NE/NP

A Two-Dimensional, Semi-Analytic Expansion Method for Nodal Calculations

by

DOE/OR/00033--T694

Scott P. Palmtag

B.S., Nuclear Engineering, University of Missouri-Rolla
(1993)

Submitted to the Department of Nuclear Engineering
in partial fulfillment of the requirements for the degree of

Master of Science in Nuclear Engineering

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

August 1995

Author ..... Scott P. Palmtag .....
Department of Nuclear Engineering
August 11, 1995

Certified by ..... A.F. Henry .....
Allan F. Henry
Professor of Nuclear Engineering
Thesis Supervisor

Certified by ..... K.F. Hansen .....
Kent F. Hansen
Professor of Nuclear Engineering
Thesis Reader

Accepted by ..... Jeffrey Freidberg .....
Jeffrey P. Freidberg
Chairman, Departmental Committee on Graduate Students

A Two-Dimensional, Semi-Analytic Expansion Method for Nodal Calculations

by

Scott P. Palmtag

Submitted to the Department of Nuclear Engineering
on August 11, 1995, in partial fulfillment of the
requirements for the degree of
Master of Science in Nuclear Engineering

Abstract

Most modern nodal methods used today are based upon the transverse integration procedure in which the multi-dimensional flux shape is integrated over the transverse directions in order to produce a set of coupled one-dimensional flux shapes. The one-dimensional flux shapes are then solved either analytically or by representing the flux shape by a finite polynomial expansion. While these methods have been verified for most light-water reactor applications, they have been found to have difficulty predicting the large thermal flux gradients near the interfaces of highly-enriched MOX fuel assemblies.

A new method is presented here in which the neutron flux is represented by a non-seperable, two-dimensional, semi-analytic flux expansion. The main features of this method are (1) the leakage terms from the node are modeled explicitly and therefore, the transverse integration procedure is not used, (2) the corner point flux values for each node are directly edited from the solution method, and a corner-point interpolation is not needed in the flux reconstruction, (3) the thermal flux expansion contains hyperbolic terms representing analytic solutions to the thermal flux diffusion equation, and (4) the thermal flux expansion contains a thermal to fast flux ratio term which reduces the number of polynomial expansion functions needed to represent the thermal flux.

This new nodal method has been incorporated into the computer code COLOR2G and has been used to solve a two-dimensional, two-group colorset problem containing uranium and highly-enriched MOX fuel assemblies. The results from this calculation are compared to the results found using a code based on the traditional transverse integration procedure.

Acknowledgments

I would like to express my graditude to my advisor, Professor Allan F. Henry, for his guidance and support during my thesis research and education at MIT. I would also like to thank Kord Smith for his many suggestions and for taking the time to work with me on this project.

Further, I would like to thank my roommates, Renaud Fournier and Prasanth Duvvur, for putting up with me over the last year and a half, and for making life at MIT even more interesting.

Finally, I would like to thank my fellowship sponsor for making my studies at MIT possible and for allowing me the flexibility to perform research on a topic of my choice.

This research was performed under appointment to the Nuclear Engineering/Health Physics Fellowship Program administered by the Oak Ridge Institute for Science and Education for the U. S. Department of Energy.

Contents

List of Figures

List of Tables

Chapter 1

Introduction and Background

1.1 Overview

For the design and operation of today's light water reactors, it is essential that methods be available that are able to predict the neutron flux, and hence the local power and reaction rates, at every location in the reactor accurately and quickly. Traditionally, finite-difference methods have been used to accomplish this goal, however, for the last 20 years, modern nodal methods have been under development which solve the neutron diffusion equation up to 2 orders of magnitude faster than finite-difference methods while still retaining the same accuracy.

Most modern nodal methods used today are based upon the transverse integration procedure. In the transverse integration procedure, the three-dimensional flux shape is integrated over the directions transverse to each direction in order to obtain a set of coupled one-dimensional flux shapes. The transverse integration procedure has been found to be effective for most light-water reactor problems encountered. However, recently there has been an interest shown in using plutonium mixed-oxide (MOX) fuel in light-water reactors. For this case, it has been found that the transverse integration procedure has difficulty predicting the steep thermal flux gradients near interfaces between UO_2 and MOX fuel.

In this thesis, a new nodal method has been developed which does not use the transverse integration procedure. Instead, the flux shape is represented by a non-

separable, two-dimensional expansion function consisting of polynomial functions and hyperbolic terms. This new nodal method has been applied to a simple test problem containing UO_2 and MOX fuel assemblies and the results are compared to a nodal method which uses the transverse integration procedure.

1.2 Background

The traditional method of solving the spatial dependence of the neutron diffusion equation has been the finite-difference (FD) method. In the FD method, the spatial domain of the reactor is subdivided into small meshes and the neutron flux is calculated for each mesh point. As the mesh size is decreased, the FD method will converge to the exact solution of the neutron diffusion equation. In order to obtain accurate results, it has been found that a mesh size on the order of 1 cm is needed. Therefore, in order to perform a three-dimensional model of a typical light-water reactor core, over a million mesh points are needed. The size of this full-core model makes the calculation very expensive by today's standards and nearly impossible to perform thirty years ago. Considering the shear size of the problem and coupled with today's desire to run many full-core models for transient analysis and fuel management optimization, it is essential that methods be available to run full-core models faster then the FD method but still retain the same accuracy.

The first generation of nodal methods used to improve on the FD method were developed 30 years ago and are characterized by the FLARE model [1]. The main characteristic of nodal methods is that large mesh sizes are used, where the size of each mesh is on the order of a fuel assembly. Using larger node sizes reduces the number of unknowns in the problem and therefore, decreases the computational time needed. In order to overcome the inaccuracy introduced with using a FD approximation with large mesh sizes, the original nodal methods used empirical correction factors to ensure that the nodal results matched experimental and/or higher order calculations. The problem with this method is that the accuracy of the empirical correction factors are limited and must be calculate a priori. Also, the solutions do not converge to the

exact solution of the diffusion equation as the mesh size is decreased, and therefore the accuracy of the method cannot be verified without performing some higher-order calculation.

Over the past 20 years, however, modern nodal methods have been developed in which the nodal coupling relations are found by using mathematically systematic methods. Most of these modern nodal methods are based upon the transverse integration procedure in which the multi-dimensional reactor flux is integrated over the directions transverse to each direction in order to reduce the three-dimensional problem to a set of coupled one-dimensional problems. The one-dimensional flux shapes are solved in order to find the nodal coupling coefficients. The nodal coupling coefficients are then used in a global reactor problem in order to calculate the eigenvalue and the volume and surface-averaged flux values for each node. The calculation to find the nodal coupling coefficients and the global calculation are usually solved iteratively until both solutions converge. One advantage that modern nodal methods have over the original nodal methods is that they converge to the exact solution of the diffusion equation as the node size is reduced, thereby bounding the error in the solution.

Several methods have been used to solve the one-dimensional flux shapes obtained from the transverse integration procedure. Analytic methods such as the Analytic Nodal Method [4] and Nodal Green's Function Method [5] are available, but due to the complexity of the problem, they are generally limited to two energy groups. Another method available to solve the one-dimensional problems is by representing the flux shapes as finite polynomial expansions. Some examples of the polynomial method are the Nodal Expansion Method [6] and the QPANDA formalism [7] used in SIMULATE–3 [8].

Once a problem is solved for the global flux values, the pin-by-pin flux distribution within each node is calculated using a flux reconstruction, or dehomogenization, method. For a detailed description of this calculation, the reader is referred to the review paper by Smith [3]. For this thesis, it shall be sufficient to point out that during the flux reconstruction, the corner point flux values must be calculated using

the one-dimensional flux shapes and some type of interpolation scheme.

Modern nodal methods employing the transverse integration procedure have been found to be very successful for most light-water reactor problems. However, with the recent interest in using plutonium mixed-oxide (MOX) fuel, it has been found that the transverse integrated polynomial nodal methods are unable to calculate the steep thermal flux gradients found near UO_2 and MOX interfaces.

Several methods have been proposed in order to improve the calculation of the steep thermal flux gradients near MOX interfaces. One method used has been to add analytic terms to the thermal flux expansion in a polynomial nodal method [9]. While this method has been successful, it still has difficulty predicting the thermal flux shape near corners where 3 UO_2 fuel-assemblies are adjacent to a MOX fuel-assembly.

Another method being used, referred to as the Analytic Function Expansion Nodal Method [16], represents the neutron flux as an expansion of non-separable analytic functions. This method does not make the transverse integration approximation and keeps the flux distribution in two-dimensions.

In this thesis a new nodal method will be introduced in which the the neutron flux will be represented by a non-separable, two-dimensional expansion of polynomial and hyperbolic functions. This method will be referred to as the two-dimensional, semi-analytic nodal method (2D-SANM).

1.3 Research Objectives

The objective of this research is to derive the nodal equations used by the 2D-SANM and develop a method that can be used to solve these equations. The solution method will then be applied to a 4-assembly colorset problem containing UO_2 and MOX fuel assemblies and the results will be compared with the results obtained from a nodal method using the transverse integration procedure.

1.4 Thesis Organization

In Chapter 2 the derivation of the 2D-SANM equations is presented along with the derivation of the necessary conditions needed in order to solve for the flux expansion coefficients.

In Chapter 3 a method is presented to solve the nodal equations using the well-known source-iteration method. This method is then applied to solve the flux distribution in a 4-assembly colorset problem containing UO_2 and MOX fuel assemblies arranged in three different patterns. The results are compared to results obtained using SIMULATE–3.

Chapter 4 includes a discussion on how this method might be expanded to a non-linear solution method, suggestions on how to improve the accuracy of the method, and a discussion on methods that can be used to extend this method to more than two energy groups and three dimensions.

Finally, Appendix A includes a listing of the computer program COLOR2G that was used to generate the results in Chapter 3.

Chapter 2

Derivation of Nodal Equations

2.1 Introduction

In this chapter the nodal equations for a two-group, semi-analytic, two-dimensional nodal expansion method (2D-SANM) will be derived. In this method, the fast neutron flux will be approximated by a non-separable, two-dimensional expansion of polynomial functions and the thermal neutron flux will be approximated by a non-separable two-dimensional expansion of polynomial and hyperbolic functions. Several conditions will then be derived to form a set of equations, which when solved, will yield the flux expansion coefficients corresponding to the solution of the diffusion equation for the problem.

2.2 Flux Expansions

The first step in the derivation of the two-dimensional nodal expansion method is to divide the radial plane of the reactor into rectangular sections called nodes. The size of each node will usually correspond to a single or quarter fuel assembly. A typical node, denoted by the coordinates ij, is shown in Figure 2-1. The dimensions of node ij are defined by the coordinates:

$$x \in [x_i, x_{i+1}] \text{ and}$$
$$y \in [y_j, y_{j+1}].$$

The widths of node ij in the x and y directions are defined as

$$h_x^i = x_{i+1} - x_i \text{ and} \tag{2.1}$$
$$h_y^j = y_{j+1} - y_j. \tag{2.2}$$

The volume of node ij is then given by the expression

$$V^{ij} = h_x^i \, h_y^j. \tag{2.3}$$

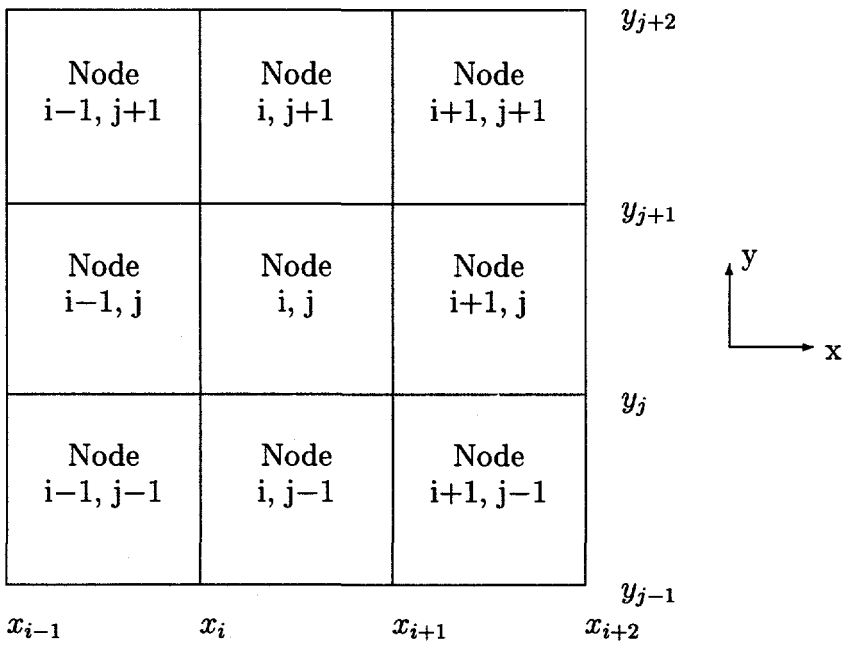

Figure 2-1: Two-dimensional coordinate system used for node ij.

In this derivation the energy dependence of the neutron flux will be represented by two energy groups. The fast flux in the interior of node ij will be approximated

by a two-dimensional, non-separable polynomial expansion given by:

$$\phi_1^{ij}(x,y) \approx \sum_{m,n=0}^{3} a_{mn}^{ij} f_m\left(\frac{2x - x_{i+1} - x_i}{2\,h_x^i}\right) f_n\left(\frac{2y - y_{j+1} - y_j}{2\,h_y^j}\right), \qquad x, y \in V^{ij} \tag{2.4}$$

where

$$\begin{aligned} f_0(\xi) &= 1, \\ f_1(\xi) &= \xi, \\ f_2(\xi) &= 3\xi^2 - \frac{1}{4}, \\ f_3(\xi) &= 2\,\xi^3, \end{aligned} \tag{2.5}$$

and $\mathbf{a}^{ij}$ is a matrix of fast expansion coefficients containing 12 non-zero elements. The non-zero elements of $\mathbf{a}^{ij}$ are given in matrix form as:

$$\mathbf{a}^{ij} = \begin{bmatrix} a_{00} & a_{01} & a_{02} & a_{03} \\ a_{10} & a_{11} & a_{12} & 0 \\ a_{20} & a_{21} & a_{22} & 0 \\ a_{30} & 0 & 0 & a_{33} \end{bmatrix}^{ij}. \tag{2.6}$$

The fast flux expansion functions given in Eq.(2.5) are similar to the expansion functions used to represent the transverse integrated flux in many codes which employ polynomial methods [2, 6, 8, 11]. One advantage to using these fast flux expansion functions is that they have the integral property:

$$\int_{-\frac{1}{2}}^{\frac{1}{2}} f_m(\xi)\,d\xi = \begin{cases} 1 & m = 0, \\ 0 & m \neq 0 \end{cases} \tag{2.7}$$

so that the volume average fast flux in node ij, denoted with a double-overline, is simply given by

$$\overline{\overline{\phi_1^{ij}}} \equiv \frac{1}{V^{ij}} \int_{x_i}^{x_{i+1}} dx \int_{y_j}^{y_{j+1}} dy\,\phi_1(x,y) = a_{00}^{ij}. \tag{2.8}$$

Most polynomial expansion methods use the same form of the fast flux expansion to also represent the thermal flux expansion. However, Esser has demonstrated that adding terms to the thermal flux expansion which represent analytic solutions to the homogeneous diffusion equation will result in better predictions of the severe flux gradient at the surfaces of highly enriched MOX assemblies [9]. Using this same motivation, the thermal flux in the interior of node ij will be approximated by a two-dimensional, non-separable expansion of polynomial and hyperbolic functions given by:

$$\begin{aligned}\phi_2^{i,j}(x,y) \approx & \sum_{m,n=0}^{5} b_{mn}^{ij}\, g_m\left(\frac{2x-x_{i+1}-x_i}{2\,h_x^i}\right) g_n\left(\frac{2y-y_{j+1}-y_j}{2\,h_y^j}\right) \\ & - b_{02}^{ij}\,\frac{2}{\kappa^{ij}}\, g_3\left(\frac{1}{2}\right) - b_{20}^{ij}\,\frac{2}{\kappa^{ij}}\, g_3\left(\frac{1}{2}\right) \\ & - b_{44}^{ij}\,\frac{8}{(\kappa^{ij})^2}\, g_5\left(\frac{1}{2}\right)\, g_5\left(\frac{1}{2}\right) + t_s^{ij}\,\phi_1^{ij}(x,y), \quad x,y \in V^{ij}\end{aligned} \tag{2.9}$$

where

$$\begin{aligned} g_0(\xi) &= 1, \\ g_1(\xi) &= \xi, \\ g_2(\xi) &= \cosh(\kappa^{ij}\,\xi), \\ g_3(\xi) &= \sinh(\kappa^{ij}\,\xi), \\ g_4(\xi) &= \cosh\left(\frac{\kappa^{ij}}{\sqrt{2}}\,\xi\right), \\ g_5(\xi) &= \sinh\left(\frac{\kappa^{ij}}{\sqrt{2}}\,\xi\right), \end{aligned} \tag{2.10}$$

and $\mathbf{b}^{ij}$ is a matrix of thermal expansion coefficients with 12 non-zero terms given in

matrix form as:

$$\mathbf{b}^{ij} = \begin{bmatrix} b_{00} & b_{01} & b_{02} & b_{03} & 0 & 0 \\ b_{10} & b_{11} & 0 & 0 & 0 & 0 \\ b_{20} & 0 & 0 & 0 & 0 & 0 \\ b_{30} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & b_{44} & b_{45} \\ 0 & 0 & 0 & 0 & b_{54} & b_{55} \end{bmatrix}^{ij} . \tag{2.11}$$

The κ^{ij} term is defined such that the thermal flux expansion functions are particular solutions to the homogeneous part of the thermal diffusion equation. The t_s^{ij} term represents the thermal to fast flux ratio for an infinite medium. The exact definitions for κ^{ij} and t_s^{ij} will be given in the next section.

The constants added to the b_{02}, b_{20}, and b_{44} terms in Eq.(2.9) are present so that the thermal volume averaged flux over node ij can be represented by

$$\overline{\overline{\phi_2^{ij}}} \equiv \frac{1}{V^{ij}} \int_{x_i}^{x_{i+1}} dx \int_{y_j}^{y_{j+1}} dy\, \phi_2(x,y) = b_{00}^{ij} + t_s^{ij}\, \overline{\overline{\phi_1^{ij}}}. \tag{2.12}$$

Using the fast and thermal flux expansions given by Eqs.(2.4) and (2.9) will give a total of 24 unknown expansion coefficients for each node in the problem. Therefore, in order to obtain a unique solution for the flux expansion coefficients, 24 linearly independent conditions must be used for each node. In the following sections, the conditions necessary to solve for the flux expansion coefficients will be given.

It should be noted that a separate flux expansion is defined for each node in the problem and therefore, each node will have its own set of flux expansion coefficients. In the following sections, the conditions necessary to solve the problem will be derived for a single node ij. In order to simplify the notation, the superscripts will be left off of the individual flux expansion coefficients. Unless otherwise stated, the expansion coefficients will refer to the arbitrary node ij.

2.3 Neutron Balance Condition

The first condition that will be imposed on the flux expansion is that a neutron balance is satisfied in the interior of each node. This condition simply states that the net neutron production rate is equal to the net leakage rate from each node.

The starting point in deriving the neutron balance equation is the few-group diffusion equation given by

$$\nabla \cdot \mathbf{J}_g(\mathbf{r}) + \Sigma_{tg}(\mathbf{r})\phi_g(\mathbf{r}) = \sum_{g'=1}^{G} \left[\frac{1}{k_{eff}} \chi_g(\mathbf{r}) \nu\Sigma_{fg'}(\mathbf{r}) + \Sigma_{gg'}(\mathbf{r}) \right] \phi_{g'}(\mathbf{r}) \tag{2.13}$$

$$\mathbf{J}_g(\mathbf{r}) = -D_g(\mathbf{r})\nabla\phi_g(\mathbf{r}) \tag{2.14}$$

where

$\mathbf{J}_g =$ net neutron current vector for group g,

$\phi_g =$ scalar neutron flux in group g,

$\Sigma_{tg} =$ total macroscopic cross section for group g,

$k_{eff} =$ reactor eigenvalue,

$\chi_g =$ fission spectrum for group g,

$\nu\Sigma_{fg} =$ number of neutrons emitted per fission times the macroscopic fission cross section for group g,

$\Sigma_{gg'} =$ macroscopic scattering cross section from group g' to group g, and

$D_g =$ diffusion coefficient for group g.

For a two-dimensional node in Cartesian geometry with spatially constant cross sections and two energy groups the diffusion equation can be written as:

$$\nabla \cdot \mathbf{J}_g(x,y) + \Sigma_{tg}^{ij}\,\phi_g(x,y) = \sum_{g'=1}^{2} \left[\frac{1}{k_{eff}} \chi_g^{ij} \nu\Sigma_{fg'}^{ij} + \Sigma_{gg'}^{ij} \right] \phi_{g'}(x,y) \tag{2.15}$$

$$\mathbf{J}_g(x,y) = -D_g^{ij}\nabla\phi_g(x,y), \quad x,y \in V^{ij}. \tag{2.16}$$

Integrating equation (2.15) over the volume of the node and applying Gauss's divergence theorem to the leakage term gives the neutron balance equation for node ij as:

$$\frac{1}{h_x^i}\left[\overline{J_{xg}^{ij}}(x_{i+1}) - \overline{J_{xg}^{ij}}(x_i)\right] + \frac{1}{h_y^j}\left[\overline{J_{yg}^{ij}}(y_{j+1}) - \overline{J_{yg}^{ij}}(y_j)\right] + \Sigma_{tg}^{ij}\,\overline{\overline{\phi_g^{ij}}}$$
$$= \sum_{g'=1}^{2}\left[\frac{1}{k_{eff}}\chi_g^{ij}\nu\Sigma_{fg'}^{ij} + \Sigma_{gg'}^{ij}\right]\overline{\overline{\phi_{g'}^{ij}}}, \quad g = 1,2 \tag{2.17}$$

where the face-average currents are defined as

$$\overline{J_{xg}^{ij}}(x) \equiv \frac{1}{h_y^j}\int_{y_j}^{y_{j+1}} dy\, J_{xg}(x,y) \tag{2.18}$$

$$\overline{J_{yg}^{ij}}(y) \equiv \frac{1}{h_x^i}\int_{x_i}^{x_{i+1}} dx\, J_{yg}(x,y) \tag{2.19}$$

and the double overlines represent the volume average fluxes that were defined in Eqs.(2.8) and (2.12).

At this point in the derivation a few simplifying assumptions will be made. The first assumption is that the widths of all nodes in both the x and y directions are equal, or

$$h = h_x^i = h_y^j, \text{ for all } i,j. \tag{2.20}$$

This assumption is justified since most light water reactors are based on cores which are composed of square fuel assemblies. The next assumption, commonly used for two-group theory in light water reactors, is that the thermal group fission spectrum is zero ($\chi_2 = 0$) and that there is no upscattering from the thermal group to the fast group ($\Sigma_{12} = 0$). Making these assumptions and noting that the removal cross section is defined as the total cross section minus the in-group scattering cross section ($\Sigma_{Rg} = \Sigma_{tg} - \Sigma_{gg}$), the fast and thermal neutron balance equations can be written as:

$$\frac{1}{h}\left[\overline{J_{x1}^{ij}}(x_{i+1}) - \overline{J_{x1}^{ij}}(x_i) + \overline{J_{y1}^{ij}}(y_{j+1}) - \overline{J_{y1}^{ij}}(y_j)\right] + \Sigma_{R1}^{ij}\overline{\overline{\phi_1^{ij}}}$$
$$= \frac{1}{k_{eff}}\left[\nu\Sigma_{f1}^{ij}\,\overline{\overline{\phi_1^{ij}}} + \nu\Sigma_{f2}^{ij}\,\overline{\overline{\phi_g^{ij}}}\right], \tag{2.21}$$

$$\frac{1}{h}\left[\overline{J_{x2}^{ij}}(x_{i+1}) - \overline{J_{x2}^{ij}}(x_i) + \overline{J_{y2}^{ij}}(y_{j+1}) - \overline{J_{y2}^{ij}}(y_j)\right] + \Sigma_{a2}^{ij}\overline{\overline{\phi_2^{ij}}} = \Sigma_{21}^{ij}\overline{\overline{\phi_1^{ij}}}. \tag{2.22}$$

The values for the face-average currents can be found by substituting the flux expansion into Fick's law and then using Eqs.(2.18) and (2.19). The volume average fluxes were given in Eqs.(2.8) and (2.12). Using these values, the neutron balance equations for node ij can be written as:

$$-\frac{6D_1}{h^2}(a_{02}+a_{20}) + \Sigma_{R1}\,a_{00} = \frac{1}{k_{eff}}\left[(\nu\Sigma_{f1} + t_s\nu\Sigma_{f2})\,a_{00} + \nu\Sigma_{f2}\,b_{00}\right], \tag{2.23}$$

and

$$\begin{aligned} &-\frac{2D_2}{h^2}\left[\kappa\sinh\left(\frac{\kappa}{2}\right)(b_{02}+b_{20}) + 4\,\sinh^2\left(\frac{\kappa}{2\sqrt{2}}\right)b_{44}\right] \\ &\quad -\frac{6t_sD_2}{h^2}(a_{02}+a_{20}) + \Sigma_{a2}\,b_{00} = (\Sigma_{21} - t_s\Sigma_{a2})\,a_{00}, \end{aligned} \tag{2.24}$$

where the superscripts ij have have been dropped from each term for simplicity.

We will now define the constant κ so that hyperbolic functions used in the thermal flux expansion functions are particular solutions to the homogeneous part of the thermal diffusion equation. The first step is to substitute Fick's Law into the thermal diffusion equation and make the previous assumptions of no upscatter and no thermal fission spectrum. The thermal diffusion equation can then be written as:

$$-D_2^{ij}\,\nabla^2\phi_2(x,y) + \Sigma_{a2}^{ij}\,\phi_2(x,y) = \Sigma_{21}^{ij}\,\phi_1(x,y), \quad x,y \in V^{i,j}. \tag{2.25}$$

By defining κ^2 as

$$\kappa^2 = \frac{h^2\Sigma_{a2}}{D_2} \tag{2.26}$$

and making the coordinate transformation

$$u = \frac{2x - x_{i+1} - x_i}{2\,h} \tag{2.27}$$

$$v = \frac{2y - y_{j+1} - y_j}{2\,h} \tag{2.28}$$

equation (2.25) can be written as

$$\nabla^2_{u,v}\phi_2(u,v) - \kappa^2\phi_2(u,v) = -\frac{h^2\,\Sigma_{21}}{D_2}\phi_1(u,v), \quad u,v \in V^{ij}. \tag{2.29}$$

From this expression, it can be seen that the hyperbolic functions of the thermal flux expansion are particular solutions to the homogeneous part of the thermal group diffusion equation. This leaves the polynomial functions of the thermal flux expansion to represent the particular solutions of the inhomogeneous thermal diffusion equation.

We can also define the t_s term in the thermal flux expansion as the thermal to fast flux ratio in an infinite, homogeneous medium. The value for t_s can be found by setting the leakage term in Eq.(2.25) to zero and solving for the ratio of the thermal to fast flux. This gives:

$$t_s^{ij} = \frac{\phi_2(x,y)}{\phi_1(x,y)} = \frac{\Sigma_{21}^{ij}}{\Sigma_{a2}^{ij}}, \quad x,y \in V^{ij}. \tag{2.30}$$

This equation states that in an infinite medium, the magnitudes of the thermal and fast flux differ by a constant factor which is independent of position. In the middle of a fuel assembly, the homogeneous flux behaves as if it were an infinite material so the t_s term should account for most of the thermal flux shape. Near the assembly boundaries, the t_s term will become less important and the thermal flux shape will be represented by the hyperbolic expansion functions.

Using the definitions of κ and t_s given in Eqs.(2.26) and (2.30), the thermal neutron balance equation, Eq.(2.24), can be written in the simplified form:

$$\frac{2}{\kappa}\sinh\left(\frac{\kappa}{2}\right)(b_{02}+b_{20}) + \frac{8}{\kappa^2}\sinh^2\left(\frac{\kappa}{2\sqrt{2}}\right) b_{44} + \frac{6t_s}{\kappa^2}(a_{02}+a_{20}) - b_{00} = 0. \tag{2.31}$$

2.4 Weighted Neutron Balance Condition

In the previous section the neutron balance equation was derived so that the net neutron production rate in a node is equal to the net neutron leakage rate. We can obtain further conditions by also requiring that the flux expansion obey the

neutron balance condition in a weighted integral sense. The weighted neutron balance equation will be found using the weighted residual technique.

The weighted residual technique is derived by first multiplying the diffusion equation for node ij, given by Eq.(2.15), by some arbitrary weight function, $w_p(x,y)$, and then integrating over the volume of node ij. This procedure leads to

$$\frac{1}{V^{ij}}\int_{x_i}^{x_{i+1}} dx \int_{y_j}^{y_{j+1}} dy\, w_p(x,y)\left[\nabla\cdot\mathbf{J}_g(x,y)+\Sigma_{tg}^{ij}\phi_g(x,y)\right]$$
$$=\frac{1}{V^{ij}}\int_{x_i}^{x_{i+1}} dx \int_{y_j}^{y_{j+1}} dy \sum_{g'=1}^{2}\left[\frac{1}{k_{eff}}\chi_g^{ij}\nu\Sigma_{fg'}^{ij}+\Sigma_{gg'}^{ij}\right] w_p(x,y)\,\phi_{g'}(x,y),\quad g=1,2. \quad (2.32)$$

This equation can be written more compactly by defining the inner product of two functions over the volume ij as

$$\langle w_p,\phi_g\rangle^{ij}\equiv\frac{1}{V^{ij}}\int_{x_i}^{x_{i+1}} dx \int_{y_j}^{y_{j+1}} dy\, w_p(x,y)\,\phi_g(x,y). \quad (2.33)$$

Using the inner product notation, the general weighted neutron balance equation for node ij can be written as

$$\langle w_p,\nabla\cdot\mathbf{J}_g\rangle^{ij}+\Sigma_{tg}^{ij}\langle w_p,\phi_g\rangle^{ij}=\sum_{g'=1}^{2}\left[\frac{1}{k_{eff}}\chi_g^{ij}\nu\Sigma_{fg'}^{ij}+\Sigma_{gg'}^{ij}\right]\langle w_p,\phi_{g'}\rangle^{ij},\quad g=1,2. \quad (2.34)$$

If $\phi_g(x,y)$ represents the true homogeneous flux solution which obeys the diffusion equation at all points in the problem, then Eq.(2.34) will be valid for any arbitrary choice of the weight function. However, we are representing the flux by a truncated expansion which is usually not general enough to match the diffusion equation at all points. Therefore, the weighted neutron balance equation will only be valid for a limited number of weight functions. It has been found that using low order expansion functions as weight functions will give accurate results [11]. This choice of weight functions is referred to as *moments weighting*. We can obtain 6 weighted nodal balance equations by using Eq.(2.34) with the three weight functions per group. The

weight functions we use are:

$$w_1(x,y) = \frac{2x - x_{i+1} - x_i}{2\,h}, \tag{2.35}$$

$$w_2(x,y) = \frac{2y - y_{j+1} - y_j}{2\,h}, \tag{2.36}$$

$$w_3(x,y) = \frac{2x - x_{i+1} - x_i}{2\,h} \cdot \frac{2y - y_{j+1} - y_j}{2\,h}, \tag{2.37}$$

It should be noted that the neutron balance equation is a special case of the weighted nodal balance equation with the weight function equal to unity.

Substituting the weight functions and the flux expansions from Eqs.(2.4) and (2.9) into the definition of the inner product gives the following *flux moments*:

$$\langle w_1, \phi_1 \rangle = \frac{1}{12} a_{10} + \frac{1}{40} a_{30} \tag{2.38}$$

$$\langle w_2, \phi_1 \rangle = \frac{1}{12} a_{01} + \frac{1}{40} a_{03} \tag{2.39}$$

$$\langle w_3, \phi_1 \rangle = \frac{1}{144} a_{11} + \frac{1}{1600} a_{33} \tag{2.40}$$

$$\langle w_1, \phi_2 \rangle = \frac{1}{12} b_{10} + \frac{\alpha_1}{\kappa} b_{30} + \frac{4}{\kappa^2} \alpha_2 \sinh\left(\frac{\kappa}{2\sqrt{2}}\right) b_{54} + t_s \langle w_1, \phi_1 \rangle \tag{2.41}$$

$$\langle w_2, \phi_2 \rangle = \frac{1}{12} b_{01} + \frac{\alpha_1}{\kappa} b_{03} + \frac{4}{\kappa^2} \alpha_2 \sinh\left(\frac{\kappa}{2\sqrt{2}}\right) b_{45} + t_s \langle w_2, \phi_1 \rangle \tag{2.42}$$

$$\langle w_3, \phi_2 \rangle = \frac{1}{144} b_{11} + \frac{2\,\alpha_2^2}{\kappa^2} b_{55} + t_s \langle w_3, \phi_1 \rangle \tag{2.43}$$

where α_1 and α_2 are defined as

$$\alpha_1 \equiv \cosh\left(\frac{\kappa}{2}\right) - \frac{2}{\kappa} \sinh\left(\frac{\kappa}{2}\right), \text{ and} \tag{2.44}$$

$$\alpha_2 \equiv \cosh\left(\frac{\kappa}{2\sqrt{2}}\right) - \frac{2\sqrt{2}}{\kappa} \sinh\left(\frac{\kappa}{2\sqrt{2}}\right). \tag{2.45}$$

Substituting the flux expansions into Fick's law and using the definition of the inner product yields the following *current-derivative moments*:

$$\langle w_1, \nabla \cdot \mathbf{J}_1 \rangle = -\frac{D_1}{h^2} \left(\frac{1}{2} a_{12} + a_{30}\right) \tag{2.46}$$

$$\langle w_2, \nabla \cdot \mathbf{J}_1 \rangle = -\frac{D_1}{h^2}\left(\frac{1}{2}\,a_{21} + a_{03}\right) \tag{2.47}$$

$$\langle w_3, \nabla \cdot \mathbf{J}_1 \rangle = -\frac{D_1}{20\,h^2}\,a_{33} \tag{2.48}$$

$$\langle w_1, \nabla \cdot \mathbf{J}_2 \rangle = -\frac{D_2}{h^2}\left(\kappa\alpha_1 b_{30} + 4\alpha_2 \sinh\left(\frac{\kappa}{2\sqrt{2}}\right) b_{54}\right) + t_s\,\frac{D_2}{D_1}\,\langle w_1, \nabla \cdot \mathbf{J}_1 \rangle \tag{2.49}$$

$$\langle w_2, \nabla \cdot \mathbf{J}_2 \rangle = -\frac{D_2}{h^2}\left(\kappa\alpha_1 b_{03} + 4\alpha_2 \sinh\left(\frac{\kappa}{2\sqrt{2}}\right) b_{45}\right) + t_s\,\frac{D_2}{D_1}\,\langle w_2, \nabla \cdot \mathbf{J}_1 \rangle \tag{2.50}$$

$$\langle w_3, \nabla \cdot \mathbf{J}_2 \rangle = -\frac{2\,D_2}{h^2}\,\alpha_2^2\, b_{55} + t_s\,\frac{D_2}{D_1}\,\langle w_3, \nabla \cdot \mathbf{J}_1 \rangle \tag{2.51}$$

Substituting the moment equations from Eqs.(2.38) thru (2.51) into the weighted neutron balance equation, Eq.(2.34), and making the assumptions that there is no thermal fission spectrum or upscatter, gives the six weighted neutron balance equations needed. These weighted neutron balance equations are listed below.

1. The fast weighted neutron balance equation using $w_1(x, y)$:

$$\begin{aligned} -\frac{D_1}{h^2}\left(\frac{1}{2}\,a_{12} + a_{30}\right) + \Sigma_{R1}\left(\frac{1}{12}a_{10} + \frac{1}{40}a_{30}\right) \\ = \frac{1}{k_{eff}}(\nu\Sigma_{f1} + t_s\nu\Sigma_{f2})\left(\frac{1}{12}a_{10} + \frac{1}{40}a_{30}\right) \\ + \frac{1}{k_{eff}}\nu\Sigma_{f2}\left(\frac{1}{12}b_{10} + \frac{\alpha_1}{\kappa}\,b_{30} + \frac{4}{\kappa^2}\,\alpha_2\,\sinh\left(\frac{\kappa}{2\sqrt{2}}\right)\,b_{54}\right) \end{aligned} \tag{2.52}$$

2. The fast weighted neutron balance equations using $w_2(x, y)$:

$$\begin{aligned} -\frac{D_1}{h^2}\left(\frac{1}{2}\,a_{21} + a_{03}\right) + \Sigma_{R1}\left(\frac{1}{12}a_{01} + \frac{1}{40}a_{03}\right) \\ = \frac{1}{k_{eff}}(\nu\Sigma_{f1} + t_s\nu\Sigma_{f2})\left(\frac{1}{12}a_{01} + \frac{1}{40}a_{03}\right) \\ + \frac{1}{k_{eff}}\nu\Sigma_{f2}\left(\frac{1}{12}b_{01} + \frac{\alpha_1}{\kappa}\,b_{03} + \frac{4}{\kappa^2}\,\alpha_2\,\sinh\left(\frac{\kappa}{2\sqrt{2}}\right)\,b_{45}\right) \end{aligned} \tag{2.53}$$

3. The fast weighted neutron balance equations using $w_3(x, y)$:

$$\begin{aligned} -\frac{D_1}{20h^2}a_{33} + \Sigma_{R1}\left(\frac{1}{144}\,a_{11} + \frac{1}{1600}a_{33}\right) \\ = \frac{1}{k_{eff}}(\nu\Sigma_{f1} + t_s\nu\Sigma_{f2})\left(\frac{1}{144}\,a_{11} + \frac{1}{1600}\,a_{33}\right) \end{aligned}$$

$$+\frac{1}{k_{eff}}\nu\Sigma_{f2}\left(\frac{1}{144}b_{11}+\frac{2\,\alpha_2^2}{\kappa^2}b_{55}\right) \tag{2.54}$$

4. The thermal weighted neutron balance equations using $w_1(x,y)$:

$$-\frac{D_2}{h^2}\left(\kappa\,\alpha_1 b_{30}+4\,\alpha_2\,\sinh\left(\frac{\kappa}{2\sqrt{2}}\right)b_{54}\right)-\frac{t_s D_2}{h^2}\left(\frac{1}{2}a_{12}+a_{30}\right)$$
$$+\Sigma_{R2}\left(\frac{1}{12}b_{10}+\frac{\alpha_1}{\kappa}b_{30}+\frac{4}{\kappa^2}\alpha_2\,\sinh\left(\frac{\kappa}{2\sqrt{2}}\right)b_{54}\right)=0 \tag{2.55}$$

5. The thermal weighted neutron balance equations using $w_2(x,y)$:

$$-\frac{D_2}{h^2}\left(\kappa\,\alpha_1 b_{03}+4\,\alpha_2\,\sinh\left(\frac{\kappa}{2\sqrt{2}}\right)b_{45}\right)-\frac{t_s D_2}{h^2}\left(\frac{1}{2}a_{21}+a_{03}\right)$$
$$+\Sigma_{R2}\left(\frac{1}{12}b_{01}+\frac{\alpha_1}{\kappa}b_{03}+\frac{4}{\kappa^2}\alpha_2\,\sinh\left(\frac{\kappa}{2\sqrt{2}}\right)b_{45}\right)=0 \tag{2.56}$$

6. The thermal weighted neutron balance equations using $w_3(x,y)$:

$$-\frac{2D_2}{h^2}\alpha_2^2 b_{55}-\frac{t_s D_2}{20\,h^2}a_{33}+\Sigma_{R2}\left(\frac{1}{144}b_{11}+\frac{2\,\alpha_2^2}{\kappa^2}b_{55}\right)=0 \tag{2.57}$$

Upon substituting the definitions of κ and t_s into the thermal weighted neutron balance equations, the coefficients corresponding to the hyperbolic expansion functions will cancel from the equations. Making this substitution gives the following simplified thermal weighted neutron balance equations:

$$4.\quad t_s(a_{12}+2\,a_{30})-\frac{\kappa^2}{6}b_{10}=0 \tag{2.58}$$

$$5.\quad t_s(a_{21}+2\,a_{03})-\frac{\kappa^2}{6}b_{01}=0 \tag{2.59}$$

$$6.\quad \frac{t_s}{5}a_{33}+\frac{\kappa^2}{36}b_{11}=0 \tag{2.60}$$

The reason that the hyperbolic thermal expansion coefficients cancel from the weighted balance equations is that they represent functions which are analytic solutions to the homogeneous part of thermal diffusion equation. Since they are analytic solution, they will cancel out of the thermal weighted nodal balance equations for any

choice of weight function. Therefore, the weighted nodal balance equation can only be used to find the expansion coefficients for the polynomial expansion functions. Since there are only 4 polynomial thermal expansion functions, only 4 thermal weighted neutron balance equations (including the neutron balance equation) can be used. If more than 4 weighted balance equations are used, the results will be equations that are not linearly independent and/or the only solution to the equations will be zero for the polynomial coefficients. However, since the fast flux expansion is made up entirely of polynomial expansion functions, up to twelve fast weighted neutron balance equations can be used.

2.5 Flux and Current Continuity Conditions

The next set of conditions that we will use is to require that the face-average flux and face-average current be continuous across the surfaces of adjacent nodes for each energy group. To derive the continuity equations, the face-average fluxes are defined as

$$\overline{\phi_{xg}^{ij}}(x) \equiv \frac{1}{h}\int_{y_j}^{y_{j+1}} \phi_g(x,y)\,dy, \text{ and} \tag{2.61}$$

$$\overline{\phi_{yg}^{ij}}(y) \equiv \frac{1}{h}\int_{x_i}^{x_{i+1}} \phi_g(x,y)\,dx, \quad g=1,2. \tag{2.62}$$

and the face-average currents have already been defined in Eqs.(2.18) and (2.19).

Referring to Figure 2-2, we can denote the $x+$ and $y+$ surfaces of a node as the surfaces on the right hand and upper sides of a node. Likewise, the surfaces on the left hand and lower sides of a node will be referred to as the $x-$ and $y-$ surfaces of a node. Using this notation, the flux and current continuity equations for the left and lower surfaces of node ij can be written as

$$\overline{\phi_{xg}^{i-1,j}}(x+) = \overline{\phi_{xg}^{ij}}(x-), \tag{2.63}$$

$$\overline{\phi_{yg}^{i,j-1}}(y+) = \overline{\phi_{yg}^{ij}}(y-), \tag{2.64}$$

$$\overline{J_{xg}^{i-1,j}}(x+) = \overline{J_{xg}^{ij}}(x-), \tag{2.65}$$

$$\overline{J_{yg}^{i,j-1}}(y+) \;=\; \overline{J_{yg}^{ij}}(y-), \quad g=1,2. \tag{2.66}$$

The flux and current will also be required to be continuous across the right hand and upper surfaces of node ij, but these conditions will "belong" to the nodes $i+1, j$ and $i, j+1$, respectively. Therefore, for 2 energy groups, there are 8 continuity equations per node.

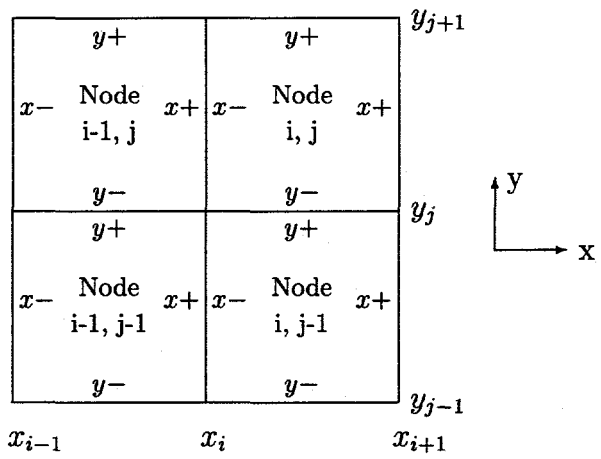

Figure 2-2: Diagram showing node surface notation.

By substituting the flux expansions from Eqs.(2.4) and (2.9) into Eqs.(2.61), (2.62), (2.18), and (2.19), and using Fick's Law to represent the current, the following face-average flux and current equations are obtained.

1. Fast face-average flux:

$$\overline{\phi_{x1}^{ij}}(x\pm) \;=\; a_{00} \pm \frac{1}{2}a_{10} + \frac{1}{2}a_{20} \pm \frac{1}{4}a_{30} \tag{2.67}$$

$$\overline{\phi_{y1}^{ij}}(y\pm) \;=\; a_{00} \pm \frac{1}{2}a_{01} + \frac{1}{2}a_{02} \pm \frac{1}{4}a_{03} \tag{2.68}$$

2. Thermal face-average flux:

$$\begin{aligned}\overline{\phi_{x2}^{ij}}(x\pm) \;=\; & b_{00} \pm \frac{1}{2}\,b_{10} + \alpha_1\, b_{20} \pm \sinh\left(\frac{\kappa}{2}\right) b_{30} + \frac{2\sqrt{2}}{\kappa}\,\alpha_2 \sinh\left(\frac{\kappa}{2\sqrt{2}}\right) b_{44} \\ & \pm \frac{2\sqrt{2}}{\kappa} \sinh^2\left(\frac{\kappa}{2\sqrt{2}}\right) b_{54} + t_s\, \overline{\phi_{x1}^{ij}}(x\pm)\end{aligned} \tag{2.69}$$

$$\overline{\phi_{y2}^{ij}}(y\pm) = b_{00} \pm \frac{1}{2} b_{01} + \alpha_1 b_{02} \pm \sinh\left(\frac{\kappa}{2}\right) b_{03} + \frac{2\sqrt{2}}{\kappa} \alpha_2 \sinh\left(\frac{\kappa}{2\sqrt{2}}\right) b_{44}$$
$$\pm \frac{2\sqrt{2}}{\kappa} \sinh^2\left(\frac{\kappa}{2\sqrt{2}}\right) b_{45} + t_s \overline{\phi_{y1}^{ij}}(y\pm) \quad (2.70)$$

3. Fast face-average current:

$$\overline{J_{x1}^{ij}}(x\pm) = -\frac{D_1}{h}\left(a_{10} \pm 3\, a_{20} + \frac{3}{2} a_{30}\right) \quad (2.71)$$

$$\overline{J_{y1}^{ij}}(y\pm) = -\frac{D_1}{h}\left(a_{01} \pm 3\, a_{02} + \frac{3}{2} a_{03}\right) \quad (2.72)$$

4. Thermal face-average current:

$$\overline{J_{x2}^{ij}}(x\pm) = -\frac{D_2}{h}\left[b_{10} \pm \kappa \sinh\left(\frac{\kappa}{2}\right) b_{20} + \kappa \cosh\left(\frac{\kappa}{2}\right) b_{30}\right.$$
$$\left. \pm 2 \sinh^2\left(\frac{\kappa}{2\sqrt{2}}\right) b_{44} + 2 \sinh\left(\frac{\kappa}{2\sqrt{2}}\right) \cosh\left(\frac{\kappa}{2\sqrt{2}}\right) b_{54}\right]$$
$$+ \frac{D_2}{D_1} t_s \overline{J_{x1}^{ij}}(x\pm) \quad (2.73)$$
$$\overline{J_{y2}^{ij}}(y\pm) = -\frac{D_2}{h}\left[b_{01} \pm \kappa \sinh\left(\frac{\kappa}{2}\right) b_{02} + \kappa \cosh\left(\frac{\kappa}{2}\right) b_{03}\right.$$
$$\left. \pm 2 \sinh^2\left(\frac{\kappa}{2\sqrt{2}}\right) b_{44} + 2 \sinh\left(\frac{\kappa}{2\sqrt{2}}\right) \cosh\left(\frac{\kappa}{2\sqrt{2}}\right) b_{45}\right]$$
$$+ \frac{D_2}{D_1} t_s \overline{J_{y1}^{ij}}(y\pm) \quad (2.74)$$

2.5.1 Discontinuity Factors

So far it has been assumed that each node is made up of a homogeneous material with spatially constant cross sections. In actual reactor problems, nodes are composed of heterogeneous materials with cross sections that vary over the spatial domain. This node heterogeneity is taken into account by using cross sections that have been *homogenized* by some higher order calculation [3]. In most modern nodal method calculations, the cross section homogenization is usually performed by modeling each type of assembly in a fully detailed transport calculation using zero-current boundary conditions (an assembly calculation). The homogenized cross sections are then found

by taking the flux-weighted average over the volume of the assembly. This technique insures that the reaction rates obtained from the heterogeneous calculation match the reaction rates from a homogeneous calculation. However, it can be shown that it is not possible to define a homogeneous diffusion constant for which reaction rates *and* current continuity are preserved for each node [3]. In order to overcome this problem, Koebke developed a method called Equivalence Theory [12] in which two extra degrees of freedom per direction are introduced into the problem by allowing the diffusion coefficient to depend on direction and by allowing the homogeneous flux to be *discontinuous* the same amount at the surfaces perpendicular to a given direction [12]. Smith generalized this concept into Generalized Equivalence Theory in which the diffusion constant is arbitrarily chosen and the homogeneous flux is allowed to be discontinuous by different amounts at each surface [13].

Using Generalized Equivalence Theory, the homogeneous flux discontinuity at each surface is taken into account by using *discontinuity factors*, denoted by f, at each nodal surface. For node ij, the surface discontinuity factors (SDF's) are defined as:

$$f_{gu-}^{i,j} = \frac{\widetilde{\phi}_{ug}^{ij}(u-)}{\overline{\phi_{ug}^{ij}}(u-)}, \tag{2.75}$$

$$f_{gu+}^{i,j} = \frac{\widetilde{\phi}_{ug}^{ij}(u+)}{\overline{\phi_{ug}^{ij}}(u+)}, \quad u = x, y \tag{2.76}$$

where $\widetilde{\phi}_{ug}^{ij}(u)$ is the physical (heterogeneous) value of the group g surface flux at u.

Using the definitions of discontinuity factors and noting that it is the heterogeneous flux that is continuous across nodal surfaces, the flux continuity equations given in Eqs.(2.63) and (2.64) can be rewritten as:

$$\overline{\phi_{xg}^{i-1,j}}(x+)\, f_{gx+}^{i-1,j} = f_{gx-}^{i,j}\, \overline{\phi_{xg}^{ij}}(x-), \tag{2.77}$$

$$\overline{\phi_{yg}^{i,j-1}}(y+)\, f_{gy+}^{i,j-1} = f_{gy-}^{i,j}\, \overline{\phi_{yg}^{ij}}(y-), \tag{2.78}$$

The values of the discontinuity factors are obtained from the assembly calculations

that were used to provide the homogenized cross sections [3].

2.5.2 Boundary Conditions

Zero-flux and zero-current boundary conditions can be taken into account for nodes on the outer edges of the problem by setting the face-average fluxes or face-average currents equal to zero.

2.6 Corner Flux Continuity Conditions

In the previous section, the face-average flux and face-average current were required to be continuous across the surfaces of any two adjacent nodes. This requirement stems from the traditional nodal scheme in which the inter-nodal flux shape is reduced to a one-dimensional problem and only the surface average values are known. In order to find the value of the flux at a corner point while using the transverse integrated method, a separate auxiliary calculation must be made [14]. Since the nodal expansion method in this paper does not use the transverse integrated approximation, the flux at the corner points can be found directly from the solution method and no separate calculation is needed. Since the corner point flux is available during the solution method, we can require the flux to be continuous at the corner points as another condition. Requiring the flux to be continuous at the corner points will impose a tighter coupling between adjacent nodes than just requiring that the face-average flux and current be continuous.

Referring to the corner point (x_i, y_j) in Figure 2-2, 4 equations can be found by requiring that the flux be continuous at the corner point. These equations are:

$$\phi_g^{i,j}(x_i, y_j) = \phi_g^{i-1,j}(x_i, y_j) \tag{2.79}$$

$$\phi_g^{i-1,j}(x_i, y_j) = \phi_g^{i-1,j-1}(x_i, y_j) \tag{2.80}$$

$$\phi_g^{i-1,j-1}(x_i, y_j) = \phi_g^{i,j-1}(x_i, y_j) \tag{2.81}$$

$$\phi_g^{i,j-1}(x_i, y_j) = \phi_g^{i,j}(x_i, y_j) \tag{2.82}$$

However, by inspection, it can be seen that only three of the four equations are linearly independent. Therefore, the corner continuity requirement gives only 3 linearly independent equations per corner.

Solving the flux expansion for each of the four corners in node ij yields the following equations for the corner point flux values.

1. Fast corner point flux values for node ij:

$$\begin{aligned}
\phi_1^{ij}(x_i, y_j) &= a_{00} - \frac{1}{2}a_{01} + \frac{1}{2}a_{02} - \frac{1}{2}a_{10} + \frac{1}{4}a_{11} - \frac{1}{4}a_{12} \\
&\quad + \frac{1}{2}a_{20} - \frac{1}{4}a_{21} + \frac{1}{4}a_{22} - \frac{1}{4}a_{03} - \frac{1}{4}a_{30} + \frac{1}{16}a_{33} \quad (2.83) \\
\phi_1^{ij}(x_i, y_{j+1}) &= a_{00} + \frac{1}{2}a_{01} + \frac{1}{2}a_{02} - \frac{1}{2}a_{10} - \frac{1}{4}a_{11} - \frac{1}{4}a_{12} \\
&\quad + \frac{1}{2}a_{20} + \frac{1}{4}a_{21} + \frac{1}{4}a_{22} + \frac{1}{4}a_{03} - \frac{1}{4}a_{30} - \frac{1}{16}a_{33} \quad (2.84) \\
\phi_1^{ij}(x_{i+1}, y_j) &= a_{00} - \frac{1}{2}a_{01} + \frac{1}{2}a_{02} + \frac{1}{2}a_{10} - \frac{1}{4}a_{11} + \frac{1}{4}a_{12} \\
&\quad + \frac{1}{2}a_{20} - \frac{1}{4}a_{21} + \frac{1}{4}a_{22} - \frac{1}{4}a_{03} + \frac{1}{4}a_{30} - \frac{1}{16}a_{33} \quad (2.85) \\
\phi_1^{ij}(x_{i+1}, y_{j+1}) &= a_{00} + \frac{1}{2}a_{01} + \frac{1}{2}a_{02} + \frac{1}{2}a_{10} + \frac{1}{4}a_{11} + \frac{1}{4}a_{12} \\
&\quad + \frac{1}{2}a_{20} + \frac{1}{4}a_{21} + \frac{1}{4}a_{22} + \frac{1}{4}a_{03} + \frac{1}{4}a_{30} + \frac{1}{16}a_{33} \quad (2.86)
\end{aligned}$$

2. Thermal corner point flux values for node ij :

$$\begin{aligned}
\phi_2^{ij}(x_i, y_j) &= b_{00} - \frac{1}{2}b_{01} - \frac{1}{2}b_{10} + \frac{1}{4}b_{11} \\
&\quad + \alpha_1(b_{02} + b_{20}) - \sinh\left(\frac{\kappa}{2}\right)(b_{03} + b_{30}) \\
&\quad + \beta\, b_{44} - \cosh\left(\frac{\kappa}{2\sqrt{2}}\right)\sinh\left(\frac{\kappa}{2\sqrt{2}}\right)(b_{45} + b_{54}) \\
&\quad + \sinh^2\left(\frac{\kappa}{2\sqrt{2}}\right) b_{55} + t_s\phi_1^{ij}(x_i, y_j) \quad (2.87) \\
\phi_2^{ij}(x_i, y_{j+1}) &= b_{00} + \frac{1}{2}b_{01} - \frac{1}{2}b_{10} - \frac{1}{4}b_{11} \\
&\quad + \alpha_1(b_{02} + b_{20}) + \sinh\left(\frac{\kappa}{2}\right)(b_{03} - b_{30}) \\
&\quad + \beta\, b_{44} + \cosh\left(\frac{\kappa}{2\sqrt{2}}\right)\sinh\left(\frac{\kappa}{2\sqrt{2}}\right)(b_{45} - b_{54})
\end{aligned}$$

$$- \sinh^2\left(\frac{\kappa}{2\sqrt{2}}\right) b_{55} + t_s \phi_1^{ij}(x_i, y_{j+1}) \tag{2.88}$$

$$\begin{aligned}
\phi_2^{ij}(x_{i+1}, y_j) &= b_{00} - \frac{1}{2} b_{01} + \frac{1}{2} b_{10} - \frac{1}{4} b_{11} \\
&\quad + \alpha_1 (b_{02} + b_{20}) - \sinh\left(\frac{\kappa}{2}\right)(b_{03} - b_{30}) \\
&\quad + \beta\, b_{44} - \cosh\left(\frac{\kappa}{2\sqrt{2}}\right) \sinh\left(\frac{\kappa}{2\sqrt{2}}\right)(b_{45} - b_{54}) \\
&\quad - \sinh^2\left(\frac{\kappa}{2\sqrt{2}}\right) b_{55} + t_s \phi_1^{ij}(x_{i+1}, y_j)
\end{aligned} \tag{2.89}$$

$$\begin{aligned}
\phi_2^{ij}(x_{i+1}, y_{j+1}) &= b_{00} + \frac{1}{2} b_{01} + \frac{1}{2} b_{10} + \frac{1}{4} b_{11} \\
&\quad + \alpha_1 (b_{02} + b_{20}) + \sinh\left(\frac{\kappa}{2}\right)(b_{03} + b_{30}) \\
&\quad + \beta\, b_{44} + \cosh\left(\frac{\kappa}{2\sqrt{2}}\right) \sinh\left(\frac{\kappa}{2\sqrt{2}}\right)(b_{45} + b_{54}) \\
&\quad + \sinh^2\left(\frac{\kappa}{2\sqrt{2}}\right) b_{55} + t_s \phi_1^{ij}(x_{i+1}, y_{j+1})
\end{aligned} \tag{2.90}$$

where β is defined as

$$\beta^{ij} = \cosh^2\left(\frac{\kappa^{ij}}{2\sqrt{2}}\right) - \frac{8}{(\kappa^{ij})^2} \sinh^2\left(\frac{\kappa^{ij}}{2\sqrt{2}}\right) \tag{2.91}$$

In addition, if discontinuity factors are used to correct for node heterogeneity as discussed in section 2.5.1, then the homogeneous flux will be discontinuous at the corner points between nodes. In order to take this discontinuity into account, *corner point discontinuity factors* (CPDF's) will be defined for each corner of a node as the ratio of the physical (heterogeneous) flux to the homogeneous flux. For the lower left corner of node ij, the group g CPDF is defined as:

$$f_{g--}^{i,j} = \frac{\widetilde{\phi}_g(x_i, y_j)}{\phi_g^{i,j}(x_i, y_j)} \tag{2.92}$$

where $\widetilde{\phi}_g(x_i, y_j)$ is the heterogeneous group g flux in the lower left corner of node ij, and $\phi_g^{i,j}(x_i, y_j)$ is the homogeneous group g flux in the lower left corner of node ij. The CPDF for the other corners of node ij are defined in an analogous manner. Using the definitions of the CPDF's, the corner point flux continuity equations given

in Eqs.(2.79) thru (2.82), can be rewritten as:

$$\phi_g^{i,j}(x_i, y_j)\, f_{g--}^{i,j} = f_{g+-}^{i-1,j}\, \phi_g^{i-1,j}(x_i, y_j) \tag{2.93}$$

$$\phi_g^{i-1,j}(x_i, y_j)\, f_{g+-}^{i-1,j} = f_{g++}^{i-1,j-1}\, \phi_g^{i-1,j-1}(x_i, y_j) \tag{2.94}$$

$$\phi_g^{i-1,j-1}(x_i, y_j)\, f_{g++}^{i-1,j-1} = f_{g-+}^{i,j-1}\, \phi_g^{i,j-1}(x_i, y_j) \tag{2.95}$$

$$\phi_g^{i,j-1}(x_i, y_j)\, f_{g-+}^{i,j-1} = f_{g--}^{i,j}\, \phi_g^{i,j}(x_i, y_j) \tag{2.96}$$

2.7 Zero-Source Condition at Corner Points

The final condition that we will impose on the flux expansion is referred to as the zero-source condition at the corner points [15, 16]. This condition is derived by drawing a square box of width $2\,dh$ around a corner point as is shown in Figure 2-3. If the dimensions of the box are taken to zero in a limiting case, there will be no volume, and hence no absorptions or fissions inside the box. Therefore, using a neutron balance condition in the box, there will be no net leakage from the box.

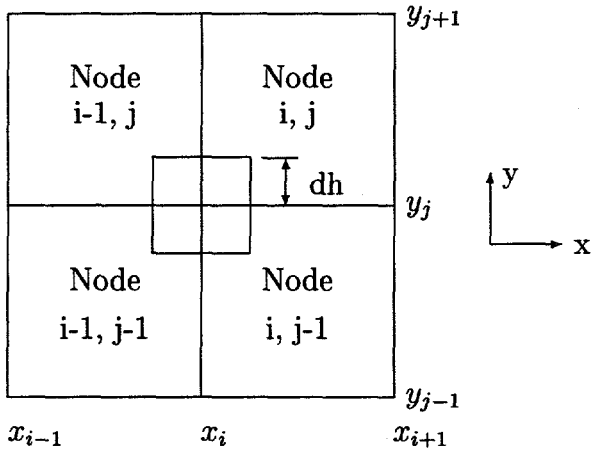

Figure 2-3: Box around a corner used for the zero-source condition.

If $L_g^{i,j}(x_i, y_j)$ is used to represent the group g leakage from the corner (x_i, y_j) into node ij, then around the corner (x_i, y_j), we have the zero-source condition

$$L_g^{i-1,j}(x_i, y_j) + L_g^{i,j}(x_i, y_j) + L_g^{i-1,j-1}(x_i, y_j) + L_g^{i,j-1}(x_i, y_j) = 0, \quad g = 1, 2. \tag{2.97}$$

The leakage term into node ij is defined as

$$L_g^{i,j}(x_i, y_j) = \frac{1}{dh}\int_{x_i}^{x_i+dh} dx\, J_y(x, y_j + dh) + \frac{1}{dh}\int_{y_j}^{y_j+dh} dy\, J_x(x_i + dh, y), \qquad (2.98)$$

and the leakage terms for the other nodes can be defined in an analogous manner. The values for the leakage terms out of the four corners of node ij are listed below.

1. Fast corner leakage equations:

$$\begin{aligned} L_1^{ij}(x_i, y_j) &= \frac{D_1}{h}\Big[-a_{01} + 3\,a_{02} - \frac{3}{2}a_{03} - a_{10} + a_{11} - 2\,a_{12} \\ &\quad + 3\,a_{20} - 2\,a_{21} + 3\,a_{22} - \frac{3}{2}a_{30} + \frac{3}{4}a_{33}\Big] \qquad (2.99) \\ L_1^{ij}(x_i, y_{j+1}) &= \frac{D_1}{h}\Big[a_{01} + 3\,a_{02} + \frac{3}{2}a_{03} - a_{10} - a_{11} - 2\,a_{12} \\ &\quad + 3\,a_{20} + 2\,a_{21} + 3\,a_{22} - \frac{3}{2}a_{30} - \frac{3}{4}a_{33}\Big] \qquad (2.100) \\ L_1^{ij}(x_{i+1}, y_j) &= \frac{D_1}{h}\Big[-a_{01} + 3\,a_{02} - \frac{3}{2}a_{03} + a_{10} - a_{11} + 2\,a_{12} \\ &\quad + 3\,a_{20} - 2\,a_{21} + 3\,a_{22} + \frac{3}{2}a_{30} - \frac{3}{4}a_{33}\Big] \qquad (2.101) \\ L_1^{ij}(x_{i+1}, y_{j+1}) &= \frac{D_1}{h}\Big[a_{01} + 3\,a_{02} + \frac{3}{2}a_{03} + a_{10} + a_{11} + 2\,a_{12} \\ &\quad + 3\,a_{20} + 2\,a_{21} + 3\,a_{22} + \frac{3}{2}a_{30} + \frac{3}{4}a_{33}\Big] \qquad (2.102) \end{aligned}$$

2. Thermal leakage equations:

$$\begin{aligned} L_2^{ij}(x_i, y_j) &= \frac{D_2}{h}\Big[-b_{01} - b_{10} + b_{11} + \kappa\sinh\left(\frac{\kappa}{2}\right)(b_{02} + b_{20}) \\ &\quad + \kappa\cosh\left(\frac{\kappa}{2}\right)(-b_{03} - b_{30}) \\ &\quad + \sqrt{2}\,\kappa\,\cosh\left(\frac{\kappa}{2\sqrt{2}}\right)\sinh\left(\frac{\kappa}{2\sqrt{2}}\right)(b_{44} + b_{55}) \qquad (2.103) \\ &\quad + \sqrt{2}\,\kappa\,\sinh^2\left(\frac{\kappa}{2\sqrt{2}}\right)(-b_{45} + b_{54})\Big] + t_s\,\frac{D_2}{D_1}L_1^{ij}(x_i, y_j) \end{aligned}$$

$$L_2^{ij}(x_i, y_{j+1}) = \frac{D_2}{h}\Big[b_{01} - b_{10} - b_{11} + \kappa\sinh\left(\frac{\kappa}{2}\right)(b_{02} + b_{20})$$

$$
\begin{aligned}
&+\kappa\cosh\left(\frac{\kappa}{2}\right)(b_{03}-b_{30}) \\
&+\sqrt{2}\,\kappa\,\cosh\left(\frac{\kappa}{2\sqrt{2}}\right)\sinh\left(\frac{\kappa}{2\sqrt{2}}\right)(b_{44}-b_{55}) \\
&+\sqrt{2}\,\kappa\,\sinh^2\left(\frac{\kappa}{2\sqrt{2}}\right)(b_{45}-b_{54})\Bigg]+t_s\,\frac{D_2}{D_1}L_1^{ij}(x_i,y_{j+1})
\end{aligned}
\tag{2.104}
$$

$$
\begin{aligned}
L_2^{ij}(x_{i+1},y_j) \;=\; &\frac{D_2}{h}\Big[-b_{01}+b_{10}-b_{11}+\kappa\sinh\left(\frac{\kappa}{2}\right)(b_{02}+b_{20}) \\
&+\kappa\cosh\left(\frac{\kappa}{2}\right)(-b_{03}+b_{30}) \\
&+\sqrt{2}\,\kappa\,\cosh\left(\frac{\kappa}{2\sqrt{2}}\right)\sinh\left(\frac{\kappa}{2\sqrt{2}}\right)(b_{44}-b_{55}) \\
&+\sqrt{2}\,\kappa\,\sinh^2\left(\frac{\kappa}{2\sqrt{2}}\right)(-b_{45}+b_{54})\Bigg]+t_s\,\frac{D_2}{D_1}L_1^{ij}(x_{i+1},y_j)
\end{aligned}
\tag{2.105}
$$

$$
\begin{aligned}
L_2^{ij}(x_{i+1},y_{j+1}) \;=\; &\frac{D_2}{h}\Big[b_{01}+b_{10}+b_{11}+\kappa\sinh\left(\frac{\kappa}{2}\right)(b_{02}+b_{20}) \\
&+\kappa\cosh\left(\frac{\kappa}{2}\right)(b_{03}+b_{30}) \\
&+\sqrt{2}\,\kappa\,\cosh\left(\frac{\kappa}{2\sqrt{2}}\right)\sinh\left(\frac{\kappa}{2\sqrt{2}}\right)(b_{44}+b_{55}) \\
&+\sqrt{2}\,\kappa\,\sinh^2\left(\frac{\kappa}{2\sqrt{2}}\right)(b_{45}+b_{54})\Bigg]+t_s\,\frac{D_2}{D_1}L_1^{ij}(x_{i+1},y_{j+1})
\end{aligned}
\tag{2.106}
$$

2.8 Summary

In this chapter the nodal equations for a two-group, semi-analytic, two-dimensional nodal expansion have been developed. In Chapter 3, a method will be developed to solve these equations for an actual reactor problem.

Chapter 3

Direct Solution of Nodal Equations

3.1 Overview

In this chapter a procedure based on the well-known source-iteration method will be described to solve the two-dimensional, semi-analytic nodal method (2D-SANM) equations. This method will then be applied to a 4-assembly colorset problem containing MOX and UO_2 fuel and the will be compared to results obtained using SIMULATE–3 [8].

3.2 Source-Iteration Method

The procedure used to solve the 2D-SANM equations directly is based on the well-known source-iteration method. In this method, the two-group flux distribution for each node in the problem is represented by a non-separable, two-dimensional flux expansion as given in Chapter 2. Using this flux expansion, there are a total of 24 unknown flux expansion coefficients for each node in the problem, and therefore, in order to find a unique solution for the flux distribution, a total of 24 conditions per node must be applied to the problem. The conditions used to find the flux distribution have been given in Chapter 2. These conditions and the number of nodal equations

per group that each condition represents is:

1. Neutron balance (1 equation per node),
2. Weighted neutron balance (3 equations per node),
3. Flux and current continuity over node surfaces (2 equations per inner surface),
4. Zero-flux or zero-current boundary conditions (1 equation per outer surface),
5. Flux continuity at corner points (3 equations at each corner where 4 nodes meet, 1 equation at each corner where 2 nodes meet), and
6. Zero-source condition at corner points (1 equation per corner).

These nodal equations can be written in a matrix form where all of the fission terms are taken to the right-hand-side of the equation. In matrix form the equations can be written as:

$$\mathbf{M}\,\boldsymbol{\Phi} = \frac{1}{\lambda}\,\mathbf{F}\,\boldsymbol{\Phi}, \tag{3.1}$$

where

$\mathbf{M} =$ 24N x 24N matrix of equation coefficients,

$\boldsymbol{\Phi} =$ 24N element column vector of flux expansion coefficients,

$\mathbf{F} =$ 24N x 24N matrix of fission terms,

$\lambda =$ the reactor eigenvalue, and

$N =$ the number of nodes in the problem.

Equation (3.1) can then be solved by using an iterative method. In order to demonstrate this method, we can set both sides of Eq.(3.1) equal to some arbitrary vector, $\mathbf{S}$, and split Eq.(3.1) into two separate equations. The vector $\mathbf{S}$ will be referred to as the fission source vector. Performing this step, Eq.(3.1) can be written as:

$$\mathbf{S}^{(k)} = \frac{1}{\lambda^{(k)}}\,\mathbf{F}\,\boldsymbol{\Phi}^{(k)}, \quad \text{and} \tag{3.2}$$

$$\mathbf{M}\,\boldsymbol{\Phi}^{(k+1)} = \mathbf{S}^{(k)}. \tag{3.3}$$

where the superscript k has been added for the iteration index.

The first step in the solution method is to calculate the fission source vector using Eq.(3.2). Initially, for $k = 0$, the flux shape and eigenvalue are taken to be unity. For each subsequent iteration, the eigenvalue and fission source vector are calculated using the latest estimate of the flux shape. One method used to calculate the eigenvalue, referred to as the power method [11], gives the eigenvalue at iteration k by the ratio:

$$\lambda^{(k)} = \lambda^{(k-1)} \frac{\langle w, \phi^{(k)} \rangle}{\langle w, \phi^{(k-1)} \rangle}, \tag{3.4}$$

where w is some arbitrary weighting function, ϕ is the neutron flux as a function of space and energy, and the angled brackets refer to the inner product over the reactor volume and all neutron energies. The choice of the weighting function is arbitrary but it is usually taken to be unity or the function $\nu\Sigma_{fg}(x, y)$. For a small class of problems with zero-current boundary conditions, the eigenvalue can be calculated directly from the neutron balance equation. Setting the leakage term to zero in the balance equation and rearranging to solve for the eigenvalue gives:

$$\lambda^{(k)} = \frac{\langle \nu\Sigma_f, \phi^{(k)} \rangle}{\langle \Sigma_a, \phi^{(k)} \rangle}. \tag{3.5}$$

Once the eigenvalue has been calculated for iteration k, the source vector is calculated using Eq.(3.2) and the latest estimate for the eigenvalue and flux vector.

The second step in the solution method is to calculate a new estimate of the flux shape using Eq.(3.3). This step brings up the major limitation to this solution method. Since the matrix $\mathbf{M}$ is not diagonally dominant, Eq.(3.3) cannot be solved using an iterative method such as the Gauss-Seidel Method. Hence, Eq.(3.3) must be solved using a direct matrix solution method such as Gaussian Elimination or LU Decomposition [17]. The disadvantage of using a direct solution method is that roundoff errors can be introduced into the solution if large matrices are solved. Due to this limitation, the direct solution method described in this section may be limited

to problems where the order of **M** is on the order of a few hundred [18]. The exact size of a problem that can be solved using this method still needs to be investigated further. This method has been applied to problems where the size of **M** is 384x384 and no roundoff errors have been observed. For larger problems where roundoff errors could be significant, a non-linear method must be used such as the one that will be outlined in Section 4.2.1. One advantage, however, to using a direct matrix solution method such as Gaussian Elimination or LU decomposition is that the inverse/decomposition of **M** only needs to be performed for the first iteration. Once the inverse/decomposition of **M** has been calculated, Eq.(3.2) reduces to a simple matrix multiplication.

After a new estimate for the flux shape has been calculated, the process is repeated until the eigenvalue and flux shape converge to within some set tolerance.

Several methods have been developed to accelerate the convergence of the source-iteration method [11]. However, since this method has only been applied to small problems, computer time has not been a factor and no acceleration methods have been used.

3.3 Problem Description

The source-iteration method described in the previous section has been applied to the 4-assembly colorset problem shown in Figure 3-1. The problem consists of 4 assemblies arranged in a square matrix with zero current boundary conditions on the outer edges of the problem.

The colorset problem has been solved using two energy groups and two material compositions arranged in three different loading patterns. The two material compositions used are a typical uranium oxide fuel assembly (UO_2) and a highly enriched mixed oxide fuel assembly (MOX) [19]. This combination of UO_2 and MOX fuel is used in order to demonstrate the advantages of the 2D-SANM in problems with very large thermal flux gradients. The cross sections used in this problem are given in Table 3.1. The nodes are assumed to be homogeneous, and therefore, no discontinuity

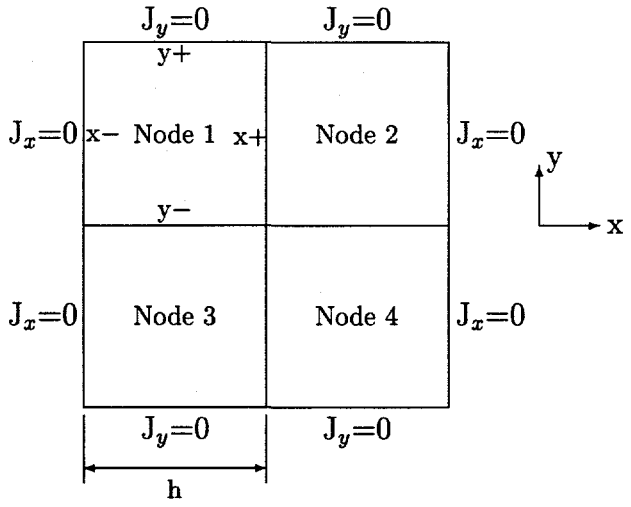

Figure 3-1: 4-assembly colorset problem with zero-current boundary conditions.

factors are used. The widths of each assembly are 10.71 cm and correspond to the half-width of an actual fuel assembly.

Table 3.1: Cross section set used in 4-assembly colorset.

Material	Group	D	Σ_a	$\nu\Sigma_f$	$\Sigma_{g'g}$	χ_g
MOX	1	1.422	0.01639	0.01423	0.0114	1.0
	2	0.277	0.32700	0.54700	0.0	0.0
UO_2	1	1.394	0.00985	0.00741	0.0165	1.0
	2	0.341	0.09100	0.14700	0.0	0.0

In order to solve this colorset problem, a computer code, called COLOR2G, has been written which solves the 2D-SANM equations for a 2x2 or a 4x4 colorset problem using the direct solution method described in Section 3.2. The program has been written entirely in FORTRAN 77 and a listing of the code used to solve a 4x4 colorset problem is included in Appendix A.

3.4 Results

The results for the 4-assembly colorset problem are shown in this section. Three different loading patterns are looked at—a checkerboard pattern, a one-dimensional

pattern, and a 3-UO_2/1-MOX loading pattern. Each loading pattern has been run using both 1 node per assembly (1 npa) and 4 nodes per assembly (4 npa). The results from COLOR2G are compared to results obtained using the SIMULATE–3 code with the semi-analytical nodal method (SANM) option [9]. A 16 npa SIMULATE problem has also been run to serve as a benchmark for the one-dimensional loading pattern, and a 64 npa SIMULATE problem has been run to serve as a benchmark for the checkerboard and 3-UO_2/1-MOX loading patterns. The number of nodes used in the benchmark problems was determined by running cases with decreasing node size until the solutions converged.

The differences found between COLOR2G and SIMULATE are due to the different flux expansion methods used by the two codes. For the fast flux representation, COLOR2G uses a non-separable, two-dimensional expansion of third-order polynomials and SIMULATE uses a one-dimensional expansion of fourth-order polynomials and the transverse leakage approximation. For the thermal flux representation in COLOR2G, a non-separable, two-dimensional expansion of linear polynomials and hyperbolic terms, along with a fast to thermal flux ratio term is used. For the thermal flux representation in SIMULATE with the SANM option, a one-dimensional expansion of fourth-order polynomials and hyperbolic terms is used along with the transverse integration approximation.

Another difference between COLOR2G and SIMULATE is in the method used to find the corner point flux values. The corner point fluxes in COLOR2G are directly edited from the solution. The corner point fluxes from SIMULATE are the result of a flux reconstruction calculation where the corner point fluxes are interpolated from the one-dimensional flux shapes [8, 14].

3.4.1 Results for One-Dimensional Loading Pattern

The first loading pattern described is the one-dimensional loading pattern shown in Figure 3-2. The flux values obtained for the one-dimensional loading are shown in Tables 3.2 and 3.3. The k-effective values obtained are shown in Table 3.4.

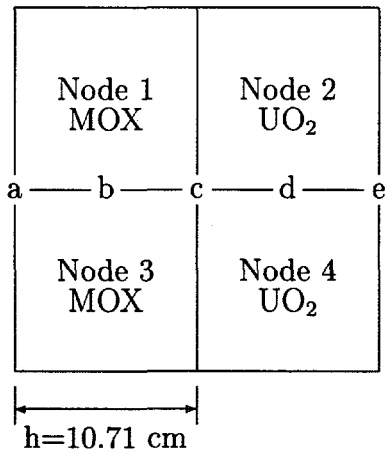

Figure 3-2: Flux positions for one-dimensional loading pattern.

Table 3.2: Fast flux values for one-dimensional loading pattern.

Position[1]	SIMULATE Results (16 npa)	Error from 16 npa SIMULATE case (%)			
		SIMULATE (4 npa)	COLOR2G (4 npa)	SIMULATE (1 npa)	COLOR2G (1 npa)
a	2.2874	0.00	-0.03	0.00	-0.20
b	2.3174	0.00	0.07	0.00	0.59
c	2.3531	0.00	-0.04	0.00	-0.19
d	2.3598	0.00	-0.07	0.00	-0.57
e	2.3791	0.00	-0.04	0.00	-0.17

[1] refer to Figure 3-2

Table 3.3: Thermal flux values for one-dimensional loading pattern.

Position[1]	SIMULATE Results (16 npa)	Error from 16 npa SIMULATE case (%)			
		SIMULATE (4 npa)	COLOR2G (4 npa)	SIMULATE (1 npa)	COLOR2G (1 npa)
a	0.079797	0.00	-0.03	-0.07	-0.07
b	0.091686	0.00	0.05	-0.01	0.45
c	0.20888	0.00	-0.05	0.00	-0.31
d	0.38873	0.00	-0.07	0.01	-0.57
e	0.42883	0.00	-0.04	0.03	-0.37

[1] refer to Figure 3-2

Table 3.4: K-effective for one-dimensional loading pattern.

	K-eff
SIMULATE-3 (16 npa)	1.246291
SIMULATE-3 (4 npa)	1.246293
COLOR2G (4 npa)	1.246256
SIMULATE-3 (1 npa)	1.246290
COLOR2G (1 npa)	1.246007

Comparing the fast flux results for the one-dimensional loading pattern, SIMULATE gives the same fast flux using 1 npa, 4 npa, and 16 npa. The maximum error in COLOR2G is 0.59% using 1 npa and 0.07% using 4 npa. Since the problem is strictly one-dimensional, the two-dimensional flux expansion used by COLOR2G is equivalent to the one-dimensional flux expansion used by SIMULATE. Therefore, the difference between the COLOR2G and SIMULATE results is attributed to the different orders of the polynomial used to represent the fast flux in the two codes. SIMULATE uses a fourth-order polynomial while COLOR2G only uses a third-order polynomial to represent the fast flux.

Comparing the thermal results from the one-dimensional loading pattern, SIMULATE has a maximum error of -0.07% using 1 npa and no error in the 4 npa case. The maximum error in COLOR2G is 0.45% for the 1 npa case and -0.07% for the 4 npa case. Again, for the one-dimensional case, the difference between the two codes is not due to the difference between the one-dimensional and two-dimensional flux expansions, but is instead attributed to the fact that SIMULATE uses a fourth-order polynomial/hyperbolic expansion and COLOR2G uses a linear polynomial/hyperbolic expansion with a thermal to fast flux ratio term.

3.4.2 Results for 3-UO_2/1-MOX Loading Pattern

The second colorset pattern described is the 3-UO_2/1-MOX loading pattern shown in Figure 3-3. The flux values for the 3-UO_2/1-MOX problem are shown in Tables 3.5 and 3.6. The k-effective values are shown in Table 3.7. The points labelled a-f in Figure 3-3 refer to corner point flux values, the points labelled g-l refer to the surface-averaged flux values, and the points labelled 1, 3, and 4 refer to the volume-averaged flux values in nodes 1, 3, and 4, respectively.

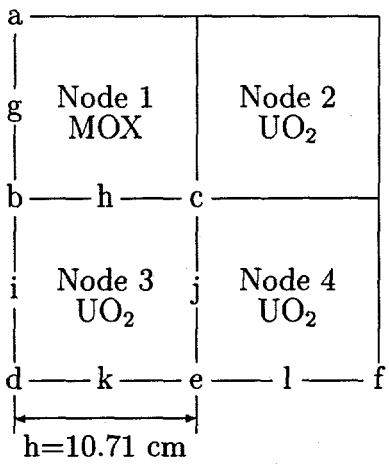

Figure 3-3: Flux positions for 3-UO_2/1-MOX loading pattern.

Table 3.5: Fast flux values for 3-UO_2/1-MOX loading pattern.

Position[1]	SIMULATE Results (64 npa)	Error from 64 npa SIMULATE case (%)			
		SIMULATE (4 npa)	COLOR2G (4 npa)	SIMULATE (1 npa)	COLOR2G (1 npa)
a	2.2732	0.02	-0.11	0.14	-1.48
b	2.3178	0.01	-0.07	0.06	-0.42
c	2.3391	0.02	-0.04	0.02	-0.14
d	2.3243	-0.01	-0.01	-0.03	0.91
e	2.3423	-0.01	-0.008	-0.06	0.04
f	2.3601	-0.01	-0.02	-0.08	-1.10
g	2.2952	0.02	0.00	0.03	-0.23
h	2.3307	0.03	-0.01	0.07	0.07
i	2.3125	0.00	-0.06	0.08	-0.10
j	2.3345	-0.03	-0.03	-0.07	-0.24
k	2.3309	-0.01	-0.00	-0.02	0.43
l	2.3537	-0.02	-0.02	-0.10	-0.48
1	2.314	0.03	0.08	0.03	0.68
3	2.321	-0.01	-0.04	-0.01	-0.16
4	2.346	0.01	-0.04	-0.05	-0.36

[1] refer to Figure 3-3

Table 3.6: Thermal flux values for 3-UO_2/1-MOX loading pattern.

Position[1]	SIMULATE Results (64 npa)	Error from 64 npa SIMULATE case (%)			
		SIMULATE (4 npa)	COLOR2G (4 npa)	SIMULATE (1 npa)	COLOR2G (1 npa)
a	0.079326	-0.05	-0.12	3.8	-0.7
b	0.20567	-0.07	-0.07	1.0	-1.2
c	0.29372	-0.91	0.58	-1.8	1.1
d	0.41952	-0.02	-0.01	-2.6	0.1
e	0.42339	-0.04	-0.01	-0.2	0.2
f	0.42734	0.02	-0.03	2.2	-0.5
g	0.090772	-0.01	-0.01	1.9	0.02
h	0.21588	-0.09	-0.12	-0.07	-0.2
i	0.38118	-0.02	-0.06	-1.3	-0.6
j	0.40045	0.21	0.13	0.5	0.1
k	0.42089	-0.09	-0.02	-1.0	-0.007
l	0.42594	0.05	-0.02	0.7	-0.1
1	0.1021	0.10	0.08	0.3	0.5
3	0.3851	-0.08	-0.10	-0.2	-0.3
4	0.4199	0.05	0.06	0.2	0.1

[1] refer to Figure 3-3

Table 3.7: K-effective for 3-UO_2/1-MOX loading pattern.

		K-eff
SIMULATE	(64 npa)	1.269908
SIMULATE	(4 npa)	1.269912
COLOR2G	(4 npa)	1.269887
SIMULATE	(1 npa)	1.269912
COLOR2G	(1 npa)	1.269740

Comparing the results for the 3-UO_2/1-MOX loading pattern, we find that SIMULATE predicts the fast corner point fluxes better than COLOR2G. The largest error in the fast corner point flux for SIMULATE is 0.14% for the 1 npa case and 0.02% for the 4 npa case. The largest error for the fast corner point flux given by COLOR2G is -1.48% for the 1 npa case and -0.11% for the 4 npa case. From these results, it can be assumed that the fourth-order polynomial representation used by SIMULATE is more important to the solution than the two-dimensional expansion used by COLOR2G. SIMULATE also gives better results for the surface-averaged and volume-averaged fast flux values.

Looking at the thermal flux though, we find that COLOR2G is better able to predict the thermal corner point fluxes. The largest error for SIMULATE is 3.8% for the 1 npa case and -0.91% for the 4 npa case. The largest error in COLOR2G is -1.2% for the 1 npa case and 0.58% for the 4 npa case. COLOR2G also gives better results for the thermal surface-averaged thermal flux values but SIMULATE gives better results for the volume-averaged thermal flux values. The reason for these results is that the two-dimensional thermal flux expansion is more important than the fourth-order polynomial representation.

3.4.3 Results for Checkerboard Loading Pattern

The third colorset pattern looked at is the checkerboard loading pattern shown in Figure 3-4. The flux values for the checkerboard problem are shown in Tables 3.8 and 3.9. The k-effective values are shown in Table 3.10. The points labelled a-d in Figure 3-4 refer to corner point flux values, the points labelled e-g refer to the surface-averaged flux values, and the points labelled 1 and 3 refer to the volume-averaged flux values in nodes 1 and 3.

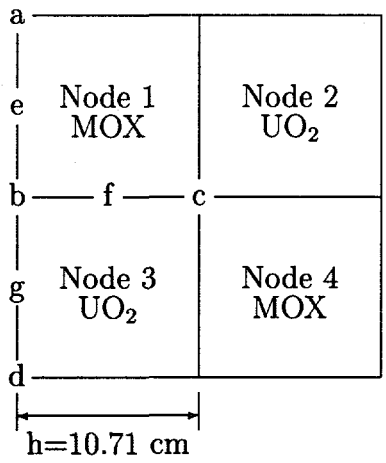

Figure 3-4: Flux positions for checkerboard loading pattern.

Table 3.8: Fast flux values for checkerboard loading pattern.

Position[1]	SIMULATE Results (64 npa)	Error from 64 npa SIMULATE case (%)			
		SIMULATE (4 npa)	COLOR2G (4 npa)	SIMULATE (1 npa)	COLOR2G (1 npa)
a	2.3197	0.00	-0.13	0.03	-2.7
b	2.3465	-0.01	-0.08	-0.08	-0.5
c	2.3546	0.17	-0.01	0.34	-0.1
d	2.3342	-0.02	-0.02	-0.05	1.9
e	2.3355	-0.01	-0.03	-0.12	-0.8
f	2.3488	0.01	-0.02	0.03	-0.1
g	2.3288	-0.01	-0.06	0.07	-0.3
1	2.346	0.00	0.05	-0.04	0.3
3	2.327	0.00	-0.06	0.03	-0.3

[1] refer to Figure 3-4

Table 3.9: Thermal flux values for checkerboard loading pattern.

Position[1]	SIMULATE Results (64 npa)	Error from 64 npa SIMULATE case (%)			
		SIMULATE (4 npa)	COLOR2G (4 npa)	SIMULATE (1 npa)	COLOR2G (1 npa)
a	0.080928	-0.05	-0.15	6.8	-2.0
b	0.20740	-0.14	-0.12	3.1	-1.9
c	0.18732	-6.09	3.16	14.3	7.4
d	0.41936	-0.05	-0.02	-5.2	0.5
e	0.092213	0.14	-0.06	2.1	-0.7
f	0.20173	0.33	0.22	0.9	0.5
g	0.38211	-0.13	-0.07	-2.0	-0.5
1	0.1013	0.17	0.14	0.4	0.4
3	0.3517	-0.14	-0.16	-0.3	-0.5

[1] refer to Figure 3-4

Table 3.10: K-effective for checkerboard loading pattern.

	K-eff
SIMULATE (64 npa)	1.246956
SIMULATE (4 npa)	1.246982
COLOR2G (4 npa)	1.246948
SIMULATE (1 npa)	1.247029
COLOR2G (1 npa)	1.246846

Comparing the fast flux results for the checkerboard pattern, the largest corner point flux error for COLOR2G is -2.7% for the 1 npa case and -0.13% for the 4 npa case. The largest corner point flux error for SIMULATE is 0.34% for the 1 npa case and 0.17% for the 4 npa case. For the the surface-average and volume-average fast flux values, the largest error in COLOR2G is -0.8% in the 1 npa case and -0.06% for the 4 npa case. In SIMULATE, the largest error in the surface-averaged and volume-averaged fast flux values is -0.12% for the 1 npa case and 0.01% for the 4 npa case. Again, from these results it can be concluded that a fourth-order polynomial representation is more important than a two-dimensional flux expansion for the fast flux.

Looking at the thermal flux in the checkerboard pattern, it can be seen that the thermal corner point flux at point c is especially difficult to calculate. The largest error in COLOR2G is 7.4% for the 1 npa case and 3.16% for the 4 npa case. For SIMULATE, the largest error for the thermal corner point fluxes is 14.3% for the 1 npa case and -6.09% for the 4 npa case. For the thermal surface-averaged and volume averaged flux values, the largest error in COLOR2G is -0.7% for the 1 npa case and 0.22% for the 4 npa case. For SIMULATE, the largest error in the surface-average and volume-average fast flux values is 2.1% for the 1 npa case and 0.33% for the 4 npa case. From these results, it can be seen that the two-dimensional thermal flux expansion is more important to the solution than the fourth-order polynomial/transverse integration combination.

3.5 Summary

In this chapter a method has been developed to solve the 2D-SANM equations given in Chapter 2 by using the well-known source iteration solution method. This solution method has been programmed into the computer code COLOR2G and the results have been compared to SIMULATE–3 for a 4-assembly colorset problem.

From the results, it has been found that SIMULATE–3 is better able to model the fast flux values in the colorset problem. This is probably due to the fourth-

order polynomial used by SIMULATE–3 to represent the fast flux expansion while COLOR2G only uses a third-order polynomial. However, it has also been shown that COLOR2G is better able to model the thermal flux values than SIMULATE–3. This is probably due to the two-dimensional thermal expansion used by COLOR2G where SIMULATE–3 uses a one-dimensional thermal flux expansion along with the transverse leakage approximation.

Chapter 4

Summary and Recommendations

4.1 Summary

The objective of this thesis research has been to derive the nodal equations for a two-dimensional, semi-analytic nodal method (2D-SANM); develop a solution method to solve the nodal equations; and to apply the 2D-SANM to a simple test problem containing UO_2 and MOX fuel assemblies.

In Chapter 2 the 2D-SANM equations were derived by representing the fast and thermal flux as an expansion of non-separable, two-dimensional functions. The fast flux expansion functions consist of third-order polynomial functions. The thermal flux expansion functions consist of a combination of linear polynomial terms, hyperbolic functions representing analytic solutions to the thermal diffusion equations, and a fast to thermal flux ratio term. The conditions needed to solve for the flux expansion coefficients were then presented, and equations were derived representing these conditions.

In Chapter 3 a method based on the well-known source-iteration method was developed to solve the 2D-SANM equations. This solution method was then implemented into the computer code COLOR2G and the code was used to solve for the flux distribution in a 4-assembly colorset problem containing UO_2 and MOX fuel assemblies arranged in three different patterns. The results were compared to results obtained using SIMULATE–3 with the SANM option. From these results, it was

found that the thermal flux representation used in the 2D-SANM performed better than the thermal flux representation used in SIMULATE–3. However, the fast flux representation in SIMULATE–3 gave more accurate results than the fast flux representation used in COLOR2G.

4.2 Recommendations for Further Work

The results from Chapter 3 have shown that the 2D-SANM can calculate the thermal flux distribution near the interfaces of MOX fuel assemblies better than polynomial nodal methods based on the transverse integration procedure. However, several questions still need to be answered before this method can be applied to a production code. First, a method has to be developed to solve large reactor problems. As is, this method is limited to fairly small problems that can be solved with a direct matrix solution method. Second, it was found that the fast flux representation used by 2D-SANM is not as accurate as a fourth-order polynomial expansion function used in combination with the transverse integration procedure. Third, this method must be expanded to three-dimensions so that full-core calculations can be performed. Each of these questions are discussed below.

4.2.1 Introduction of non-linear iteration method

One of the major limitations to the 2D-SANM, as mentioned in Chapter 3, is that the size of the problem is limited by the size of the matrix of nodal equation coefficients. Since the nodal equations cannot be put into a diagonally dominant form, the equations must be solved using a direct matrix solution method. For large matrices, a direct matrix solution method can introduce round-off errors into the solution.

The solution to this problem is to introduce a non-linear iteration method similar to the the method proposed by Smith [20] for the QUANDRY code. In this method, the nodal solution is broken into two steps. The first step is to solve for the nodal coupling coefficients (discontinuity factors) for each node using the 2D-SANM coupling equations. The second step is then to use the nodal coupling coefficients in a

modified finite-difference solution to perform the fission source and flux iterations for the global problem. The two steps are iterated upon until the fission source, flux, and coupling coefficients converge to within some tolerance.

The main difference between the QUANDRY non-linear iteration method and the 2D-SANM non-linear iteration method is that, instead of performing a nodal coupling calculation for each nodal interface in the problem, a nodal coupling solution will be performed for each node in the problem. For example, a finite difference solution can be obtained by using some initial guess for the discontinuity factors. Using the currents from the finite difference solution as boundary conditions, a 3x3 2D-SANM calculation can be performed for each node in the problem in order to obtain improved discontinuity factors. The improved discontinuity factors can be used to obtain a new finite-difference solution, and the process can be repeated until all the parameters converge.

4.2.2 Extension to a fourth-order polynomial for fast flux expansion

For the test cases run in Chapter 3, it was shown that the third-order polynomial fast flux expansion did not give results as accurate as the fourth-order fast flux expansion used in SIMULATE–3. In order to overcome this problem, the fast flux expansion in the 2D-SANM can be extended to fourth-order by adding additional expansion coefficients to the problem.

4.2.3 Extension of method to three-dimensions and multiple energy groups

The 2D-SANM can be extended to three dimensions either by adding more unknowns and representing the flux by a non-separable, three-dimensional flux expansion or by using a combination two-dimension/transverse integration representation. In the combinational approach, the two-dimensional flux expansion can be used to represent the flux in the radial direction. The axial flux direction can then be represented by a

one-dimensional flux shape using a transverse integration procedure.

The 2D-SANM can be also be extended to problems with as many energy groups as desired. The addition of more groups can be handled by either treating the additional group flux expansions as polynomial expansions as the fast group is now treated, or as a combination of polynomial and hyperbolic functions like the thermal flux expansion is treated. The extension of strict polynomial flux expansion to multigroups has already been demonstrated by Gehin [11].

Bibliography

[1] D. L. Delp, D. L. Fischer, J. M. Harriman, and M. J. Stedwell. *FLARE, A Three-Dimensional Boiling Water Reactor Simulator.* Number GEAP-4598. General Electric Company, 1964.

[2] R. D. Lawrence. "Progress in Nodal Methods for the Solution of the Neutron Diffusion and Transport Equations". *Progress in Nuclear Energy*, 17(3):271, 1986.

[3] K. S. Smith. "Assembly Homogenization Techniques for Light Water Reactor Analysis". *Progress in Nuclear Energy*, 17(3):303, 1986.

[4] Kord S. Smith. *An Analytical Nodal Method for Solving the Two-Group, Multidimensional, Static and Transient Neutron Diffusion Equation.* Nuclear Engineering Thesis, Massachusetts Institute of Technology, Nuclear Engineering Department, 1979.

[5] R. D. Lawrence. *A Nodal Green's Function Method for Multidimensional Neutron Diffusion Calculations.* Thesis, Univ. of Illinois, Urbana, Nuclear Engineering Program, 1979.

[6] H. Finnemann, F. Bennewitz, and M. R. Wagner. "Interface Current Techniques for Multidimensional Reactor Calculations". *Atomkernenergie*, 30:123, 1977.

[7] K. S. Smith. "QPANDA: An Advanced Nodal Method for LWR Analyses". *Transactions of the American Nuclear Society*, 50:532, 1985.

[8] K. S. Smith et al. *SIMULATE-3 Methodology.* Number SOA-89/4. Studsvick of America, Inc., 1989.

[9] P. D. Esser and K. S. Smith. "A Semianalytic Two-Group Nodal Model for SIMULATE–3". *Transactions of the American Nuclear Society*, 68:220, June 1993.

[10] A. F. Henry. *Nuclear Reactor Analysis.* M.I.T. Press, Cambridge, MA, 1975.

[11] Jess C. Gehin. *A Quasi-Static Polynomial Nodal Method for Nuclear Reactor Analysis.* Ph.D. Thesis, Massachusetts Institute of Technology, Nuclear Engineering Department, September 1992.

[12] K. Koebke. "Advances in Homogenization and Dehomogenization". In *Proc. Conf. Advances in Mathematical Methods for the Solution of Nuclear Engineering Problems*, volume II, page 59, Munich, April 1981. American Nuclear Society.

[13] Kord S. Smith. *Spatial Homogenization Methods for Light Water Reactors.* Ph.D. Thesis, Massachusetts Institute of Technology, Nuclear Engineering Department, 1980.

[14] K. R. Rempe, K. S. Smith, and A. F. Henry. "Simulate-3 Pin Power Reconstruction: Methodology and Benchmarking". In *Proc. of the International Reactor Physics Conference*, Jackson, WY, September 1988. American Nuclear Society.

[15] H. S. Khalil. *The Applications of Nodal Methods to PWR Analysis.* Ph.D. Thesis, Massachusetts Institute of Technology, Nuclear Engineering Department, January 1983.

[16] Jae Man Noh and Nam Zin Cho. "A New Approach of Analytic Basis Function Expansion to Neutron Diffusion Nodal Calculation". *Nuclear Science and Engineering*, 116:165, 1994.

[17] Kendall E. Atkinson. *An Introduction to Numerical Analysis.* John Wiley and Sons, New York, second edition, 1989.

[18] William H. Press et al. *Numerical Recipes in FORTRAN: the Art of Scientific Computing.* Cambridge University Press, New York, 1992.

[19] Kord S. Smith. personnel correspondance, April 1995.

[20] K. S. Smith. "Nodal Method Storage Reduction by Nonlinear Iteration". *Transactions of the American Nuclear Society*, 44:265, 1983.

Appendix A

Listing of Computer Code

COLOR2G

```
c=======================================================
c  program name:  COLOR2G
c  author:        Scott Palmtag
c  last update:   11 August 1995
c
c  This program solves a 4x4 colorset problem with 4 npa
c  zero current boundary conditions on all sides
c
c  For a description of the theory used in this program,
c  refer to the document:                                  10
c
c  Palmtag, Scott, "A Two-Dimensional, Semi-Analytic Expansion
c       Method for Nodal Calculations", M.S. Thesis, Massachusetts
c    Institute of Technology, Department of Nuclear Engineering,
c    August 1995.
c
c  General notes:
c
c  (1)  This program was developed on a Sun workstation but
c       was written in standard fortran so it should be    20
c       portable to other systems.  Problems might arise
c       in the subroutine introstuff and in the timing
c       routines.
c
c  (2)  The matrix LU decomposition was performed using
c       LAPACK routines.  Other routines (such as those
c       from numerical recipes) could easily be substituted
c       if the LAPACK routines are not available.
c       Documentation for the LAPACK routines can be found
c       using the world-wide-web at the URL:               30
```

```
c
c    (3) all calculations are performed using double precision
c        accuracy
c
c=======================================================
c
c  Listing of main variables used in program:
c
c  maxc      maximum number of flux coefficients (24*maxsize)
c  maxsize   number of nodes in problem (parameter=16)                    40
c  a(i,j)    array of equation coefficients
c  x()       array of flux coefficients
c  b()       source vector array/array of flux coefficients
c  h         node width
c  epso      convergence tolerance
c  fs        fission reaction rate
c  ab        absorption reaction rate
c  keff      reactor eigenvalue (double precision)
c  imat()    array of material composition (integer array)
c  time1()   array of total times (system+user) (real array)              50
c  time2()   array of system times (real array)
c  iout      integer value output unit number
c
c  trans     character value used with LAPACK routines
c  pivot()   array used with LAPACK routines (integer)
c  info      integer value returning status of calls to LAPACK
c                routines (integer)
c
c=======================================================
c                                                                        60
c  Common block: (this section must be included in a
c                 separate file called 'common.f' during
c                 compilation)
c
c     parameter (maxmat=4,neq=24)
c     integer maxmat, neq
c     double precision ts(maxmat), kappa(maxmat),
c    *  nufis(2*maxmat), abs(2*maxmat), scat(2*maxmat), diff(maxmat*2),
c    *  alpha1(maxmat), alpha2(maxmat), beta(maxmat), epsinu(maxmat)
c     common /xs/ nufis, abs, scat, diff, ts, kappa, epsinu,             70
c    *     alpha1, alpha2, beta
c
c  Listing of variables used in common block:
c
c  maxmat    maximum number of materials in problem (integer parameter)
c  neq       number of equations per node (integer parameter)
c  ts()      thermal to fast flux values for each material
c  kappa()   kappa values for each material
c  nufis()   nu*fission cross sections for each material and group
c  abs()     absorption cross sections for each material and group        80
c  scat()    downscatter cross sections for each material
c  diff()    diffusion constants for each material and group
c  alpha1()  alpha1 values for each material
c  alpha2()  alpha2 values for each material
```

```
c  beta()    beta values for each material
c  epsinu()  fission energy yield (ev) divided by nu for each material
c
c=======================================================
c
c  structure of input file                                          90
c
c  The input is read from standard input.
c
c  A sample input file is:
c
c      Case 1: 1-D problem
c      h     5.355
c      isize 4
c      imat  1 1 2 2
c            1 1 2 2                                               100
c            1 1 2 2
c            1 1 2 2
c      epso  1.0E-7
c      maxit 60
c      xnorm 4.7950974E-10
c      nmat  2
c
c      MOX assembly
c      diff  1.422 .277
c      abs   .01639 .327                                           110
c      nufis .01423 .547
c      scat  .0114 0.0
c
c      UO2 assembly
c      diff  1.394 .341
c      abs   .00985 .091
c      nufis .00741 .147
c      scat  .0165 0.0
c
c  The structure for the input file is:                             120
c
c  Line 1:  title card
c  Section 1 (options):
c    List of options.  Each option is optional.  If the option
c    is omitted, the default value will be used.  The options
c    can be in any order.  The format for this section is that
c    the option name must be in the first 5 columns and then the
c    value of the option is read in free format after that.
c    The options are listed below with the default value listed
c    in parentheses:                                             130
c       h      node width (20.0)
c       isize  must be 4, denoting a 4x4 matrix (4)
c       imat   material number for each node (16 integer values) (all 1's)
c          epso   convergence criteria (1.0e-4)
c          maxit  maximum number of iterations (100)
c          nmat   number of material types (1)
c          xnorm  flux normalization constant (1.0)
c    Section 2 (material cross sections):
```

```
c     Section 2 is terminated by a blank line.    Section 2 is then
c     repeated for each type of material in problem (nmat materials)          140
c     The first line of section 2 is always the title card for the material
c     The format for each preceeding line is cross section name in the
c     first five columns followed by the cross section value for groups 1
c     and 2.    Each cross section must be included, there is no default
c     values.
c     The cross sections are:
c          diff   macroscopic diffusion constants
c          abs    macroscopic absorption cross sections
c          nufis  macroscopic fission cross sections * nu
c          scat   macroscopic downscatter cross sections (only one value needed) 150
c          epsinu thermal fission yield/nu (optional)
c   Section 2 (optional format):
c     Section 2 can also be input in the following format in order to
c     resemble the input to SIMULATE:
c     Line 1:  title card for material
c     Line 2:  'mac ' followed by group 1 diffusion constant, group 2 diffusion
c         constant, downscattering xs, group 1 abs xs, group 2 abs xs, group 1
c         nufis, group 2 nufis, epsinu
c
c=============================================================              160
c
c  Listing of subroutines and functions and general program flow:
c
c  Part 1:    Read input
c      introstuff:
c      getinput:
c
c  Part 2:    Fill a-matrix
c      filla4:
c      fcorner:                                                             170
c      tcorner:
c      xcontinuity:
c      ycontinuity:
c      fcsource:
c      tcsource:
c      balance1:
c      balance2:
c      zerocurry:
c      zerocurrx:
c                                                                           180
c  Part 3:  apply source iteration solution method
c      fillb4:
c
c  Part 4:  edits
c      aedit4:
c      fastcorn:
c      tcorner1:
c      tcorner2:
c      tcorner3:
c      tcorner4:                                                            190
c      tcurr1:
c      tcurr2:
```

```
c      tcurr3:
c      tcurr4:
c
c
c=======================================================
c
c*  main routine
c                                                                    200
      program color2g
      parameter (maxc=384, maxsize=16)
      implicit double precision (a-h,o-z)
      double precision a(maxc,maxc), x(maxc), b(maxc),
     *  h, omega, epso, fs, ab, err, x1, x2, keff, xnorm
      integer imat(maxsize), pivot(maxc), info, maxit, iout
      character*1 trans
      real etime, dtime, time1(10), time2(10), tm(2)

      include 'common.f'                                             210

      double precision zero, one
      data zero, one, iout / 0.0, 1.0, 12 /
      data trans /'N'/

      time1(1) = dtime(tm)
      time2(1) = tm(2)

c*
c*  read input                                                       220
c*
      open (iout, file='output')
      call introstuff(iout)
      call getinput(maxsize, h, epso, maxit, xnorm, omega, imat,
     *      isize, iout)

      if (isize.ne.4) then
        write (*,10) isize
        write (iout,10) isize
        stop                                                         230
      endif
  10  format (' ** isize error, isize = ',i4,', should be 4 **')

      time1(2) = dtime(tm)
      time2(2) = tm(2)

c*
c*  fill a-matrix with constants
c*
                                                                     240
      call filla4 (maxc, imat, a, h, iout)

      time1(3) = dtime(tm)
      time2(3) = tm(2)

      write (iout,200)
```

```
 200  format (/,'1 main solution routine ')

c*
c*  get an initial guess for the coefficients                          250
c*

      do 30 i=1,maxc
         x(i) = zero
  30  continue
      do 35 i=1,maxsize
        n = (i-1)*neq
        x(1+n) = one
  35  continue
                                                                       260
      fs = zero
      ab = zero
      do 50 i=1,maxsize
        fs = fs + nufis(imat(i)*2-1)
        fs = fs + nufis(imat(i)*2) * ts(imat(i))
        ab = ab + abs(imat(i)*2-1)
        ab = ab + abs(imat(i)*2) * ts(imat(i))
   50 continue
      keff = fs / ab
      write (iout,*)                                                   270
      write (iout,*) 'initial guess for coefficients...'
      write (iout,65) 1, (x((j-1)*neq+1) ,j=1,maxsize)
      write (iout,65) 13,(x((j-1)*neq+13),j=1,maxsize)
      write (iout,*)
      write (iout,*) 'initial k-eff = ',keff
   65 format (2x,i2,1p,4e15.5,/,(4x,4e15.5))

c*
c*  factor c-matrix using LU decomp routine
c*  (dgetrf is a LAPACK routine)                                        280
c*

      call dgetrf (maxc, maxc, a, maxc, pivot, info)
      if (info.ne.0) then
        write (iout,72) info
        write (*,72) info
        stop
      endif
  72  format (' ** dgetrf info = ',i4,' ** ')
                                                                       290
c*
c*  start inner/outer iteration scheme to solving a*x=b
c*    1.  calculate b() using current x()
c*    2.  solve for new x()
c*    3.  calculate new keff
c*    4.  renormalize
c*    5.  check for convergence
c*

      iter = 0                                                         300
```

```
   90 format(/'           k-eff ',12x,'k-eff ',8x,'coefficient',
     *       /'iteration     value ',12x,'change',8x,' error    ')
      write (iout,90)
      write (iout,160) iter, keff
  100 continue
      iter = iter + 1

c*
c*  fill b-vector with fission terms
c*                                                                    310

      call fillb4(maxc,imat,keff,x,b)

c*
c*  call solver to find x()
c*  (dgetrs is a lapack routine)
c*

      call dgetrs(trans, maxc, 1, a, maxc, pivot, b, maxc, info)
      if (info.ne.0) then                                             320
        write (*,*) '** dgetrs returned info = ',info,' **'
        write (iout,*) '** dgetrs returned info = ',info,' **'
        stop
      endif

c*
c*  calculate new k-eff
c*

      fs = zero                                                       330
      ab = zero
      do 120 i=1,maxsize
        n = (i-1)*neq
        fs = fs + nufis(imat(i)*2-1)*b(n+1)
        fs = fs + nufis(imat(i)*2)*b(n+1)*ts(imat(i))
        fs = fs + nufis(imat(i)*2)*b(n+13)
        ab = ab + abs(imat(i)*2-1)*b(n+1)
        ab = ab + abs(imat(i)*2)*b(n+1)*ts(imat(i))
        ab = ab + abs(imat(i)*2)*b(n+13)
  120 continue                                                        340
      x1 = keff
      keff = fs / ab
      errk = keff - x1

c*
c*  check for convergence
c*

      err = -one
      do 150 i=1,maxc                                                 350
         x1 = dabs(b(i) - x(i))
         if (x1.gt.err) err = x1
  150 continue
      write (iout,160) iter, keff, errk, err
```

```
  160 format (1x,i4,5x,f16.12,2x,1p,e14.6,2x,e14.6)

c*
c*  normalize to one fission source and accelerate
c*
      do 130 i=1,maxc                                                    360
         b(i) = b(i) / fs
         x(i) = omega * b(i) + (one - omega)*x(i)
  130 continue

      if ((err.gt.epso).and.(iter.lt.maxit)) goto 100
      if (iter.ge.maxit) then
         write (iout,330)
         write (*,330)
      endif
  330 format (' ** maximum number of iterations met **'/)                370
  340 format (10x,'Node 1',9x,'Node 2',9x,'Node 3',9x,'Node 4')

c*
c*  finished with iterations!!
c*

c*
c*  do final normalization to 1 watt/cm * xnorm
c*
      write (iout,*)                                                     380
      write (iout,*) 'normalizing results to xnorm W/length '
      write (iout,*) 'xnorm = ',xnorm
      x1 = dble(0.0)
      do 345 i=1,maxsize
        n = (i-1)*neq
        write (iout,348) i, epsinu(imat(i))
        x2 = nufis(imat(i)*2-1)*b(n+1)
     *       + nufis(imat(i)*2)*(b(n+13) + b(n+1)*ts(imat(i)))
        x1 = x1 + x2 * epsinu(imat(i))
  345 continue                                                           390
      x1 = x1 * h * h / xnorm
  348 format (' node ',i2,2x,1p,e13.5,' W/fission/nu ')
      write (iout,*) 'total normalization factor = ',x1
      do 360 i=1,maxc
         x(i) = x(i) / x1
  360 continue

      time1(4) = dtime(tm)
      time2(4) = tm(2)
                                                                         400
c*
c*  print results
c*

      write (iout,*)
      write (iout,*) 'results...'
      write (iout,*) ' keff = ',keff
      write (iout,*) ' after ',iter,' iterations '
```

```
      write (iout,*)
      write (iout,*) ' coefficients'                                    410
      write (iout,340)
      write (iout,365) (i,(x(i+(j-1)*neq),j=1,8),i=1,neq)
      write (iout,*)
      write (iout,365) (i,(x(i+(j-1)*neq),j=9,16),i=1,neq)
      write (iout,*)
  365 format (2x,i2,1p,8e15.5)

      open (10,file='ACOEF',form='unformatted')
      write (10) maxsize
      write (10) (kappa(imat(i)),i=1,16),(ts(imat(i)),i=1,maxsize)     420
      write (10) (diff(imat(i)*2-1)/h,i=1,16),
     *           (diff(imat(i)*2)/h,i=1,16)
      write (10) (imat(i),i=1,16)
      write (10) keff, h
      write (10) xnorm
      do 400 j=1,maxsize
        write (10) (x(24*(j-1)+i),i=1,24)
  400 continue
      close (10)
                                                                        430
      time1(5) = dtime(tm)
      time2(5) = tm(2)
      time1(6) = etime(tm)
      time2(6) = tm(2)

      write (iout,*)
      write (iout,410)
      write (iout,420) 'start    ',time2(1), time1(1)
      write (iout,420) 'read input',time2(2), time1(2)
      write (iout,420) 'fill a   ',time2(3), time1(3)                   440
      write (iout,420) 'solution ',time2(4), time1(4)
      write (iout,420) 'edits    ',time2(5), time1(5)
      write (iout,420)
      write (iout,420) 'total    ',time2(6), time1(6)
      write (iout,420)
  410 format (/' Computer time used for different routines: ',
     *        /14x,' system  total')
  420 format (2x,a10,2x,2f8.2)

      call aedit4 (iout)                                                450

      close (iout)
      write (*,*) 'done'
      end

c=======================================================
c
c   subroutine to print introductory information
c   3/3/95
c                                                                       460
c   Note:    this subroutine contains many system calls
c          that may only be valid for sun workstations.
```

```
c
c   If you find problems with this subroutine, it can
c   be edited out without effecting the rest of the
c   program
c
c=======================================================

      subroutine introstuff(iout)                                    470

      integer i, getlog, hostnm, iout
      character start*24, fdate*24, htemp*60
      data htemp / 'htemp' /

      write (iout,*)  'Two-D Semi-Analytic Nodal Program '
      start = fdate()
      do 5 i=1,24
         if (ichar(start(i:i)).eq.0) start(i:i)=' '
   5  continue                                                       480
      write (iout,*) ' date: ',start
      i = getlog(htemp)
      write (iout,*) ' user: ',htemp
      i = hostnm(htemp)
      write (iout,*) ' host: ',htemp

      return
      end

c=======================================================           490
c
c   subroutine to read COLOR2G input file
c   7/10/95
c
c=======================================================

      subroutine getinput(maxsize, h, epso, maxit, xnorm, omega, imat,
     *       isize, iout)
      implicit double precision (a-h,o-z)
      integer nmat, imat(maxsize), maxit, isize                      500
      double precision h, epso, omega, xnorm
      double precision t1, zero

      include 'common.f'

      integer i
      character cc*5, cx*75
      data zero / 0.0 /

c*                                                                   510
c*  set defaults
c*

      isize = 4
      h = dble(20.0)
      do 5 i=1,maxsize
```

```
        imat(i) = 1
  5   continue
      epso = dble(1.0E-4)
      maxit = 100                                                    520
      omega = dble(1.2)
      nmat = 1
      xnorm = dble(1.0)

c*
c*  open input file and print intro stuff
c*

      write (*,*) 'reading input..'
      write (iout,100)                                               530
      read (5,20) cc, cx
      write (iout,101) cc,cx

c*
c*  read block 1 values
c*

  50  read (5,20) cc, cx
      if (cc.eq.'h   ') then
         read(cx,*) h                                                540
      elseif (cc.eq.'imat ') then
         read(cx,*) (imat(i),i=1,4)
         read(*,*) (imat(i),i=5,16)
      elseif (cc.eq.'isize') then
         read(cx,*) isize
      elseif (cc.eq.'epso ') then
         read(cx,*) epso
      elseif (cc.eq.'maxit') then
         read(cx,*) maxit
      elseif (cc.eq.'omega') then                                    550
         read(cx,*) omega
      elseif (cc.eq.'nmat ') then
         read(cx,*) nmat
      elseif (cc.eq.'xnorm') then
         read(cx,*) xnorm
      elseif (cc.eq.'    ')  then
         goto 60
      else
         write (iout,30) cc
      endif                                                          560
      goto 50
  20  format (a5, a75)
  30  format (' ** input value ''',a5,''' not understood, skipping ** ')

c*
c*  print block 1 input values
c*

  60  continue
      write (iout,200)                                               570
```

```
      write (iout,210) h, isize
      write (iout,220) nmat
      do 90 i=1,isize*isize
        write (iout,240) imat(i),i
   90 continue
      write (iout,270) maxit
      write (iout,250) epso
      write (iout,260) omega
      write (iout,280) xnorm
  100 format (/,'1subroutine getinput ')                                  580
  101 format (/,' title: ',a5, a75)
  200 format (/,' input values: ')
  210 format (2x,f12.6,' h - node width ',
     *       /,8x,i6,' n x n matrix ')
  220 format (8x,i6,' maximum number of materials in input')
  240 format (8x,i6,' material for node ',i2)
  250 format (2x,1p,e12.4,' epso (outer convergence) ')
  260 format (2x,1p,e12.4,' omega (acceleration) ')
  270 format (8x,i6,' maximum number of iterations ')
  280 format (2x,1p,e12.4,' normalization ')                              590

c*
c* check input values
c*

      if (isize.ne.4) then
         write (iout,32)
         write (*,32)
         stop
      endif                                                               600
      if (nmat.gt.maxmat) then
         write (iout,70) nmat, maxmat
         write (*,70) nmat, maxmat
         stop
      endif
      do 15 i=1,4
        if (imat(i).gt.nmat) then
           write (iout,80) i, imat(i), nmat
           stop
        endif                                                             610
   15 continue

   32 format (' ** invalid isize value, use either 2 or 4 ** ')
   70 format ('** maximum number of materials exceeded **',
     *       /,'**   number of materials = ',i3,' **',
     *       /,'**   maximum number      = ',i3,' **')
   80 format ('** error: imat ',i2,' = ',i3,
     *  ' but nmat is only ',i3,' **')

c*                                                                        620
c*  read material cross sections
c*

      do 300 n = 1, nmat
```

```
      diff(2*n-1)  = -dble(2.0)
      abs(2*n-1)   = -dble(2.0)
      nufis(2*n-1) = -dble(2.0)
      scat(2*n-1)  = -dble(2.0)
      epsinu(n)    = dble(1.325E-11)                                    630
      scat(2*n) = dble (0.0)
      read (5,20) cc, cx
      write (iout,330) n, cc, cx

400   read (5,20) cc, cx
      if (cc.eq.'diff ') then
         read(cx,*) diff(2*n-1), diff(2*n)
      elseif (cc.eq.'abs ') then
         read(cx,*) abs(2*n-1), abs(2*n)
      elseif (cc.eq.'nufis') then                                       640
         read(cx,*) nufis(2*n-1), nufis(2*n)
      elseif (cc.eq.'scat ') then
         read(cx,*) scat(2*n-1)
      elseif (cc.eq.'enu ') then
         read(cx,*) epsinu(n)
      elseif (cc.eq.'mac ') then
         read(cx,*) diff(2*n-1), diff(2*n), scat(2*n-1), abs(2*n-1),
     *              abs(2*n), nufis(2*n-1), nufis(2*n), epsinu(n)
      elseif (cc.eq.'    ')  then
         goto 410                                                       650
      endif
      goto 400
410   continue
      if (diff(2*n-1).lt.zero) then
         write (iout,360) 'diff ',n
         stop
      endif
      if (abs(2*n-1).lt.zero) then
         write (iout,360) 'abs ',n
         stop                                                           660
      endif
      if (nufis(2*n-1).lt.zero) then
         write (iout,360) 'nufis',n
         stop
      endif
      if (scat(2*n-1).lt.zero) then
         write (iout,360) 'scat ',n
         stop
      endif
                                                                        670
      t1 = abs(2*n)/diff(2*n)
      kappa(n) = dsqrt(t1) * h
      ts(n) = scat(2*n-1)/abs(2*n)

      t1 = kappa(n) / dble(2.0)
      alpha1(n) = dcosh(t1) - dsinh(t1) / t1
      t1 = kappa(n) / dble(2.0) / dsqrt(dble(2.0))
      alpha2(n) = dcosh(t1) - dsinh(t1) / t1
```

```
        beta(n) = dcosh(t1)*dcosh(t1) - dsinh(t1)*dsinh(t1)/t1/t1
                                                                      680
        write (iout,340) 1, diff(2*n-1),abs(2*n-1),
     *          nufis(2*n-1),scat(2*n-1)
        write (iout,340) 2, diff(2*n), abs(2*n), nufis(2*n), scat(2*n)
        write (iout,345) epsinu(n)
        write (iout,350) kappa(n), ts(n)
        write (iout,355) alpha1(n), alpha2(n), beta(n)
  300 continue
c     close (5)

                                                                      690
  330 format (/' material ',i2,': ',a5,a75,
     *     /,'  g     diff',8x,'abs',9x,'nufis',7x,'scat')
  340 format (1x,i3,5f12.6)
  345 format (/,2x,1p,e12.5,' joules/fission/nu ')
  350 format (/,2x,1p,e12.5,' kappa',/,2x,e12.5,' ts ')
  355 format (2x,1p,e12.5,' alpha1',/,2x,e12.5,' alpha2 ',
     *     /,2x,e12.5,' beta ')
  360 format (' ** ',a5,' not found for material ',i2,' ** ')

      return                                                          700
      end

c=====================================================
c   Scott Palmtag
c   update:  4/12/95
c
c   amatrix.f
c   fills a-matrix with constants
c
c   input:                                                            710
c     maxc    dimensions of a-matrix
c     imat    array of material properties
c     h       node width
c     iout    logical unit to write output
c
c   output:
c     a       array of constants
c
c=====================================================
c                                                                     720
      subroutine filla4 (maxc, imat, a, h, iout)
      implicit double precision (a-h,o-z)
      integer imat(*), maxc, i, j, n, iout
      double precision a(maxc,maxc), h, t1, t2, zero
      include 'common.f'
      double precision one, two
      data zero, one, two / 0.0, 1.0, 2.0 /
      isize = 4

      write (iout,110)                                                730
  110 format (/'1 routine filla   ')
      write (iout,*) ' h   = ',h
```

```
      write (iout,*) ' maxc = ',maxc

c*
c*  initially fill a-matrix with zeros
c*
      do 10 i=1,maxc
         do 10 j=1,maxc
            a(i,j) = zero                                          740
  10  continue
      nrow = 0

c*
c*  weighted nodal balances for each node
c*

      do 30 n=1,isize*isize
        icol = (n-1)*neq
        call balance1 (maxc, nrow, icol, imat(n), a, h)            750
        call balance2 (maxc, nrow, icol, imat(n), a)
  30  continue

      write (iout,*) ' nrow = ',nrow,' after balance equations'
      if (nrow.ne.128) stop

c*
c*  zero-current b.c.
c*
                                                                   760
      j = 1
      do 32 i=1,isize
        icol = (j-1) * neq
        call zerocurrx(maxc, icol, nrow, kappa(imat(j)), -one, a)
        icol = (j+isize-2) * neq
        call zerocurrx(maxc, icol, nrow, kappa(imat(j+isize-1)),one, a)
        j = j + isize
        icol = (i-1) * neq
        call zerocurry(maxc, icol, nrow, kappa(imat(i)),  one, a)
        icol = (i+11) * neq                                        770
        call zerocurry(maxc, icol, nrow, kappa(imat(i+12)),-one, a)
  32  continue

      write (iout,*) ' nrow = ',nrow,' after zero current eqs.'
      if (nrow.ne.160) stop

c*
c*  fill continuity equations
c*
                                                                   780
      call xcontinuity (maxc, nrow, 1, 2, imat, a)
      call xcontinuity (maxc, nrow, 2, 3, imat, a)
      call xcontinuity (maxc, nrow, 3, 4, imat, a)
      call xcontinuity (maxc, nrow, 5, 6, imat, a)
      call xcontinuity (maxc, nrow, 6, 7, imat, a)
      call xcontinuity (maxc, nrow, 7, 8, imat, a)
```

```
      call xcontinuity (maxc, nrow, 9,10, imat, a)
      call xcontinuity (maxc, nrow,10,11, imat, a)
      call xcontinuity (maxc, nrow,11,12, imat, a)
      call xcontinuity (maxc, nrow,13,14, imat, a)                    790
      call xcontinuity (maxc, nrow,14,15, imat, a)
      call xcontinuity (maxc, nrow,15,16, imat, a)

      call ycontinuity (maxc, nrow, 1, 5, imat, a)
      call ycontinuity (maxc, nrow, 2, 6, imat, a)
      call ycontinuity (maxc, nrow, 3, 7, imat, a)
      call ycontinuity (maxc, nrow, 4, 8, imat, a)
      call ycontinuity (maxc, nrow, 5, 9, imat, a)
      call ycontinuity (maxc, nrow, 6,10, imat, a)
      call ycontinuity (maxc, nrow, 7,11, imat, a)                    800
      call ycontinuity (maxc, nrow, 8,12, imat, a)
      call ycontinuity (maxc, nrow, 9,13, imat, a)
      call ycontinuity (maxc, nrow,10,14, imat, a)
      call ycontinuity (maxc, nrow,11,15, imat, a)
      call ycontinuity (maxc, nrow,12,16, imat, a)

      write (iout,*) ' nrow = ',nrow,' after continuity equations'
      if (nrow.ne.256) stop

c*                                                                    810
c*  fill center corner equations
c*

      do 40 i=1,12
         if (mod(i,isize).eq.0) then
            write (iout,*) '  skipping lower right corner of ',i
            goto 40
         endif
         i1 = (i-1) * neq
         i2 = i * neq                                                 820
         i3 = (i+isize-1) * neq
         i4 = (i+isize)*neq

         nrow = nrow + 1
         call fcorner (maxc,nrow,i1, one,-one, one,a)
         call fcorner (maxc,nrow,i2,-one,-one,-one,a)
         nrow = nrow + 1
         call fcorner (maxc,nrow,i2,-one,-one, one,a)
         call fcorner (maxc,nrow,i3, one, one,-one,a)
         nrow = nrow + 1                                              830
         call fcorner (maxc,nrow,i3, one, one, one,a)
         call fcorner (maxc,nrow,i4,-one, one,-one,a)
         nrow = nrow + 1
         call fcorner (maxc,nrow,i1, one,-one, ts(imat(i)),a)
         call fcorner (maxc,nrow,i2,-one,-one,-ts(imat(i+1)),a)
         call tcorner (maxc,nrow,i1, one,-one, one, a,kappa(imat(i)))
         call tcorner (maxc,nrow,i2,-one,-one,-one, a,kappa(imat(i+1)))
         nrow = nrow + 1
         call fcorner (maxc,nrow,i2,-one,-one, ts(imat(i+1)),a)
         call fcorner (maxc,nrow,i3, one, one,-ts(imat(i+isize)),a)   840
```

```
      call tcorner (maxc,nrow,i2,-one,-one, one, a,kappa(imat(i+1)))
      call tcorner (maxc,nrow,i3, one, one,-one, a,
     *           kappa(imat(i+isize)))
      nrow = nrow + 1
      call fcorner (maxc,nrow,i3, one, one, ts(imat(i+isize)),a)
      call fcorner (maxc,nrow,i4,-one, one,-ts(imat(i+isize+1)),a)
      call tcorner (maxc,nrow,i3, one, one, one, a,
     *           kappa(imat(i+isize)))
      call tcorner (maxc,nrow,i4,-one, one,-one, a,
     *           kappa(imat(i+isize+1)))                                  850

 40   continue

      write (iout,*) ' nrow = ',nrow,' after center corner point eq.'
      if (nrow.ne.310) stop

c*
c*  fill edge corner point equations
c*
      do 50 i=1,3                                                         860

      i1 = (i-1)*neq
      i2 = i*neq
      nrow = nrow + 1
      call fcorner (maxc,nrow, i1, one, one,  one,a)
      call fcorner (maxc,nrow, i2,-one, one, -one,a)
      nrow = nrow + 1
      call fcorner (maxc,nrow, i1, one, one,  ts(imat(i)),a)
      call fcorner (maxc,nrow, i2,-one, one, -ts(imat(i+1)),a)
      call tcorner (maxc,nrow, i1, one, one,  one, a, kappa(imat(i)))     870
      call tcorner (maxc,nrow, i2,-one, one, -one, a, kappa(imat(i+1)))

      i3 = (i+11)*neq
      i4 = (i+12)*neq
      nrow = nrow + 1
      call fcorner (maxc,nrow, i3, one,-one,  one,a)
      call fcorner (maxc,nrow, i4,-one,-one, -one,a)
      nrow = nrow + 1
      call fcorner (maxc,nrow, i3, one,-one, ts(imat(i+12)),a)
      call fcorner (maxc,nrow, i4,-one,-one,-ts(imat(i+13)),a)            880
      call tcorner (maxc,nrow, i3, one,-one, one, a, kappa(imat(i+12)))
      call tcorner (maxc,nrow, i4,-one,-one,-one, a, kappa(imat(i+13)))

 50   continue

      do 55 j=1,9, 4
         write (iout,*) '  j=',j,'  (should be 1, 5, 9)'

      i1 = (j-1)*neq
      i2 = (j+isize-1)*neq                                                890
      nrow = nrow + 1
      call fcorner (maxc,nrow, i1, -one,-one,  one,a)
      call fcorner (maxc,nrow, i2, -one, one, -one,a)
      nrow = nrow + 1
```

```
      call fcorner (maxc,nrow, i1, -one,-one,  ts(imat(j)),a)
      call fcorner (maxc,nrow, i2, -one, one, -ts(imat(j+isize)),a)
      call tcorner (maxc,nrow, i1, -one,-one,  one, a, kappa(imat(j)))
      call tcorner (maxc,nrow, i2, -one, one, -one, a,
     *              kappa(imat(j+isize)))
                                                                        900
      i1 = (j+2)*neq
      i2 = (j+2+isize)*neq
      nrow = nrow + 1
      call fcorner (maxc,nrow, i1, one,-one, one,a)
      call fcorner (maxc,nrow, i2, one, one,-one,a)
      nrow = nrow + 1
      call fcorner (maxc,nrow, i1, one,-one,  ts(imat(j+3)),a)
      call fcorner (maxc,nrow, i2, one, one, -ts(imat(j+3+isize)),a)
      call tcorner (maxc,nrow, i1, one,-one,  one, a, kappa(imat(j+3)))
      call tcorner (maxc,nrow, i2, one, one, -one, a,                   910
     *              kappa(imat(j+3+isize)))

   55 continue

      write (iout,*) ' nrow = ',nrow,' after edge corner point eq.'
      if (nrow.ne.334) stop

c*
c*  fill zero source condition equations for absolute corners
c*                                                                      920

      i = 1
      nrow = nrow + 1
      t1 = diff(imat(i)*2-1)
      call fcsource (maxc,a,nrow,0, -one, one, t1)
      nrow = nrow + 1
      t1 = ts(imat(i))*diff(imat(i)*2)
      call fcsource (maxc,a,nrow,0, -one, one, t1)
      call tcsource (maxc,a,nrow,0, -one, one, imat(i))
      write (iout,*) ' filling corner ',i                               930

      i = isize
      nrow = nrow + 1
      t1 = diff(imat(i)*2-1)
      call fcsource (maxc,a,nrow,(i-1)*neq, one, one, t1)
      nrow = nrow + 1
      t1 = ts(imat(i))*diff(imat(i)*2)
      call fcsource (maxc,a,nrow,(i-1)*neq, one, one, t1)
      call tcsource (maxc,a,nrow,(i-1)*neq, one, one, imat(i))
      write (iout,*) ' filling corner ',i                               940

      i = isize*isize-isize+1
      nrow = nrow + 1
      t1 = diff(imat(i)*2-1)
      call fcsource (maxc,a,nrow,(i-1)*neq,-one,-one, t1)
      nrow = nrow + 1
      t1 = ts(imat(i))*diff(imat(i)*2)
      call fcsource (maxc,a,nrow,(i-1)*neq,-one,-one, t1)
```

```
      call tcsource (maxc,a,nrow,(i-1)*neq,-one,-one, imat(i))
      write (iout,*) ' filling corner ',i                                    950

      i = isize*isize
      nrow = nrow + 1
      t1 = diff(imat(i)*2-1)
      call fcsource (maxc,a,nrow,(i-1)*neq, one,-one, t1)
      nrow = nrow + 1
      t1 = ts(imat(i))*diff(imat(i)*2)
      call fcsource (maxc,a,nrow,(i-1)*neq, one,-one, t1)
      call tcsource (maxc,a,nrow,(i-1)*neq, one,-one, imat(i))
      write (iout,*) ' filling corner ',i                                    960

c
c  fill edge zero-source conditions
c

      do 70 i=1,3

c top row

        nrow = nrow + 1                                                      970
        i1 = i
        i2 = i + 1
        t1 = diff(imat(i1)*2-1)
        t2 = diff(imat(i2)*2-1)
        call fcsource (maxc,a,nrow,(i1-1)*neq,  one, one, t1)
        call fcsource (maxc,a,nrow,(i2-1)*neq, -one, one, t2)
        write (iout,*) ' filling top  edge ',i1,' ',i2

        nrow = nrow + 1
        t1 = ts(imat(i1))*diff(imat(i1)*2)                                   980
        t2 = ts(imat(i2))*diff(imat(i2)*2)
        call fcsource (maxc,a,nrow,(i1-1)*neq,  one, one, t1)
        call fcsource (maxc,a,nrow,(i2-1)*neq, -one, one, t2)
        call tcsource (maxc,a,nrow,(i1-1)*neq,  one, one, imat(i1))
        call tcsource (maxc,a,nrow,(i2-1)*neq, -one, one, imat(i2))

c bottom row

        nrow = nrow + 1
        i1 = i + 12                                                          990
        i2 = i1 + 1
        t1 = diff(imat(i1)*2-1)
        t2 = diff(imat(i2)*2-1)
        call fcsource (maxc,a,nrow,(i1-1)*neq,  one,-one, t1)
        call fcsource (maxc,a,nrow,(i2-1)*neq, -one,-one, t2)
        write (iout,*) ' filling bott edge ',i1,' ',i2

        nrow = nrow + 1
        t1 = ts(imat(i1))*diff(imat(i1)*2)
        t2 = ts(imat(i2))*diff(imat(i2)*2)                                   1000
        call fcsource (maxc,a,nrow,(i1-1)*neq,  one,-one, t1)
        call fcsource (maxc,a,nrow,(i2-1)*neq, -one,-one, t2)
```

```
        call tcsource (maxc,a,nrow,(i1-1)*neq,  one,-one, imat(i1))
        call tcsource (maxc,a,nrow,(i2-1)*neq, -one,-one, imat(i2))

c left side

        nrow = nrow + 1
        i1 = (i-1)*4 + 1
        i2 = i1 + 4                                                    1010
        t1 = diff(imat(i1)*2-1)
        t2 = diff(imat(i2)*2-1)
        call fcsource (maxc,a,nrow,(i1-1)*neq, -one,-one, t1)
        call fcsource (maxc,a,nrow,(i2-1)*neq, -one, one, t2)
        write (iout,*) ' filling left  edge ',i1,' ',i2

        nrow = nrow + 1
        t1 = ts(imat(i1))*diff(imat(i1)*2)
        t2 = ts(imat(i2))*diff(imat(i2)*2)
        call fcsource (maxc,a,nrow,(i1-1)*neq, -one,-one, t1)          1020
        call fcsource (maxc,a,nrow,(i2-1)*neq, -one, one, t2)
        call tcsource (maxc,a,nrow,(i1-1)*neq, -one,-one, imat(i1))
        call tcsource (maxc,a,nrow,(i2-1)*neq, -one, one, imat(i2))

c right side

        nrow = nrow + 1
        i1 = 4 * i
        i2 = i1 + 4
        t1 = diff(imat(i1)*2-1)                                        1030
        t2 = diff(imat(i2)*2-1)
        call fcsource (maxc,a,nrow,(i1-1)*neq,  one,-one, t1)
        call fcsource (maxc,a,nrow,(i2-1)*neq,  one, one, t2)
        write (iout,*) ' filling right edge ',i1,' ',i2

        nrow = nrow + 1
        t1 = ts(imat(i1))*diff(imat(i1)*2)
        t2 = ts(imat(i2))*diff(imat(i2)*2)
        call fcsource (maxc,a,nrow,(i1-1)*neq,  one,-one, t1)
        call fcsource (maxc,a,nrow,(i2-1)*neq,  one, one, t2)          1040
        call tcsource (maxc,a,nrow,(i1-1)*neq,  one,-one, imat(i1))
        call tcsource (maxc,a,nrow,(i2-1)*neq,  one, one, imat(i2))

  70  continue

c*
c*  fill center zero-source condition
c*
      do 80 i=1,12
        if (mod(i,isize).eq.0) then                                    1050
           write (iout,*) ' skipping lower right corner of ',i
           goto 80
        endif
        i1 = i
        i2 = i + 1
        i3 = i + 4
```

```
        i4 = i + 5
        write (iout,*) ' filling corner ',i1,' ',i2,' ',i3,' ',i4

        nrow = nrow + 1                                                    1060
        t1 = diff(imat(i1)*2-1)
        t2 = diff(imat(i2)*2-1)
        t3 = diff(imat(i3)*2-1)
        t4 = diff(imat(i4)*2-1)
        call fcsource (maxc,a,nrow,(i1-1)*neq, one,-one, t1)
        call fcsource (maxc,a,nrow,(i2-1)*neq,-one,-one, t2)
        call fcsource (maxc,a,nrow,(i3-1)*neq, one, one, t3)
        call fcsource (maxc,a,nrow,(i4-1)*neq,-one, one, t4)

        nrow = nrow + 1                                                    1070
        t1 = ts(imat(i1))*diff(imat(i1)*2)
        t2 = ts(imat(i2))*diff(imat(i2)*2)
        t3 = ts(imat(i3))*diff(imat(i3)*2)
        t4 = ts(imat(i4))*diff(imat(i4)*2)
        call fcsource (maxc,a,nrow,(i1-1)*neq, one,-one, t1)
        call fcsource (maxc,a,nrow,(i2-1)*neq,-one,-one, t2)
        call fcsource (maxc,a,nrow,(i3-1)*neq, one, one, t3)
        call fcsource (maxc,a,nrow,(i4-1)*neq,-one, one, t4)
        call tcsource (maxc,a,nrow,(i1-1)*neq, one,-one, imat(i1))
        call tcsource (maxc,a,nrow,(i2-1)*neq,-one,-one, imat(i2))         1080
        call tcsource (maxc,a,nrow,(i3-1)*neq, one, one, imat(i3))
        call tcsource (maxc,a,nrow,(i4-1)*neq,-one, one, imat(i4))

 80   continue

      write (iout,*) ' nrow = ',nrow,' after zero-source conditions'
      if (nrow.ne.isize*isize*neq) stop

c*
c*  normalize sinh and cosh terms                                          1090
c*

      do 220 n=1,isize*isize
        icol = (n-1)*neq
        s1 = dsinh(kappa(imat(n))/two)
        t2 = kappa(imat(n)) / two / dsqrt(two)
        c2 = dcosh(t2)
        s2 = dsinh(t2)
        do 200 i=1,maxc
           a(i,17+icol) = a(i,17+icol) / alpha1(imat(n))                   1100
           a(i,18+icol) = a(i,18+icol) / s1
           a(i,19+icol) = a(i,19+icol) / alpha1(imat(n))
           a(i,20+icol) = a(i,20+icol) / s1
           a(i,21+icol) = a(i,21+icol) / beta(imat(n))
           a(i,22+icol) = a(i,22+icol) / c2 / s2
           a(i,23+icol) = a(i,23+icol) / c2 / s2
           a(i,24+icol) = a(i,24+icol) / s2 / s2
 200    continue
 220  continue
                                                                           1110
```

```
      return
      end

c=======================================================
c
c  subroutine to fill b vector (fission source vector)
c
c  note:  sinh and cosh expansion terms are normalized!
c
c  c() is the fission source vector,                                    1120
c  x() is the array of flux coefficients
c
c=======================================================

      subroutine fillb4(maxc, imat, keff, x, b)
      implicit double precision (a-h,o-z)
      integer maxc, nrow, icol, imat(*)
      double precision b(maxc), x(maxc), keff
      double precision f1, f2, t1, t2, t3, kap1, kap2, two, zero
      include 'common.f'                                               1130
      data zero, two / 0.0, 2.0 /

      do 50 i=1,maxc
         b(i) = zero
 50   continue

      nrow = 0
      do 280 n=1,16

        f2 = nufis(2*imat(n)) / keff                                   1140
        f1 = nufis(2*imat(n)-1) / keff + ts(imat(n)) * f2
        kap1 = kappa(imat(n))
        kap2 = kap1 / dsqrt(two)
        t1 = (dcosh(kap1/two)/dsinh(kap1/two) - two/kap1) / kap1
        t2 = (two - two * two/kap2*dsinh(kap2/two)/dcosh(kap2/two))
     *     / kap2 / kap2
        t3 = (dcosh(kap2/two) - two/kap2*dsinh(kap2/two))
     *     / (dcosh(kap2/two) + two/kap2*dsinh(kap2/two)) /kap2 / kap2
        icol=(n-1)*neq
                                                                       1150
        nrow = nrow + 1
        b(nrow) = f1 * x(1+icol) +  f2 * x(13+icol)

        nrow = nrow + 1
        b(nrow) = f1 * (x(5+icol)/dble(12.0) + x(11+icol)/dble(40.0))
     *    + f2 * (x(15+icol)/dble(12.0) + x(20+icol)*t1
     *    + x(23+icol) * t2)

        nrow = nrow + 1
        b(nrow) = f1 * (x(2+icol)/dble(12.0) + x(4+icol)/dble(40.0))   1160
     *    + f2 * (x(14+icol)/dble(12.0) + x(18+icol)*t1
     *    + x(22+icol) * t2)

        nrow = nrow + 1
```

```
        b(nrow) = f1 * (x(6+icol) + x(12+icol)*dble(0.09))
     *    + f2 * (x(16+icol) + dble(144.0) * t3 * x(24+icol))

        nrow = nrow + 4

  280 continue                                                          1170

      return
      end

c=======================================================
c
c   subroutine to fill flux corner point equations
c
c   this subroutine is used by the subroutine filla4
c                                                                       1180
c   x and y denote which corner the value is for.    For
c   example for the top left node (x,y)=(-1,1) for the
c   top right node (x,y)=(1,1), etc.
c
c=======================================================

      subroutine fcorner (maxc,nrow,icol,x,y,ts,a)
      integer maxc, nrow, icol
      double precision a(maxc,maxc), half, fourth, x, y, ts
      data half, fourth / 0.5, 0.25 /                                   1190
      a(nrow,1+icol) = ts
      a(nrow,2+icol) = ts * half * y
      a(nrow,3+icol) = ts * half
      a(nrow,4+icol) = ts * fourth * y
      a(nrow,5+icol) = ts * half * x
      a(nrow,6+icol) = ts * fourth * x * y
      a(nrow,7+icol) = ts * fourth * x
      a(nrow,8+icol) = ts * half
      a(nrow,9+icol) = ts * fourth * y
      a(nrow,10+icol)= ts * fourth                                      1200
      a(nrow,11+icol)= ts * fourth * x
      a(nrow,12+icol)= ts * fourth * fourth * x * y
      return
      end

      subroutine tcorner (maxc,nrow,icol,x,y,sign,a,kap)
      integer maxc, nrow, icol
      double precision x, y, sign, a(maxc,maxc), kap,
     *  half, fourth, s1, alpha, t2, s2, c2, two
      data half, fourth, two / 0.5, 0.25, 2.0 /                         1210
      s1 = dsinh(kap/two)
      alpha = dcosh(kap/two) - s1*two/kap
      t2 = kap / dsqrt(two) / two
      s2 = dsinh(t2)
      c2 = dcosh(t2)
      a(nrow,13+icol) =  sign
      a(nrow,14+icol) =  sign * half * y
      a(nrow,15+icol) =  sign * half * x
```

```
      a(nrow,16+icol) =  sign * fourth * x * y
      a(nrow,17+icol) =  sign * alpha                                    1220
      a(nrow,19+icol) =  sign * alpha
      a(nrow,18+icol) =  sign * y * s1
      a(nrow,20+icol) =  sign * x * s1
      a(nrow,21+icol) =  sign * (c2 * c2 - s2 * s2 / t2 / t2)
      a(nrow,22+icol) =  sign * y * s2 * c2
      a(nrow,23+icol) =  sign * x * s2 * c2
      a(nrow,24+icol) =  sign * s2 * s2 * x * y
      return
      end
                                                                         1230
c=======================================================
c
c   subroutine to fill continuity equations in x-direction
c   between nodes n1 and n2
c
c   this subroutine is used by the subroutine afilla4
c
c   note:  n1 has to be on the left and n2 has to be on the right
c
c=======================================================                 1240

      subroutine xcontinuity (maxc, nrow, n1, n2, imat, a)
      include 'common.f'
      integer maxc, nrow, ic1, ic2, n1, n2, imat(*)
      double precision a(maxc,maxc), k1a, k2a, k1b, k2b,
     *  s1a, s2a, s1b, s2b, c1a, c2a, c1b, c2b
      double precision one, half, two, three
      data one, half, two, three / 1.0, 0.5, 2.0, 3.0 /

      if (n1.gt.n2) then                                                 1250
         write (*,20)
         stop
      endif
 20   format (' ** error: n1.gt.n2 in subroutine xcontinuity ** ')

      ic1 = (n1-1)*neq
      ic2 = (n2-1)*neq

      k1a = kappa(imat(n1))
      c1a = dcosh(k1a/two)                                               1260
      s1a = dsinh(k1a/two)
      k2a = kappa(imat(n1)) / dsqrt(two)
      c2a = dcosh(k2a/two)
      s2a = dsinh(k2a/two)
      k1b = kappa(imat(n2))
      c1b = dcosh(k1b/two)
      s1b = dsinh(k1b/two)
      k2b = kappa(imat(n2)) / dsqrt(two)
      c2b = dcosh(k2b/two)
      s2b = dsinh(k2b/two)                                               1270

c  x fast flux continuity equations
```

```
      nrow = nrow + 1
      a(nrow,1+ic1) = one
      a(nrow,5+ic1) = half
      a(nrow,8+ic1) = half
      a(nrow,11+ic1)= half*half
      a(nrow,1+ic2) = -one
      a(nrow,5+ic2) =  half                                          1280
      a(nrow,8+ic2) = -half
      a(nrow,11+ic2)=  half*half

c  x thermal flux continuity equations

      nrow = nrow + 1
      a(nrow,1+ic1) =  one  * ts(imat(n1))
      a(nrow,5+ic1) =  half * ts(imat(n1))
      a(nrow,8+ic1) =  half * ts(imat(n1))
      a(nrow,11+ic1)=  half * half * ts(imat(n1))                    1290
      a(nrow,1+ic2) = -one  * ts(imat(n2))
      a(nrow,5+ic2) =  half * ts(imat(n2))
      a(nrow,8+ic2) = -half * ts(imat(n2))
      a(nrow,11+ic2)=  half * half * ts(imat(n2))
      a(nrow,13+ic1) = one
      a(nrow,15+ic1) = half
      a(nrow,19+ic1) = alpha1(imat(n1))
      a(nrow,20+ic1) = s1a
      a(nrow,21+ic1) = two / k2a * s2a * alpha2(imat(n1))
      a(nrow,23+ic1) = two / k2a * s2a * s2a                         1300
      a(nrow,13+ic2) = -one
      a(nrow,15+ic2) =  half
      a(nrow,19+ic2) = -alpha1(imat(n2))
      a(nrow,20+ic2) =  s1b
      a(nrow,21+ic2) = -two / k2b * s2b * alpha2(imat(n2))
      a(nrow,23+ic2) =  two / k2b * s2b * s2b

c   current continuity, group 1

      nrow = nrow + 1                                                1310
      a(nrow, 5+ic1) = -diff(imat(n1)*2-1)
      a(nrow, 8+ic1) = -diff(imat(n1)*2-1)*three
      a(nrow,11+ic1) = -diff(imat(n1)*2-1)*half*three
      a(nrow, 5+ic2) =  diff(imat(n2)*2-1)
      a(nrow, 8+ic2) = -diff(imat(n2)*2-1)*three
      a(nrow,11+ic2) =  diff(imat(n2)*2-1)*half*three

c   current continuity, group 2

      nrow = nrow + 1                                                1320
      a(nrow, 5+ic1) = -ts(imat(n1)) * diff(imat(n1)*2)
      a(nrow, 8+ic1) = -ts(imat(n1)) * diff(imat(n1)*2)*three
      a(nrow,11+ic1) = -ts(imat(n1)) * diff(imat(n1)*2)*half*three
      a(nrow, 5+ic2) =  ts(imat(n2)) * diff(imat(n2)*2)
      a(nrow, 8+ic2) = -ts(imat(n2)) * diff(imat(n2)*2)*three
      a(nrow,11+ic2) =  ts(imat(n2)) * diff(imat(n2)*2)*half*three
```

```
      a(nrow,15+ic1) = -diff(imat(n1)*2)
      a(nrow,19+ic1) = -diff(imat(n1)*2) * k1a * s1a
      a(nrow,20+ic1) = -diff(imat(n1)*2) * k1a * c1a
      a(nrow,21+ic1) = -diff(imat(n1)*2) * two * s2a * s2a                    1330
      a(nrow,23+ic1) = -diff(imat(n1)*2) * two * s2a * c2a
      a(nrow,15+ic2) =  diff(imat(n2)*2)
      a(nrow,19+ic2) = -diff(imat(n2)*2) * k1b * s1b
      a(nrow,20+ic2) =  diff(imat(n2)*2) * k1b * c1b
      a(nrow,21+ic2) = -diff(imat(n2)*2) * two * s2b * s2b
      a(nrow,23+ic2) =  diff(imat(n2)*2) * two * s2b * c2b

      return
      end
                                                                              1340
c=============================================================
c
c   subroutines to fill continuity equations in y-direction
c   between two nodes.
c
c   this subroutine is used by the subroutine afilla4
c
c   note:    n1 is top node, n2 is bottom node
c
c=============================================================                 1350

      subroutine ycontinuity (maxc, nrow, n1, n2, imat, a)
      include 'common.f'
      integer maxc, nrow, ic1, ic2, n1, n2, imat(*)
      double precision a(maxc,maxc), k1a, k2a, k1b, k2b,
     *  s1a, s2a, s1b, s2b, c1a, c2a, c1b, c2b
      double precision one, half, two, three
      data one, half, two, three / 1.0, 0.5, 2.0, 3.0 /

      if (n1.gt.n2) then                                                      1360
         write (*,20)
         stop
      endif
   20 format (' ** error: n1.gt.n2 in subroutine ycontinuity ** ')

      ic1 = (n1-1)*neq
      ic2 = (n2-1)*neq
      k1a = kappa(imat(n1))
      c1a = dcosh(k1a/two)
      s1a = dsinh(k1a/two)                                                    1370
      k2a = kappa(imat(n1)) / dsqrt(two)
      c2a = dcosh(k2a/two)
      s2a = dsinh(k2a/two)
      k1b = kappa(imat(n2))
      c1b = dcosh(k1b/two)
      s1b = dsinh(k1b/two)
      k2b = kappa(imat(n2)) / dsqrt(two)
      c2b = dcosh(k2b/two)
      s2b = dsinh(k2b/two)
                                                                              1380
```

```
c  y-flux continuity equations, group 1

      nrow = nrow + 1
      a(nrow,1+ic1)  = one
      a(nrow,2+ic1)  =-half
      a(nrow,3+ic1)  = half
      a(nrow,4+ic1)  =-half*half
      a(nrow,1+ic2) = -one
      a(nrow,2+ic2) = -half
      a(nrow,3+ic2) = -half                                          1390
      a(nrow,4+ic2) = -half*half

c  y thermal flux continuity equations

      nrow = nrow + 1
      a(nrow,1+ic1)  = one  * ts(imat(n1))
      a(nrow,2+ic1)  =-half * ts(imat(n1))
      a(nrow,3+ic1)  = half * ts(imat(n1))
      a(nrow,4+ic1)  =-half * ts(imat(n1)) * half
      a(nrow,1+ic2) = -one  * ts(imat(n2))                           1400
      a(nrow,2+ic2) = -half * ts(imat(n2))
      a(nrow,3+ic2) = -half * ts(imat(n2))
      a(nrow,4+ic2) = -half * ts(imat(n2)) * half
      a(nrow,13+ic1) =  one
      a(nrow,14+ic1) = -half
      a(nrow,17+ic1) =  alpha1(imat(n1))
      a(nrow,18+ic1) = -s1a
      a(nrow,21+ic1) =  two / k2a * s2a * alpha2(imat(n1))
      a(nrow,22+ic1) = -two / k2a * s2a * s2a
      a(nrow,13+ic2) = -one                                          1410
      a(nrow,14+ic2) = -half
      a(nrow,17+ic2) = -alpha1(imat(n2))
      a(nrow,18+ic2) = -s1b
      a(nrow,21+ic2) = -two / k2b * s2b * alpha2(imat(n2))
      a(nrow,22+ic2) = -two / k2b * s2b * s2b

c  y current continuity, group 1

      nrow = nrow + 1
      a(nrow,2+ic1) = -diff(imat(n1)*2-1)                            1420
      a(nrow,3+ic1) =  diff(imat(n1)*2-1)*three
      a(nrow,4+ic1) = -diff(imat(n1)*2-1)*half*three
      a(nrow,2+ic2) =  diff(imat(n2)*2-1)
      a(nrow,3+ic2) =  diff(imat(n2)*2-1)*three
      a(nrow,4+ic2) =  diff(imat(n2)*2-1)*half*three

c   current continuity, group 2

      nrow = nrow + 1
      a(nrow,2+ic1) = -ts(imat(n1)) * diff(imat(n1)*2)               1430
      a(nrow,3+ic1) =  ts(imat(n1)) * diff(imat(n1)*2)*three
      a(nrow,4+ic1) = -ts(imat(n1)) * diff(imat(n1)*2)*half*three
      a(nrow,2+ic2) =  ts(imat(n2)) * diff(imat(n2)*2)
      a(nrow,3+ic2) =  ts(imat(n2)) * diff(imat(n2)*2)*three
```

```
      a(nrow,4+ic2) =  ts(imat(n2)) * diff(imat(n2)*2)*half*three
      a(nrow,14+ic1) = -diff(imat(n1)*2)
      a(nrow,17+ic1) =  diff(imat(n1)*2) * k1a * s1a
      a(nrow,18+ic1) = -diff(imat(n1)*2) * k1a * c1a
      a(nrow,21+ic1) =  diff(imat(n1)*2) * two * s2a * s2a
      a(nrow,22+ic1) = -diff(imat(n1)*2) * two * s2a * c2a                    1440
      a(nrow,14+ic2) =  diff(imat(n2)*2)
      a(nrow,17+ic2) =  diff(imat(n2)*2) * k1b * s1b
      a(nrow,18+ic2) =  diff(imat(n2)*2) * k1b * c1b
      a(nrow,21+ic2) =  diff(imat(n2)*2) * two * s2b * s2b
      a(nrow,22+ic2) =  diff(imat(n2)*2) * two * s2b * c2b

      return
      end

c=========================================================              1450
c
c  subroutine to fill zero source condition equations
c
c  this subroutine is used by the subroutine afilla4
c
c  see the explanation under filling the corner point
c  equations to explaine the x and y notation for the nodes
c
c=========================================================
                                                                        1460
      subroutine fcsource (maxc,a,nrow,icol,x,y,d)
      integer maxc,nrow,icol
      double precision a(maxc,maxc), x, y, d
      double precision half, two, three
      data half, two, three / 0.5, 2.0, 3.0 /
      a(nrow,2+icol) =  d * y
      a(nrow,3+icol) =  d * three
      a(nrow,4+icol) =  d * three * half * y
      a(nrow,5+icol) =  d * x
      a(nrow,6+icol) =  d * x * y                                       1470
      a(nrow,7+icol) =  d * two * x
      a(nrow,8+icol) =  d * three
      a(nrow,9+icol) =  d * two * y
      a(nrow,10+icol)=  d * three
      a(nrow,11+icol)=  d * three * half * x
      a(nrow,12+icol)=  d * dble(0.75) * x * y
      return
      end

      subroutine tcsource (maxc,a,nrow,icol,x,y,mat)                    1480
      integer maxc, nrow, icol
      double precision a(maxc,maxc), kap1, kap2, c1, s1, c2, s2,
     *   two, x, y, d
      include 'common.f'
      data two / 2.0 /
      d = diff(mat*2)
      kap1 = kappa(mat)
      kap2 = kap1/dsqrt(two)
```

```
      c1 = kap1 * dcosh(kap1/two)
      s1 = kap1 * dsinh(kap1/two)                                          1490
      c2 = dcosh(kap2/two)
      s2 = dsinh(kap2/two)
      a(nrow,14+icol) = d * y
      a(nrow,15+icol) = d * x
      a(nrow,16+icol) = d * x * y
      a(nrow,17+icol) = d * s1
      a(nrow,18+icol) = d * c1 * y
      a(nrow,19+icol) = d * s1
      a(nrow,20+icol) = d * c1 * x
      a(nrow,21+icol) = d * kap2 * two * c2 * s2                           1500
      a(nrow,22+icol) = d * kap2 * (two * c2 * c2 - dble(1.))  * y
      a(nrow,23+icol) = d * kap2 * (two * c2 * c2 - dble(1.))  * x
      a(nrow,24+icol) = d * kap2 * two * c2 * s2 * x * y
      return
      end

c=======================================================
c
c  subroutines to fill weighted balance equations
c                                                                          1510
c  balance 1 fills the fast balance equations,
c  balance 2 fills the thermal balance equations,
c
c=======================================================
c
      subroutine balance1 (maxc, nrow, icol, mat, a, h)
      implicit double precision (a-h,o-z)
      integer maxc
      double precision a(maxc,maxc), a1, a2, six, half
      include 'common.f'                                                   1520
      data half, six / 0.5, 6.0 /

      a1 = -diff(mat*2-1)/h/h
      a2 = abs(mat*2-1) + scat(mat*2-1)

      nrow = nrow + 1
      a(nrow,1+icol) = a2
      a(nrow,3+icol) = six * a1
      a(nrow,8+icol) = six * a1
                                                                           1530
      nrow = nrow + 1
      a(nrow,7+icol)  = a1 * half
      a(nrow,5+icol)  = a2 / dble(12.0)
      a(nrow,11+icol) = a2 / dble(40.0) + a1

      nrow = nrow + 1
      a(nrow,9+icol) = a1 * half
      a(nrow,2+icol) = a2 / dble(12.0)
      a(nrow,4+icol) = a2 / dble(40.0) + a1
                                                                           1540
      nrow = nrow + 1
      a(nrow,6+icol)  = a2
```

```
      a(nrow,12+icol) = a2 * dble(0.09) + a1 * dble(7.2)

      return
      end

      subroutine balance2 (maxc, nrow, icol, mat, a)
      implicit double precision (a-h,o-z)
      integer maxc                                                    1550
      double precision a(maxc,maxc), kap1, kap2, two
      include 'common.f'
      data two / 2.0 /

      kap1 = kappa(mat)
      kap2 = kappa(mat)/dsqrt(two)

      nrow = nrow + 1
      a(nrow,3+icol) = dble(6.0) * ts(mat)
      a(nrow,8+icol) = dble(6.0) * ts(mat)                            1560
      a(nrow,13+icol)= -kap1 * kap1
      a(nrow,17+icol)= two * kap1 * dsinh(kap1/two)
      a(nrow,19+icol)= two * kap1 * dsinh(kap1/two)
      a(nrow,21+icol)= dble(8.0) * (dsinh(kap2/two)**2)

      nrow = nrow + 1
      a(nrow,7+icol)  =  dble(0.5) * ts(mat)
      a(nrow,11+icol) =  ts(mat)
      a(nrow,15+icol) = -kap1 * kap1 / dble(12.0)
                                                                      1570
      nrow = nrow + 1
      a(nrow,4+icol)  = ts(mat)
      a(nrow,9+icol)  = dble(0.5) * ts(mat)
      a(nrow,14+icol) = -kap1 * kap1 / dble(12.0)

      nrow = nrow + 1
      a(nrow,12+icol) =-dble(7.2) * ts(mat)
      a(nrow,16+icol) = kap1 * kap1

      return                                                          1580
      end

c=====================================================
c
c  zero current boundary conditions
c
c    x and y are double precision number equaling 1.0 or -1.0 depending
c    on which side of the node the equation is for
c
c    Note that the diffusion coefficient and h aren't used because      1590
c  the equation are set equal to zero, and therefore, D and h can
c  be cancelled from the equations
c
c  Also, the ts*flux1 terms aren't needed in the thermal equations
c    because they can be eliminated by using the fast zero-current
c    equations.
```

```
c
c=======================================================

      subroutine zerocurry (maxc, icol, nrow, k1, y, a)                 1600
      implicit double precision (a-h,o-z)
      double precision k1, s2, c2, y, two, a(maxc,maxc)
      integer icol, nrow
      data two / 2.0 /
      s2 = dsinh(k1/two/dsqrt(two))
      c2 = dcosh(k1/two/dsqrt(two))
      nrow = nrow + 1
      a(nrow,2+icol) =  dble(1.0)
      a(nrow,3+icol) =  dble(3.0) * y
      a(nrow,4+icol) =  dble(1.5)                                       1610
      nrow = nrow + 1
      a(nrow,14+icol)=  dble(1.0)
      a(nrow,17+icol)=  k1 * dsinh(k1/two) * y
      a(nrow,18+icol)=  k1 * dcosh(k1/two)
      a(nrow,21+icol)=  two * s2 * s2 * y
      a(nrow,22+icol)=  two * s2 * c2
      return
      end

      subroutine zerocurrx (maxc, icol, nrow, k1, x, a)                 1620
      implicit double precision (a-h,o-z)
      double precision k1, s2, c2, x, two, a(maxc,maxc)
      integer icol, nrow
      data two / 2.0 /
      s2 = dsinh(k1/two/dsqrt(two))
      c2 = dcosh(k1/two/dsqrt(two))
      nrow = nrow + 1
      a(nrow,5+icol)  =  dble(1.0)
      a(nrow,8+icol)  =  dble(3.0) * x
      a(nrow,11+icol) =  dble(1.5)                                      1630
      nrow = nrow + 1
      a(nrow,15+icol) =  dble(1.0)
      a(nrow,19+icol) =  k1 * dsinh(k1/two) * x
      a(nrow,20+icol) =  k1 * dcosh(k1/two)
      a(nrow,21+icol) =  two * s2 * s2 * x
      a(nrow,23+icol) =  two * s2 * c2
      return
      end

c=======================================================             1640
c=======================================================
c
c   edit subroutine for a 16 node problem
c
c   the following subroutines/functions make up the edit
c   module of color2g:
c       aedit4
c       fastcorn
c       tcorner1
c       tcorner2                                                      1650
```

```
c       tcorner3
c       tcorner4
c       tflux1
c       tflux2
c       tflux3
c       tflux4
c       tcurr1
c       tcurr2
c       tcurr3
c       tcurr4                                                    1660
c
c   These programs derive all of their input from an
c   interface file created by color2g, therefore these
c   programs can easily be separated from the main color2g
c   module in order to create a stand-alone program
c
c   tflux is thermal face-averaged flux
c
c=============================================================
                                                                  1670
      subroutine aedit4 (iout)
      implicit double precision (a-h,o,z)
      parameter (maxc=384)
      integer imat(16), iout
      double precision x(maxc), keff, h, three, one,
     *  k(16), ts(16), d1(16), d2(16), g1(16,9), g2(16,9), xnorm,
     *  fastcorn, tflux, tcorner1, tcorner2, tcorner3, tcorner4,
     *  tcurr1, tcurr2, tcurr3, tcurr4, t1, t2, t3
      data three, one / 3.0, 1.0 /
                                                                  1680
c*  note that d1 and d2 are normalized diffusion coefficients

c*  read interface file

      open (10,file='ACOEF',form='unformatted',status='old')
      read (10) isize
      if (isize.ne.16) then
         write (*,*) 'ACOEF error'
         stop
      endif                                                       1690
      read (10) (k(i),i=1,isize), (ts(i),i=1,isize)
      read (10) (d1(i),i=1,isize), (d2(i),i=1,isize)
      read (10) (imat(i),i=1,isize)
      read (10) keff, h
      read (10) xnorm
      do 5 j=1,isize
      read (10) (x(24*(j-1)+i),i=1,24)
   5  continue
      close (10)
                                                                  1700
c*  print material composition map

      write (iout,20)
      write (iout,50) (imat(i),i=1,16)
```

```
      write (iout,55) keff, h
      write (iout,60) xnorm
   20 format (/'1edit subroutine '/)
   50 format ('  +',4('====+'),/,'  |',4(i3,'|'),
     *      /,'  +',4('====+'),/,'  |',4(i3,'|'),
     *      /,'  +',4('====+'),/,'  |',4(i3,'|'),
     *      /,'  +',4('====+'),/,'  |',4(i3,'|'),
     *      /,'  +',4('====+'))
   55 format (/' k-eff = ',f14.9,/,' pitch = ',f12.5,/)
   60 format (' power = ',1p,e13.5,' W/cm ')

c*  calculate flux values for all of the nodes

      neq = 24
      do 80 n=1,isize
        icol = (n-1)*neq
        g1(n,1) = fastcorn(x,icol,-one, one)
        g1(n,2) = x(1+icol) + x(2+icol)*dble(0.5) + x(3+icol)*dble(0.5)
     *          + x(4+icol)*dble(0.25)
        g1(n,3) = fastcorn(x,icol, one, one)
        g1(n,4) = x(1+icol) - x(5+icol)*dble(0.5) + x(8+icol)*dble(0.5)
     *          - x(11+icol)*dble(0.25)
        g1(n,5) = x(1+icol)
        g1(n,6) = x(1+icol) + x(5+icol)*dble(0.5) + x(8+icol)*dble(0.5)
     *          + x(11+icol)*dble(0.25)
        g1(n,7) = fastcorn(x,icol,-one,-one)
        g1(n,8) = x(1+icol) - x(2+icol)*dble(0.5) + x(3+icol)*dble(0.5)
     *          - x(4+icol)*dble(0.25)
        g1(n,9) = fastcorn(x,icol, one,-one)
        g2(n,1) = g1(n,1) * ts(n) + tcorner1(x,icol)

        t1 = k(n) / dsqrt(dble(2.0)) / dble(2.0)
        t2 = dsinh(t1) / (t1*dcosh(t1) + dsinh(t1))
        t3 = dsinh(t1) / dcosh(t1) / t1
        tflux = x(13+icol)+x(14+icol)*dble(0.5)+x(17+icol)+x(18+icol)
     *    + x(21+icol) * t2 + x(22+icol) * t3
        g2(n,2) = g1(n,2) * ts(n) + tflux

        g2(n,3) = g1(n,3) * ts(n) + tcorner2(x,icol)

        tflux = x(13+icol)-x(15+icol)*dble(0.5)+x(19+icol)-x(20+icol)
     *   + x(21+icol) * t2 - x(23+icol) * t3
        g2(n,4) = g1(n,4) * ts(n) + tflux

        g2(n,5) = g1(n,5) * ts(n) + x(13+icol)

        tflux = x(13+icol)+x(15+icol)*dble(0.5)+x(19+icol)+x(20+icol)
     *   + x(21+icol) * t2 + x(23+icol) * t3
        g2(n,6) = g1(n,6) * ts(n) + tflux

        g2(n,7) = g1(n,7) * ts(n) + tcorner3(x,icol)

        tflux = x(13+icol)-x(14+icol)*dble(0.5)+x(17+icol)-x(18+icol)
     *   + x(21+icol) * t2 - x(22+icol) * t3
```

```
      g2(n,8) = g1(n,8) * ts(n) + tflux
                                                                        1760
      g2(n,9) = g1(n,9) * ts(n) + tcorner4(x,icol)
   80 continue

c*  print flux values

      write (iout,*) 'group 1 flux '
      write (iout,*)
      write (iout,110)
      do 90 j=1,15,4
        write (iout,100) (g1(j,i),i=1,3), (g1(j+1,i),i=1,3),            1770
     *     (g1(j+2,i),i=1,3), (g1(j+3,i),i=1,3)
        write (iout,100) (g1(j,i),i=4,6), (g1(j+1,i),i=4,6),
     *     (g1(j+2,i),i=4,6), (g1(j+3,i),i=4,6)
        write (iout,100) (g1(j,i),i=7,9), (g1(j+1,i),i=7,9),
     *     (g1(j+2,i),i=7,9), (g1(j+3,i),i=7,9)
        write (iout,110)
  90  continue

      write (iout,*)
      write (iout,*) 'group 2 flux'                                     1780
      write (iout,*)
      write (iout,110)
      do 95 j=1,15,4
        write (iout,100) (g2(j,i),i=1,3), (g2(j+1,i),i=1,3),
     *        (g2(j+2,i),i=1,3), (g2(j+3,i),i=1,3)
        write (iout,100) (g2(j,i),i=4,6), (g2(j+1,i),i=4,6),
     *        (g2(j+2,i),i=4,6), (g2(j+3,i),i=4,6)
        write (iout,100) (g2(j,i),i=7,9), (g2(j+1,i),i=7,9),
     *        (g2(j+2,i),i=7,9), (g2(j+3,i),i=7,9)
        write (iout,110)                                                1790
  95  continue

c*  calculate surface currents

      do 300 n=1, 16
        icol = (n-1)*neq
        g1(n,1) = -d1(n)*(x(2+icol) + x(3+icol)*three
     *        + x(4+icol)*dble(1.5))
        g1(n,2) = -d1(n)*(x(5+icol) - x(8+icol)*three
     *        + x(11+icol)*dble(1.5))                                   1800
        g1(n,3) = -d1(n)*(x(5+icol) + x(8+icol)*three
     *        + x(11+icol)*dble(1.5))
        g1(n,4) = -d1(n)*(x(2+icol) - x(3+icol)*three
     *        + x(4+icol)*dble(1.5))
        g2(n,1) = tcurr1(x,icol,d2(n),k(n)) - d2(n)*ts(n)*(x(2+icol)
     *        + x(3+icol)*three + x(4+icol)*dble(1.5))
        g2(n,2) = tcurr2(x,icol,d2(n),k(n)) - d2(n)*ts(n)*(x(5+icol)
     *        - x(8+icol)*three + x(11+icol)*dble(1.5))
        g2(n,3) = tcurr3(x,icol,d2(n),k(n)) - d2(n)*ts(n)*(x(5+icol)
     *        + x(8+icol)*three + x(11+icol)*dble(1.5))                 1810
        g2(n,4) = tcurr4(x,icol,d2(n),k(n)) - d2(n)*ts(n)*(x(2+icol)
     *        - x(3+icol)*three + x(4+icol)*dble(1.5))
```

```
  300 continue

c*  print surface currents

      write (iout,*)
      write (iout,*) 'group 1 currents'
      write (iout,*)
      write (iout,110)                                                    1820
      write (iout,120) (g1(n,1), n=1,4)
      write (iout,130) (g1(n,2), g1(n,3), n=1,4)
      write (iout,120) (g1(n,4), n=1,4)
      write (iout,110)
      write (iout,120) (g1(n,1), n=5,8)
      write (iout,130) (g1(n,2), g1(n,3), n=5,8)
      write (iout,120) (g1(n,4), n=5,8)
      write (iout,110)
      write (iout,120) (g1(n,1), n=9,12)
      write (iout,130) (g1(n,2), g1(n,3), n=9,12)                         1830
      write (iout,120) (g1(n,4), n=9,12)
      write (iout,110)
      write (iout,120) (g1(n,1), n=13,16)
      write (iout,130) (g1(n,2), g1(n,3), n=13,16)
      write (iout,120) (g1(n,4), n=13,16)
      write (iout,110)
      write (iout,*)
      write (iout,*) 'group 2 currents'
      write (iout,*)
      write (iout,110)                                                    1840
      write (iout,120) (g2(n,1), n=1,4)
      write (iout,130) (g2(n,2), g2(n,3), n=1,4)
      write (iout,120) (g2(n,4), n=1,4)
      write (iout,110)
      write (iout,120) (g2(n,1), n=5,8)
      write (iout,130) (g2(n,2), g2(n,3), n=5,8)
      write (iout,120) (g2(n,4), n=5,8)
      write (iout,110)
      write (iout,120) (g2(n,1), n=9,12)
      write (iout,130) (g2(n,2), g2(n,3), n=9,12)                         1850
      write (iout,120) (g2(n,4), n=9,12)
      write (iout,110)
      write (iout,120) (g2(n,1), n=13,16)
      write (iout,130) (g2(n,2), g2(n,3), n=13,16)
      write (iout,120) (g2(n,4), n=13,16)
      write (iout,110)


  100 format (' |',4(3f10.5,'|'))
  110 format (' +',4(31('='),'+'))                                        1860
  120 format (' |',4(10x,f10.5,10x,'|'))
  130 format (' |',4(f10.5,10x,f10.5'|'))


  200 continue
      return
```

```
      end

c=======================================================
c  this function computes the fast flux at the corner                  1870
c  of a node
c
c  x is the array of flux coefficients
c  icol is the node index used with 'x'
c  a is the x-coordinate of the corner (either -1 or +1)
c  b is the y-coordinate of the corner (either -1 or +1)
c
c  for example for the top left node (a,b)=(1,-1), for the
c  bottom right node (a,b)=(1,-1)
c                                                                      1880
c=======================================================

      double precision function fastcorn (x, icol, a, b)
      integer icol
      double precision a1, x(384), fourth, a, b
      data fourth / 0.25 /
      a1 = x(1+icol)
     *  + x(2+icol) * dble(0.5) * b
     *  + x(3+icol) * dble(0.5)
     *  + x(4+icol) * fourth * b                                        1890
     *  + x(5+icol) * dble(0.5) * a
     *  + x(6+icol) * fourth * a * b
     *  + x(7+icol) * fourth * a
     *  + x(8+icol) * dble(0.5)
     *  + x(9+icol) * fourth * b
     *  + x(10+icol)* fourth
     *  + x(11+icol)* fourth * a
     *  + x(12+icol)* fourth * fourth * a * b
      fastcorn = a1
      return                                                           1900
      end

c=======================================================
c
c  these functions return the thermal corner point fluxes
c  for nodes corners labeled 1-4 respectively
c
c  x is the array of flux coefficients, icol is the index
c  for the x array.
c                                                                      1910
c=======================================================

      double precision function tcorner1(x,icol)
      double precision x(384)
      tcorner1 = x(13+icol) + dble(0.5)*( x(14+icol) - x(15+icol) )
     *    - dble(0.25)*x(16+icol)
     *    + x(17+icol) + x(19+icol) + x(18+icol) - x(20+icol)
     *    + x(21+icol) + x(22+icol) - x(23+icol) - x(24+icol)
      return
      end                                                              1920
```

```
      double precision function tcorner2(x,icol)
      double precision x(384)
      tcorner2 = x(13+icol) + dble(0.5)*( x(14+icol) + x(15+icol) )
     *    + dble(0.25)*x(16+icol)
     *    + x(17+icol) + x(19+icol) + x(18+icol) + x(20+icol)
     *    + x(21+icol) + x(22+icol) + x(23+icol) + x(24+icol)
      return
      end
      double precision function tcorner3(x,icol)
      double precision x(384)                                              1930
      tcorner3 = x(13+icol) - dble(0.5)*( x(14+icol) + x(15+icol) )
     *    + dble(0.25)*x(16+icol)
     *    + x(17+icol) + x(19+icol) - x(18+icol) - x(20+icol)
     *    + x(21+icol) - x(22+icol) - x(23+icol) + x(24+icol)
      return
      end
      double precision function tcorner4(x,icol)
      double precision x(384)
      tcorner4 = x(13+icol) - dble(0.5)*( x(14+icol) - x(15+icol) )
     *    - dble(0.25)*x(16+icol)                                          1940
     *    + x(17+icol) + x(19+icol) - x(18+icol) + x(20+icol)
     *    + x(21+icol) - x(22+icol) + x(23+icol) - x(24+icol)
      return
      end

c=======================================================
c
c  thermal face average currents for node surfaces labeled 1-4
c
c  x is the array of flux coefficients, icol is the index used          1950
c  with the x array, diff is the normalized diffusion equation
c  for the node, and kappa is the kappa value for the node
c
c=======================================================

      double precision function tcurr1(x,icol,diff,kap)
      double precision x(384), a1, diff, kap, t1, t2, two, t3, kap2, one
      data one, two / 1.0, 2.0 /
      t1 = dcosh(kap/two) / dsinh(kap/two)
      t2 = kap / (t1 - two/kap)                                            1960
      kap2 = kap/dsqrt(two)/two
      t3 = (dcosh(kap2)/dsinh(kap2))**2 - one/kap2/kap2
      a1 = x(14+icol) + x(17+icol)*t2 + x(18+icol)*kap*t1
     *       + x(21+icol) * two / t3
     *       + x(22+icol) * two
      tcurr1 = -a1 * diff
      return
      end
      double precision function tcurr2(x,icol,diff, kap)
      double precision x(384), a1, diff, kap, t1, t2, two, t3, kap2, one   1970
      data one, two / 1.0, 2.0 /
      t1 = dcosh(kap/two) / dsinh(kap/two)
      t2 = kap / (t1 - two/kap)
      kap2 = kap/dsqrt(two)/two
```

```
      t3 = (dcosh(kap2)/dsinh(kap2))**2 - one/kap2/kap2
      a1 = x(15+icol) - x(19+icol)*t2 + x(20+icol)*t1*kap
     *       - x(21+icol) * two / t3
     *       + x(23+icol) * two
      tcurr2 = -a1 * diff
      return                                                          1980
      end
      double precision function tcurr3(x,icol,diff, kap)
      double precision x(384), a1, diff, kap, t1, t2, two, t3, kap2, one
      data one, two / 1.0, 2.0 /
      t1 = dcosh(kap/two) / dsinh(kap/two)
      t2 = kap / (t1 - two/kap)
      kap2 = kap/dsqrt(two)/two
      t3 = (dcosh(kap2)/dsinh(kap2))**2 - one/kap2/kap2
      a1 = x(15+icol) + x(19+icol)*t2 + x(20+icol)*t1*kap
     *       + x(21+icol) * two / t3                                  1990
     *       + x(23+icol) * two
      tcurr3 = -a1 * diff
      return
      end
      double precision function tcurr4(x,icol,diff, kap)
      double precision x(384), a1, diff, kap, t1, t2, two, t3, kap2, one
      data one, two / 1.0, 2.0 /
      t1 = dcosh(kap/two) / dsinh(kap/two)
      t2 = kap / (t1 - two/kap)
      kap2 = kap/dsqrt(two)/two                                       2000
      t3 = (dcosh(kap2)/dsinh(kap2))**2 - one/kap2/kap2
      a1 = x(14+icol) - x(17+icol)*t2 + x(18+icol)*kap*t1
     *       - x(21+icol) * two / t3
     *       + x(22+icol) * two
      tcurr4 = -a1 * diff
      return
      end

c=======================================================
                                                                      2010
```