phi}$ is a finite difference approximation (obtained, as indicated in **appendix B**, for the left hand side of **equation 2.1.7**):

$$- \nabla . D_{ng}(\underline{r}) \nabla \phi_g(\underline{r}) + \sigma_{rg}(\underline{r}) \phi_g(\underline{r}) - \sum_{g'} \sigma_{gg'}(\underline{r}) \phi_{g'}(\underline{r}) \ \text{, and}$$

$F \underline{\phi}$ is the equivalent approximation for the right hand side (without the $1/k$ factor):

$$\chi_g \sum_{g'} \nu\sigma_{fg'}(\underline{r}) \phi_{g'}(\underline{r}) \quad (g=1,2,\ldots,G \ , g'=1,2,\ldots,G) \ .$$

In more detailed form, **equation 4.2.1** is

$$\begin{bmatrix} -\nabla.D_1\nabla + \sigma_{r1} & -\sigma_{12} & -\sigma_{13} & \cdots \\ -\sigma_{21} & -\nabla.D_2\nabla + \sigma_{r2} & -\sigma_{23} & \cdots \\ -\sigma_{31} & -\sigma_{32} & -\nabla.D_3\nabla + \sigma_{r3} & \cdots \\ \cdot & \cdot & \cdot & \cdots \\ \cdot & \cdot & \cdot & \cdots \\ \cdot & \cdot & \cdot & \cdots \end{bmatrix} \begin{bmatrix} \underline{\phi}_1 \\ \underline{\phi}_2 \\ \cdot \\ \cdot \\ \cdot \\ \underline{\phi}_G \end{bmatrix}$$

$$= \frac{1}{k} \begin{bmatrix} \chi_1\nu\sigma_{f1} & \chi_1\nu\sigma_{f2} & \chi_1\nu\sigma_{f3} & \cdots \\ \chi_2\nu\sigma_{f1} & \chi_2\nu\sigma_{f2} & \chi_2\nu\sigma_{f3} & \cdots \\ \chi_3\nu\sigma_{f1} & \chi_3\nu\sigma_{f2} & \chi_3\nu\sigma_{f3} & \cdots \\ & & & \cdots \\ \cdot & \cdot & \cdot & \cdots \\ \cdot & \cdot & \cdot & \cdots \end{bmatrix} \begin{bmatrix} \underline{\phi}_1 \\ \underline{\phi}_2 \\ \cdot \\ \cdot \\ \cdot \\ \underline{\phi}_G \end{bmatrix} \qquad (4.2.2)$$

For the complete matrix M of **equation 4.2.2**,

$$\sigma_{ij} \neq \sigma_{ji} \ (i \neq j) \ ,$$

consequently there is no symmetry relation equivalent to that for any diagonal component of **equation 4.2.2**. Row diagonal dominance

$$m_{ii} \geq \sum_{j \neq i} | m_{ij} |$$

no longer applies. The matrix, however, is diagonally dominant by column:

$$m_{ii} \geq \sum_{j \neq i} | m_{ji} | \ .$$

Each of the diagonal matrix components of M are sparse, real, irreducible, symmetric, positive definite (*i.e.* Stieltjes) block tridiagonal matrices with elements

$$m_{ij} = m_{ji}, \quad (i = 1,2,...,N; j = 1,2,...,N),$$

$$m_{ij} \leq 0 \quad i \neq j \ ,$$

$$m_{ii} \geq \sum_{j \neq i} | m_{ij} | \ ,$$

whereas the source terms on the right are non-negative (and not all zero).

Birkhoff and Varga [1958] demonstrated the existence and uniqueness of a positive solution for **equation 4.2.1**, with a corresponding real eigenvalue greater in modulus than all other eigenvalues of the matrix equation, by application of the Perron-Frobenius theory of non-negative matrices. Solution of the eigenvalue matrix equations involves the determination of only the largest modulus eigenvalue and its corresponding eigenvector. The simplest approach is the power method [Wachspress 1966], where

$$\underline{\phi}^{(n+1)} = \frac{1}{k^{(n)}} M^{-1} F \underline{\phi}^{(n)} \ ,$$

$$\frac{1}{k^{(n+1)}} = \frac{1}{k^{(n)}} \frac{< \underline{w} , M^{-1} F \underline{\phi}^{(n)} >}{< \underline{w} , M^{-1} F \underline{\phi}^{(n+1)} >} \ ,$$

and <,> denotes inner product and $\underline{w}$ is an arbitrary weighting vector.

The rate of convergence is determined by the dominance ratio $|k_2| / |k_1|$, the ratio of the two largest moduli eigenvalues. It is slow for most problems, and notoriously slow for those with a high dominance ratio. For large reactors, this is typically of the order of 0.95 or larger. Although fast reactors are more compact and have lower dominance ratios [Ferguson and Derstine 1977], even these may be as high as 0.9. Because convergence is slow, it is accelerated by Chebyshev extrapolation. In practice, it is more attractive to extrapolate neutron sources rather than the flux. In POW3D, the approach is similar to that adopted in POW, where weighted averages of three earlier fission sources are used [Pollard 1973].

Determination of the largest eigenvalue requires repeated calculation of

$$M^{-1} F \underline{\phi}^{(n-1)} \ . \qquad (4.2.3)$$

4.3 Time-dependent and Source Problems

Time-dependent and source problems require solution of the matrix equation

$$A \underline{\phi} = \underline{s} \ .$$

For the kinetics problem $\underline{s}$ collects all the quantities $\underline{\phi}_{;p-1}$ and $\underline{C}_{;p-1}$ from the previous time step, as well as any external source. Solution of both the time-dependant and steady-state fluxes requires the calculation of

$$A^{-1} \underline{s} \ . \qquad (4.3.1)$$

4.4 Matrix Solution

Of course, M^{-1} or A^{-1} are never computed, and iterative techniques normally are used to solve the linear systems of equations. The symmetry and row diagonal dominance of M is lost for the multigroup form but column diagonal dominance remains. Consequently, until now the Gauss-Seidel (GS) technique has been the standard iterative solution method for the outer energy group block of **equation 4.2.2**.

Equation 4.2.2 is solved by block techniques. Fortunately, however, more advanced techniques are available for the inner blocks. The new iterative technique MINI does not require symmetry and provides an appropriate and novel approach for hastening convergence of the outer block.

The matrix A for source and time-dependent problems also lacks symmetry, however, the same conclusions for MINI apply. In addition, it is not always possible to guarantee that for the time-dependent equation the matrix is even column dominant. Fortunately, practical studies have shown that there are no problems in obtaining converged results for stable configurations.

5. OVERVIEW OF POW3D COMPUTATIONAL STRUCTURE

The schematic representation of POW3D embodies concepts of programming and mathematics, to give an overall flow of logic. Not all the concepts have been explained; this will be done in sections (MINI, **sections 6 and 7**; ICCG, **sections 6** and **7**; energy groups and region rebalance, **section 8**). Some of these concepts, such as the calculation of source components, look remarkably simple. This is misleading since quite large time savings were obtained by reworking that part of the code. These subroutines were worked over so often that it is no longer possible to recognise the original physical intent of the calculation from the code. This leads to the dilemma of choosing between a highly efficient code at run time, or a set of well-structured, self-documenting and understandable instructions. Computer scientists [Dahl *et al.* 1972] would emphatically favour the latter; however, the former philosophy is chosen for all the important sections of POW3D because the time savings are so enormous that the range of problems handled by POW3D is greatly enhanced.

A logic summary of POW3D follows, but some details are simplified and others omitted:

```
procedure main
$t=t_o$
*calculate* $t_p$, cross sections and coefficient matrix

*if* kinetics calc.   *then* $\underline{R}_g$ = rhs eqn 3.2.1 for $g = 1,2,...,G$
*if* source calc.     *then* $\underline{R}_g$ = external source for $g = 1,2,...,G$
                      *else* $\underline{R}_g = 0$ for $g = 1,2,...,G$

*generate* trial fluxes  $\underline{\Phi}_g$; $g = 1,2,...,G$

$\chi_g = \chi_{pg}(1-\beta) + \sum_d \chi_{dg}\beta_d$
 *calculate* fission production   $\underline{F} = \sum_g \frac{\nu}{k} \sigma_{fg} \cdot \underline{\Phi}_g$

*if* eigenvalue calc.   *then* *renormalise* fluxes to given power level
*if* not kinetics       *then* p=0

*loop* kinetics from 1 to p  ($t_f = t_o + \sum_{p=1}^{P} \delta t_p$)
  *loop* outer = 1 to maximum outers
    *calculate* number of upscatter passes (upmax)
    *** POSSIBLE GLOBAL COARSE MESH SOLUTION OF OUTER CALCULATION ***

    *if* global group-space rebalance
    *then begin* collapse
            $\hat{\underline{\Phi}}_g \leftarrow \underline{\Phi}_g$, $\hat{M}_{gg'} \leftarrow M_{gg'}$
            $\hat{\underline{S}} \leftarrow \underline{S}_g$ ; $g = 1,2,...,G$;  $g' = 1,2,...,G$
            call INNER ($\hat{M}$, $\hat{\underline{\Phi}}$, $\hat{\underline{S}}$, upmax,coarse mesh)
    *end*

    *** FINE MESH CALCULATION OF OUTER CALCULATION ***
  CALL INNER (M, $\underline{\phi}$, $\underline{s}$ , upmax,fine mesh)
   if kinetics then $\chi_g = \chi_{(2)g}(\delta t)$
            else $\chi_g = \chi_{pg}(1-\beta) + \sum_d \chi_{dg}\beta_d$

      calculate fission production    $\underline{F} = \sum_g \frac{\nu}{k}\sigma_{fg'}\,\underline{\phi}_{g'}$
   if eigenvalue calc. then calculate k
   if converged then finish for eigenvalue and source calcs.
   if search and partial convergence then begin adjust parameters
                                             calculate necessary matrix terms
                                       end
   extrapolate $\underline{F}$ with Chebyshev method
   if eigenvalue calc. then renormalise $\underline{\phi}_g$;g = 1,2,...,G
  end loop outer
  update precursor concentration
  $t = t + \delta t$
  adjust parameters for kinetics variations
  recompute necessary coefficient matrix terms

end loop kinetics
procedure INNER (M, $\underline{\phi}$ , $\underline{s}$, upmax , mesh)
loop $l_{up}$ from 1 to upmax
   $g_1=1$, $g_f=G$, $\Delta g = 1$
   if $l_{up} \neq 1$ then $g_1 = g_{up}$ (first group with upscatter into it)
   if adjoint then $g_1 = G$ , $g_f=1$, $\Delta g = -1$
   loop from $g = g_1$ to $g_f$, step $\Delta g$
        if ($g = g_{up}$ and $l_{up}=1$ and convergence difficult, mesh fine)
        then apply energy rebalance $\underline{\phi}_{g'}$; $g' = 1,2,...,G$
        if MINI SOURCE then   $\underline{s}_g = \underline{R}_g + \chi_g \underline{F} + \sum_{g' \neq g} \sigma_{gg'}\,\underline{\phi}_{g'}$        (shown only for eigenvalue problem)
                                $+ \sum_{g'>g} \gamma_{g'}\,\sigma_{gg'}\,\underline{\phi}_{g'}$
                           $M_{gg} = M_{gg} + \sum_{g'>g} \gamma_{g'}\,\sigma_{gg'}$
                      else $s_g = \underline{R}_g + \chi_g \underline{F} + \sum_{g' \neq g} \sigma_{gg'}\,\underline{\phi}_{g'}$ (GS source)
        if (convergence difficult and method $\neq$ SLOR and mesh fine)
        then region rebalance $\underline{\phi}_g$
        if (convergence difficult and method = SLOR with $\omega$ determined and mesh fine)
        then region rebalance $\underline{\phi}_g$
        if coarse mesh option
           then if disjunctive partitioning and region balance weighting
                     then solve $M_{gg}\,\underline{\phi}_g = \underline{s}_g$ by MINI
                     else solve $M_{gg}\,\underline{\phi}_g = \underline{s}_g$ by direct means
        if (fine mesh option)
          then solve $M_{gg}\,\underline{\phi}_g = \underline{s}_g$ by SLOR, MINI or ICCG, or combinations of same
     end loop g

end loop $l_{up}$

end procedure INNER
```

6. ITERATIVE SOLUTION OF LINEAR EQUATIONS IN POW3D

6.1 Introduction

Three iterative techniques,

(i) MINI,

(ii) SLOR, and

(iii) Conjugate gradient,

are available within POW3D to solve the large system of sparse linear equations at the heart of the code. The authors believe MINI to be the most suitable and have made this the default option for the code. The SLOR method is well known [Young 1971] and is treated no further other than to consider the appropriate data handling.

When the code POW3D was planned, there was a need for a more efficient iterative technique than SLOR. Although it is a robust and mathematically well-established method, something better was sought since the third dimension was included. MINI was devised and tested in a two-dimensional mode [Barry and Pollard 1977] and the promise it showed encouraged its implementation in three dimensions. This promise has been more than fulfilled for test and real world reactor problems in both two and three spatial dimensions. It also gave an unexpected bonus by providing an alternative to the Gauss-Siedel (GS) method for accelerating convergence in energy.

6.2 MINI

Details of MINI are given by [Barry and Pollard 1977, 1978, 1981], however, only its implementation in a three-dimensional code is discussed here. The method was prompted by a novel concept for making an algorithm more implicit [Noble 1975]. MINI applies this concept to solving systems of linear equations. Essentially, it takes the unknowns at any stage of the GS method that are not updated and writes them implicitly in terms of the updated diagonal term.

The method was developed and refined with an experimental approach. Its final form was greatly influenced by discoveries resulting from the use of computer graphics to investigate various formulations of implicit schemes. A general proof for its convergence has yet to be established, although there is a proof for a restricted set of circumstances. Absence of the proof, although disappointing from a mathematical viewpoint, is not unusual in numerical computations. Experience indicates that it has always converged for any system of equations for which GS is an appropriate method.

6.3 Multi-dimensional MINI

Although MINI can be used in a point form, it is much more effective when used as a block technique (as it is in POW3D). The generalised MINI block approach is now given for the linear systems arising from various neutron diffusion equations, written in the form

$$\sum_{j=1}^{N} a_{ij} x_j = b_i, \quad (i=1,2,...,N). \tag{6.3.1}$$

A block solution process is used for non-overlapping ordered partitions m = 1,2,...,M. If M=N, the block process degenerates to a point method. For example, m=1 might denote an x-line on which (y,z) is fixed, and m=2 its immediate neighbouring line. Alternatively, it may denote an (x,y) plane, or a particular energy group. In POW3D, MINI works as a block system that is three levels deep. An energy layer contains an (x,y,z) layer which, in turn, contains an (x,y) layer which finally contains an (x) or (y) layer. In default, the final choice of layer is made by POW3D which selects the most detailed of the (x) and (y) directions as the inner one.

The following sets of indices are introduced for a partial stage of iteration pass n through **equation 6.3.1**:

J_m is the set of indices j for elements x_j to be updated together as a block,

J_m^- is the set of indices j for elements x_j, that have been updated,
$J_m^- = J_1 \cup J_2 \cup \cdots \cup J_{m-1}$.

and J_m^+ is the set of indices j for elements x_j, that have not been updated,
$J_m^+ = J_{m+1} \cup J_{m+2} \cup \dots \cup J_M$.

In this notation, the basic block GS iteration process may be written as

$$J_m^- a_{ij} x_j^{(n)} + J_m a_{ij} x_j^{(n)} + J_m^+ a_{ij} x_j^{(n-1)} = b_i \qquad (6.3.2)$$

$$i \in J_m, \; m = 1,2,\dots,M,$$

where the J's used in **equation 6.3.2** also denote partial summation over the index j (for example, $J_m^- = \sum_{j \in J_m^-}$).

The MINI approach for hastening convergence of the basic process given by **equation 6.3.2**, seeks to use a better estimate for the last term $J_m^+ a_{ij} x_j^{(n-1)}$ on the left hand side. The estimate would be better if the components $x_j^{(n-1)}$ were replaced by those of the iterative pass $x_j^{(n)}$. In MINI, the unknown current pass term $x_j^{(n)}$, is replaced by the diagonally coupled term

$$x_j^{(n-1)} + \gamma_{ij}^{(n)} (x_i^{(n)} - x_i^{(n-1)}),$$

where $\gamma_{ij}^{(n)}$ are extrapolating parameters yet to be determined. If they are known the GS process is transformed by MINI to

$$J_m a_{ij} x_j^{(n)} + (J_m^+ a_{ij} \gamma_{ij}^{(n)}) x_i^{(n)} \qquad (6.3.3)$$

$$= b_i - J_m^- a_{ij} x_j^{(n)} - J_m^+ a_{ij} (x_j^{(n-1)} - \gamma_{ij}^{(n)} x_i^{(n-1)}) \;,$$

$$i \in J_m, \quad m = 1,2,\dots,M.$$

The block of equations **6.3.3** for the set J_m may be solved by

(i) a direct method,

(ii) a further iterative process, or

(iii) even recursive or repeated use of the MINI process.

Method (iii) is the standard option available in POW3D.

The determination of $\gamma_{ij}^{(n)}$ needs to be undertaken. The problems of interest in neutron diffusion studies have positive solutions x_i. The MINI iterative process requires that even intermediate solutions $x_i^{(n+1)}$ be positive. As a step towards ensuring positive solutions, the right hand side of **equation 6.3.3** should be positive. This is achieved by applying the following stringent but not absolutely necessary conditions to the extrapolating parameters:

$$0 \le \gamma_{ij}^{(n)} < x_j^{(n)}/x_i^{(n)} \;. \qquad (6.3.4)$$

To assume positive solutions for $x_j^{(n)}$, $j \in J_m$, the block matrix of coefficients with elements on the left hand side of **equation 6.3.3**

$$a_{ik} + (J_m^+ a_{ij} \gamma_{ij}^{(n)}) \delta_{ik}, \quad i \in J_m, \; k \in J_m \;,$$

where δ_{ik} = 1 if i = k
0 otherwise,

must not contain excessively large γ's. Although somewhat restrictive, let g_m be an upper γ limit for the nth block, provided that

$$\gamma_{ij}^{(n)} < g_m \ ,$$

the block matrix of coefficients, will yield positive solutions. When the results are collected, the overall restriction is

$$0 \le \gamma_{ij}^{(n)} < g_{ij}^{(n)} \ ,$$

where

$$g_{ij}^{(n)} = \min (g_m, x_j^{(n)}/x_i^{(n)}) \ .$$

A preliminary calculation of the block γ limits (g_m) could be carried out, or g_m could be lowered empirically if there were negative solutions; however, here we use

$$g_m = 1 \ . \qquad (6.3.5)$$

This limit is used throughout the calculations, even though occasional negative intermediate solutions may be found for non-symmetric matrices. Of course, it is necessary for the process to converge; the restrictions to maintain positive intermediate solutions are only part of the overall restrictions. **Condition 6.3.5** then becomes desirable.

The working equations for the extrapolating parameters [Barry and Pollard 1977, 1978] are derived by assuming that they vary slowly between iterations, *i.e.*

$$\gamma_{ij}^{(n)} = \gamma_{ij}^{(n-1)} \ .$$

The parameters are given by

$$\gamma_{ij}^{(n)} = \begin{array}{ll} \bar{\gamma}_{ij}^{(n)} & \text{if } \bar{\gamma}_{ij}^{(n)} \le g_{ij}^{(n)} \\ g_{ij}^{(n)} & \text{otherwise} \end{array} \ ,$$

where

$$\bar{\gamma}_{ij}^{(n)} = \begin{array}{ll} | \delta_j^{(n)}/\delta_i^{(n)} | & \text{if } | \delta_j^{(n)}/\delta_i^{(n)} | \le 1 \\ | \delta_i^{(n)}/\delta_j^{(n)} | & \text{otherwise} \ , \end{array} \qquad (6.3.6)$$

and $\delta_j^{(n)} = x_j^{(n)} - x_j^{(n-1)}$.

In this case, $\gamma_{ij}^{(0)}$ is obtained either from the last overall pass through the current layer, or it is taken to be zero (a GS iteration). The expolating parameters are not stored, but rather $\delta_i^{(n)}$ is saved after each iterative pass for use in the next round with **equation 6.3.6**.

For the purposes of computation, an additional rule is introduced to stop the right hand side from becoming zero in certain instances when

$$\gamma_{ij}^{(n)} = x_j^{(n)}/x_i^{(n)} \ .$$

A slightly smaller value is required by the inequality **6.3.4**, so

$$g_{ij}^{(n)} = \min (g_m, 0.99999\, x_j^{(n)}/x_i^{(n)}) \ .$$

6.4 MINI for the Inner Two Layers

Implementation of MINI within POW3D is now presented for the inner two layers. The presentation is given in terms of a physical interpretation of the neutron diffusion equation. Finite difference representations are based on those given in **appendices A** and **B**, however, some variations will soon become apparent. Alternatively, the MINI implementation could be discussed in the general matrix form of **section 6.3**, applied directly to the finite difference equations of **appendices A** and **B**; however, the physical approach is more enlightening.

No matter which neutron diffusion problem is studied, the linear system of finite difference equations arising from any spatial discretisation may be considered to have the general form

$$-\nabla . D_g \nabla \phi_g + \sigma_{rg} \phi_g = s_g \ , \qquad (6.4.1)$$

where σ_{rg} denotes all the non-leakage (or removal) terms and s_g denotes the scattering and fission contributions based on previous estimates of the flux and any other neutron source. The solution process for the spatial dimensions iterates through the 'layers' as follows:

(i) z-direction 'between (x,y) plane' inners for the mth pass;

(ii) '(x,y) plane' inners for the nth pass (and mth between plane pass); and

(iii) x line direction solution (n,m). (Solved with a direct method.)

For ease of presentation, consider D to be constant, as is the mesh width h_y. If group dependence is dropped, the numerical approximation for leakage in the y direction becomes

$$-\frac{\partial}{\partial y} D \frac{\partial \phi}{\partial y} \approx C_y (-\phi_{j-1} + 2\phi_j - \phi_{j+1}) \ ,$$

where $C_y = D/h_y^2$, and ϕ_{j-1}, ϕ_j, ϕ_{j+1} are successive values of flux along a line in the y direction. As the iteration proceeds in this direction, the flux ϕ_{j+1} is not available for the present pass n. In the normal approach, the approximation for the y-direction leakage is

$$-\left[\frac{\partial}{\partial y} D \frac{\partial \phi}{\partial y}\right]^{(n,n-1)}_{(m)} \approx C_y \left[-\phi_{j-1}^{(n,m)} + 2\phi_j^{(n,m)} - \phi_{j+1}^{(n-1,m)}\right] \ ,$$

in which emphasis is given, on the left, to the appearance of the fluxes from the two iterations n-1 and n. The usual MINI approximation

$$\phi_{j+1}^{(n,m)} \approx \phi_{j+1}^{(n-1,m)} + \gamma_{yj}^{(n-1,m)} \left[\phi_j^{(n,m)} - \phi_j^{(n-1,m)}\right] \ ,$$

is used and a better leakage approximation

$$-\left[\frac{\partial}{\partial y} D \frac{\partial \phi}{\partial y}\right]^{(n,n)}_{(m)} \approx -\left[\frac{\partial}{\partial y} D \frac{\partial \phi}{\partial y}\right]^{(n,n-1)}_{(m)} - \\ - C_y \gamma_y^{(n-1,m)} (\phi^{(n,m)} - \phi^{(n-1,m)})$$

is obtained, where the subscript j, denoting the particular line for γ and ϕ, has been omitted. Similarly

$$-\left[\frac{\partial}{\partial z} D \frac{\partial \phi}{\partial z}\right]^{(n)}_{(m,m)} \approx -\left[\frac{\partial}{\partial z} D \frac{\partial \phi}{\partial z}\right]^{(n)}_{(m,m-1)} - \\ - C_z \gamma^{(.,m-1)} (\phi^{(n,m)} - \phi^{(.,m-1)}) \ ,$$

where the notation (.,m-1) denotes results obtained from converged fluxes of the previous z pass. (The diagonal leakage term always remains implicit as $\phi^{(n,m)}$.) When the above results are collected, the approximation for **equation 6.4.1** becomes

$$-\left[\frac{\partial}{\partial x} D_g \frac{\partial \phi_g}{\partial x}\right]^{(n,n)}_{(m)} - \left[\frac{\partial}{\partial y} D_g \frac{\partial \phi_g}{\partial y}\right]^{(n,n-1)}_{(m)} - \\ - \left[\frac{\partial}{\partial z} D_g \frac{\partial \phi_g}{\partial z}\right]^{(n)}_{(m,m-1)} + \\ + \left[\sigma_{rg} - C_{yg} \gamma_{yg}^{(n-1,m)} - C_{zg} \gamma_{zg}^{(.,m-1)}\right] \phi_g^{(n,m)} \\ = s_g - \left[C_{yg} \gamma_{yg}^{(n-1,m)} \phi_g^{(n-1,m)} + C_{zg} \gamma_{zg}^{(.,m-1)} \phi_g^{(.,m-1)}\right] \ , \qquad (6.4.2)$$

which is also the result for a general mesh with appropriate changes to C_{yg} and C_{zg}. **Equation 6.4.2** is similar to the normal approximations, except for the reduced removal terms and the compensating terms on the right.

Like SLOR, the MINI approach (**equation 6.4.2**) is an extrapolation of the basic GS method ($\gamma = 0$). Either MINI or SLOR (or for that matter GS) may be used for a particular direction. This enables a combined approach using the three methods for the whole reactor calculation. When SLOR is used for the (x,y) plane and MINI is used between planes (z), the diagonal terms in the SLOR matrix will alter because of the changing removal term (**equation 6.4.2**). Ideally, the optimum SLOR relaxation parameters should be continually re-determined, necessitating computational overhead, however, in the POW3D implementation they are assumed to be insensitive to such changes. (This is rather questionable but SLOR-MINI was never considered as a suitable option even though it was easily implemented in the code.)

6.5 Conjugate Gradient Method

The linear system of equations

$$A\underline{x} = \underline{b}$$

is once more considered to correspond to a one-energy subsystem of **equation 4.2.2**. The matrix is symmetric, irreducible, has positive diagonal elements with negative or zero off-diagonal elements, and is both row and column dominant. The conventional conjugate gradient ~~method~~ for solving $A\underline{x} = \underline{b}$ is greatly improved through an incomplete Choleski refinement [Meijerink and van der Vorst 1977]. The incomplete Choleski conjugate gradient (ICCG) method involves an approximate factoring of

$$A = LDL^T ,$$

where L is a lower triangular matrix retaining the sparseness of A, and D is a diagonal matrix. The evaluation of elements in L and D is defined by **equation 6.5.2**. Once the approximate LDL^T decomposition is complete, the algorithm is as follows

Select $\underline{x}_o$ as an initial approximation to $\underline{x}$,

$$\underline{r}_o = \underline{b} - A\underline{x}_o ,$$

$$\underline{t}_o = (LDL^T)^{-1}\underline{r}_o ,$$

$$\alpha_i = <\underline{r}_i,(LDL^T)^{-1}\underline{r}_i> / <\underline{t}_i,A\underline{t}_i> ,$$

$$\underline{x}_{i+1} = \underline{x}_i + \alpha_i\underline{t}_i$$

$$\underline{r}_{i+1} = \underline{b} - A\underline{x}_{i+1} \text{ or } \underline{r}_i - \alpha_i A\underline{t}_i ,$$

$$\beta_i = <\underline{r}_{i+1},(LDL^T)^{-1}\underline{r}_{i+1}> / <\underline{r}_i,(LDL^T)^{-1}\underline{r}_i> , \text{ and}$$

$$\underline{t}_{i+1} = (LDL^T)^{-1}\underline{r}_{i+1} + \beta_i\underline{t}_i . \qquad (6.5.1)$$

Computation of the residual $\underline{r}_{i+1}$ may be performed more economically in the form

$$\underline{r}_{i+1} = \underline{r}_i - \alpha_i A\underline{t}_i .$$

Reid [1971] has advocated that this is the preferred way for computing the residual. The longer form

$$\underline{r}_{i+1} = \underline{b} - A\underline{x}_{i+1}$$

is used in POW3D because large errors in the initial estimate for $\underline{x}$ may cause roundoff errors to swamp iterative corrections to $\underline{r}_{i+1}$ based on $\underline{r}_i$. We have found that false convergence occurs when the initial estimates involve significant error. With the use of single precision arithmetic to resolve the intermediate stages of the conjugate gradient algorithm and, at times, great uncertainty in estimates for the magnitude of the trial fluxes, the choice appears desirable.

In POW3D, the approximate (LDL^T) decomposition follows the traditional Choleski approach, whereby L and D are computed by

$$L_{ij} = a_{ij} - \sum_{k=1}^{j-1} L_{ik} d_{kk} L_{jk} , \qquad (6.5.2)$$

except $L_{ij} = 0$ if $a_{ij} = 0$,

$d_{ii} = 1 / L_{ii}$

The non-zero infill for L not only allows it to retain the original sparseness of A but an unexpected computational bonus is received. For the five or seven point finite difference operators used for the two and three-dimensional diffusion codes, the elements of L are those of the original matrix A. This result is established in **appendix C**.

7. COMPUTATIONAL IMPLEMENTATION OF ITERATIVE TECHNIQUES

7.1 Introduction

All to often, once a numerical technique has been formulated in mathematical terms, there is a tendency to dismiss the computational implementation as a trivial exercise. This is unfortunate because what appears most efficient from a straight mathematical viewpoint may not be so when limitations of computer capacity and function are considered. The storage required for neutron diffusion calculations is large; even the largest machine at the time when POW3D development was commenced did not have sufficient primary storage to hold all matrix coefficients and unknowns. Nor at the time was it realistic to anticipate any change in the immediate future, because reactor designers would attempt to incorporate more details in their existing models as computer facilities increased. Consequently, similar consideration must be given to adequate data structures and I/O handling as was given to the mathematical study of convergence.

The development of POW3D coincided with a major change in the IBM memory architecture. The code was designed to function efficiently in both memory environments. The form in which a technique is best implemented for a particular problem on a computer may appear at first glance quite different to that of the original mathematical definition. Such is the case for MINI and ICCG. We believe that implementation of the numerical techniques belongs to the discipline of mathematics. Accordingly, the ways in which MINI and ICCG function within POW3D are discussed in detail. The implementation of three-dimensional SLOR has many features in common with MINI, particularly in the data structures for the third dimension. Consequently, its implementation is not discussed further.

The philosophy of POW3D is based on the assumption that for any energy group, sufficient memory is available for a complete two-dimensional model, and for a two-dimensional subsection of a three-dimensional model. This was a practical assumption in view of then scientific computers. It was only in the third spatial dimension (and energy) that secondary data transfer is necessary. Arrays suitably dimensioned to hold the two-dimensional forms in POW3D were generated dynamically under the VARRAY system [Cox and Pollard 1978]. Under UNIX this is replaced with a C routine call to 'malloc' [Robinson unpublished]. This overcomes a weakness of FORTRAN requiring storage to be allocated explicitly. Should sufficient primary space within the requested region be unavailable to accommodate a two-dimensional section of the user's model, the calculation is terminated. The user may request a larger region or simplify the model.

At first sight, the availability of virtual storage systems appears to take responsibility for handling data transfers and the management of storage away from the scientific programmer. Blind acceptance of virtual storage presents problems because such systems are not universal. This affects the portability of any code which relies upon them exclusively. Limitations in the address range may present problems for complex reactor calculations, particularly when the number of energy groups modelled is large. (Development of POW3D commenced when the limit for the IBM system was eight megabytes without the use of extended architecture.) In addition, it is probably best for the user to organise data transfers personally. This is particularly so when the most efficient type of organisation is known in advance. The alternative is to rely on a more general paging algorithm which in some cases, exchanges data in an inappropriate fashion. These algorithms retain the most recently used pages and swap out the others. Frequently it is the page that was dispatched to the disk (because it was used some time ago) which is required next in iterative approaches rather than the page just used (and still in memory).

When initial planning for POW3D commenced, the AAEC operated an IBM360/65 computer with two megabytes of primary storage. It was known that this machine would not be replaced by an IBM3031 (with virtual facilities) for at least three years. Before then it was anticipated that a significant proportion of the code would be operational. It was this knowledge, plus the desire to maintain a code with some portability to non-IBM compatible installations which greatly influenced the data structures chosen. To achieve this, the code had to run without relying on virtual memory. It was decided, however, not to ignore the potential of the virtual systems for later generation computers, particularly as this would be the environment in which POW3D would earn its keep. A method that attempts to exploit the best of the real and virtual worlds evolved from these constraints.

Instead of relying on the virtual operating systems, and to permit the code to run efficiently in both the real and virtual modes, POW3D employs its own data handling techniques. Efficiency was originally achieved through the internal VIRTUL system [Pollard, unpublished] which relied on very fast Assembler I/O routines for direct access files [Cawley 1977]. Later the Assembler I/O routines were replaced with FORTRAN versions for portability considerations. The VIRTUL system allows data to be fetched and stored as though they are being retained there and the whole operation is transparent to the user. The basic unit of supported data corresponds to one (x,y) plane filled with information (an array on size N_xN_y, where N_x and N_y are the number of grid points in the respective directions) arising from the finite difference idealisation of the z coordinate direction. All data transfer is controlled through three indices; the type of data (*i.e.* matrix coefficients, flux, source, δ's), the relevant (z) plane indicator and, if appropriate, the energy group.

Under VIRTUL, all of the spare primary storage for the specified region size is divided into (x,y) planes, then allocated among the fluxes, coefficients, *etc.*, on a basis which can be varied by the code writer. Once all the available real store is used up, the remainder of the data is assigned to disc storage. The code writer is no longer concerned with the destination or source of the data. An attempt by the code (under VIRTUL) to read (or write) data assigned to real storage, simply results in a fast move operation which transfers data to or from the appropriate POW3D working array. In the case of disc storage, the fast I/O routines transfer the data directly between the appropriate arrays and the disc. (A memory sharing system is also used for some data.) This is an approach that functioned successfully under the older machine architecture.

For computers with the newer architecture, three approaches are available under the VIRTUL system which do not require alteration to the code, however, only (i) and (iii) are supported on the Fujitsu VP.

i. A region of arbitrary size may be specified (say 10 Mbytes). The data for arrays that do not fit into the requested region are transferred to and from disc storage directly through read/write instructions within the VIRTUL code.

ii. A region of small size may be specified and the virtual I/O facilities of the computer operating system allowed instead to perform the data transfers when the VIRTUL system requests a transfer.

iii. A maximum sized region may be specified, so that wherever possible, data transfers are memory to memory and are under direct control of the VIRTUL facility.

No matter which option is in force, the performance of the code is of greatest benefit when the computer is lightly loaded on large problems.

Unless sufficient fast storage is available to hold a minimum number of working planes of information, it is inefficient to commence the calculation. For mathematical manipulation, the code requires a number of working arrays (based on the x,y plane). These are in addressable memory. Before the advent of IBM's virtual operating system (MVS), the necessity of their presence prevented the code from commencing a calculation when memory was unavailable. Under MVS, and UNIX on virtual machines, the requirements are identical, but the calculation may commence even if there is little chance of obtaining the memory. In such an environment, there is always the possibility of page 'thrashing' when the machine is heavily loaded.

The different iterative algorithms are implemented in an environment in which the amount of directly addressable memory required for their efficient working is the same. Each uses the space in very different ways. For MINI and SLOR, double precision arithmetic was clearly necessary, whereas at the design stage ICCG was expected to perform successfully with single precision operations in most sections. (The LDL^T incomplete decomposition, however, is performed in double precision.) Because ICCG requires more work vectors than either SLOR or MINI, there were spatial condiderations which were more dominant factors (than speed) in avoiding double precision arithmetic where possible.

7.2 Data Block Structure for ICCG

To understand the computational implementation of ICCG, a modified mathematical form, suited to the data block structure of the three-dimensional form,

$$\begin{bmatrix} A_{11} & A_{12} & & & \\ A_{21} & A_{22} & A_{23} & & \\ & A_{32} & A_{33} & A_{34} & \\ & & & \ddots & \\ & & & A_{N_zN_{z-1}} & A_{N_zN_z} \end{bmatrix} \begin{bmatrix} \underline{\phi}_1 \\ \underline{\phi}_2 \\ \underline{\phi}_3 \\ \vdots \\ \underline{\phi}_{N_z} \end{bmatrix} = \begin{bmatrix} \underline{s}_1 \\ \underline{s}_2 \\ \underline{s}_3 \\ \vdots \\ \underline{s}_{N_z} \end{bmatrix} \qquad (7.2.1)$$

is used. (Remember that only the data is block structured, not ICCG.) In addition to the sparse matrix A and vectors $\underline{\phi}$ and $\underline{s}$, six working vectors of length $N_xN_yN_z$ are introduced for convenience. These are $\underline{t}$, $\underline{d}$, $\underline{r}$, $\underline{h}$, $\underline{u}$ and $\underline{v}$. The components of the block tridiagonal matrix A used in the formulation are as given in structure **7.2.1**, whereas the sub-blocks L_{ij} and D_{ii} (i,j = 1,2,...,N_z) represent the matrices appropriate to the incomplete Choleski decomposition of A, which is presumed to be computed previously. The vector notation $\underline{\phi}_k$ represents the kth subvector block of $\underline{\phi}$ of length N_xN_y when $\underline{\phi}$ is ordered $\underline{\phi} = \{\underline{\phi}_1, \underline{\phi}_2, ..., \underline{\phi}_{N_z}\}$. In the following description, any improperly formed matrix or vector operation is ignored.

Solution of $A\underline{\phi} = \underline{s}$ by ICCG

```
*start* perform incomplete $LDL^T$ decomposition of A
      choose $\underline{\phi}^{(0)}$
      initialise $\underline{t}^{(0)} = \underline{0}$, $\underline{d}^{(0)} = \underline{0}$, $\gamma^{(0)} = 1$, $\alpha=\beta=0$
*loop* i=1 to maximum iterations ($N_xN_yN_z$)
      $\underline{t}_1^{(i)} = \underline{d}_1^{(i-1)} + \beta \underline{t}_1^{(i-1)}$
      $\underline{\phi}_1^{(i)} = \underline{\phi}_1^{(i-1)} + \alpha \underline{t}_1^{(i)}$
      *if* (any $\alpha \underline{t}_1^{(i)} > \varepsilon$) *then* set not converged flag
      $\underline{r}_1^{(i)} = \underline{0}$
      *loop* k = 1 to $N_z$
            $\underline{t}_{k+1}^{(i)} = \underline{d}_{k-1}^{(i-1)} + \beta \underline{t}_{k-1}^{(i-1)}$
            $\underline{\phi}_{k+1}^{(i)} = \underline{\phi}_{k+1}^{(i-1)} + \alpha \underline{t}_{k+1}^{(i)}$
            *if* (any $\alpha \underline{t}_{k+1}^{(i)} > \varepsilon$) *then* set not converged flag
            *if* ($k=N_z$ & converged) *then* finish.
            $\underline{r}_k^{(i)} = \underline{s}_k - A_{kk}\,\underline{\phi}_k^{(i)} - A_{kk+1}\,\underline{\phi}_{k+1}^{(i)} - \underline{r}_k^{(i)}$
            $\underline{h}_k^{(i)} = L^{-1}_{kk}\,(\underline{r}_k^{(i)} - L_{kk-1}\,\underline{h}_{k-1}^{(i)})$
            $\underline{r}_{k+1}^{(i)} = A_{k+1\,k}\,\underline{\phi}_k^{(i)}$
    end loop k
    $\beta_1 = \beta_2 = \beta_3 = \gamma^{(i)} = 0$
    $k = N_z$
    $\underline{d}_k^{(i)} = L_{kk}^{-1}\,D_{kk}^{-1}\,\underline{h}_k$
    loop l = 1 to $N_z$
            $\underline{u}_k^{(i)} = A_{kk}\,\underline{t}_k^{(i)} + A_{kk-1}\,\underline{t}_{k-1}^{(i)} + A_{kk+1}\,\underline{t}_{k+1}^{(i)}$
            $\beta_2 = \beta_2 + \underline{d}_k^{(i)T}\,\underline{u}_k$
            $\beta_3 = \beta_3 + \underline{t}_k^{(i)T}\,\underline{u}_k$
            $\gamma^{(i)} = \gamma^{(i)} + \underline{r}_k^{(i)T}\,\underline{d}_k$
            $\underline{v}_k^{(i)} = A_{kk}\,\underline{d}_k^{(i)} + A_{kk+1}\,\underline{d}_{k+1}^{(i)}$
            $\underline{d}_{k-1}^{(i)} = (L_{k-1\,k-1}^{T})^{-1}\,(D_{k-1\,k-1}^{-1}\,\underline{h}_{k-1}^{(i)} - L_{k-1\,k}^{T}\,\underline{d}_k^{(i)})$
            $\underline{v}_k^{(i)} = \underline{v}_k^{(i)} + A_{kk-1}\,\underline{d}_{k-1}^{(i)}$      $(A_{kk-1} <=> A_{k-1\,k})$
            $\beta_1 = \beta_1 + \underline{d}_k^{(i)T}\,\underline{v}_k^{(i)}$
            $\beta_2 = \beta_2 + t_k^{(i)T}\,\underline{v}_k^{(i)}$
            $k = k - 1$
    end loop l
    $\beta = \gamma^{(i)} / \gamma^{(i-1)}$
    $\alpha = \gamma^{(i)} / (\beta_1 + \beta(\beta_2 + \beta(\beta_3)))$
end loop i
end
```

7.3 Computer Implementation of Three-dimensional ICCG

The implementation of ICCG in POW3D is presented in a format that lies somewhere between the high-level language and a mathematical definition. To understand the process adequately, the following data structures and notation are introduced.

$\underline{\phi}$ represents a storage vector (in VIRTUL memory) containing the three-dimensional flux, ordered in terms of directly transferable subvectors

$$\underline{\phi} = \{\,\underline{\phi}_k\,;\,k=1,2,\ldots,N_z\,\}\ .$$

Each $\underline{\phi}_k$ is also ordered

$$\underline{\phi}_k = \{\,\underline{\phi}_{jk}\,;\,j=1,2,\ldots,N_y\,\}$$ and, at the lowest level,

$$\{\,\underline{\phi}_{jk} = \ \phi_{ijk}\,;\,i=1,2,\ldots,N_x\,\}\ .$$

$\underline{s}$ represents the right hand side of **equation 7.2.1**, is stored in VIRTUL memory, and is structured like $\underline{\phi}$.

$\underline{e}$, $\underline{d}$, $\underline{t}$, $\underline{h}$ are work vectors held in VIRTUL memory and structured like $\underline{\phi}$. In the algorithm they are always subscripted.

A represents the matrix of **equation 7.2.1** held in VIRTUL memory. The matrix is block structured like **equation 7.2.1** and each block may be identified directly and transferred, *e.g.*

$A_{k,k-1}$, A_{kk}, or $A_{k,k+1}$.

$\underline{e}$, $\underline{d}$, $\underline{t}$, $\underline{x}$, $\underline{x}^*$, $\underline{r}$, $\underline{b}$, $\underline{u}$, $\underline{v}$ are temporary working vectors kept in real computer memory. Each represents a plane filled with information (N_xN_y locations and are subscripted in the ICCG algorithm). (Note that $\underline{e}$, $\underline{d}$, and $\underline{t}$ should not be confused with the larger vectors of the same name held in VIRTUL memory.) To assist identification they are not subscripted. Nine vectors are used for convenience of description, however, $\underline{u}$, $\underline{e}$ and $\underline{x}^*$ may share the same physical locations, as is the case for $\underline{v}$ and $\underline{b}$. Different names are selected only to assist understanding; $\underline{x}$, $\underline{x}^*$ and $\underline{b}$ are double precision vectors.

A_1, A_2, A_3 are three computer memory matrices each capable of holding an individual block submatrix of equation 7.2.1. To save storage and time throughout this operation, only non-zero matrix elements are actually stored and transferred.

$\leftarrow$ represents a fetch operation from VIRTUL to real memory. In each fetch operation, the information for a full *(x,y)* plane is transferred to the appropriate vector or matrix. The transfer may be achieved by one of the following mechanisms within the VIRTUL system:

(i) a direct access read from disc storage,

(ii) a move from other directly addressable memory, and

(iii) a move from virtual memory.

The fetch operation applies both to block subvectors, *e.g.*
$\underline{x} \leftarrow \underline{\phi}_i$ (the ith *(x,y)* plane of $\underline{\phi}$ is fetched),
or block matrices, *e.g.*
$A_1 \leftarrow A_{k\,k+1}$ (the kth off-diagonal block is fetched).
$\rightarrow$ represents a storage operation from real to VIRTUL memory.
It follows a similar convention to $\leftarrow$.

= represents computational assignment.

$\{t_j\}_i$ denotes the ith element of the jth subvector of $\underline{t}$.

$\{A_1\}_{ij}$ denotes the $(i,j)^{th}$ element of the block submatrix currently being held in A_1.

diag(A) is a vector whose components are the diagonal elements of A.

$[\underline{x}]$ denotes a matrix formed with $\underline{x}$ as its diagonal.

$\Leftrightarrow$ denotes equivalence of storage.

◺ denotes the lower triangular portion of a matrix.

◹ denotes the upper triangular portion of a matrix

Throughout the description any improperly formed summations, vector and matrix operations or data transfers are ignored. In the FORTRAN implementation, great care was taken to avoid unnecessary operations and storage requirements were minimised.

```
*** solves A $\underline{\phi}$ = $\underline{s}$  equation 7.2.1 ***
*** ICCG 3-D implementation ***
*** Assume estimates of $\underline{\phi}_k$ (k =1,2,...,$N_z$ ) have been made and stored ***
*** $LDL^T$ decomposition of A ***

loop k = 1 to $N_z$
    $A_1 \leftarrow A_{k-1\,k}$, $A_2 \leftarrow A_{k\,k}$, $A_3 \leftarrow A_{k\,k+1}$
    $\underline{t} \leftarrow \underline{e}_{k-1}$
    loop i = 1 to $N_xN_y$
        $e_i = \{diag(A_2)\}_i - \sum_{j=1}^{i-1} \{A_2\}_{ij}^2 e_j - \{diag(A_1)\}_i^2 t_i$
        $e_i = 1 / e_i$                                            $(D_{ii} = 1/L_{ii})$
    end loop i
    $\underline{e} \rightarrow \underline{e}_k$                                          (store diagonal D)
end loop k
```

*** conjugate gradient section ***

```
$\underline{t} = \underline{0}$
$\underline{t} \rightarrow \underline{t}_k$ , $\underline{t} \rightarrow \underline{d}_k$ ; k=1,2,...,$N_z$
$\alpha = \beta = 0$, $\gamma^{(o)} = 1$

loop i = 1 to maximum iterations
    $\underline{d} \leftarrow \underline{d}_1$, $\underline{t} \leftarrow \underline{t}_1$, $\underline{x} \leftarrow \underline{\Phi}_1$
    $\underline{t} = \underline{d} + \beta\,\underline{t}$
    if (any | $\alpha\,\underline{t}_m$ | / | $\underline{x}_m$ | > $\varepsilon$ ;  m =1,2,...,$N_xN_y$) then  set not converged flag
    $\underline{x} = \underline{x} + \alpha\,\underline{t}$
    $A_2 \leftarrow A_{11}$, $A_3 \leftarrow A_{12}$
    $\underline{r} \leftarrow \underline{0}$
    loop k = 1 to $N_z$
        $\underline{d} \leftarrow \underline{d}_{k+1}$, $\underline{t} \leftarrow \underline{t}_{k+1}$, $\underline{x}^* \leftarrow \underline{\Phi}_{k+1}$
        $\underline{t} = \underline{d} + \beta\underline{t}$
        $\underline{t} \rightarrow \underline{t}_{k+1}$
        if (any  | $\alpha t_m$ | / | $x^*_m$ |  > $\varepsilon$ ; m =1,2,...,$N_xN_y$) then set not converged flag
        $\underline{x}^* = \underline{x}^* + \alpha\underline{t}$
        $\underline{x}^* \rightarrow \underline{\Phi}_k$
        if  (k.eq.$N_z$.and. not converged flag unset) then  finish
        $\underline{b} \leftarrow \underline{s}_k$
        $\underline{r} = \underline{b} - A_2\,\underline{x} - A_3\,\underline{x}^* - \underline{r}$
        $\underline{r} \rightarrow \underline{r}_k$
        $\underline{t} \leftarrow \underline{h}_{k-1}$
        diag($A_2$) $\leftarrow 1/\underline{e}_k$                                    (fetch $L_{ii}$ ; i=1,2,...,$N_xN_y$ )
        d = [◺($A_2$)]$^{-1}$ ( $\underline{r} - A_1\,\underline{t}$ )
        $\underline{d} \rightarrow \underline{h}_k$
        $A_1 = A_3$                  ($A_1 = A_{k+1\,k}$ by symmetry)
        $A_2 \leftarrow A_{k+1\,k+1}$, $A_3 \leftarrow A_{k+1\,k+2}$
        $\underline{r} = A_3\,\underline{x}$
        $\underline{x} = \underline{x}^*$
    end loop k
    $\beta_1 = \beta_2 = \beta_3 = \gamma^{(i)} = 0$
    k = $N_z$
    $\underline{r} \leftarrow \underline{h}_k$
    diag($A_2$) $\leftarrow 1/\underline{e}_k$                                    (fetch $L_{ii}$ ; i=1,2,...,$N_xN_y$ )
    $\underline{d}$ = [ ◹ ($A_2$)]$^{-1}$[diag($A_2$)]$\underline{r}$                                    $(L^T)^{-1} D^{-1}$
    $\underline{d} \rightarrow \underline{d}_k$
    loop l = 1 to $N_z$
        $\underline{t} \leftarrow \underline{t}_k$, $r \leftarrow \underline{t}_{k-1}$, $\underline{u} \leftarrow \underline{t}_{k+1}$
        $\underline{u} = A_1\,\underline{r} + A_2\,\underline{t} + A_3\,\underline{u}$
        $\underline{r} \leftarrow \underline{r}_k$
        $\beta_2 = \underline{d}^T\,\underline{u} + \beta_2$
        $\beta_3 = \underline{t}^T\,\underline{u} + \beta_3$
        $\gamma^{(i)} = \gamma^{(i)} + \underline{r}^T\,\underline{d}$
        $\underline{r} \leftarrow \underline{d}_{k-1}$
        $\underline{v} = A_2\,\underline{d} + A_3\,\underline{r}$
        diag($A_2$) $\leftarrow \underline{e}_{k-1}$
        $\underline{r} \leftarrow \underline{h}_{k-1}$
        $\underline{r}$ =  [diag($A_2$)]$^{-1}$ $\underline{r}$
        $A_3 = A_1$
        $A_1 \leftarrow A_{k-1\,k-2}, A_2 \leftarrow A_{k-1\,k-1}$
        $\underline{x} = \mathrm{diag}(A_2)$                                    (Save diagonal D)
        $\mathrm{diag}(A_2) = 1/\underline{e}_k$
        $\underline{x}^* = [\,◥\,(A_2)]^{-1}(\underline{r} - A_3\underline{d})$
        $\mathrm{diag}(A_2) = \underline{x}$                                       (restore $A_2$)
        $\underline{v} = \underline{v} + A_1\underline{x}^*$
        $\beta_1 = \beta_1 + \underline{d}^T\underline{v}$
        $\beta_2 = \beta_2 + \underline{t}^T\underline{v}$
        $\underline{d} = \underline{x}^*$
        $\underline{d} \rightarrow \underline{d}_{k-1}$
        $k = k-1$
    end loop l
    $\beta = \gamma^{(i)}/\gamma^{(i-1)}$
    $\alpha = \gamma^{(i)}/(\beta_1 + \beta_2\beta + \beta_3\beta^2)$
end loop i
finish  error condition
end  ICCG
```

POW3D offers ICCG as a possible means of solution for two-dimensional problems. The algorithm outlined above is more complex than required for two-dimensional data structures, so separate code is used for efficiency. For such problems, all of the spatial information resides in directly addressable memory while the solution is performed. The procedure for two-dimensional calculations follows the conventional mathematical formulation (**section 6.5**) and is not presented.

7.4 Computer Implementation of Three-dimensional MINI

The MINI process of POW3D outlined in **sections 6.2, 6.3 and 6.4** is a block procedure for solving the linear systems of **equations 7.2.1**. In this respect, both MINI and SLOR differ markedly from ICCG. A description of the double block process to solve this three-dimensional system of equations is given. The third level, with MINI for the energy layers, is not presented but its implementation is a logical extension of that used for the three spatial dimensions.

The three-dimensional form of MINI requires the use of two distinct γs at each grid point. One is related to the forward line (the 'in-plane' γ_1) and the other related to the forward plane (the 'out-of-plane' γ_2 as shown in **figure 1**; (a third γ_e, not shown, is necessary for the energy group MINI). On the first pass it is possible either to set $\gamma = 0$ and obtain a GS approach, or to calculate a somewhat arbitrary value of γ (perhaps from the last outer iteration). In POW3D, the default option is a GS start on the first outer iteration for each energy group of any new model. On subsequent outers, knowledge remaining from the previous outer is used to obtain a true MINI iteration for the first inner iteration.

Like SLOR and two-dimensional ICCG, the inner-most (or two-dimensional) MINI block requires no data transfers because all the necessary data are established in computer memory by the outer block driver. A similar notation to that defined in **section 7.3** is employed for MINI. MINI, however, uses the following working vectors $\underline{x}$, $\underline{b}$, $\underline{\Gamma}$, $\underline{h}$, $\underline{d}$ and $\underline{e}$ of length N_xN_y. In practice, it is more convenient to compute and use the elements of $\underline{\Gamma}$ as they are required, but the algorithm is more succinct in the form shown here:

```
*** solves $A\underline{\phi} = \underline{s}$        equation 7.2.1 ***
*** 3-D MINI algorithm ***
***assume estimates of $\underline{\phi}_k$ $(k = 1,2,\ldots,N_z)$ have been made and stored ***

loop n=1 to maximum iterations
    $\underline{b} \leftarrow \underline{s}_1$
    $\underline{x} \leftarrow \underline{\phi}_1$
    $A_2 \leftarrow A_{11}, A_3 \leftarrow A_{12}$
    loop k = 1 to $N_z$
        $\underline{h} \leftarrow \underline{\phi}_{k+1}$
        $\underline{d} \leftarrow \delta^{\cdot}_{k+1}$                         (out of plane $\underline{\delta}$'s for current plane k)
        $\underline{e} \leftarrow \delta^{\cdot}_{k+1}$                         (out of plane $\underline{\delta}$'s for next plane k+1)
        for k ≠ $N_z$ loop l = 1 to $N_xN_y$
                $\bar{\gamma}_l = \min(|e_l/d_l|, |d_l/e_l|)$
                $\underline{\Gamma}_l = \min(\gamma_l, |y_l/x_l|)$
                end loop l
            $\underline{b} = \underline{b} - A_3\underline{h} + \underline{\Gamma}^T A_3 x$
            $\mathrm{diag}(A_2) = \mathrm{diag}(A_2) + \mathrm{diag}([\underline{\Gamma}^T A_3])$
            $\underline{e} = \underline{x}$
            $\underline{d} \leftarrow \underline{\delta}_k$                      (in plane δ's from previous pass of plane k)

*** invoke plane MINI routine to solve A $\underline{x} = \underline{b}$ ***
            Call plane MINI ($A_2$, $\underline{x}$, $\underline{b}$, $\underline{d}$)

*** solution returned in $\underline{x}$, in plane δ's in $\underline{d}$***

            $\underline{d} \rightarrow \underline{\delta}_k$                    (store within plane δ's)
            $\underline{x} \rightarrow \underline{\phi}_k$                    (store solution for kth z plane)
            $\underline{d} = \underline{x} - \underline{e}$
            $\underline{d} \rightarrow \underline{\delta}_k$                    (store out of plane δ's)
            if (any $|d_l/e_l| > \varepsilon$) then set not converged flag
            $A_1 = A_3$, $A_2 \leftarrow A_{k+1\,k+1}$, $A_3 \leftarrow A_{k+1\,k+2}$
            $\underline{b} \leftarrow \underline{s}_{k+1}$
            $\underline{b} = \underline{b} - A_1\underline{x}$
            $\underline{x} = \underline{h}$
        end loop k
if (not converged flag unset) then finish
end loop n
finish error condition

    Procedure plane MINI ($\underline{A}$, $\underline{x}$, $\underline{b}$, $\underline{d}$)

*** A, $\underline{x}$, $\underline{b}$, $\underline{d}$ are block (line) addressable ***

loop n = 1 to maximum iterations
loop j = 1 to $N_y$
    for j ≠ $N_y$ loop i = 1 to $N_x$
            $\bar{\gamma}_i = \min(|\{\underline{d}_{j+1}\}_i/\{\underline{d}_j\}_i|, |\{\underline{d}_j\}_i/\{\underline{d}_{j+1}\}_i|)$
            $\Gamma_i = \min(\gamma_i, |\{\underline{x}_{j+1}\}_i/\{\underline{x}_j\}_i|)$
            end loop i
    $\underline{b}_j = \underline{b}_j - A_{j\,j-1}\underline{x}_{j-1} + \underline{\Gamma}^T A_{j\,j+1}\underline{x}_{j+1}$
    $\mathrm{diag}(A_{jj}) = \mathrm{diag}(A_{jj}) + \mathrm{diag}([\underline{\Gamma}^T A_{j\,j+1}])$
    $\underline{d}_j = (A_{jj})^{-1}\underline{b}_j$                                    (solve tridiagonal system)
    if (any $|\{\underline{d}_j\}_i| > \varepsilon$) then set not converged flag
    $\underline{x}_j = \underline{x}_j - \underline{d}_j$
end loop j
if (not converged flag unset) then return
end loop n
return error condition
end procedure plane MINI
```

All of the mathematical schemes were developed in FORTRAN. Experience with the block methods MINI and SLOR indicated that much of the time was spent in converging individual (x,y) planes instead of terating in the z direction. Being a global method, ICCG spends time in a more even-handed fashion. Consequently, it was possible and computationally attractive to rewrite the inner-most blocks for MINI and SLOR in Assembler language [Cox 1976]. Even with an optimised FORTRAN compiler, this resulted in significant time savings. Although it was possible to do the same for ICCG, the enormity of the task of handling the third dimension made it less attractive. Only FORTRAN versions, however, are now provided or reasons of portability.

MINI works harder at converging the (x,y) plane than it does in the z direction. In general, it requires ewer z iterations than ICCG and this bestows on MINI a great computational advantage because far fewer /O transfers are required for large reactor problems. This affects the turnaround time for such jobs by orders of magnitude.

In addition, the storage limitations require ICCG to have more intermediate I/O transfers than MINI. The number of I/O transfers and floating point arithmetic operations per z iteration pass for the POW3D mplementation of the three iterative schemes is given in **table 1**. The I/O penalty of matrix block transfers is arger by a factor of three for ICCG. SLOR requires fewer I/O vector transfers but MINI is not badly penalised.

It is hard to evaluate the number of floating point operations per z pass because ICCG is not a block scheme, but clearly ICCG requires fewer per z pass. When obtaining a converged result, the advantage of ICCG may be illusory because more z passes are required. SLOR needs slightly fewer operations than MINI per pass but the performance of the latter more than makes up for the difference in floating point operations.

TABLE 1

NUMBER OF I/O TRANSFERS AND FLOATING POINT OPERATIONS PER z ITERATION FOR THE 3-DIMENSIONAL FORMS OF ITERATIVE SCHEMES AS IMPLEMENTED IN POW3D

N_x,N_y,N_z are the number of grid points in each direction, and K is the number of iterations necessary to converge the two-dimensional subsystem.

Type of Operation	SLOR	MINI	ICCG
Matrix block transfers	N_z	N_z	$3N_z$
Vector block transfers	$3N_z$	$7N_z$	$20N_z$
*/ floating point arithmetic	$(3+10K)N_xN_y$	$(9+12K)N_xN_y$	$37N_xN_y$
+- floating point arithmetic	$(5+6K)N_xN_y$	$(5+7K)N_xN_y$	$32N_xN_y$

7.5 Some Results

Results for a typical three-dimensional reactor system are given. The study is for a fast benchmark, sodium-cooled breeder reactor (LMFBR) investigated by Buchel *et al.* [1977]. Two (x,y,z) geometry versions are considered:

B1 of $20 \times 20 \times 19$ mesh points, and

B2 of $39 \times 39 \times 37$, a half spacing of B1 mesh points.

The data are available in four energy groups. The results are recorded in **table 2** for the various solution strategies. Because this is a fast reactor (no upscatter in thermal groups), it does not demonstrate the great advantage displayed by MINI in that dimension. For the simpler B1 geometry, there are few differences in central processor unit (CPU) time required, but for the more involved B2 geometry the SLOR method performs poorly. The MINI-MINI combination and ICCG use similar CPU times, but when driving the z direction iteration, MINI requires fewer passes.

Results were obtained on an IBM3031 with the computing region requested for the VIRTUL structure of POW3D equal to the available real storage of the machine. To enable comparison, all mathematical sections were compiled in FORTRAN. As it was not possible to obtain exclusive use of the machine, elapsed times have been discounted to allow for the possible presence of other jobs by the formula

$$\text{Real elapsed time} = \text{Measured elapsed time} \times \frac{\text{CPU time in frequently invoked routines}}{\text{Elapsed time in same routines}}$$

The CPU time spent in arranging I/O transfers (this is the total CPU time in the VIRTUL routines and includes all times that the IBM computer books against the work) for the three methods indicates that ICCG takes considerably longer. The relative cost is not so high as predicted in **table 1**, because other routines in POW3D also require a considerable overhead. (Much time was spent improving the functioning of these routines.)

TABLE 2

IBM3031 TIMING CONSIDERATIONS FOR THE LMFBR STUDY

*Convergence was not achieved in the time available;
extrapolation was used to obtain these figures

Method (x,y)	Method (z)	Total CPU Time (min)	I/O CPU Time (min)	I/O Elapsed Time (min)	Number I/O Calls × 1000	Total Plane Iterations	Total z Iterations
				B1 — Coarse Mesh Spacing			
SLOR	SLOR	17.8	1.8	16.8	36	11 069	226
SLOR	MINI	18.3	2.3	22.4	44	9 845	194
MINI	SLOR	24.9	2.7	26.5	56	13 353	333
MINI	MINI	21.2	3.0	31.3	57	8 621	213
ICCG	SLOR	24.0	2.1	11.2	44	7 777	267
ICCG	MINI	24.0	1.7	12.7	57	5 944	215
ICCG		20.7	5.	62.5	102		207
				B2 — Fine Mesh Spacing			
SLOR	SLOR	279.4	9.9	73.6	153	57 690	744
SLOR	MINI	204.3	12.0	144.7	155	30 455	366
MINI	SLOR	310	12.7	152.0	194	52 946	687
MINI	MINI	189.8	10.8	111.6	149	24 540	306
ICCG	SLOR*	353	8.9	98.9	442	33 220	634
ICCG	MINI	212.7	10.5	118.3	57	15 253	282
ICCG		228.1	25.3	296.2	401		454

The I/O elapsed time (**table 2**, column 3) is the time that elapses while POW3D is involved in purely I/O activity. (It is not a measure of total elapsed time before results are returned. Assuming no other computer users, the total elapsed time is obtained by adding the individual times from columns 1 and 3.) The elapsed time values show ICCG to be a very poor performer. Although the SLOR-SLOR combination gains some advantage in I/O elapsed time, MINI-MINI emerges as the better performer when elapsed and CPU times are combined for the fine mesh problem.

7.6 Evaluation of Iterative Procedures

Experimentation carried out so far suggests that the three iterative schemes are all contenders for solving spatial aspects of the neutron diffusion problem. The characteristics of each scheme are considerably different and the 'best' choice probably depends upon the specific problem and the priority with which a solution is required.

Generally, ICCG is the fastest for two-dimensional problems (CPU time) and for small three-dimensional problems, although the savings are not necessarily significant. As the complexity of the problem increases, MINI-MINI and ICCG appear comparable in CPU time, but I/O penalties in data organisation favour MINI. The amount of disk storage required for the larger problems is significant, the requirements of ICCG being greater than either SLOR or MINI. The requirements for all methods are sufficiently large to present real difficulties. The working data sets are best allocated to different units for efficient disk head movements, and ICCG again suffers most because it uses more data sets. Single precision arithmetic is adequate for ICCG on most problems although it presented significant difficulties for one of the models tested. Recourse to double precision intermediate working space in memory and on disk files would have placed too great a burden on most computing facilities when the code was first developed.

The I/O overhead per iteration is a minimum for SLOR-SLOR, but the SLOR-driven z direction is the most inefficient scheme in terms of number of iterations, and this counterbalances SLOR-SLOR's advantage.

As an energy group driver, MINI is significantly more efficient than GS when considerable upscatter occurs. Even without significant upscatter, little additional expense is involved and it is worth retaining MINI as the default option for energy group iterations.

The authors believe MINI to be the best default option for the nuclear code.

8. ACCELERATING CONVERGENCE BY COARSE MESH REBALANCE

8.1 Introduction

In POW3D, the convergence of the group and spatial iterative schemes may be accelerated through coarse mesh rebalance (CMR). At present, there is no attempt to accelerate convergence of the eigenvalue problem with the variational approach. Several schemes were tested on two-dimensional problems; these routines are available within POW3D. One scheme, however, appeared so computationally superior that it was the only one to be implemented for the third spatial dimension (and energy).

Consider the linear system

$$A\underline{\phi} = \underline{s} \quad , \qquad (8.1.1)$$

where $\underline{\phi}_o$ is our initial estimate of $\underline{\phi}$. Coarse mesh rebalance (CMR) uses a set of appropriate trial (or basis) vectors $\underline{x}_i$ (i = 1,2,...,M) to obtain a better estimate for $\underline{\phi}$:

$$\underline{\phi} \approx \underline{\phi}_o + \sum_{i=1}^{M} a_i \underline{x}_i \quad . \qquad (8.1.2)$$

The number of trial vectors M is usually considerably less than N, the order of the original system **8.1.1**. The unknown coefficients a_i of **equation 8.1.2** are obtained by solving a much reduced matrix system

$$A\underline{a} = \underline{k} \quad . \qquad (8.1.3)$$

The solution may be carried out by MINI or direct techniques, the choice being influenced by the type of trial vector used. The reduced system of equations is obtained by weighted residuals. If the **approximation 8.1.2** is used in **equation 8.1.1** a residual

$$\underline{r} = A\underline{\phi} - \underline{s}$$

and the unknowns a_i (i = 1,2,...,M) are determined by making $\underline{r}$ orthogonal to a set of weighting vectors $\underline{w}_i$

(l = 1,2,...,M), *i.e.*

$$<\underline{w}_l, \underline{r}> = 0 \quad (l = 1,2,...,M) \ ,$$

giving the relationship

$$<\underline{w}_l, A\underline{\phi}_o + \sum_{i=1}^{M} a_i A\underline{x}_i - \underline{s}> = 0 \quad (l = 1,2,...,M)$$

or

$$\sum_{i=1}^{M} <\underline{w}_l, A\underline{x}_i> a_i = <\underline{w}_l, \underline{s} - A\underline{\phi}_o> \quad (l = 1,2,...,M) \ . \qquad (8.1.4)$$

Several choices of trial vectors are available, and they form a considerable number of combinations with suitable weighting vectors. There are two basic forms of CMR - multiplicative and additive corrections. The multiplicative form is obtained by writing the improved estimate **8.1.2** as

$$\phi \approx \sum_{i=1}^{M} a_i \underline{x}_i \ , \qquad (8.1.5)$$

where the values for $\underline{x}_i$ are selected to reflect the structure of the estimate $\underline{\phi}_o$ of $\underline{\phi}$. This can be achieved by introducing M partitioning operators P_j so that

$$\underline{x}_j = P_j \underline{\phi}_o \quad (j = 1,2,...,M) \ , \qquad (8.1.6)$$

where values of P_j are chosen so that

$$\underline{\phi}_o = \sum_{j=1}^{M} P_j \underline{\phi}_o \quad \text{i.e.} \quad I = \sum_{j=1}^{M} P_j \ .$$

The M × M system of linear equations arising with the multiplicative form looks a little simpler than the general form **8.1.4**:

$$\sum_{i=1}^{M} <\underline{w}_l, A P_i \underline{\phi}_o> a_i = <\underline{w}_l, \underline{s}> \quad (l = 1,2,...,M). \qquad (8.1.7)$$

The partitioning operator P_j divides the spatial (x,y and z) and energy dimensions into a suitable form. In POW3D, the partitioning is done so that the non-zero elements of $\underline{x}_i$ are connected geometrically. POW3D automatically positions the coarse mesh lines of the subdivision so that the reactor fuel is the prime determinant of their position. Non-fissile materials and interfaces between materials are the secondary criteria considered for their allocation. In POW3D, the order of the coarse grid is approximately the square root of the order of the fine mesh.

The location of the coarse grid should vary with the degree of mathematical sophistication involved in the choice of partitioning operators. The code allows the knowledgeable physicist to force an estimate of optimal coarse mesh partitioning; this has proved useful for experimental purposes. The automatic generation in POW3D was designed to accommodate the first form of partitioning to be described (*i.e.* the disjunctive type).

Three forms of partitioning have been tested with the code, but only one (**8.3.1**) is available for the third dimension. The others were implemented in two-dimensional form but the results did not seem to warrant a three-dimensional implementation [Barry 1982]. Two of these forms of partitioning involve multiplicative corrections **8.1.5** and **8.1.6**; in the third, the vector $\underline{\phi}_o$ is readmitted (**8.1.2** and **8.1.4**) as an additional type of correction. The trial vectors are then a set of prescribed vectors unrelated to the solution obtained for $\underline{\phi}_o$ so far.

In all cases, the forms of partitioning generate sets of M basis vectors which are linearly independent. For the disjunctive methods, the basis vectors are also mutually orthogonal and the P_i are projection operators.

8.2 Types of Partitioning

8.2.1 Disjunctive partitioning

The spatial domain D, over which the discretised form of the diffusion equation is defined, is divided into M subdomains D_m. Any vector $\underline{\phi}$ of length N (where each element of the vector corresponds to a value of $\phi(\underline{r})$, defined over D) is partitioned so that each element of $\underline{\phi}$ belongs to one and only one subdomain D_m.

The partitioning matrix P is simply a diagonal N x N matrix, where certain diagonal elements are unity. For a one-dimensional problem, P_j has a single block unit matrix structure which is not preserved for higher dimensions in POW3D as the non-zero elements become scattered. The structure now depends upon the spatial ordering of the elements of $\underline{\phi}$. A two-dimensional geometric representation of disjunctive partitioning is given in **figure 2**. The coarse grid divides the fine grid into four subdomains D_1, D_2, D_3 and D_4. The division of the fine geometric mesh is such that each fine mesh point belongs to only one subdomain D_m. Consequently, the operations $P_1\underline{\phi}_o$, $P_2\underline{\phi}_o$, $P_3\underline{\phi}_o$ and $P_4\underline{\phi}_o$ simply pick out values of $\underline{\phi}_o$ defined over the four subdomains. The improved approximation is

$$\underline{\phi} = \sum_{i=1}^{M} a_i P_i \underline{\phi}_o \ .$$

It effects an acceleration of convergence by rescaling the flux estimate $\underline{\phi}_o$ by a constant a_i over each subdomain.

8.2.2 Multiplicative 'pyramid' partitioning

The disjunctive methods destroy any continuity exhibited by the flux estimate at coarse mesh boundaries, so a more complex scheme of 'pyramid' (or, more correctly, 'pagoda' [Nakamura 1977]) was incorporated. This attempts to overcome the introduced discontinuity in $\underline{\phi}$ when a more realistic form of correction is applied across the subdomain D_m. This form operates in POW3D for two-dimensional geometries only.

The spatial domain D is divided into M subdomains D_j which is defined in terms of I spatial subregions R_i. This time, however, an element of $\underline{\phi}_0$ may be common to more than one subregion (*i.e.* the subdomains are not mutually exclusive). Typical positions of the coarse mesh lines are shown in **figure 3**. These divide the fine mesh to form 6 subregions (R_i) which are combined to form 12 subdomains associated with the corners of R_i. In this case, any fine mesh point lying on a coarse meshline of subdivision belongs to all subregions associated with that line. The partitioning matrices P_j (j = 1,2,...,M) are again diagonal, but the non-zero terms are defined over four adjacent subregions associated with each subdomain (for two-dimensional form). For the partitioning operator defined over the domain D_j (consisting of four regions centred at the coarse grid mesh node coinciding with the (k,l)th fine mesh point (X_k, Y_l)) the non-zero diagonal elements of P_j are defined by two-dimensional pagoda functions, namely:

$$P_j(x,y) = \frac{(X_{k+1}-x)(Y_{l+1}-y)}{(X_{k+1}-X_k)(Y_{l+1}-Y_l)} \ , \qquad (8.2.1)$$

$$\text{for} \quad X_k \le x \le X_{k+1}, \quad Y_l \le y \le Y_{l+1}$$

$$= \frac{(X_{k-1}-x)(Y_{l+1}-y)}{(X_{k-1}-X_k)(Y_{l+1}-Y_l)} \ ,$$

$$\text{for} \quad X_{k-1} \le x \le X_k, \quad Y_l \le y \le Y_{l+1}$$

$$= \frac{(X_{k+1}-x)(Y_{l-1}-y)}{(X_{k+1}-X_k)(Y_{l-1}-Y_l)} \ ,$$

$$\text{for} \quad X_k \le x \le X_{k+1}$$
$$Y_{l-1} \le y \le Y_l \quad \text{and}$$
$$= \frac{(X_{k-1}-x)(Y_{l-1}-y)}{(X_{k-1}-X_k)(Y_{l-1}-Y_l)} \, ,$$
$$\text{for} \quad X_{k-1} \le x \le X_k$$
$$Y_{l-1} \le y \le Y_l \; . \qquad (8.2.1)$$

These definitions **8.2.1** also identify indirectly the domain D_j, as that set of fine grid points associated with non-zero elements of P_j. Again $\sum_{j=1}^{M} P_j = I$. Both the disjunctive and pyramid forms of partitioning lead to an M x M system of the form **8.1.7**. (The latter is only available in a test version of POW3D.)

8.2.3 Additive pyramid partitioning

The additive form of partitioning **8.1.2** involves trial vectors that are unrelated to the original trial estimate $\underline{\phi}_0$. There are considerable time savings when establishing the matrix system **8.1.4** because the matrix on the left hand side is independent of $\underline{\phi}_0$ and hence does not alter for each group upscatter pass or eigenvalue iteration.

In POW3D, the piecewise continuous bilinear polynomials **8.2.1** — pagoda function — are selected as the trial vectors. These are defined over subdomains and are identical to those used with pyramid partitioning. Again there is only a two-dimensional implementation of additive partitioning in POW3D.

8.3 Types of Weighting

Two types of weighting are applied within POW3D. These are

(i) Galerkin weighting

$\underline{w}_l = \text{diag}\{P_l\}$ (additive CMR) ;

(ii) Region balancing weighting —

$\underline{w}_l = P_l \underline{1}$;

where P_l is the disjunctive partitioning matrix and $\underline{1}$ is a unit vector.

All combinations of basis and weighting vectors are possible; however, for reasons discussed by Barry [1982], only the following combinations were ever implemented in the code:

	Weighting	Partitioning	Equation
(i)	Region balancing	disjunctive multiplicative	(8.3.1)
(ii)	Region balancing	multiplicative pyramid	(8.3.2)
(iii)	Region balancing	additive pyramid	(8.3.3)
(iv)	Galerkin	additive pyramid	(8.3.4)

Of these only the first was implemented for three-dimensional geometries.

8.4 Rebalancing in Energy

A very simple yet effective rebalance technique known as *energy rebalancing* is available. All the spatial (x,y and z) information is 'collapsed' to a single point. This is achieved through a single disjunctive partitioning operation (*i.e.* $P_g = I$ (g = 1,2,...,G)) for each energy group equation in turn, and the use of an associated weighting vector $\underline{w}_g = \underline{1}$.

Energy rebalancing is an operation which is performed once for each outer iteration for the eigenvalue problem, or once only for a source problem. The operation is performed when the first energy group of the eigenvalue **equation 4.2.2**, or the equivalent source equation, with neutron upscatter into it is encountered. In the case of the eigenvalue problem, after all the fluxes have been rebalanced, they are renormalised to a predetermined source strength (power level). Because of the small number of energy groups there is no advantage in collapsing the groups still further.

When POW3D was being used as a research tool, it seemed that the twin concepts of rebalancing the fluxes in energy and space could be combined with very little additional effort. This form of global rebalancing was attempted and the possible combinations are outlined in **section 9**. By and large, the authors do not recommend the limited forms of global rebalancing studied.

8.5 Properties of the Coarse Mesh Rebalance Matrices

The coarse mesh rebalance equations generated by POW3D are of considerably reduced order because of the square root method used for allocating coarse mesh grid points. It is possible in many instances to solve these efficiently by direct methods, because the time spent on their solution is insignificant to that spent on the fine mesh. Nevertheless, for a general purpose code it is attractive to solve the reduced equations by the same routines that handle the fine mesh. This is particularly so if the CMR method is ever to extend to a multigrid technique [Brandt 1977].

The properties of the reduced matrix system **8.1.3** depend upon those of the original fine mesh matrix, and the types of partitioning and weighting. The following fine mesh matrix is appropriate to a single energy group in a multigroup model:

(i) real symmetric;

(ii) irreducible;

(iii) diagonal elements are positive, off-diagonal elements are zero or negative;

(iv) diagonally dominant (row and column).

When the weighting/partitioning combination **8.3.1** is used, symmetry and row dominance are lost. Column dominance remains and is sufficient to guarantee GS type convergence, but SLOR and ICCG are no longer appropriate because of lack of symmetry. Consequently, MINI is the iterative scheme used in POW3D for the first weighting/partitioning combination. Unfortunately, when pyramid partitioning is used for combinations **8.3.2** and **8.3.3**, most of the desirable matrix properties are lost and no iterative scheme can be relied on to converge. The matrix equation may even have negative solutions which violate physical considerations. In POW3D, negative intermediate solutions are adjusted in a physically significant way. For Galerkin weighting, however, it is possible to establish [Barry 1982] that the reduced matrices are positive definite for all forms of partitioning.

Some of the mathematical results may be extended to the eigenvalue problem, but at this stage POW3D uses CMR only at each outer pass through the eigenvalue problem. This is the default option within POW3D. Use of CMR is avoided when SLOR is used for the inner two layers until the optimum relaxation factors have been evaluated.

The weighting/partitioning combinations were tested extensively in two dimensions. It was concluded that only the first combination **8.3.1** would be worthwhile in three dimensions. Coarse mesh rebalancing is very effective in hastening convergence. The reasons for its effectiveness are open to speculation, however, its ability to remove rapidly the low frequency error components [Brandt 1977] from the solution estimate seems important. Although the iterative techniques on the fine mesh will rapidly remove the high frequency error components, they have difficulty in doing so with error components of low frequency. A Fourier analysis of MINI and CMR [Barry *et al.* 1983] supports the conjecture of Brandt [1977].

9. CONTROLLING AVAILABLE MATHEMATICAL OPTIONS

9.1 Introduction

It is not our intention to describe the ways in which POW3D is used to model a reactor. These will be described in a companion report [Barry *et al.*, forthcoming]. Accordingly, descriptions of the input data are restricted to those items which control the available mathematical algorithms.

The order of these records is somewhat important, full details will also be given by Barry *et al.* [forthcoming]. Options relating to the choice of available mathematical methods are described. Like all POW3D input data, a free format layout applies [Bennett and Pollard 1967]. In the description given for each parameter, the code default option is given in *italics.*

9.2 method

method = i_1 i_2 is used to specify the solution methods used for the iterative solution of the sparse linear equations. The two parameters are appropriate to the three layers in a three-dimensional multigroup study:

i_1 = *mini* *slor* *iccg*	for the innermost layer; (x,y) planes.
i_2 = *mini* *slor*	for the outside (z) driver of the innermost block. Only applicable for three-dimensional systems

When i_1 is coded as *iccg*, the second parameter is ignored. For example, the full MINI requirement is set thus:

method = *minimini.*

Apart from the restrictions on ICCG, any of the other block combinations are allowed, however, users are reminded that MINI (**section 6.4**) continually alters the diagonal terms of the inner matrices, and for those blocks it may have some effect on SLOR. *minimini* is the default option in POW3D.

9.3 methsc

methsc	=	*mini* | *gs*	This specifies the method used to handle the energy dimension.
mininf	=	i_1,i_2,i_3	This is coded only when MINI is used for the spatial geometry solution.

In the MINI process, should all the γs (**section 6.3**) be zero, a MINI iteration is the same as an ordinary GS iteration. When a new problem is commenced because there are no prior iterations, a GS start is appropriate; this is forced by the code. When a new outer iteration (eigenvalue problem) or an upscatter pass is encountered, it is necessary in the spatial domain to choose between a fresh GS start or MINI, based on somewhat older estimates of the flux. On test problems, it was discovered that a true MINI start was better despite the fact that only very old flux estimates were available to determine the γs. The user can alter this default for outer iterations or upscatter passes.

i_1 = 0	Each (x,y) plane solution starts with all the γs = 0 for the start of every outer iteration or upscatter pass.
= 1	Each (x,y) plane solution is started with all the γ's set as a result of some previous iteration if available, otherwise i_1 = 0 is chosen. This is the default option.
i_2	As for i_1 but applies to the MINI process in the third spatial dimension.

i_3 The minimum number of iterations that must be taken by each application of MINI even though convergence may have been achieved earlier. The default is 2.

9.4 minins

minins = *0*
= *1* This serves the same function for determining the γ's associated with energy, as **mininf** did for the spatial γ's on new outer iterations.

9.5 jmbal

This feature is no longer supported

9.6 jbpymr

This feature is no longer supported

9.7 Calculation Termination Data

Termination data for a calculation are initialised to a default set of values by POW3D and, for the usual ob, these values should be adequate. Full details of quantities used here will be given in Barry *et al.* forthcoming]. Except for error termination (when POW simply 'browses' through the remaining data), a calculation ceases when one or more of the following termination conditions is met:

(a) If,

i. overall neutron balance accuracy $\leq$ **acclam(1)**, default 1.E-4;

ii. local fission source accuracy $\leq$ **accfo(1)**, default 1.E-4; and

iii. critical k accuracy $\leq k_{acc}$ if a *search* is requested.

(b) If machine time exceeds a specified limit (default 90 per cent of the time remaining for the current job step).

(c) if number of outer iterations, n $\geq$ **nol**, a set limit (default 100).

(d) if **nil**, the set limit to the number of inner iterations = 0.

A user would probably only be interested in the above under the following circumstances:

(a) to reduce (the already tight) accuracy for a lengthy calculation;

(b) if the anticipated run time is uncertain;

(c) if a calculation is not physically meaningful; and

(d) if a previous flux dump is to be entered for **edit**

For the user requiring more than a brief run through the termination conditions, further details are provided below.

9.8 Accuracy Termination (acclam and accfo)

POW3D uses two main quantities to assess convergence of a calculation:

(i) overall neutron balance, $b^{(n)}$, and

(ii) local reaction (usually fission source) error, $s_{err}^{(n)}$, both of which change with each outer iteration n. In addition, for a criticality **search** we need to consider a further quantity:

(iii) effective multiplication error, $| k^{(n)} - k_{reqd} |$.

Let $s_{ij}^{(n)}$ be the total reaction of indicated type for outer iteration number n in a box about the mesh point (i,j), then we have the following meanings:

for a real eigenvalue calculation

$$s_{ij}^{(n)} = \sum_g {}_g\phi_{ij}^{(n)} \int_{(ij)} \nu\sigma_{fg} \, dV,$$

for a real source calculation, as above if any reactor material is fissile, otherwise

$$s_{ij}^{(n)} = \sum_g {}_g\phi_{ij}^{(n)} \int_{(ij)} \sigma_{nofiss,g} \, dV,$$

and **nofiss** is usually 2 (removals);

for an adjoint calculation

$$s_{ij}^{(n)} = \sum_g {}_g\phi_{ij}^{(n)*} \chi_{pg}.$$

We then have

(i) $b^{(n)} = \sum_i \sum_j s_{ij}^{(n)} / \sum_i \sum_j s_{ij}^{(n-1)}$

(ii) $S_{err}^{(n)} = \left\{ \max_{i,j} \left[s_{ij}^{(n)} / s_{ij}^{(n-1)} \right] - \min_{i,j} \left[s_{ij}^{(n)} / s_{ij}^{(n-1)} \right] \right\} S^{(n-1)} / S^{(n)}$

with i and j taken over the whole reactor for which $s_{ij}^{(n-1)} \neq 0$ and S is the total source value.

POW utilises a convergence countdown scheme from trial solution stage (MOP=3) to ultimate convergence stage (MOP=0). Briefly, for the usual problem,

Stage	Type	Mode of Change
3	Power iteration	$b^{(n)} \leq$ **acclam**(3) and $S^{(n)}_{err} \leq$ **accfo**(3)
2	Chebyschev extrapolation $4 \leq$ order ≤ 6	$b^{(n)} \leq$ **acclam**(2) and $S^{(n)}_{err} \leq$ **accfo**(2)
1	As above but more confident use $6 \leq$ order ≤ 10	$b^{(n)} \leq$ **acclam**(1) and $S^{(n)}_{err} \leq$ **accfo**(1)
0		CONVERGED (provided $\mid k^{(n)} - k_{reqd} \mid \leq k_{acc}$) for a **search** calculation

The default values for accuracy limits may be changed:

acclam(1)=1.-4,**acclam**(2)=1.-3,**acclam**(3)=1.-2,
accfo(1)=1.-4,**accfo**(2)=1.-3,**accfo**(3)=1.-2)
using the data, for example, acclam=1.-3
accfo=1.-3,
where here and normally only the final accuracy would be changed. Note: The tight limits for acclam=1.-5, accfo=1.-5 are seldom, if ever, required.

9.9 Performance tuning parameters

Details on performance tuning parameters are given in Appendix D.

10. CONCLUSIONS

The default values for accuracy limits may be changed:

acclam(1)=1.-4,**acclam**(2)=1.-3,**acclam**(3)=1.-2,
accfo(1)=1.-4,**accfo**(2)=1.-3,**accfo**(3)=1.-2)
using the data, for example, acclam=1.-3
accfo=1.-3,
where here and normally only the final accuracy would be changed. Note: The tight limits for **acclam**=*1.-5*, **accfo**=*1.-5* are seldom, if ever, required.

1. ACKNOWLEDGEMENTS

We are indebted to Mr G Robinson, for a very thorough and constructive review. His comments, based on a detailed reading of our work, have enhanced greatly this report. We also acknowledge many long conversations with Professor G. Doherty (Wollongong University) over convergence aspects of MINI, and his suggestion of considering ICCG. His help in seeking the elusive proof of MINI's convergence is greatly appreciated.

2. REFERENCES

Barry, J.M. [1982] - Multi-dimensional Neutron Diffusion. Ph.D Thesis, Wollongong University.

Barry, J.M., Harrington, B.V., Pollard, J.P. [forthcoming] - POW3D user's guide. ANSTO/E report in preparation.

Barry, J.M., Jenkinson, J.H., Pollard, J.P. [1983] - Acceleration of iterative solution of large systems of linear equations by finite element methods. *J. Austral. Math., Soc.*, **B25**:190-216.

Barry, J.M., Pollard, J.P. [1977] - Method of implicit non-stationary iteration for solving neutron diffusion linear equations. *Ann. Nucl. Energy*, **4**:485-493.

Barry, J.M., Pollard, J.P. [1978] - Application of the method of implicit non-stationary iteration (MINI) to 3D neutron diffusion problems. *Ann. Nucl. Energy*, **6**:121-131.

Barry, J.M., Pollard, J.P. [1982] - Solution of neutron diffusion equations by implicit non-stationary iteration. *Proc. Conf. on Numerical Solutions of Partial Differential Equations*, University of Melbourne, 1981. North Holland, Amsterdam, pp.605-622.

Bennett, N.W., Pollard, J.P. [1967] - SCAN — a free input subroutine for the IBM360. AAEC/TM399.

Birkhoff, G., Varga, R.S. [1958] - Reactor criticality and non-negative matrices. *J. Soc. Indust. Appl. Math.*, 6(4)354-377.

Brandt, A. [1977] - Multi level adaptive solutions to boundary-value problems. *Math. Comput.*, **31**(138)333-390

Bondarenko, I.I. (ed.) [1964] - Group Constants for Nuclear Reactor Calculation. Consultants Bureau, New York.

Buckel, G., Kufner, K., Stehle, B. [1977] - Benchmark calculations for a sodium-cooled breeder reactor by two- and three-dimensional diffusion methods. *Nucl. Sci. Eng.*, **64**:75-89.

Cawley, R.J. [1976] - AAEC private communication.

Cawley, R.J. [1977] - AAEC private communication.

Cox, G.W., Pollard, J.P. [1978] - Appendix to - SKAN A free input labelled output variable dimensioning routine for the IBM360 computer. AAEC/E431.

Cox, G.W. [1976] - AAEC private communication.

Dahl, O.J., Dijkstra, E.W., Hoare, C.A.R. [1972] - Structured Programming. Academic Press, New York.

Ferguson, D.R., Derstine, K.L. [1977] - Optimised iteration strategies and data management considerations for fast reactor finite difference diffusion theory codes. *Nucl. Sci. Eng.*, **64**:593-604.

Hansen, G.G., Roach, W.H. [1961] - Six and sixteen group cross sections for fast and intermediate critical assemblies. LAMS-2543.

Hassitt, A. [1962] - A computer prgram to solve the multigroup diffusion equation. TRG-299(R).

Hopkins, D.R., Oakes, D.B. [1968] - The two-dimensional, multigroup diffusion code, GOG. AEEW-R532.

Meijerink, J.A., van der Vorst, H.A. [1977] - An iterative solution method for linear systems of which the coefficient matrix is a symmetric M-matrix. *Math. Comput.*, **31**(137)148-162.

Nakamura, S. [1977] - Computational Methods in Engineering and Science with Applications to Fluid Dynamics and Nuclear Systems. John Wiley, New York.

Noble, B. [1976] - Private communication.

Pollard, J.P. [1968] - Subroutine SOK — iterative solution of linear equations by the method of averaging functional corrections. AAEC/E192.

Pollard, J.P. [1973] - Numerical Methods used in Neutronics Calculations. Ph.D thesis, University of New South Wales.

Pollard , J.P. [1974] - AUS module POW — a general purpose 0, 1, and 2D, multigroup diffusion code including feedback-free kinetics. AAEC/E269.

Pollard, J.P. [1978] - A free input lablelled output variable dimensioning routine for the IBM360 computer. AAEC/E431.

Reid, J.K. [1971] - On the method of conjugate gradients for the solution of large sparse systems of linear equations. In *Large Sparse Sets of Linear Equations* (Reid, J.K. ed). Academic Press, New York.

Robinson, G.S. [1975] - AUS — The Australian modular scheme for reactor neutronic computations. AAEC/E369.

Stacey, W.M. [1969] - Space-Time Nuclear Reactor Kinetics. Academic Press, New York.

Wachspress, E.L. [1966] - Iterative Solution of Elliptic Systems and Applications to the Neutron Diffusion Equations of Reactor Physics. Prentice Hall, New Jersey.

Young, D.M. [1971] - Iterative Solution of Large Linear Systems. Academic Press, New York.

APPENDIX A

TEMPORAL INTEGRATION OF THE TIME-DEPENDENT NEUTRON DIFFUSION EQUATION

A sketch is given of the numerical methods used in POW3D to handle the temporal integration of the multigroup diffusion equation **(2.1.1)** and the precursor concentration **(2.1.2)**. The method is due essentially to Pollard [1973] and designed for the two-dimensional code POW [Pollard 1974], the forerunner of POW3D. The method is based on direct time integration according to the method of Stacey [1969] and aims to include the precursor concentration in a natural way.

Assume that the flux and precursor concentration solutions up to the time t_{p-1} are known, and the solution at time $t_p = (t_{p-1} + \delta t)$ is sought. **Equation 2.1.2** can be written as

$$\frac{\partial}{\partial t}[e^{\lambda_d t} C_d(\underline{r},t)] = \beta_d e^{\lambda_d t} \sum_g \frac{\nu}{k} \sigma_{fg}(\underline{r},t)\phi_g(\underline{r},t) \quad (d=1,2,\dots,D) \quad , \qquad \text{(A1)}$$

which, on integration, yields

$$C_d(\underline{r};p) = C_d(\underline{r};p-1)e^{-\lambda_d \delta t} + $$
$$+\beta_d \int_{t_{p-1}}^{t_p} e^{-\lambda_d (t_p - t)} \sum_g \frac{\nu}{k} \sigma_{fg}(\underline{r},t)\phi_g(\underline{r},t)dt \quad , \qquad \text{(A2)}$$

with the notation showing that $C_d(\underline{r},t)$ is now discretised. Across each time step the flux is assumed to vary linearly,

$$\phi_g(\underline{r},t) = \left[\frac{t_p - t}{\delta t}\right] \phi_g(\underline{r};p-1) + \left[\frac{t - t_{p-1}}{\delta t}\right] \phi_g(\underline{r};p) \quad , \qquad \text{(A3)}$$

and each cross section is assumed to be constant,

$$\text{i.e.} \quad \sigma(\underline{r},t) = \sigma(\underline{r},\bar{t}_p) \quad , \qquad \text{(A4)}$$

where

$$\sigma(\underline{r},\bar{t}_p) = \int_{t_{p-1}}^{t_p} \sigma(\underline{r},t)dt/\delta t \ . \qquad \text{(A5)}$$

Equation A5 is necessary, because within each step, the cross section may otherwise have had a discontinuity (in function value or slope). Normally, however,

$$\bar{t}_p = \frac{1}{2}(t_{p-1} + t_p) \ .$$

By making the change in variable

$$t = t_{p-1} + \tau\,\delta t \quad ,$$

equation A2 becomes

$$C_d(\underline{r};p) = C_d(\underline{r};p-1)e^{-\lambda_d \delta t} + \beta_d\,\delta t \int_0^1 e^{-\lambda_d t(1-\tau)} \sum_g \frac{\nu}{k} \sigma_{fg}(\underline{r};\bar{t}_p) \quad x$$
$$x \quad [(1-\tau)\,\phi_g(\underline{r};p-1) + \tau\,\phi_g(\underline{r};p)]d\tau \ , \quad (d=1,2,\dots,D) \ . \qquad \text{(A7)}$$

The functions

$$F_n(x) = \int_0^1 e^{x(1-\tau)}\tau^{n-1}\,d\tau \quad ,$$

$$F_n'(x) = \int_0^1 e^{x(1-\tau)}\tau^{n-1}(1-\tau)\,d\tau$$

are introduced, and **equation A7** becomes

$$C_d(\underline{r};p) = C_d(\underline{r};p-1)\, e^{-\lambda_d \delta t} + \beta_d\, \delta t \sum_g \frac{\nu}{k}\, \sigma_{fg}(\underline{r},T_p) \quad x$$

$$x \quad [F_1'(-\lambda_d\delta t)\,\phi_g(\underline{r};p-1) + F_2(-\lambda_d\delta t)\,\phi_g(\underline{r};p)]\ , \quad (d=1,2,\ldots,D). \qquad \text{(A8)}$$

The multigroup diffusion equation can be integrated in a similar fashion. The terms are basically of the form

$$I = \int_{t_{p-1}}^{t_p} \sigma_g(\underline{r},t)\,\phi_g(\underline{r},t)\,dt \quad ,$$

and, with the assumptions A3 and A4, the integral becomes

$$I = \frac{\delta t}{2}\,\sigma_g(\underline{r},t_p)\,[\phi_g(\underline{r};p-1) + \phi_g(\underline{r};p)] \quad .$$

To deal with the integrated precursor terms from A1, the precursor equation is integrated directly to give

$$\int_{t_{p-1}}^{t_p} \lambda_d\, C_d(\underline{r};t)\,dt = \beta_d \frac{\delta t}{2} \sum_g \frac{\nu}{k}\,\sigma_{fg}(\underline{r},T_p)\,[\phi_g(\underline{r};p-1) + \phi_g(\underline{r};p)] +$$

$$+\, C_d(\underline{r};p-1) - C_d(\underline{r};p) \quad . \qquad \text{(A9)}$$

Substitution of equation A8 for $C_d(\underline{r};p)$ in equation A9 and replacement of the integrated precursor term from A1 with A9 leads to the required time integrated diffusion equation:

$$- \nabla \cdot D_{n,g}(\underline{r},T_p)\ \nabla\phi_g(\underline{r};p) + [\,\sigma_{rg}(\underline{r},T_p) + 2/(v_g\,\delta t)]\,\phi_g(\underline{r};p) -$$

$$- \sum_{g'} \sigma_{gg'}(\underline{r},T_p)\,\phi_{g'}(\underline{r};p) - \chi_{(2)g}(\delta t) \sum_{g'} \frac{\nu}{k}\,\sigma_{fg'}(\underline{r};T_p)\,\phi_{g'}(\underline{r};p)$$

$$= \nabla \cdot D_{n,g}(\underline{r},T_p)\ \nabla\phi_g(\underline{r};p-1) - [\sigma_{rg}(\underline{r},T_p) - 2/(v_g\delta t)]\,\phi_g(\underline{r};p-1)$$

$$+ \sum_{g'} \sigma_{gg'}(\underline{r},T_p)\,\phi_{g'}(\underline{r};p-1) + \chi_{(1)g}(\delta t) \sum_{g'} \frac{\nu}{k}\,\sigma_{fg'}(\underline{r},T_p)\,\phi_g(\underline{r};p-1)$$

$$+ 2\sum_d \chi_{dg}\lambda_d\, G_0(\lambda_d\,\delta t)\, C_d(\underline{r};p-1) + S_g(\underline{r};T_p) \quad , \qquad \text{(A10)}$$

where the fission spectrum functions are given by

$$\chi_{(1)g}(\delta t) = \chi_{pg}(1-\beta) + \sum_d \chi_{dg}\,\beta_d\, G_1(\lambda_d\,\delta t) \quad ,$$

$$\chi_{(2)g}(\delta t) = \chi_{pg}(1-\beta) + \sum_d \chi_{dg}\,\beta_d\, G_2(\lambda_d\,\delta t) \quad ,$$

$$G_0(x) = F_1(-x) \ ,$$

$$G_1(x) = 1 - 2F_1'(-x) \ , \quad \text{and}$$

$$G_2(x) = 1 - 2F_2(-x) \ .$$

The integral functions $F_n(x)$, $F_n'(x)$ and $G_n(x)$ are evaluated through a single Pade approximation [Pollard 1973]. This is necessary because of the sensitivity of the analytic functions in evaluation on a computer. Integration by parts of the spectrum function $F_n(x)$, results in the recurrence relationship

$$F_n(x) = \left[x\,F_{n+1}(x) + 1\right]/n$$

Now $\quad F_1(x) = \left[e^x - 1\right]/x$

which presents difficulties for computer evaluation for small x. Consequently, the Pade approximation

$$F_3(x) \approx \frac{3600 - 180x + 30x^2 + x^3 + bx^4}{10800 - 3240x + 360x^2 - 15x^3} \quad \text{for } -1 \le x \le 0$$

where b = $3.5711027315\times10^{-2}$ is employed. Direct evaluation is used for $x < -1$ which is not subject to excessive roundoff error and the overall approximation is continuous for the choice of b. The relation

$$F_n'(x) = F_n(x) - F_{n+1}(x)$$

is necessary to enable evaluation of the remaining spectrum functions.

APPENDIX B

SPATIAL DISCRETISATION OF THE NEUTRON DIFFUSION EQUATION

B1. INTRODUCTION

The derivation of the finite difference representation for the three spatial dimensional diffusion equation is given for a single energy group. The derivation is for an (x,y,z) rectangular geometry divided by a grid system that consists of planes parallel to the appropriate axes. (The special cases, 2D cylinder and 1D sphere are not treated here, but are derived so that accurate volumes are retained.)

Spatial integration of the diffusion **equation A10** is carried out over integration boxes surrounding each grid point. A typical box is shown in **figure B1**; the box is subdivided into eight smaller boxes to permit integration over varying materials. Approximations are then sought for the resulting integrals. The required flux solution $\phi_g(\underline{r},t)$ is calculated at the intersection of the grid lines (the edge flux method) rather than the computationally simpler (centre flux) method using grid centres. The edge flux method results in a more accurate approximation for the neutron leakage about a point.

To identify the materials making up the region of integration in **figure B1,** the sub-boxes are numbered as indicated, specifying the furthest diagonal from the centre (x,y,z) and using an abbreviated notation of successive digits 'abc' to denote a point (x+(a-1)..., y+(b-1)..., z+(c-1)...), where '...' is the half mesh spacing appropriate to the indicated axis: 1-(222), 2-(022), 3-(002), 4-(202), 5-(220), 6-(020), 7-(000), 8-(200), and the grid dimensions are shown.

The non-leakage terms such as fission emission $(\nu\sigma_f)$ are considered first, and then the neutron leakage term $(-\nabla . D_n \nabla)$. The following notation is used:

l = 1,2,3,4,5,6,7,8 as an index for each sub-box,
m_l identifies the material for box l,
$\sigma(m)$ the cross section of the material filling a sub-box,
h_j is the grid width as indicated in **figure B1**; j = 1,2,3,4,6,7,
h max($h_1, h_2, h_3, h_4, h_6, h_7$)
v_l is the volume of the l^{th} sub-box,
ϕ_{ijk} is the flux value at the point (i,j,k).

B2. INTEGRATION OF NON-LEAKAGE TERMS

Approximations to the non-leakage integrals of the form

$$I = \int_v \sigma(\underline{r})\phi(\underline{r})d\underline{r}$$

are given first, where for convenience the solution is simplified from that of **equation A10**. $\phi(\underline{r},t)$ is expanded by a Taylor's series about the grid point (i,j,k):

$$\phi(\underline{r}) = \phi_{ijk} + O(h)$$

and when the first approximation is used, the integral becomes

$$I = \phi_{ijk}\int_v \sigma(\underline{r})d\underline{r} + O(h^4),$$

since $v \approx h^3$

$$I = \phi_{ijk} \sum_{l} \sigma(m_l)\, v_l + O(h^4).$$

The discretised approximation is then

$$\int_v \sigma(\underline{r})\, \phi(\underline{r})\, d\underline{r} \approx \phi_{ijk} \sum_{l=1}^{8} \sigma(m_l)\, v_l .$$

The centre flux method would yield a computationally more efficient approximation of the same order of accuracy, however, the edge flux system is used because it provides a more accurate approximation for the leakage term.

B3 INTEGRATION OF THE LEAKAGE TERMS

The leakage integral

$$I = \int_v \nabla .\, D_n(\underline{r})\, \nabla\phi(\underline{r}) d\underline{r}$$

is transformed by Green's theorem to

$$I = - \int_{\text{box-surface}} n .\, D_n(\underline{r}) \nabla \phi(\underline{r})\, ds , \qquad \text{(B3.1)}$$

and suitable approximations are sought which are required to satisfy the internal boundary conditions **2.1.5** and **2.1.6**. The discrete approximation for **B3.1** is given by

$$I = \sum_{l=1}^{8} I_l , \qquad \text{(B3.2)}$$

where I_l denotes integral approximations over the external surface of each of the eight sub-boxes. Attention is restricted first to boxes not touching the external boundaries. The summation in **B3.2** is broken down into summation over 24 box surfaces:

$$I = \sum_{j=1}^{24} I_j ,$$

where the 24 surfaces are ordered counter clockwise and the diagonally opposite nodes are used to specify the surface *viz*:

1(121-222), 2(121-022), 3(121-020), 4(121-220), 5(011-022), 6(011-002), 7(011-000), 8(011-020), 9(101-202), 10(101-002), 11(101-000), 12(101-200), 13(211-222), 14(211-202), 15(211-200), 16(211-220), 17(112-222), 18(112-022), 19(112-002), 20(112-202), 21(110-220), 22(110-020), 23(110-000), 24(110-200).

Consider a typical surface 13(211-222) then the leakage component $D_x(m_1)\frac{\partial\phi}{\partial x}$ may be approximated

$$D_x(m_1)\frac{\partial\phi}{\partial x}\bigg|_{13} = D_x(m_1)\frac{\partial\phi}{\partial x}\bigg|_0 + \frac{h_4}{2}\frac{\partial}{\partial x} D_x(m_1)\frac{\partial\phi}{\partial x}\bigg|_0 + $$
$$+ y\frac{\partial}{\partial y} D_x(m_1)\frac{\partial\phi}{\partial x}\bigg|_0 + z\frac{\partial}{\partial z} D_x(m_1)\frac{\partial\phi}{\partial x}\bigg|_0 + O(h^2).$$

Integrating this over the surface (13) yields

$$I_{13} = -\iint D_x(m_1)\left.\frac{\partial\phi}{\partial x}\right|_{13} dydz = -\left[\left[D_x(m_1)\left.\frac{\partial\phi}{\partial x}\right|_0 + \frac{h_4}{2}\frac{\partial}{\partial x}D_x(m_1)\left.\frac{\partial\phi}{\partial x}\right|_0\right]\frac{h_1h_6}{4} + \right.$$
$$\left. + \frac{h_1^2h_6}{16}\frac{\partial}{\partial y}D_x(m_1)\left.\frac{\partial\phi}{\partial x}\right|_0 + \frac{h_1h_6^2}{16}\frac{\partial}{\partial z}D_x(m_1)\left.\frac{\partial\phi}{\partial x}\right|_0\right] + O(h^4) \qquad \text{(B3.3)}$$

and by substituting the expansion

$$D_x(m_1)\left[\phi_{ijk} - \phi_{i+1jk}\right] = -\left[h_4 D_x(m_1)\left.\frac{\partial\phi}{\partial x}\right|_0 + \frac{h_4^2}{2}\frac{\partial}{\partial x}D_x(m_1)\left.\frac{\partial\phi}{\partial x}\right|_0\right] + O(h^3)$$

in **equation B3.3**, the result

$$I_{13} = \frac{h_1h_6D_x(m_1)}{4h_4}\left[\phi_{ijk}-\phi_{i+1jk}\right] - \frac{h_1^2h_6}{16}\frac{\partial}{\partial y}D_x(m_1)\left.\frac{\partial\phi}{\partial x}\right|_0 -$$
$$- \frac{h_1h_6^2}{16}\frac{\partial}{\partial z}D_x(m_1)\left.\frac{\partial\phi}{\partial x}\right|_0 + O(h^4) \qquad \text{(B3.4)}$$

is obtained. On the opposite face, 5(011-022), a similar result **(B3.5)** emerges

$$I_5 = \frac{h_1h_6D_x(m_2)}{4h_2}\left[\phi_{ijk}-\phi_{i-1jk}\right] - \frac{h_1^2h_6}{16}\frac{\partial}{\partial y}D_x(m_2)\left.\frac{\partial\phi}{\partial x}\right|_0 +$$
$$+ \frac{h_1h_6^2}{16}\frac{\partial}{\partial z}D_x(m_2)\left.\frac{\partial\phi}{\partial x}\right|_0 + O(h^4). \qquad \text{(B3.5)}$$

Combining **equations B3.4** and **B3.5**, while taking care to satisfy the internal boundary conditions **2.1.5** and **2.1.6** produces

$$I_{13} + I_5 = \frac{h_1h_6}{4}\left\{\frac{D_x(m_1)}{h_4}\left[\phi_{ijk}-\phi_{i+1jk}\right] + \frac{D_x(m_2)}{h_2}\left[\phi_{ijk}-\phi_{i-1jk}\right]\right\} + O(h^4)\,.$$

In a similar way, the following integral approximations may be obtained:

$$I_{17} + I_{21} = \frac{h_4h_1}{4}\left\{\frac{D_z(m_1)}{h_6}\left[\phi_{ijk}-\phi_{ijk+1}\right] + \frac{D_z(m_5)}{h_7}\left[\phi_{ijk}-\phi_{ijk-1}\right]\right\} + O(h^4)$$

$$I_{18} + I_{22} = \frac{h_2h_1}{4}\left\{\frac{D_z(m_2)}{h_6}\left[\phi_{ijk}-\phi_{ijk+1}\right] + \frac{D_z(m_6)}{h_7}\left[\phi_{ijk}-\phi_{ijk-1}\right]\right\} + O(h^4)$$

$$I_{14} + I_6 = \frac{h_3h_6}{4}\left\{\frac{D_x(m_4)}{h_4}\left[\phi_{ijk}-\phi_{i+1jk}\right] + \frac{D_x(m_3)}{h_2}\left[\phi_{ijk}-\phi_{i-1jk}\right]\right\} + O(h^4)$$

$$I_4 + I_{12} = \frac{h_4h_7}{4}\left\{\frac{D_y(m_5)}{h_1}\left[\phi_{ijk}-\phi_{ij+1k}\right] + \frac{D_y(m_8)}{h_3}\left[\phi_{ijk}-\phi_{ij-1k}\right]\right\} + O(h^4)$$

$$I_3 + I_{11} = \frac{h_2h_7}{4}\left\{\frac{D_y(m_6)}{h_1}\left[\phi_{ijk}-\phi_{ij+1k}\right] + \frac{D_y(m_7)}{h_3}\left[\phi_{ijk}-\phi_{ij-1k}\right]\right\} + O(h^4)$$

$$I_2 + I_{10} = \frac{h_2h_6}{4}\left\{\frac{D_y(m_2)}{h_1}\left[\phi_{ijk}-\phi_{ij+1k}\right] + \frac{D_y(m_3)}{h_3}\left[\phi_{ijk}-\phi_{ij-1k}\right]\right\} + O(h^4)$$

$$I_1 + I_9 = \frac{h_4h_6}{4}\left\{\frac{D_y(m_1)}{h_1}\left[\phi_{ijk}-\phi_{ij+1k}\right] + \frac{D_y(m_4)}{h_3}\left[\phi_{ijk}-\phi_{ij-1k}\right]\right\} + O(h^4)$$

$$I_{16} + I_8 = \frac{h_1h_7}{4}\left\{\frac{D_x(m_5)}{h_4}\left[\phi_{ijk}-\phi_{i+1jk}\right] + \frac{D_x(m_6)}{h_2}\left[\phi_{ijk}-\phi_{i-1jk}\right]\right\} + O(h^4)$$

$$I_{20} + I_{24} = \frac{h_4h_3}{4}\left\{\frac{D_z(m_4)}{h_6}\left[\phi_{ijk}-\phi_{ijk+1}\right] + \frac{D_z(m_8)}{h_7}\left[\phi_{ijk}-\phi_{ijk-1}\right]\right\} + O(h^4)$$

$$I_{19} + I_{23} = \frac{h_2h_3}{4}\left\{\frac{D_z(m_3)}{h_6}\left[\phi_{ijk}-\phi_{ijk+1}\right] + \frac{D_z(m_7)}{h_7}\left[\phi_{ijk}-\phi_{ijk-1}\right]\right\} + O(h^4)$$

$$I_{15} + I_7 \quad = \quad \frac{h_3 h_7}{4}\left\{\frac{D_x(m_8)}{h_4}\left[\phi_{ijk} - \phi_{i+1\,jk}\right] + \frac{D_x(m_7)}{h_2}\left[\phi_{ijk} - \phi_{i-1\,jk}\right]\right\} \quad + \quad O(h^4)\ .$$

Consequently, the approximation of $O(h^4)$ for the leakage is

$$\begin{aligned} -\int \nabla . D_n(\underline{r}) \nabla \phi(\underline{r})\, d\underline{r} \quad = \quad & a^1_{ijk}\,\phi_{ij+1\,k} \;+\; a^2_{ijk}\,\phi_{i-1\,jk} \;+\; a^3_{ijk}\,\phi_{ij-1\,k} \\ & + a^4_{ijk}\,\phi_{i+1\,jk} \;+\; a^5_{ijk}\,\phi_{ijk} \;+\; a^6_{ijk}\,\phi_{ijk+1} \;+\; a^7_{ijk}\,\phi_{ijk-1}\,. \end{aligned}$$

If I'_1 is the leftmost term of $\{I_1 + I_9\}$ and I'_9 is the rightmost term, *etc.*, then

$$\begin{aligned}
a^1_{ijk} &= -\left[\frac{h_4 h_6}{4} D_y(m_1) + \frac{h_2 h_6}{4} D_y(m_2) + \frac{h_2 h_7}{4} D_y(m_6) + \frac{h_4 h_7}{4} D_y(m_5)\right] / h_1 && (= I'_1 + I'_2 + I'_3 + I'_4) \\
a^2_{ijk} &= -\left[\frac{h_1 h_6}{4} D_x(m_2) + \frac{h_3 h_6}{4} D_x(m_3) + \frac{h_3 h_7}{4} D_x(m_7) + \frac{h_1 h_7}{4} D_x(m_6)\right] / h_2 && (= I'_5 + I'_6 + I'_7 + I'_8) \\
a^3_{ijk} &= -\left[\frac{h_4 h_6}{4} D_y(m_4) + \frac{h_2 h_6}{4} D_y(m_3) + \frac{h_2 h_7}{4} D_y(m_7) + \frac{h_4 h_7}{4} D_y(m_8)\right] / h_3 && (= I'_9 + I'_{10} + I'_{11} + I'_{12}) \\
a^4_{ijk} &= -\left[\frac{h_1 h_6}{4} D_x(m_1) + \frac{h_3 h_6}{4} D_x(m_4) + \frac{h_3 h_7}{4} D_x(m_8) + \frac{h_1 h_7}{4} D_x(m_5)\right] / h_4 && (= I'_{13} + I'_{14} + I'_{15} + I'_{16}) \\
a^6_{ijk} &= -\left[\frac{h_4 h_1}{4} D_z(m_1) + \frac{h_2 h_1}{4} D_z(m_2) + \frac{h_2 h_3}{4} D_z(m_3) + \frac{h_4 h_3}{4} D_z(m_4)\right] / h_6 && (= I'_{17} + I'_{18} + I'_{19} + I'_{20}) \\
a^7_{ijk} &= -\left[\frac{h_4 h_1}{4} D_z(m_5) + \frac{h_2 h_1}{4} D_z(m_6) + \frac{h_2 h_3}{4} D_z(m_7) + \frac{h_4 h_3}{4} D_z(m_8)\right] / h_7 && (= I'_{21} + I'_{22} + I'_{23} + I'_{24}) \\
a^5_{ijk} &= -\sum_{m \neq 5} a^m_{ijk} && \text{(B3.6)}
\end{aligned}$$

It is necessary to study the behaviour of the approximation at boundaries. Consider the face B(111-122) as an external boundary, then

$$\begin{aligned} D_x(m_2)\left.\frac{\partial\phi}{\partial x}\right|_B \quad = \quad & D_x(m_2)\left.\frac{\partial\phi}{\partial x}\right|_o + y\frac{\partial}{\partial y} D_x(m_2)\left.\frac{\partial\phi}{\partial x}\right|_o + \\ & + z\frac{\partial}{\partial z} D_x(m_2)\left.\frac{\partial\phi}{\partial x}\right|_o + O(h^2)\,, \end{aligned}$$

and upon integration over the external face

$$\begin{aligned} -\iint D_x(m_2)\left.\frac{\partial\phi}{\partial x}\right|_B dydz \quad = \quad & -\left[\frac{h_1 h_6}{4} D_x(m_2)\left.\frac{\partial\phi}{\partial x}\right|_o + \frac{h_1^2 h_6}{16}\frac{\partial}{\partial y} D_x(m_2)\left.\frac{\partial\phi}{\partial x}\right|_o + \right. \\ & \left. + \frac{h_1 h_6^2}{16}\frac{\partial}{\partial z} D_x(m_2)\left.\frac{\partial\phi}{\partial x}\right|_o\right] + O(h^4). \qquad \text{(B3.7)} \end{aligned}$$

On the opposite face (not on a boundary), NB(011-022), the integrated term is given by **equation B.3.5** as

$$\iint D_x(m_2)\frac{\partial\phi}{\partial x}\bigg|_{NB} dy\,dz = \frac{h_1h_6}{4h_2}D_x(m_2)\left[\phi_{ijk}-\phi_{i-1jk}\right] + \frac{h_1^2h_6}{16}\frac{\partial}{\partial y}D_x(m_2)\frac{\partial\phi}{\partial x}\bigg|_o + $$
$$+\frac{h_1h_6^2}{16}\frac{\partial}{\partial z}D_x(m_2)\frac{\partial\phi}{\partial x}\bigg|_o + O(h^4). \qquad \text{(B3.8)}$$

Adding equation **B3.7** and equation **B3.8** leads to

$$-\iint D_x(m_2)\frac{\partial\phi}{\partial x}\bigg|_{B} dydz + \iint D_x(m_2)\frac{\partial\phi}{\partial x}\bigg|_{NB} dydz$$
$$= \frac{h_1h_6}{4h_2}D_x(m_2)\left[\phi_{ijk}-\phi_{i-1jk}\right] - \frac{h_1h_6}{4}D_x(m_2)\frac{\partial\phi}{\partial x}\bigg|_o + O(h^4). \qquad \text{(B3.9)}$$

The second term in **equation B3.9** can be replaced directly by using either boundary condition **2.1.3** or **2.1.4**. Consequently, the leakage approximations given by **B3.6**, and modified according to **B3.10** and **B3.11** are suitable at all grid points. The modifications are

$$a^m_{ijk}=0 \quad \text{for} \quad \begin{array}{l} m=1,\ j=N_y \\ m=2, i=1 \\ m=3, j=1 \\ m=4,\ i=N_x \\ m=6, k=N_z \\ m=7, k=1 \end{array} \qquad \text{(B3.10)}$$

for both reflective and extrapolated boundaries;

$$a^5_{ijk}=-\sum_{m\neq 5}a^m_{ijk} + \frac{1}{12}[h_1+h_3][h_6+h_7]\left[\delta_{i1}/d'_{i=1} + \delta_{iN_x}/d'_{i=N_x}\right]$$
$$+\frac{1}{12}[h_2+h_4][h_6+h_7]\left[\delta_{j1}/d'_{j=1} + \delta_{jN_y}/d'_{j=N_y}\right]$$
$$+\frac{1}{12}[h_1+h_3][h_2+h_4]\left[\delta_{k1}/d'_{k=1} + \delta_{kN_z}/d'_{k=N_z}\right] \qquad \text{(B3.11)}$$

where $\delta_{nm}=1$ for $n=m$, 0 for $n\neq m$, $d'_{i=1,j=1,..}$ are the boundary extrapolation distances in transport mean free paths, with $d'=0.71$ for a free boundary and $d'=\infty$ for a reflective boundary and the appropriate h is zero on an external boundary (*e.g.* $h_4=0$ when $i=N_x$).

APPENDIX C

INCOMPLETE CHOLESKI DECOMPOSITION

Theorem. Let A be the matrix corresponding to the finite difference representation adopted in **appendix** B, then approximate decomposition of the matrix $A = LDL^T$ given by

$$l_{ij} = a_{ij} - \sum_{k=1}^{j-1} l_{ik} d_{kk} l_{jk} \quad \text{for } a_{ij} \neq 0$$

$$= 0 \quad \text{otherwise} ,$$

and $$d_{ii} = l_{ii}^{-1}$$

results in a lower triangular matrix *L*, whose elements are

$$l_{ij} = a_{ij} \quad (i \neq j) \ .$$

Proof. The proof is presented for the seven-point finite difference representation corresponding to the discretisation of the three spatial dimensional form. The result for fewer dimensions is established as portion of the larger proof.

For the seven-point finite difference representation, the elements a_{ij} of the matrix A that are non-zero can occur only in the following instances:

$$\begin{array}{lcl} j = 1 & & i = j \\ j = i \pm 1 & & i = j \pm 1 \\ j = i \pm s & \text{or} & i = j \pm s \\ j = i \pm (s+t) & & i = j \pm (s+t) \ , \end{array}$$

where s and t are given by $s = N_x$ and $t = N_x N_y$, for the ordering of ϕ used here. Now, because no infill is permitted unless a non-zero element occurs in the corresponding position in A, the lower triangular elements l_{ij} are zero except for the following three cases:

(i) $i = j + 1$,

(ii) $i = j + s$,

(iii) $i = j + s + t$.

The elements of L for case (i) are

$$l_{j+1\,j} = a_{j+1\,j} - \sum_{k=1}^{j-1} l_{j+1\,k} d_{kk} l_{jk}.$$

From the non-zero conditions applying to a_{ij}, both $l_{j+1\,k}$ and l_{jk} can be non-zero only for $k = j$. Because this combination is excluded from the summation, it immediately follows that

$$l_{j+1\,j} = a_{j+1\,j} \ .$$

Cases (ii) and (iii) follow in a similar fashion

APPENDIX D

PERFORMANCE TUNING PARAMETERS

The user of POW3D has the opportunity of setting several parameters which might influence the efficiency with which a computation is undertaken. These are described now.

D1 CONTROL OF REGION REBALANCE

rbal $r_1, r_2, r_3, r_4, r_5, r_6,$ an array used to specify the circumstances under which region rebalance might be enabled or disabled.

1,1,2,1,1,1 are the default values used by POW3D.

Region rebalance is always disabled when the optimal extrapolation parameter (ω), for the SLOR iterative scheme is being determined. The user is able to unconditionally disable region rebalance at all other times by specifying

rbal = 0,0,0,0,0,0 (or in more compact form **rbal**=6*0)

in POW3D data. The user, however, has the option of specifying control in a finer fashion through a more selective use of the six parameters available.

For example, r_2 and r_3 are used to specify limits on the group passes for which region rebalance is disabled.

If ig_{pass} represents the count of current pass through the various neutron groups for a particular inner iteration, then region rebalance is disabled for

$ig_{pass} < r_2$ or when $ig_{pass} > r_3$.

(The default is no region rebalance for $ig_{pass} < 1$ or $ig_{pass} > 2$.)

Region rebalance is also avoided for the following stages of outer loop convergence by setting the appropriate flag to zero.

Stage	Acceleration	Error	Flag
3	Power iteration	$b^{(n)} \leq$ **acclam**(3)	$r_6 = 0$
2	Chebyschev extrapolation $4 \leq$ order ≤ 6	$b^{(n)} \leq$ **acclam**(2) and $S^{(n)}_{err} \leq$ **accfo**(2)	$r_5 = 0$
1	As above but more confident use $6 \leq$ order ≤ 10	$b^{(n)} \leq$ **acclam**(1) and $S^{(n)}_{err} \leq$ **accfo**(1)	$r_4 = 0$

Specifying, r_4, r_5, or r_6 for the appropriate stage of convergence as 1, ensures that with other conditions permitting, region rebalance is enabled for that stage of every outer iteration. The use of an other positive integer permits region rebalance only if the outer iteration is a multiple of that integer. (i.e. r_4 = 3 permits region rebalance for every third outer iteration, provided the converged process is in Stage 1).

Region rebalance is normally disabled for the first outer iteration, or after a restart. Should the user wish to force region rebalance in these circumstances, r_1 must be set so that

r_1 * accfo(1) > 1.

D2 CONTROL OF GROUP (ENERGY) REBALANCE

gbal g_1,g_2,g_3,g_4,g_5,g_6, an array used to specify the circumstances under which group rebalance might be enabled or disabled.

1,1,1,1,1,1 are the default values used by POW3D.

The user is able to unconditionally disable group rebalance at all times by specifying

gbal = 0,0,0,0,0,0 (or in more compact form **gbal**=6*0)

in POW3D data. The user, however, has the option of specifying control in a finer fashion through a more selective use of the six parameters available.

For example, g_2 and g_3 are used to specify limits on the group passes for which group rebalance is disabled. The default (see below), is that group rebalance is only applied for the first group pass of each outer iteration. (Group balance is applied across all groups, when the first energy group with upscatters into it is encountered.)

If ig_{pass} represents the count of current pass through the various neutron groups for a particular inner iteration, then group rebalance is disabled for
$ig_{pass} < g_2$ or when $ig_{pass} > g_3$.
(The default is no group rebalance for $ig_{pass} > 1$.)

Group rebalance is also avoided for the following stages of outer loop convergence by setting the appropriate flag to zero.

Stage	Acceleration	Error	Flag
3	Power iteration	$b^{(n)} \leq$ **acclam(3)**	$g_6 = 0$
2	Chebyschev extrapolation $4 \leq$ order ≤ 6	$b^{(n)} \leq$ **acclam(2)** and $S_{err}^{(n)} \leq$ **accfo(2)**	$g_5 = 0$
1	As above but more confident use $6 \leq$ order ≤ 10	$b^{(n)} \leq$ **acclam(1)** and $S_{err}^{(n)} \leq$ **accfo(1)**	$g_4 = 0$

Specifying g_4, g_5, or g_6 for the appropriate stage of convergence as 1, ensures that with other conditions permitting, group rebalance is enabled for that stage of every outer iteration. The use of any other positive integer permits group rebalance only if the outer iteration is a multiple of that integer. (i.e. $g_4 = 3$ permits group rebalance for every third outer iteration, provided the converged process is in Stage 1).

Group rebalance is normally disabled for the first outer iteration, or after a restart. Should the user wish to force group rebalance in these circumstances, g_1 must be set so that

g_1 * **accfo**(1) > 1.

D3 CONTROL OF UPSCATTER PASSES

The user is able to control the upscatter strategy of POW3D through use of the parameter nig. The parameter consists of three numbers, each of which is made up of three separate two digit numbers concatenated to form a six digit integer. The default values used by the code are

nig = 3*080201

Which of the three nig numbers is used, will be determined by the stage of the outer iterative process (as used for region rebalance, **section D1**). For example, the first **nig** number is used when one is most confident in the state of convergence of the process, and the third during the preliminary stage when confidence is least. The state of convergence is determined by POW3D.

For a selected state of convergence, the components of the appropriate **nig** entity are used as described now by the rather pragmatic but effective procedure:

Consider a six digit entity

nig = $\text{nig}_1 \times 10000 + \text{nig}_2 \times 100 + \text{nig}_3$.

The number of upscatter passes i2 is determined

for real calculations

$$\begin{aligned} i2 &= \text{nig}_1 \\ i1 &= \max(\text{nig}_2, 1) \\ j2 &= \text{nig}_3 \\ i2 &= \max(i2, j2) \\ j2 &= \text{nig}_3 + \frac{ng - ngs}{2} + \frac{ng - ngs}{4} \\ j2 &= \min(i2, j2) \\ i2 &= j2 \end{aligned}$$

for adjoint calculations

$$\begin{aligned} i2 &= \text{nig}_1 \\ i1 &= \max(\text{nig}_2, 1) \\ j2 &= \text{nig}_3 \\ i2 &= \max(i2, j2) \end{aligned}$$

where $ng - ngs$ is the number of groups involved in the upscatter process and the integer division process is performed as defined in FORTRAN. For example, a real 10 group problem with 6 upscattering groups would take $i2 = j2 = \min(8,5) = 5$ upscattering passes at most.

A loop is set up to iterate over the groups with upscatter for at most i2 passes. Under the following conditions, however, it is possible to take an early exit at the end of a loop pass for real calculations:

i. spatial convergence has been achieved for all groups,
ii. the maximum number of iterations for any group with upscatter is $\leq$ i1 .

For adjoint calculations, an early exit is taken when:

i. spatial convergence has been achieved for all groups,
ii. at least j2 "upscatter" passes are completed,
iii. the maximum number of iterations for any group with "upscatter" is $\leq$ i1 .

D4 SPECIFYING THE ACCURACY REQUIRED FOR SOLUTION OF THE INNER SYSTEMS OF EQUATIONS

accfi	a_1, a_2	two real numbers used to specify the relative accuracy of inner iterations,
	0.07,0.1	are the default values used by POW3D.

The current accuracy of the overall process (raccfo) is computed in POW3D at every outer iteration and that value (combined with **accfi**) is used to determine an accuracy to which the linear equations will be solved when one of the various linear equation solving strategies is employed. The accuracy is set:

raccfo * a_1	for two-dimensional problems, for three dimensional problems solved with **iccg**, for planes of a three-dimensional problem solved with **slor** or **mini**,

raccfo*a_2	for the third-dimension solver (**slor** or **mini**) when a two level inner solution strategy (*e.g.* minimini) is employed.

D5 SPECIFYING THE MINIMUM NUMBER OF ITERATIONS FOR ITERATIVE SOLUTION OF LINEAR EQUATIONS

Irrespective of convergence having been obtained for the iterative solution of the linear systems of equations, POW3D will force a minimum number of iterations to be performed before an exit from that loop is permitted. The rules for the three iterative methods **mini**, **slor** and **iccg** are slightly different. The rules for **mini** were given in **section 9.3**. The parameter mininf(3) specifies the minimum number of iterations that must be taken before an exit is permitted. The same parameter is used to control the exit procedure for **iccg**.

For **slor** the situation is far more complicated, this is due primarily for the need to compute a good estimate of the over-relaxation parameter whilst the iterative system is being solved. The following pragmatic algorithm is used and the user is able to exert some influence through the parameter **minit**.

minit	m_1,m_2,m_3	three integer numbers through which the algorithm is controlled,
	50,10,2	are the default values used by POW3D.

Let Min_{it} be the minimum number of spatial iterations that must be performed.

If the over-relaxation factor is not determined

Min_{it} $= m_1$ for the first pass (first outer and first inner)
$= m_2$ other passes
POW3D will calculate Min_{it} if Min_{it} determined by the above rules is zero. This would not appear to be a wise option for the user to force, as the calculation is based on the eigenvalue estimate which is unlikely to be accurate if the over-relaxation factor has not yet been determined accurately. (See **section 9.8** on **acclam**.)

If the over-relaxation factor is determined

Min_{it} $= m_3$ however, an exit is not automatically allowed on the first iteration even if m_3=1 is specified. Under that condition the exit test is carried out on a previously POW3D computed limit (if not a restart) or a currently computed POW3D limit (if a restart).

When POW3D calculates Min_{it} it is always bounded $m_1 \geq Min_{it} \geq m_3$.

D6 MORE ON SPECIFYING ACCURACY FOR TERMINATION

In **sections 9.7-9.8** the issue of stopping criteria was introduced. The user has two parameters **acclam** anc **accfo** though which control over convergence and stopping may be exerted.

accfo	a_1,a_2,a_3	are used for the three stages of convergence already described,
	1.E-4, 1.E-3, 1.E-2	are the default values.

The parameter **accfo** is used primarily to test for convergence on local fission accuracy where it alsc determines the stage of convergence. The paradigms adopted to converge the calculation in each stage are rather different. The parameter is also used in the decision strategies for region rebalance and th component a_1 in limiting the accuracy demanded by POW3D of the linear equation algorithms.

acclam a_{11},a_{21},a_{31} a_{12},a_{22},a_{32}	are used for two purposes in the three stages of convergence.
1.E-4, 1.E-3, 1.E-2 5.E-2, 5.E-2, 5.E-2	are the default values.

The first tripple a_{11},a_{21},a_{31} are the values of **acclam** treated in **sections 9.7-9.8** and are directed at achieving overall neutron balance in the calculation.

The second tripple are used in determining convergence at each stage of the dominance calculation.

D7 ESTABLISHING AN ESTIMATE FOR THE TRIAL FLUX

Where a trial flux is computed to commence a POW3D calculation, the shape of that flux may be influenced by the use of two parameters:

trflx $t_1,t_2,$

0., 0.975 are the default values used by POW3D.

The parameters are used to "harden" the cos function used to approximate the Bessel function

$$\cos(ws) * \cos(c * ws) * \frac{\cos(t_1 * ws)}{\cos(t_2 * ws)},$$

where ws and c are appropriate arguments.

D8 ITERATION LIMITS

nol the maximum number of outer iterations permitted before the code decides convergence has not been achieved (default 100).

nil the maximum number of inner iterations permitted before the code decides convergence has not been achieved (default 100).

D9 CONTROL OF DOMINANCE DETERMINATION

The dominance ratio is determined by the code and its value is the parameter underpinning the Chebychev acceleration method used in POW3D. A determination of this parameter to a suitable accuracy is an involved procedure and the user may influence the strategy of the calculation through:

nop nop_1, nop_2, nop_3

where the appropriate element of **nop** selected is determined by the stage of convergence achieved (as identified in **section C1**). Each element itself consists of two components concatenated through

$\mathbf{nop}_i = n_1 * 100 + n_2.$

The number of outers iterations that must completed before dominance is recalculated is given by n_2, whilst the maximum number of outer iterations permitted before dominance is recalculated is n_1. The default values used are

nop 1006, 0604, 0404.

D10 CONTROL OVER OMEGA DETERMINATION (SLOR METHOD)

somega initial value used for ω.

The initial value of ω used in the **slor** iterative solution algorithm for the solution of systems of linear equations is specified in **somega.** The default used in POW3D is 1.

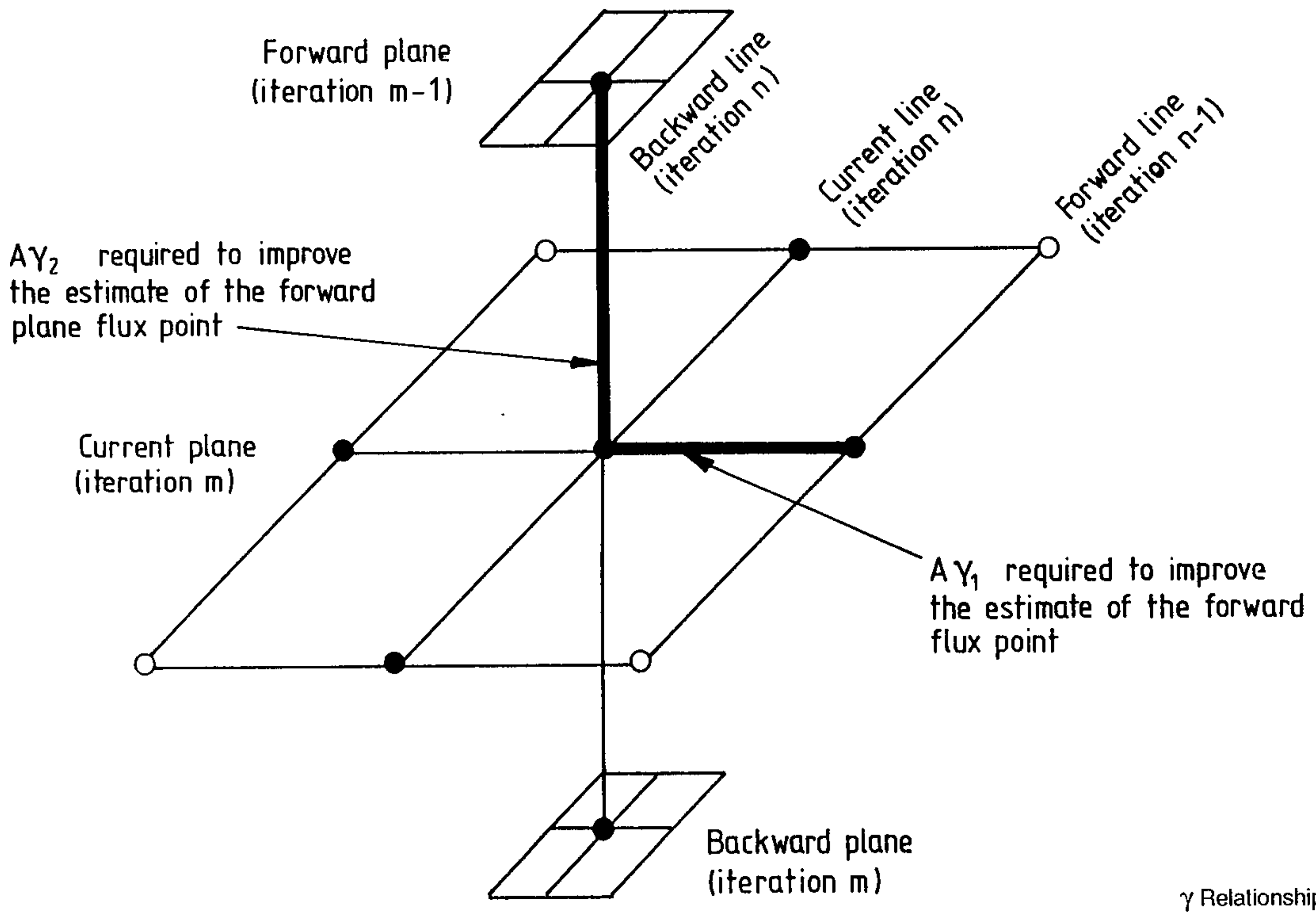
acomg2 controls the accuracy in estimating ω.

This is the square of the accuracy that is required for the determination of an updated value of ω.

D11 CONTROL OVER MINI GAMMA LIMIT

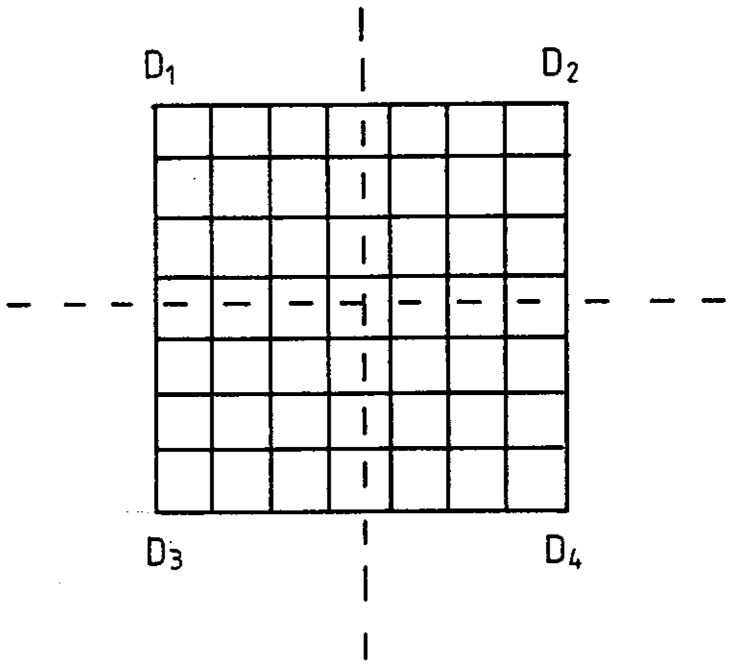
The upper γ limit (**section 6.3**), applied to the **mini** iterative method (in energy alone) may be altered through the parameter **sgamlt**. The default settings (an upper γ limit of 1) are

sgamlt 1000*1.



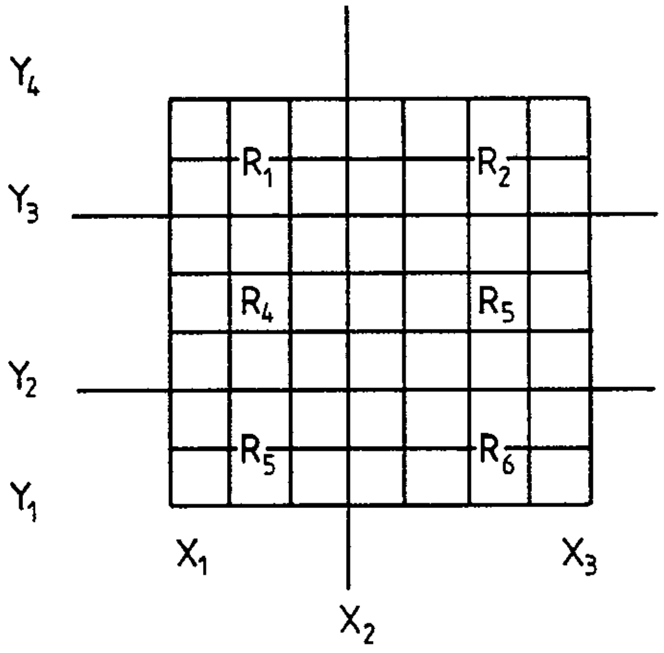
γ Relationship for 3-D problem

Figure 1

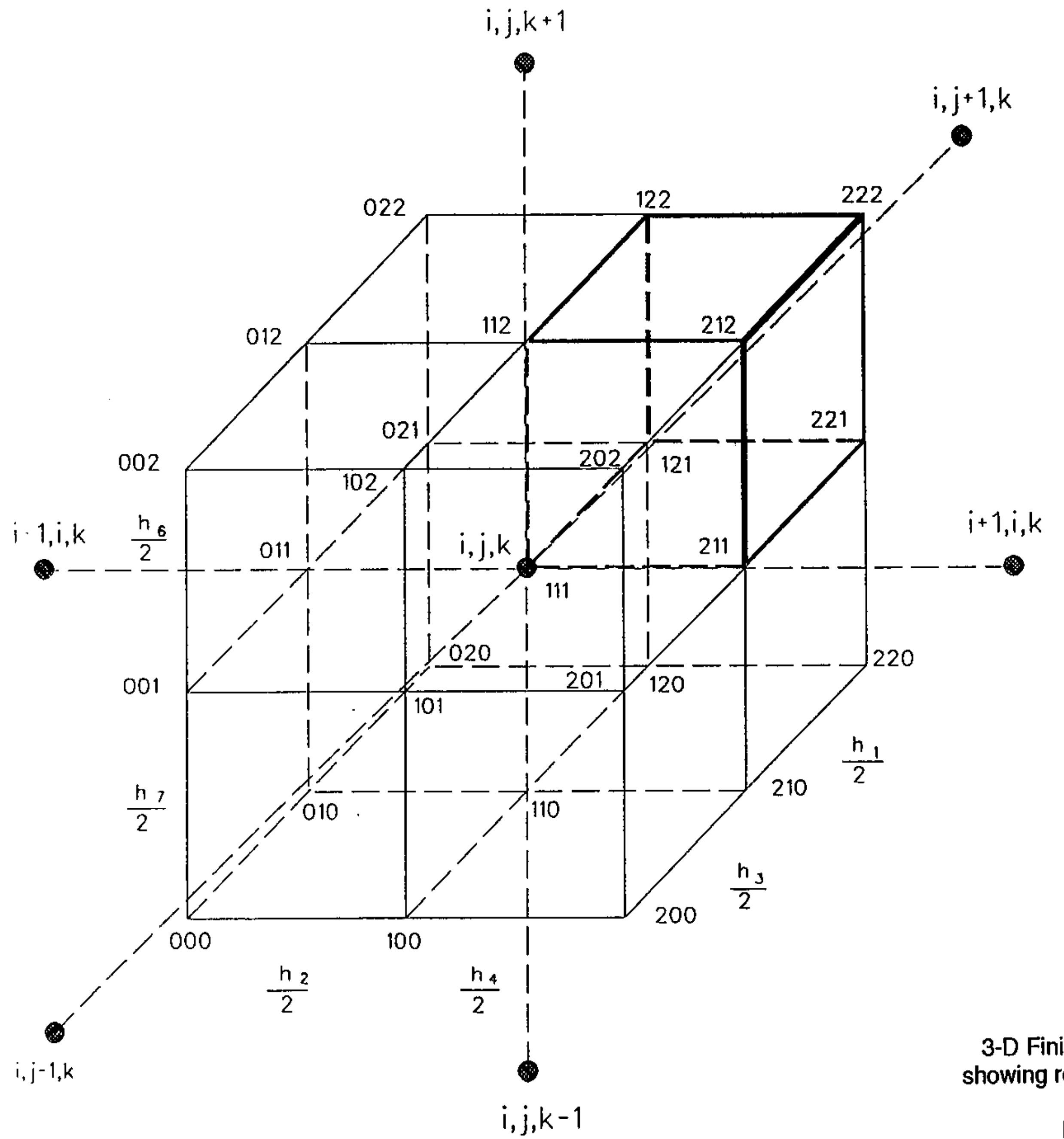


Disjunctive partitioning of coarse grid for 4 subdomains D_m

Figure 2



Coarse mesh grid for pyramid partitioning
Figure 3



3-D Finite difference grid
showing region of integration

Figure B1