NOTE TO USERS

This reproduction is the best copy available.

UNIVERSITY OF CINCINNATI

5/31/ 1990

I hereby recommend that the thesis prepared under my supervision by Haluk Utku

entitled Application of the Temporal Subdomain Method to the Finite Element Formulation of Two Dimensional Reactor Dynamics Equations

be accepted as fulfilling this part of the requirements for the degree of Doctor of Philosophy

Approved by:

APPLICATION OF THE TEMPORAL SUBDOMAIN METHOD TO THE FINITE ELEMENT FORMULATION OF TWO DIMENSIONAL REACTOR DYNAMICS EQUATIONS

A Dissertation submitted to the Division of Graduate Studies and Research of the University of Cincinnati

in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

in the Department of Mechanical, Industrial and Nuclear Engineering of the College of Engineering

June 1990

by

Haluk UTKU

B.S. Astronomy-Physics, University of Istanbul, Turkey, 1978

M.S. Nuclear Engineering, Technical University of Istanbul, Turkey, 1984

To my mother, brother
and late father

ACKNOWLEDGEMENTS

Several people definitely deserve recognition and the expression of appreciation from the author for their contribution during the course of the study.

My advisor, Professor John M. Christenson has supported and helped me by giving informative suggestions from the crawling days of this study to the present stage. I am grateful to him.

The author is indebted to Professor Alvin Shapiro. During my years in the Nuclear Engineering program, he has been always helpful. It was his lecture teaching me the initial steps of the finite element method.

I would like to thank Professor R. Huston for his contribution as the member of committee.

Warm thanks are due to Won Seok Park for the friendship he showed. This thesis would have been costly without his office supplies.

The author wishes to conclude this section with special gratitude to his relatives Sabahattin Benlikol, Bulent Orgun, and his brother Faruk Utku for providing the initial support.

ABSTRACT

The temporal subdomain method based on the Ritz-Galerkin method is investigated as a method for the solution of space-time dependent neutron dynamics equations. In the temporal subdomain finite element method, the time domain is divided into subdomains and within each subdomain the unknown coefficients of the time dependent trial functions are determined by making the residual of an appropriate functional orthogonal to the step function.

The spatial aspect of the problem was formulated in a two-dimensional rectangular coordinate system subdivided into a regular array of contiguous triangular finite elements. Within each element and within each neutron group the flux was represented by a linear polynomial. System condition number calculations as a function of the number of nodes were carried out for a series of static eigenvalue problems. The condition number calculations supported the conclusion that, for the finite element formulation used in this study, the global system equations are well-behaved and any additional programming effort to minimize the solution error caused by the matrix inversion procedure is not necessary. These conclusions were confirmed by the numerical results obtained from two computer codes (OPUS-G and OPUS-SOR) that were developed to support the investigation. OPUS-G used Gaussian elimination (a direct method) to solve the system equations, while OPUS-SOR used Successive Over Relaxation (an iterative

method) to solve the system equations. Numerical experiments were performed with both programs for a variety of static and dynamic problems. The result of the experiments were compared with those obtained by other programs using other methods. The comparison confirmed the stability of this formulation of the subdomain method and seems to indicate that more accurate results (for the same number of mesh points) can be obtained using this method than are commonly attained using other numerical procedures.

CONTENTS

CHAPTER 1

INTRODUCTION

The development of numerical approximation methods has been one of the main objectives of applied research ever since the availability of the computers. These studies have generally been directed toward devising more accurate and efficient approximation methods. As with the other fields of science, the neutron diffusion problem has been the subject of many studies of this type.

The most well-known and widely applied numerical method used to study the diffusion of neutrons in a nuclear fission reactor is the finite difference method. Two and three dimensional finite difference models for reactor problems have yielded good results. However the finite difference method requires small meshes with a corresponding large number of unknowns in order to accurately describe the behavior of a nuclear reactor.

Synthesis methods have also been developed and applied to the nuclear reactor problem. In this method, the solution is expanded in terms of functions representing various transient states of the problem. However, the selection of appropriate

expansion functions varies from one system to another, making it difficult to apply the method to a wide range of reactor problems. Furthermore, analytic error bounds for the approximation are not known.

Another widely used approximate method is the nodal method. In this method, the reactor is divided into a small number of regions and the regions are coupled through equations for the neutron flux or neutron current. Trial functions are defined in each region, which vanish outside the region. The drawback is the selection of the trial functions and definition of the proper coupling coefficients.

The finite element method is another numerical approximation method that has recently been used to study the neutron diffusion problem. In this method, the region of interest is divided into smaller regions. In each region, the unknown function is defined in terms of piecewise polynomials and the error introduced by the approximation is minimized within the small subregion.

Although there may be a debate, particularly on eigenvalue search calculations, the above methods yield nearly the same value for the reactor multiplication factor, k_{eff}, because k_{eff} is an integral value. However, recent studies have shown that for a given mesh size, the finite element method gives better approximation for k_{eff} than the finite difference method. The importance of an accurate determination of k_{eff} is illustrated by the statement that a -0.75 % error in the k_{eff}

estimate of a U-235 thermal reactor is the equivalent of saying that the designed reactor needs additional fuel of an amount equivalent to a reactivity insertion of more than 1$. Since accuracy and error bounds of the finite element method are well established, the method can be applied to a wide range of problems to determine the accuracy of the results obtained by any of the other previously mentioned methods. These types of comparisons become especially important in dynamic calculations where the best estimate of the true response of a particular region or point is required.

The finite element method can be incorporated into a variety of different numerical techniques. This thesis is concerned with the application of the subdomain collocation method to the neutron dynamic equation in conjunction with the Ritz-Galerkin finite element method applied in space domain.

1.2. The Finite element method

The finite element method is based on variational concepts presented in discretized way. Rayleigh was the first to introduce variational techniques for solving engineering problems. He proposed substituting an unknown function by a series of known ones with undetermined coefficients. The coefficients can then be obtained by minimizing a functional. The method was further developed by Ritz. A common difficulty is the formulation of the functional even when the governing

equation is known. In such cases Galerkin's error minimization method can be applied, which needs only the governing equations and boundary conditions.

The advantage of the finite element method over the finite difference approach is mainly because of its flexibility in handling complex geometries and arbitrary boundary conditions. The choice of higher order polynomial trial functions results in increased accuracy for a given mesh size compared to the finite difference method. The construction of trial functions in one or multi-dimensions automatically satisfies the continuity of the unknown function between the elements. Continuity of derivatives of the function can also be satisfied by choosing higher order polynomials, that is, piecewise continuous polynomials permit imposing conditions on the continuity or discontinuity of derivatives of the unknown functions.

Some scientists believe that the idea of finite element was first initiated by Courant's article[1] in 1943. In his article, Courant used triangular elements to generate trial functions to which he applied the variational method to obtain approximate solutions of column torsion problems. The suggestion of partitioning the domain into smaller subdomains first appeared in solid mechanics studies.[2],[3] What followed was the rapid construction of finite element theories and their formulations for various applied problems. For specific applications, the literature on the method of finite elements

is enormous. A survey covering more than 2000 cited papers and books was given by Whiteman[4]. Another bibliography was compiled by Norrie and Vries.[5]

Although the idea of solving the diffusion equation utilizing trial functions is not new[6] , Kang and Hansen[7] were one of the earliest to introduce the finite element method in reactor calculations. They applied the method, with Hermite polynomials, to the space, energy, and time dependent neutron diffusion equation. Ohnishi[8] proposed the use of linear triangular elements for the spatial dependence in static diffusion calculations together with a discrete ordinates treatment of the angular variables for transport calculations, but his paper did not include numerical results. Semenza et al.[9],[10], applied both triangular and rectangular elements to static multigroup equations. They employed Cholesky decomposition for the inner iteration and the standard power iteration for outer iteration, with an option for Chebyshev acceleration. Kaper et al.[11],[12] applied a higher-order approximation with up to fifth degree Lagrange polynomials to the two-dimensional static diffusion equation. They employed Cholesky decomposition to solve the within group equations and Chebyshev acceleration for the outer iteration.

The code NEPTUN[13] used the finite element method to calculate solution to the static multi-group neutron diffusion equation in 2 and 3 dimensions. For within group calculations, Successive Over Relaxation method (SOR) or Cholesky

factorization was employed. Coarse mesh rebalancing and the Chebyshev acceleration method was used for the outer iteration. This code was extended further to include dynamic calculations by Kavenoky and Lautard[14]. The same researchers also employed the finite element method in depletion calculations[15]. Another study was done at the Stuttgart University[16],[17]. In that work, the finite element formulation of the 2 and 3 dimensional multi-group diffusion equations were solved by the conjugate gradient method. The most recent application to 3D static multigroup diffusion equation has been at the Technical University of Istanbul[18]. In that study, the Lagrangien bilinear elements are utilized and the fission source iteration is accelerated by a coarse mesh rebalancing.

The previous studies on the solution of static multigroup neutron diffusion equation show that higher accuracy is obtained by the finite element method compared to other well known methods. There have also been a few studies of the finite element approximation of dynamic reactor behavior. Hennart[19] applied the finite element method for point kinetic equations, concluding that results are promising for space-time calculations. Kang and Hansen[7] proposed their Hermite method but they did not include any comparison study. Meade[20] applied the collocation method to the space variables and several finite difference approaches to the time variable in the solution of the one dimensional kinetic equation.

In this thesis, an approximate solution is sought for the space-time dependent neutron dynamic equation coupled with precursor equations. Kantorovich's method[21] is utilized in the polynomial approximation of the neutron flux. The space dependent inner product operations are carried out by the Ritz-Galerkin method. The time dependent flux is represented by a linear polynomial and the subdomain method is utilized in the integration of time variable. The subdomain method was originally suggested by Biezeno[22],[23] and it was Biezeno's presentation[24] of the subdomain method which prompted Courant's remark[25] that led Crandall[26] to choose the name **Method of weighted residuals**. Later, the subdomain collocation method was also recognized as a special case of the method of moments when the weighting function is selected to be unity. The subdomain method was first used for point kinetics by Brittan[27] and later by Kaganova[28]. To our knowledge this thesis is the first application of the temporal subdomain method to the finite element formulation of the two dimensional reactor kinetic equations.

In Chapter 2, the variational method and the Ritz method are discussed. Weighted residual methods are briefly presented and Galerkin method is explained. Finally, the error involved in the finite element method is reviewed. In Chapter 3, the weak formulation and the choice of trial functions are presented. With this formulation the two dimensional, two group neutron diffusion equations can be reduced to the set of

ordinary differential equations.

The finite element matrices encountered in structural analysis are stiff and sensitive to singularities that make the resulting matrices nearly ill-conditioned. Furthermore, in general, the matrix of the discretized system is irregular and sparse, which may make the data transfer of matrix elements of the finite element method slow and inefficient. These reasons suggest that direct inversion methods, such as the Cholesky method, will work better than iterative methods for the solution of finite element formulation of structural problems. Partly for historical reasons, the same approach (matrix inversion by a direct method) has been used in most finite element reactor physics calculations. The condition number of a matrix is an indication of possible errors for direct solution methods and slow convergence for iterative methods. By evaluating the condition numbers for different problems, and by solving the problems by Gauss elimination with static condensation technique(as a direct method) and by Successive over relaxation (SOR) method(as an iterative method), we can justify the necessity of using a certain solution method in reactor physics problems. Chapter 4 is concerned with the solution of neutron dynamics equation, the subdomain method is introduced and applied to the time dependent part of the neutron dynamic equation coupled with precursor equations. Numerical results are presented at the end of Chapters 3 and 4.

CHAPTER 2

In this chapter, the variational technique is outlined and the Ritz method is discussed as an illustration of the application of the variational technique in a finite dimensional space. We will see how the Ritz method and Galerkin method become equivalent to each other when certain choices are made. We close the chapter by reviewing the order of the errors involved in the finite element method.

2.1. THE VARIATIONAL TECHNIQUE

In the solution of many differential equations, the solution has the property that it minimizes a certain functional, **F**, defined on a real space V, that is, the function which provides the solution is a stationary point of such a functional. Thus, the task of solving a boundary value problem is equivalent to that of finding a function in V that makes **F** stationary, and the problem is called variational formulation of the boundary value problem.

Consider the neutron diffusion equation for the neutrons

in energy group g as a boundary value problem. Using the same notation as in reference [18] the diffusion equation for group g can be written as

$$\nabla . D_g(\mathbf{r}) \nabla \Phi_g(\mathbf{r}) + \Sigma_{r,g} \Phi_g(\mathbf{r}) = q_g(\mathbf{r}), \qquad (2.1)$$

with the following boundary conditions

$$\mathbf{n} . D_g(\mathbf{r}) \nabla \Phi_g(\mathbf{r}) + \frac{1}{2} \Phi_g(\mathbf{r}) = 0, \qquad \mathbf{r} \in S_v \text{ (vacuum b. c.)} \qquad (2.2)$$

$$\mathbf{n} . D_g(\mathbf{r}) \nabla \Phi_g(\mathbf{r}) = 0, \qquad \mathbf{r} \in S_r \text{ (reflective b. c.)} \qquad (2.3)$$

where

$D_g(\mathbf{r})$ = Diffusion coefficient of the system for the neutrons in energy group g at location $\mathbf{r}$.

$\Sigma_{r,g}(\mathbf{r})$ = Macroscopic removal cross section of the system for the neutrons in energy group g at $\mathbf{r}$.

$q_g(\mathbf{r})$ = The source of neutrons in the system for the energy group g at $\mathbf{r}$.

$\Phi_g(\mathbf{r})$ = The neutron flux in energy g at $\mathbf{r}$.

S_v is a convex reactor surface beyond which there is no neutron diffusion and no source of neutrons. S_r is a reactor symmetry surface. $\mathbf{n}$ is a unit vector normal to the reactor surface. The problem is to find a solution to the system of equations represented by (2.1) [g=1,2,..,G] satisfying the boundary conditions (2.2) and (2.3).

Consider the following functional having continuous

partial derivatives of order up to 2

$$F[\phi_g(\mathbf{r})] = \int_D \{D_g(\mathbf{r})[\nabla\phi_g(\mathbf{r})]^2 + \Sigma_{r,g}(\mathbf{r})\phi^2(\mathbf{r}) - 2q_g(\mathbf{r})\phi_g(\mathbf{r})\}d\mathbf{r} + \tfrac{1}{2}\int_{Sv} \phi_g^2(\mathbf{r})dS. \quad (2.4)$$

Assume that the function, $\Phi_g(\mathbf{r})$, minimizes the above functional. Since $\phi_g(\mathbf{r})$ and $\Phi_g(\mathbf{r})$ are in the same space, $\phi_g(\mathbf{r})$ can be represented in terms of $\Phi_g(\mathbf{r})$ plus some deviation

$$\phi_g(\mathbf{r}) = \Phi_g(\mathbf{r}) + \delta(\mathbf{r}). \quad (2.5)$$

If (2.5) is substituted in (2.4), we arrive at,

$$F[\phi_g(\mathbf{r})] = F[\Phi_g(\mathbf{r})] + 2\delta F_g[\delta(\mathbf{r})] + \delta^2 F[\delta(\mathbf{r})], \quad (2.6)$$

where

$$\delta F[\delta(\mathbf{r})] = \int_D \{D_g(\mathbf{r})[\nabla\Phi_g(\mathbf{r})][\nabla\delta(\mathbf{r})] + \Sigma_{r,g}(\mathbf{r})\Phi_g(\mathbf{r})\delta(\mathbf{r}) - q_g(\mathbf{r})\delta(\mathbf{r})\}d\mathbf{r} + \tfrac{1}{2}\int_{Sv} \Phi_g(\mathbf{r})\delta(\mathbf{r})dS, \quad (2.7)$$

$$\delta^2 F[\delta(\mathbf{r})] = \int_D \{D_g(\mathbf{r})[\nabla\delta(\mathbf{r})]^2 + \Sigma_{r,g}(\mathbf{r})\delta^2(\mathbf{r})\}d\mathbf{r} + \tfrac{1}{2}\int_{Sv} \delta^2(\mathbf{r})dS. \quad (2.8)$$

It is clear that the second derivative of the functional $F[\phi_g(\mathbf{r})]$ is always positive. Hence, a sufficient requirement

for $\Phi_g(\mathbf{r})$ to minimize $F[\phi_g(\mathbf{r})]$ is to make the first derivative of the functional equal to zero. If Green's theorem is utilized for the first term in (2.7) we obtain

$$\int_D D_g(\mathbf{r})[\nabla\Phi_g(\mathbf{r})][\nabla\delta(\mathbf{r})]d\mathbf{r} = \int_D \delta(\mathbf{r})\nabla.[D_g(\mathbf{r})\nabla\Phi_g(\mathbf{r})]d\mathbf{r} + \int_S \delta(\mathbf{r})D_g(\mathbf{r})\mathbf{n}.\nabla\Phi_g(\mathbf{r})dS. \qquad (2.9)$$

By substituting (2.9) in (2.7), we get

$$\delta F[\delta(\mathbf{r})] = \int_D \{ -\nabla.D_g(\mathbf{r})\nabla\Phi_g(\mathbf{r}) + \Sigma_{r,g}(\mathbf{r})\Phi_g(\mathbf{r}) - q_g(\mathbf{r})\}\delta(\mathbf{r})d\mathbf{r} + \int_{Sv} \{ \mathbf{n}.D_g(\mathbf{r})\nabla\Phi_g(\mathbf{r}) + \tfrac{1}{2}\Phi_g(\mathbf{r}) \}\delta(\mathbf{r})dS + \int_{Sr} \{ \mathbf{n}.D_g(\mathbf{r})\nabla\Phi_g(\mathbf{r}) \}\delta(\mathbf{r})dS. \qquad (2.10)$$

As can be seen from (2.10), the sufficient condition to make the first derivative equal to zero is that $\Phi_g(r)$ is the solution of (2.1) and satisfies the boundary conditions (2.2) and (2.3). Hence we conclude that solutions to (2.1) which satisfy the associated boundary conditions do indeed minimize the functional.

The functional given by (2.4) was defined in a continuous way and its solution lies in an infinite-dimensional space V. Many engineering problems can not be solved analytically.

Instead the solution is sought in a finite dimensional subspace V_N.

In the following sections, we omit subscript g and assume that the discussions are valid for the g group equations. This implies that the following methods are applied on a group by group bases.

2.1.2. THE RITZ METHOD

To obtain an approximate solution of a practical boundary value problem some kind of discretization is nearly always required. In the Ritz method case, this goal is achieved by selecting an N dimensional subspace $\mathbf{V}_N$ which belongs to the space $\mathbf{V}$. A function $\Phi(\mathbf{r})$ in the subspace V_N is defined as the set of N linearly independent functions (or basis) in the form of

$$\bar{\Phi}(\mathbf{r}) = \sum_{i=1}^{N} c_i u_i(\mathbf{r}) \ , \tag{2.11}$$

where the c_i, called the Ritz coefficients, are unknown parameters.

In order to develop the Ritz approximation, we need the following definitions. We define the inner product by

$$(q,\Phi) = \int_D q(\mathbf{r})\Phi(\mathbf{r})d\mathbf{r} \ , \tag{2.12}$$

the L^2 norm by

$$\|\Phi\| = (\Phi,\Phi)^{1/2} , \tag{2.13}$$

and the bilinear form, B, by

$$B(\Phi,\Phi) = \int_D \{ D(\mathbf{r})[\nabla\Phi(\mathbf{r})]^2 + \Sigma_r(\mathbf{r})\Phi^2(\mathbf{r}) \}d\mathbf{r}. \tag{2.14}$$

From the discussion in the preceding section we know that the solution $\Phi(\mathbf{r})$ of (2.1) minimizes the functional

$$F[\Phi(\mathbf{r})] = B(\Phi,\Phi) - 2(q,\Phi). \tag{2.15}$$

Using the approximate solution (2.11) and the bilinear property of B(,) we find that

$$F[\Phi(\mathbf{r})] = B\Big(\sum_{i=1}^{N} c_i u_i, \sum_{i=1}^{N} c_i u_i\Big) - 2\Big(q, \sum_{i=1}^{N} c_i u_i\Big), \tag{2.16}$$

$$= \sum_{i,j=1}^{N} c_i c_j B(u_i,u_j) - 2\sum_{i=1}^{N} c_i (q,u_i), \tag{2.17}$$

or in matrix form

$$F[\Phi(\mathbf{r})] = c^T K c - 2c^T f , \tag{2.18}$$

where

$$c = [\, c_1, c_2, .., c_N \,]^T ,$$

$$f = [\,(q,u_1), (q,u_2), (q,u_3), (q,u_N)\,]^T ,$$

$$K = \begin{bmatrix} B(u_1,u_1) & B(u_2,u_1) & \cdots & B(u_N,u_1) \\ B(u_1,u_2) & B(u_2,u_2) & \cdots & B(u_N,u_2) \\ \vdots & \vdots & & \vdots \\ B(u_1,u_N) & B(u_2,u_N) & \cdots & B(u_N,u_N) \end{bmatrix} . \quad (2.19)$$

Once the set $\{u_i\}$ is chosen, the computation of the coefficients c_i in (2.11) is simple. The functional becomes, after carrying out the inner product operation (i.e., integrating over the domain), an ordinary function of coefficients. The unknown coefficients can be defined by taking the derivative of $F[\Phi(\mathbf{r})]$ with respect to c_i

$$\frac{\delta F}{\delta c_i} = 0 \qquad i=1,2,\ldots,N \qquad (2.20)$$

or in matrix notation

$$\frac{\delta F}{\delta \mathbf{c}} = \mathbf{K}\mathbf{c} - \mathbf{f} = 0. \qquad (2.21)$$

There are some comments which should be made in the derivation of the previous equations:

1. The matrix **K** is symmetric. There are cases that **K** may not be symmetric but we are not concerned with asymmetry in this study.

2. The set of basis functions used for the Ritz method can be divided into two general categories: dimension independent and dimension dependent. The characteristic property of dimension

independent basis functions is that in passing from dimension N to (N+1), we simply add a new function u_{N+1} to the old set. In the case of dimension dependent basis functions, we replace the old set of N functions by an entirely new set of (N+1) functions. Furthermore, dimension dependent basis functions make **K** a sparse matrix. Dimension independent basis functions usually do not have this property.

3. Once we state that **K** is symmetric, arriving at (2.21) from (2.18) can be easily justified. Differentiating (2.18)

$$\delta F = \delta c^T K c + c^T K \delta c + 2\delta c^T f \ .$$

Since **K** is symmetric

$$\delta c^T K c = c^T K \delta c \ ,$$

Hence

$$\delta F = 2\delta c^T (Kc - f) = 0,$$

$$Kc - f = 0.$$

In practice, the region of interest is divided into subdomains called elements. Equation (2.21) is applied within each element and the overall result is assembled element by element in the following form

$$\sum^{e} \left(\frac{\delta F}{\delta c}\right)^{e} = 0.$$

II.II. WEIGHTED RESIDUAL METHODS

The weighted residual method is an appealing mathematical tool, because it not only handles linear self-adjoint problems, but it also gives a good approximation for non-self-adjoint systems as well. Consider the equation

$$A\Phi(\mathbf{r}) = q(\mathbf{r}), \tag{2.22}$$

where A is a linear differential operator. To apply the method of weighted residuals, an approximate solution is sought in the form

$$\bar{\Phi}(\mathbf{r}) = \sum_{i=1}^{N} c_i u_i(\mathbf{r}) \ , \tag{2.23}$$

with the condition that the u_i are linearly independent.

Variable dependency of the trial functions, u_i, is optional. They may be a function of one or all of the independent variables. The c_i are either undetermined parameters or undetermined functions of one or more of the independent variables.

Since (2.23) is an approximation, it does not satisfy (2.22). A measure of error is given by the residual, R, which is

$$R = \bar{q} - A \sum_{i=1}^{N} c_i u_i(\mathbf{r}) \ . \tag{2.24}$$

Here '-' over the source vector indicates that an approximation similar to (2.23) is applied to this term.

As the number of approximating functions increases, the residual decreases. Since the residual does not vanish, N weighted averages of the residual are set equal to zero

$$\int_D W_k(\mathbf{r}) R d\mathbf{r} = 0. \quad k=1,\ldots,N \ , \tag{2.25}$$

where $W_k(\mathbf{r})$ is one of N arbitrary weighting functions. The variables in D include all the independent variables which the c_i do not depend on.

If the c_i are undetermined parameters, N algebraic equations result, which are solved for the c_i. When the c_i are undetermined functions, (2.25) yields N simultaneous differential equations. The method of undetermined parameters is also called the direct method. The method of undetermined functions is known as the semidirect method, or **Kantorovich method.**

Substituting (2.24) into (2.25) and writing the result in matrix notation, we get

$$(\mathbf{Au},\mathbf{W})\mathbf{c} = (\mathbf{q},\mathbf{W}). \tag{2.26}$$

If the choice $\mathbf{W} = \mathbf{u}$ is made, the weighting functions are the trial functions themselves. Then, the method is called Galerkin method

$$(\mathbf{Au},\mathbf{u})\mathbf{c} = (\mathbf{q},\mathbf{u}). \tag{2.27}$$

We will write the following theorem without proof.

Theorem[29]: If $(\mathbf{Au},\mathbf{u})$ satisfies

$$(\mathbf{Au},\mathbf{u}) \geq \tau \|u\|^2 , \tag{2.28}$$

where τ is an arbitrary positive constant, then the solution of (2.27) exists and is unique. Furthermore A is said to be positive definite.

Applying these results to the one-group neutron diffusion equation, let

$$A = -\nabla.D(\mathbf{r})\nabla + \Sigma_r(\mathbf{r}), \tag{2.29}$$

and define

$$B_1(\bar{\Phi},W) = \int_D \{ D(\mathbf{r})\nabla\bar{\Phi}(\mathbf{r})\nabla W(\mathbf{r}) + \Sigma_r(\mathbf{r})\bar{\Phi}(\mathbf{r})W(\mathbf{r}) \}d\mathbf{r}, \tag{2.30}$$

$$B_2(\bar{\Phi},W) = \int_S \tfrac{1}{2}\bar{\Phi}W dS , \tag{2.31}$$

$$B(\bar{\Phi},W) = B_1(\bar{\Phi},W) + B_2(\bar{\Phi},W) . \tag{2.32}$$

We assume that $\bar{\Phi}$, and W belong to the Sobolev space ($\bar{\Phi}$,W ϵ Æ) where $\bar{\Phi}$,W = 0 on the surface S of the domain.

Then from the Green's theorem

$$(A\bar{\Phi},W) = B_1(\bar{\Phi},W) - B_3(\bar{\Phi},W) , \tag{2.33}$$

where

$$B_3(\bar{\Phi},W) = \int_S \{ \mathbf{n}.D(\mathbf{r})\bar{\nabla}\bar{\Phi}(\mathbf{r}) \} W dS \ . \qquad (2.34)$$

From the boundary condition (2.2), we can write

$$B_3(\bar{\Phi},W) = -B_2(\bar{\Phi},W) \ , \qquad (2.35)$$

which yields,

$$(A\bar{\Phi},W) = B(\bar{\Phi},W) . \qquad (2.36)$$

In the case of the Galerkin method, W = u, and this choice together with the use of (2.23) and (2.36) leads to a system of equations in the form of

$$\mathbf{Kc} = \mathbf{f}, \qquad (2.37)$$

which is the same matrix equation as (2.21). Therefore we say that for the self-adjoint systems the Ritz method and the Galerkin method are equivalent.

2.3. ERRORS IN THE FINITE ELEMENT METHOD

Application of the finite element method to the mathematical model of a physical system yields discrete equations. Errors from various sources give rise to an approximate solution. The sources of errors can be defined as the equation errors, transformation of equation errors into solution errors, and manipulation errors[30]. These errors are discussed in this

section.

2.3.1. Equation errors

Equation Discretization Errors

In the finite element method the exact solution is approximated by trial functions of the following form

$$\bar{\phi}(\mathbf{r},t) = \mathbf{c}(t)^T\mathbf{u}(\mathbf{r}) \ , \tag{2.38}$$

where

$$\mathbf{u}(\mathbf{r}) = [\ u_1(\mathbf{r}),\ u_2(\mathbf{r}),\ \ldots,\ u_N(\mathbf{r})\ ]^T,$$
$$\mathbf{c}^T(t) = [\ c_1(t),\ c_2(t),\ \ldots,\ c_N(t)\ ],$$

and $c_j(t)$ represents $c(\mathbf{r}_j,t)$ (i.e., the solution at the j th. node at time t) and $u_j(\mathbf{r})$ is the j th. trial function.

Application of the Ritz-Galerkin method yields a discretization error denoted by r_d. As the number of trial functions increases,the discretization error decreases

$$\lim_{N \to \infty} r_d \to 0.$$

For a given N, it is possible to express r_d in terms of the higher order derivatives of the response and mesh sizes of the finite element by taking advantage of the remainder theorem of the power series expansions. If the response of the actual system is sufficiently smooth (that is, if the higher order derivatives are not varying rapidly in the finite

elements) then by using the approximate solution itself, r_d may be approximately evaluated by the remainder theorem and we obtain the following relationship

$$||r_d|| = O(h^p),$$

where h is the mesh size and the power p is proportional to the degree of the interpolation rule used.

Equation Round-off Errors

In order to obtain an approximate answer from the discretized system, the finite element matrices are calculated and assembled into a global matrix. The computed global matrix is at least slightly different from the exact one. The difference comes from round-off error arising from the finite digit representation of the matrix elements.

The numbers used in the computer arithmetic are represented with floating-point numbers. Let a_1 denote the actual real number, $|a_1|$ its magnitude and $fl(a_1)$ its machine representation, then, the round-off error is

$$r_r = fl(a_1) - a_1 , \tag{2.39}$$

and

$$|r_r| \leq |a_1| 2^{-t},$$

where 't' is the number of binary bits used for the mantissa in the machine representation. Let a_1 and b_1 denote two real numbers, (a_1+b_1) and a_1b_1, their actual sum and product, and $fl(a_1+b_1)$ and $fl(a_1b_1)$ denote the corresponding machine

results. It can be shown that the following hold

$$r_r = fl(a_1+b_1) - (a_1+b_1),$$
$$|r_r| \leq (|a_1|+|b_1|)2^{-t}, \qquad (2.40)$$

and

$$r_r = fl(a_1b_1) - a_1b_1,$$
$$|r_r| \leq |a_1||b_1|2^{-t}. \qquad (2.41)$$

We can write similar inequalities for the corresponding matrix operations. Let **a** and **b** represent two vectors of order N, and $\mathbf{fl(a^Tb)}$ denote the result of $\mathbf{a^Tb}$ product carried out in a t-bit mantissa machine, then

$$\mathbf{r_r = fl(a^Tb) - a^Tb},$$
$$||\mathbf{r_r}|| \leq ||\mathbf{a}||||\mathbf{b}||2^{-t}N. \qquad (2.42)$$

Similarly, if **A** and **B** denote P*N and N*Q matrices, and **fl(AB)** is the result obtained for **AB** product in a t-bit mantissa, then

$$\mathbf{F = fl(AB) - AB},$$
$$||\mathbf{F}|| \leq ||\mathbf{A}||||\mathbf{B}||2^{-t}N. \qquad (2.43)$$

In the above $||.||$ is the Euclidian norm or subordinate norm of the matrices. From the above discussion, it is clear that in order to minimize the equation round-off error one

should use large t (i.e., double precision) in the construction of the global matrix.

Using (2.40) with $a_1 > 0$, and $b_1 = -c_1$ with $c_1 > 0$, the relative error in the computed quantity $fl(a_1-c_1)$ is

$$\frac{|r_r|}{|fl(a_1-c_1)|} \leq 2^{-t} \frac{|a_1|+|c_1|}{|fl(a_1-c_1)|} . \qquad (2.44)$$

If $|a_1-c_1|$ is sufficiently small, the round-off error overwhelms the computed quantity $fl(a_1-c_1)$. This type of difference occurs, for example, in computing the nodal coordinates of element vertices. For a given mantissa length t, there is a limit to the mesh refinement beyond which the computed quantities may be 100% erroneous.

Usually the equation round-off error, r_r, may be kept negligibly small by using sufficiently large t with reasonably sized finite elements.

2.3.2. Transformation of equation errors into solution errors

Because of the equation errors discussed above we in fact solve

$$(\mathbf{A}+\delta\mathbf{A})\mathbf{c}'=\mathbf{f}+\delta\mathbf{f} , \qquad (2.45)$$

instead of

$$\mathbf{Ac} = \mathbf{f}, \qquad (2.46)$$

where $\mathbf{c}'= \mathbf{c} + \mathbf{e}$ is the solution vector and $\mathbf{e}$ is called solution error. The solution error may be considered as the

transformation of the equation errors by the system response. Substituting (2.45) into (2.46) and solving for **e** leads to

$$\mathbf{e} = (\mathbf{A}+\delta\mathbf{A})^{-1}(\delta\mathbf{f}-\delta\mathbf{A}\mathbf{c}) \ , \qquad (2.47)$$

which shows how the equation errors produce errors in the solution. The errors defined are the absolute errors, whereas one usually wants to have a true measure of the errors and desires to express the error in terms of relative magnitude. By using (2.45) and (2.46), the following can be obtained[30]

$$\frac{||\mathbf{e}||}{||\mathbf{c}||} \leq ||\mathbf{A}||\,||\mathbf{A}^{-1}||\left[\frac{||\delta\mathbf{f}||}{||\mathbf{f}||} + \alpha\frac{||\delta\mathbf{A}||}{||\mathbf{A}||}\right] , \qquad (2.48)$$

where

$\alpha = (1-||\mathbf{A}||\,||\mathbf{A}^{-1}||\,||\delta\mathbf{A}||/||\mathbf{A}||)^{-1}$.

$\frac{||\mathbf{e}||}{||\mathbf{c}||}$ = the norm of the relative errors in the system response.

$\frac{||\delta\mathbf{f}||}{||\mathbf{f}||}, \frac{||\delta\mathbf{A}||}{||\mathbf{A}||}$ = the norms of the relative errors in the equation

$||\mathbf{A}||\,||\mathbf{A}^{-1}||$ = condition number of the system.

The last relationship defines the **condition number** of the system (or simply of the matrix A). Since the magnitude of the condition number is directly proportional to the bound on the relative error (Eq.(2.48)), it is usually regarded as an

important measure of the difficulty of obtaining an accurate numerical solution of the system equation. Using Euclidian norm for the matrices it can be shown that[30]

$$||\mathbf{A}||\,||\mathbf{A}^{-1}|| = L_1/L_n \ , \tag{2.49}$$

where L_1 is the largest, and L_n is the smallest eigenvalue of the system matrix $\mathbf{A}$. When (2.49) is a large number, we say that significant errors in the solution are more likely. The behavior of the condition number will be investigated for various cases in the next chapter.

2.3.3. Manipulation Errors

So far, we discussed the equation errors and the transformation of these errors into solution errors. During the solution of discrete equations by means of some solution algorithm additional errors are incorporated into the solution. We define these additional errors as manipulation errors.

Two sources of the manipulation errors are the finiteness of the computer word length and the solution algorithms. The first causes further round-off errors and the second may cause alteration of the growth patterns of the equation error. Let $\mathbf{g}$ denote the manipulation errors, we define

$$\mathbf{g} = \mathbf{c}_c - \mathbf{c}' \ , \tag{2.50}$$

where $\mathbf{c}_c$ is the computed response and $\mathbf{c}'$ is the solution of

(2.45). Let $ is the exact solution of the mathematical model the total solution error is

$$g + e = \$ - c_c \ . \tag{2.51}$$

The solution algorithms for equation (2.45) may be divided into two classes: Direct methods and Indirect methods. Consider first the Gauss elimination method as an example of a direct method. In this method the **A** matrix is decomposed into lower and upper triangular matrices

$$\mathbf{A} + \delta\mathbf{A} = \mathbf{LU} \ , \tag{2.52}$$

and (2.45) takes the following form

$$\mathbf{LUc'} = \mathbf{f} + \delta\mathbf{f} \ . \tag{2.53}$$

This is called the decomposition part of the algorithm. The forward pass part of the algorithm forms

$$\mathbf{Lz} = \mathbf{f} + \delta\mathbf{f} \ . \tag{2.54}$$

Finally, in the backward pass part

$$\mathbf{Uc'} = \mathbf{z} \ . \tag{2.55}$$

However, in practice round-off errors are introduced into the computed quantities and instead of (2.52), (2.54), (2.55), and (2.45) we have

$$\mathbf{A} + \delta\mathbf{A} = \mathbf{LU} - \blacktriangle\mathbf{A} \ , \tag{2.56}$$

$$\mathbf{Lz} = (\mathbf{f} + \delta\mathbf{f}) + \mathbf{b} \ , \tag{2.57}$$

$$\mathbf{Uc}_c = \mathbf{z} - \mathbf{h} \ , \tag{2.58}$$

and

$$[(\mathbf{A} + \delta\mathbf{A}) + \blacktriangle\mathbf{A}]\mathbf{c}_c = (\mathbf{f} + \delta\mathbf{f}) + \blacktriangle\mathbf{f} \ , \qquad (2.59)$$

where

$$\blacktriangle\mathbf{f} = \mathbf{b} + \mathbf{L}\mathbf{h} \ ,$$

and $\blacktriangle\mathbf{A}$, $\mathbf{b}$, $\mathbf{h}$ are the round-off errors associated with decomposition, forward pass, and backward pass, respectively. The $\blacktriangle\mathbf{A}$ and $\blacktriangle\mathbf{f}$ in the solution are the result of the manipulation errors.

Among the indirect methods, Gauss-Seidel iteration (with over relaxation and under relaxation) is often used. In this method, (2.45) is first scaled with $\mathbf{diag}[(a_{ii} + \delta a_{ii})^{-1}]$ such that

$$\mathrm{diag}[(a_{ii} + \delta a_{ii})^{-1}](\mathbf{A} + \delta\mathbf{A}) = -\mathbf{L} + \mathbf{I} - \mathbf{U} \ , \qquad (2.60)$$

and

$$\mathrm{diag}[(a_{ii} + \delta a_{ii})^{-1}](\mathbf{f} + \delta\mathbf{f}) = \mathbf{q} \ , \qquad (2.61)$$

so that (2.45) can be rewritten as

$$\mathbf{c}' = \mathbf{q} + \mathbf{L}\mathbf{c}' + \mathbf{U}\mathbf{c}', \qquad (2.62)$$

and the k+1 Gauss-Seidel iterate (without relaxation) is

$$\mathbf{c}'_{k+1} = \mathbf{q} + \mathbf{L}\mathbf{c}'_{k+1} + \mathbf{U}\mathbf{c}'_k \ . \quad \text{for } k=1,2,.. \qquad (2.63)$$

In practice, due to round-off errors during the manipulation, instead of (2.60), (2.61), (2.62), and (2.63), one has, respectively

$$\text{diag}[(a_{ii}+\delta a_{ii})^{-1}](\mathbf{A}+\delta\mathbf{A}) = -\mathbf{L}+\mathbf{I}-\mathbf{U} - \text{diag}[(a_{ii}+\delta a_{ii})^{-1}]\blacktriangle\mathbf{A}, \quad (2.64)$$

$$\text{diag}[(a_{ii}+\delta a_{ii})^{-1}](\mathbf{f}+\delta\mathbf{f}) = \mathbf{q}+\mathbf{d}, \quad (2.65)$$

$$\mathbf{c}_c = \mathbf{q} + \mathbf{d} + \mathbf{L}\mathbf{c}_c + \mathbf{U}\mathbf{c}_c - \text{diag}[(a_{ii}+\delta a_{ii})^{-1}]\blacktriangle\mathbf{A}\mathbf{c}_c , \quad (2.66)$$

and

$$\mathbf{c}_{c,k+1} = \mathbf{q} + \mathbf{d} + \mathbf{L}\mathbf{c}_{c,k+1} + \mathbf{U}\mathbf{c}_{c,k} + \mathbf{d}_k. \quad \text{for } k=1,2,.. \quad (2.67)$$

After the iteration process has converged one has instead of (2.45)

$$[(\mathbf{A}+\delta\mathbf{A}) + \blacktriangle\mathbf{A}]\mathbf{c}_c = (\mathbf{f}+\delta\mathbf{f}) + \blacktriangle\mathbf{f} , \quad (2.68)$$

where now

$$\blacktriangle\mathbf{f} = \text{diag}[(a_{ii}+\delta a_{ii})^{-1}](\mathbf{d} + \sum_k \mathbf{d}_k). \quad (2.69)$$

The approximate solution obtained for (2.45) by a computer is essentially the solution of (2.59) or (2.68). Thus, the knowledge of $\blacktriangle\mathbf{A}$ and $\blacktriangle\mathbf{f}$ could enable one to obtain the solution $\mathbf{c}'$ of (2.45). The computation of $\blacktriangle\mathbf{A}$ and $\blacktriangle\mathbf{f}$ is a very difficult process and usually no attempt is made to compute their actual values. In general the magnitudes of $\blacktriangle\mathbf{A}$ and $\blacktriangle\mathbf{f}$ are smaller when $(\mathbf{A}+\delta\mathbf{A})$ has a small condition number.

CHAPTER 3

In this chapter we consider the solution of time-independent neutron diffusion problems and the condition number of the associated matrices for several problems encountered in the utilization of the finite element method to solve the neutron diffusion equation.

In Section 3.1 we will first review the weak formulation for the application of diffusion problems. In Section 3.2, the choice of the trial functions is presented. Section 3.3 is concerned with the derivation of the reduced equations. In Section 3.4, the numerical methods for the solution of system of equations are discussed. We close the chapter by presenting the results for several numerical experiments in Section 3.5

3.1. THE WEAK FORMULATION OF THE MULTI-GROUP EQUATIONS.

The general multi-group neutron diffusion equations predicting the neutron behavior throughout the reactor are

$$\nabla\cdot D_g(\mathbf{r})\nabla\Phi_g(\mathbf{r}) + \Sigma_{r,g}\Phi_g(\mathbf{r}) = \sum_{\substack{g'=1\\ g'\neq g}}^{G} \{\Sigma_{s,gg'}(\mathbf{r})\Phi_{g'}(\mathbf{r}) + (1/k)p_g\tau\Sigma_{f,g'}(\mathbf{r})\Phi_{g'}(\mathbf{r})\}, \quad (3.1)$$

with the boundary conditions given by (2.2) and (2.3). Here,

$\Sigma_{s,gg'}(\mathbf{r})$= Macroscopic scattering cross-section from group g' to group g at location $\mathbf{r}$.

$\Sigma_{f,g'}(\mathbf{r})$ = Macroscopic fission cross-section in group g' at $\mathbf{r}$.

τ = Average number of neutrons produced from a fission event.

p_g= The fraction of neutrons falling into energy group g.

and the other terms were defined in Chapter 2. To simplify the appearance of Eq.(3.1), we define

$$q_g(\mathbf{r}) = \sum_{\substack{g'=1\\ g'\neq g}}^{G} \{\Sigma_{s,gg'}(\mathbf{r})\Phi_{g'}(\mathbf{r}) + \frac{1}{k}p_g\tau\Sigma_{f,g'}(\mathbf{r})\Phi_{g'}(\mathbf{r})\}, \quad (3.2)$$

$$A_g = \nabla\cdot D_g(\mathbf{r})\nabla + \Sigma_{r,g}, \quad (3.3)$$

so that (3.1) becomes

$$A_g\Phi_g(\mathbf{r}) = q_g(\mathbf{r}), \quad (3.4)$$

for each of "G" groups. The weak form of (3.4) is

$$(A_g\Phi_g, w_g) = (q_g, w_g), \quad (3.5)$$

where (,) indicates the inner product operation defined by

Eq.(2.12). Alternatively, (3.4) and (3.5) can be rewritten in more general forms

$$\mathbf{A\Phi(r) = q(r)} \tag{3.6}$$

$$\mathbf{(A\Phi,w) = (q,w)}, \tag{3.7}$$

where **A** is a diagonal matrix in the form

$$\mathbf{A} = \begin{bmatrix} A_1 & & & \\ & A_2 & & \\ & & . & \\ & & & A_G \end{bmatrix}$$

and $\mathbf{\Phi}$, **q**, and **w** are column vectors

$$\mathbf{\Phi} = [\ \Phi_1 \quad \Phi_2 \quad . \quad . \quad \Phi_G\ ]^T$$

$$\mathbf{q} = [\ q_1 \quad q_2 \quad . \quad . \quad q_G\ ]^T$$

$$\mathbf{w} = [\ w_1 \quad w_2 \quad . \quad . \quad w_G\ ]^T$$

We consider the reactor core as a bounded region, D, in the two dimensional Euclidian space E_2. The region D consists of a finite number of subregions D_e, within each subregion the material properties do not vary and the boundary, δD, of core region D is piecewise smooth.

Definition[11] 1: We say that a function $\Phi(\mathbf{r})$ is Hölder continuous in the set **D** with modulus K and exponent α ($K > 0$, $0 < \alpha < 1$) if at any two points $\mathbf{r'}$ and $\mathbf{r''}$ in **D**

$$|\Phi(\mathbf{r'}) - \Phi(\mathbf{r''})| \leq K\ |\mathbf{r'} - \mathbf{r''}|^{\alpha},$$

where $|\mathbf{r}''-\mathbf{r}'|$ is the Euclidian distance in E_2.

Definition 2: $C_{l,\alpha}(\mathbf{D})$ is the Banach space of all functions Φ that are "l" times continuously differentiable in **D** and whose "l" th derivatives are Hölder continuous in **D** with exponent α, and for which the value of

$$|\Phi|_{l,\alpha,D}=\sum_{k=0}^{l-1}\sum_{(k)}\max_{\mathbf{r}\epsilon\mathbf{D}}|\Phi^{(k)}(\mathbf{r})|+\sum_{(l)}\sup_{\mathbf{r}\epsilon\mathbf{D}}\frac{|\Phi^{(l)}(\mathbf{r}')-\Phi^{(l)}(\mathbf{r}'')|}{|\mathbf{r}'-\mathbf{r}''|^{\alpha}}$$

is finite. In this definition, $\Phi^{(k)}$ denotes an arbitrary derivative of order k of Φ with respect to **r**, and $\Sigma_{(k)}$ denotes summation over all possible derivatives of order k. The quantity $|\Phi|_{l,\alpha,D}$ defines a norm in $C_{l,\alpha}(\mathbf{D})$. $C_{l,\alpha}(D)$ is the set of functions belonging to $C_{l,\alpha}(\mathbf{D})$ for all $D'\subset D$.

Equation (3.6) is an elliptic boundary value problem and its classical solution is available(i.e., the solution exists and it is at least twice continuously differentiable). With the above definitions we try to relax some of the restrictions on the problem so that we can construct the weak form of the equation. The following theorem, given without proof, utilizes the above definitions and states that the solution is still of the classical type[11,31].

Theorem 3.1: The boundary value problem (3.6) has a unique generalized solution, $\Phi(\mathbf{r})$, in Hilbert space; $\Phi(\mathbf{r})$ belongs to the class $C_{o,\alpha}(\mathbf{D})$ and $C_{2,\alpha}(D_e)$, where e=1,...,E. Moreover,

$D(\mathbf{r})(\delta\Phi/\delta r)$ exists and is Hölder continuous in each D_e up to the boundary δD_e, except possibly at the vertices on δD_e.

Making the obvious generalization of one-group results of (2.36) in Chapter 2 we can write

$$(A\Phi,w) = B(\Phi,w),$$

where now

$$A = -\nabla.D(r)\nabla + \Sigma_r(r),$$

$$B(\Phi,w) = \int_D \{ D(r)\nabla\Phi(r)\nabla w(r)^T + \Sigma_r(r)\Phi(r)w(r)^T \}dr$$

$$+(1/2)\int \Phi(r)w(r)^T ds .$$

Hence, we can define our problem in terms of the bilinear form

$$B(\Phi,w) = (q,w). \qquad (3.8)$$

Therefore the generalized solution of (3.6) can be stated by (3.8). However, (3.8) has functions, Φ, having first derivatives only, and its coefficients and its inhomogeneous part are only piecewise continuous.

The following lemma states some of the properties of $B(\Phi,w)$.

Lemma 3.1: If the coefficients of the boundary value problem (3.6) are positive definite and piecewise continuous, then the

bilinear form $B(\Phi,w)$ defined by (3.8) has the following properties

(i) $B(\Phi,w)$ is a bilinear symmetric functional.

(ii) There exists a constant C such that

$$|B(\Phi,w)| \leq C\|\Phi\| \ \|w\|$$

(iii) The bilinear form $B(\Phi,w)$ is positive definite, meaning that there exists a positive constant τ such that

$$B(\Phi,\Phi) \geq \tau\|\Phi\|^2 \ .$$

In the Galerkin procedure, the region of interest D_M is considered to be the subspace of the Hilbert space. We divide the region in a way that the partitioning occurs along the material interfaces. This preserves the property of piecewise continuity.

In (3.8), Φ is considered to be the approximate solution and represented by

$$\bar{\Phi}_g(\mathbf{r}) = \sum_{i=1}^{M} c_{g,i}u_{g,i}(\mathbf{r}) \ , \qquad (3.9)$$

Here

$$\Phi(\mathbf{r}) = [\ \bar{\Phi}_1, \ \bar{\Phi}_2, .., \ \bar{\Phi}_g\ ].$$

The basis functions, w, are represented by

$$w_{g,i} = u_{g,i} \quad 1 \leq i \leq M, \ 1 \leq g \leq G \qquad (3.10)$$

The expansion coefficients can be determined by applying the Galerkin scheme to the weak form of the multi-group diffusion equation (3.5)

$$B(\bar{\Phi}_g, u_{g,i}) = (q_g, u_{g,i}) \quad 1 \le i \le M,\ 1 \le g \le G. \quad (3.11)$$

This procedure leads to M*G linear algebraic equations for the coefficients $c_{g,i}$, where G is the number of energy groups and M is the number of nodes in the system.

Theorem 3.2: The problem (3.5) has a unique solution Φ_g in Hilbert space. For any finite dimensional subspace D_M of Hilbert space, the approximate problem (3.11) has a unique solution Φ_g in D_M. The following estimate holds for the error of the approximate solution in the Sobolov norm:

$$\|\bar{\Phi}_g - \Phi_g\| \le \tau \|b_g\| \operatorname*{Inf}_{s \in D_M} \|s - \Phi_g\|,$$

where b is a function in Hilbert space such that

$$(q_g, u_g) = \langle b_g, u_g \rangle ,$$

$$\langle b_g, u_g \rangle = \int_D (b_g u_g + \frac{\delta b_g}{\delta \mathbf{r}} \frac{\delta u_g}{\delta \mathbf{r}}) d\mathbf{r},$$

and s is an arbitrary function in D_M.

Furthermore, let $\{D_M\}_{M=1}^{\infty}$ be any sequence of finite dimensional subspaces in Hilbert space for which

$$\lim_{M \to \infty} \{ \operatorname*{Inf}_{s \in D_M} \|s - \Phi_g\| \} = 0,$$

where Φ_g is the generalized solution in Hilbert space. Then, if $\Phi_{M,g}$ is the approximate solution in D_M,

$$\lim_{M \to \infty} \|\bar{\Phi}_{M,g} - \Phi_g\| = 0.$$

3.2. DERIVATION OF THE TRIAL FUNCTIONS

To apply the finite element method to practical reactor physics problems, we divide the spatial domain in the reactor into E contiguous linear triangular elements. The linear triangular element is a subdomain with three nodes at the apices. That is, the neutron flux is represented by a linear polynomial with three coefficients. To derive the trial functions, the following conditions are imposed:

1. The neutron flux is assumed to be continuous across element boundaries, and no derivative values of the neutron flux are used to determine the polynomials. These types of polynomials are called Lagrangian types.
2. The three trial functions, defined within the element will have the magnitude of unity at one apex of the triangle and zero at the others.
3. The trial functions vanish outside of the triangular element.

The neutron flux in (3.9) is represented as the summation over nodes and neutron energy groups. In the following derivation the energy dependency will be omitted. Furthermore we will consider the flux only within an element e instead of the summation over nodes within the whole region. Within e the flux in a particular group has the form

$$\overline{\Phi}_e(x,y) = a_e x + b_e y + d_e \ , \qquad (3.12)$$

and we require that

$$\bar{\Phi}_e(x_l, y_l) = c_l \tag{3.13}$$

at node 1. A consistent node numbering is a must and the vertices of the element A_e are represented in counter clockwise order by (x_i, y_i), (x_j, y_j), and (x_k, y_k) as shown in Figure (3.1). The use of condition (3.13) in (3.12) leads to

$$\begin{aligned} c_i &= a_e x_i + b_e y_i + d_e \\ c_j &= a_e x_j + b_e y_j + d_e \\ c_k &= a_e x_k + b_e y_k + d_e \ . \end{aligned} \tag{3.14}$$

From (3.14), the coefficients a_e, b_e, d_e can be determined

$$a_e = (2A_e)^{-1}*[(y_j - y_k)c_i + (y_k - y_i)c_j + (y_i - y_j)c_k] \ , \tag{3.15}$$

$$b_e = (2A_e)^{-1}*[(x_k - x_j)c_i + (x_i - x_k)c_j + (x_j - x_i)c_k] \ , \tag{3.16}$$

$$d_e = (2A_e)^{-1}*[(x_j y_k - x_k y_j)c_i + (x_k y_i - x_i y_k)c_j + (x_i y_j - x_j y_i)c_k] \ , \tag{3.17}$$

where the area of triangle, A_e, is

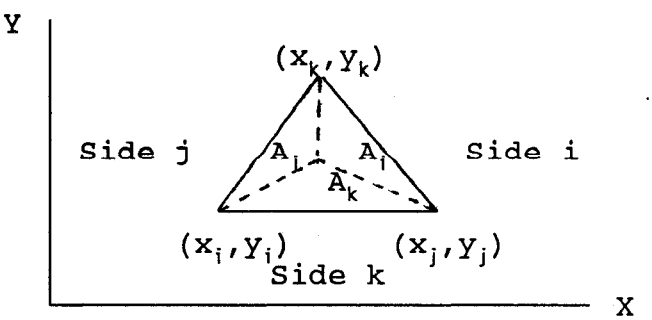

Figure (3.1). Linear Triangular Element

$$A_e = (1/2)\ \det \begin{bmatrix} 1 & x_i & y_i \\ 1 & x_j & y_j \\ 1 & x_k & y_k \end{bmatrix}$$

Substituting Eqs.(3.15) through (3.17) into (3.12) we obtain

$$\Phi_e(x,y) = u_{i,e}(x,y)c_i + u_{j,e}(x,y)c_j + u_{k,e}(x,y)c_k \ , \qquad (3.18)$$

where the expression for trial function "i" is given by

$$u_{i,e}(x,y) = [A_{i,e}x + B_{i,e}y + D_{i,e}]*(2A_e)^{-1} \ , \qquad (3.19)$$

with

$$A_{i,e} = (y_j - y_k)$$
$$B_{i,e} = (x_k - x_j)$$
$$D_{i,e} = (x_j y_k - x_k y_j) \ .$$

The expressions for $u_{j,e}$ and $u_{k,e}$ are obtained by rotating i,j, and k. Note that the trial functions $u_l(x,y)$ become unity at the corresponding apex and zero at other apices.

The equation (3.18) can also be derived using the area coordinates of the triangle. The area coordinate of a point lying within a triangle can be best explained by Figure (3.1) and the following definitions

$$S_i = \frac{A_i}{A_e} \ ; \qquad S_j = \frac{A_j}{A_e} \ ; \qquad S_k = \frac{A_k}{A_e} \ .$$

Since $A_i + A_j + A_k = A_e$, therefore $S_i + S_j + S_k = 1$. It can be shown that $S_i = u_i$, $S_j = u_j$, and $S_k = u_k$.

Integration formulas can be developed by utilizing the area coordinates and they simplify the evaluation of the matrices. The following equality is useful in the derivation of integrals in the following section[32]

$$\int_{A_e} S_i^p S_j^q S_k^r = \frac{p!q!r!}{(p+q+r+2)!} 2A_e \ . \qquad (3.20)$$

3.3. DERIVATION OF THE SYSTEM OF EQUATIONS

Once the choice of the trial functions and the type of the elements are made, next step is to derive the element matrices for the g th. group integral equation. To be consistent with the definitions made in the previous sections, the system properties are assumed to be homogeneous within the triangular elements. The open form of (3.11) in the element e is

$$\{D_{g,e}\int_{A_e}\left[\frac{\delta \mathbf{u}}{\delta x}\frac{\delta \mathbf{u}^T}{\delta x}+\frac{\delta \mathbf{u}}{\delta y}\frac{\delta \mathbf{u}^T}{\delta y}\right]dA+\Sigma_{r,g,e}\int_{A_e}\mathbf{u}\mathbf{u}^T dA+\frac{1}{2}\int_{S_{v,e}}\mathbf{u}\mathbf{u}^T dS\}\mathbf{c}_{g,e}$$

$$=\int_{A_e}\mathbf{u}q_{g,e}dA \ . \qquad (3.21)$$

In (3.21) A_e indicates an integration over the area of the

triangular element and $S_{v,e}$ indicates a line integration along the sides of the element upon which the vacuum boundary condition is imposed. The line integral is carried out only if the element border is on a vacuum boundary. Although $\mathbf{u}_g(x,y)$ in (3.11) was selected to be energy dependent, one usually chooses trial functions to be independent of energy. Since we have followed this convention, the subscript g has been dropped from the **u** vector in equation (3.21).

u is a vector with three coefficients and the integrations in (3.21) are basically matrix inner product operations. The first term in the first integral is

$$\mathbf{M}_{g,e,1} = \int \{ D_{g,e} \begin{bmatrix} \frac{\delta u_i}{\delta x} \\ \frac{\delta u_j}{\delta x} \\ \frac{\delta u_k}{\delta x} \end{bmatrix} \begin{bmatrix} \frac{\delta u_i}{\delta x} & \frac{\delta u_j}{\delta x} & \frac{\delta u_k}{\delta x} \end{bmatrix} \} dA \ ,$$

and the result of this integration is

$$\mathbf{M}_{g,e,1} = \frac{D_{g,e}}{4A_e} \begin{bmatrix} a_i^2 & a_i a_j & a_i a_k \\ a_i a_j & a_j^2 & a_j a_k \\ a_i a_k & a_j a_k & a_k^2 \end{bmatrix} \ ,$$

Similarly the second term in the first integral is

$$\mathbf{M}_{g,e,2} = \frac{D_{g,e}}{4A_e} \begin{bmatrix} b_i^2 & b_i b_j & b_i b_k \\ b_i b_j & b_j^2 & b_j b_k \\ b_i b_k & b_j b_k & b_k^2 \end{bmatrix}.$$

The second integral in (3.21) can be evaluated by using (3.20). The result is

$$\mathbf{H}_{g,e} = \frac{\Sigma_{r,g,e} * A_e}{12} \begin{bmatrix} 2 & 1 & 1 \\ 1 & 2 & 1 \\ 1 & 1 & 2 \end{bmatrix}.$$

In the above derivation we assume that the node numbering of the triangular element is counter clockwise, that is, i is the first node, j is the second, k is the third node of triangular element as shown in Figure (3.1).

The last integral on the left hand side of (3.21) is the boundary integral. This integral is performed only if the side of the triangular element coincides with a non-reentrant vacuum boundary. This requires us to assign labels to the sides of triangular elements. The lines ij, jk, and ki will be indicated as the "Bottom = B", "Right = R", and "Left = L" sides, respectively. Using this notation, the result of the line integrations for a B, R, and L side along a vacuum boundary are respectively,

$$\mathbf{B}_e = \frac{l_B}{12}\begin{bmatrix} 2 & 1 & 0 \\ 1 & 2 & 0 \\ 0 & 0 & 0 \end{bmatrix},$$

$$\mathbf{R}_e = \frac{l_R}{12}\begin{bmatrix} 0 & 0 & 0 \\ 0 & 2 & 1 \\ 0 & 1 & 2 \end{bmatrix},$$

$$\mathbf{L}_e = \frac{l_L}{12}\begin{bmatrix} 2 & 0 & 1 \\ 0 & 0 & 0 \\ 1 & 0 & 2 \end{bmatrix},$$

where the l's are the lengths of the related sides. The reflective boundary condition (2.3) was not considered in the development of (3.21) because it is a Neumann type boundary condition with zero on the right hand side. Integration along a reflective boundary does not make any contribution to the equation and is therefore omitted. In most neutron diffusion problems instead of the mixed type boundary condition of the kind (2.2), the Dirichlet type (or essential) boundary condition is considered

$$\Phi_b = V(\mathbf{r}) \ ,$$

where $\mathbf{r}$ is the coordinate on the boundary and V is prescribed function of $\mathbf{r}$. This type of boundary condition is taken into consideration during the numerical calculation. In this study,

the Penalty method is utilized to include the Dirichlet type boundary conditions. In the Penalty method, assuming that V_i is the known value of the flux at node i, the diagonal term of the i th. row of the LHS matrix is multiplied by a large number, K, in the order of 10^8 - 10^{20} and the right hand side vector is multiplied by KV_i. The following shows this manipulation

$$m_{1j} + m_{2j} + \ldots + Km_{ii} + \ldots + m_{Nj} = KV_i q_i \ .$$

The integral on the RHS of (3.21) involves the derivation of source term. There are two approaches to evaluate the integration implied by the (,) operation: the lump and consistent approximations. The lump approximation is that in the calculation of an element of a source vector, the g' group flux in the integral is assumed to be equal to the nodal value of corresponding element. An element of the matrix is evaluated by the following operation

$$(u, \Phi_{g'})_i = \Phi_{g',i} \int_{A_e} u_i(x,y)\, dA \ ,$$

which yields a diagonal matrix. In the consistent approximation the trial functions of the g' group flux are assumed to have the same form as the group flux

$$\overline{\Phi}_{g'}(x,y) = \mathbf{u}^T(x,y)\, \mathbf{c}_{g',e} \ .$$

The lumped approximation was utilized in reference [33] and both approximations were compared in [18]. The result of comparison suggests that the consistent approximation is superior to the lump approximation. In this work, the consistent approximation was utilized in the calculation of the source term. The right hand side of (3.11) with the utilization of (3.20) becomes

$$(\mathbf{q}_{g',e},\mathbf{u}) = \sum_{\substack{g'=1 \\ g'\neq g}}^{G} \{ \mathbf{S}_{gg',e} + \frac{p_g}{k} \mathbf{F}_{f,g',e} \} \mathbf{c}_{g',e} , \qquad (3.22)$$

where

$$\mathbf{S}_{gg',e} = \frac{\Sigma_{gg',e} A_e}{12} \begin{bmatrix} 2 & 1 & 1 \\ 1 & 2 & 1 \\ 1 & 1 & 2 \end{bmatrix} ,$$

$$\mathbf{F}_{gg',e} = \frac{\Sigma_{f,g',e} A_e}{12} \begin{bmatrix} 2 & 1 & 1 \\ 1 & 2 & 1 \\ 1 & 1 & 2 \end{bmatrix} ,$$

and

$$\mathbf{c}_{g',e} = [\, c_i \quad c_j \quad c_k \,]_{g'}^{T} .$$

Finally, within the e th. triangular element, the static neutron diffusion equation for the g th. group assumes the following form

$$[\ \mathbf{M}_{g,e,1} + \mathbf{M}_{g,e,2} + \mathbf{H}_{g,e} + \delta_{B,e}\mathbf{B}_e + \delta_{R,e}\mathbf{R}_e + \delta_{L,e}\mathbf{L}_e\ ]\mathbf{c}_{g,e}$$

$$= \sum_{\substack{g'=1 \\ g' \neq g}}^{G} \{\ \mathbf{S}_{gg',e} + \frac{p_g}{k}\mathbf{F}_{f,g',e}\ \}\mathbf{c}_{g',e}\ , \qquad (3.23)$$

where

$$\delta_{j,e} = \begin{cases} 1 & \text{if the j th. side of the triangle facing the vacuum boundary.} \\ 0 & \text{otherwise.} \end{cases}$$

The above element matrices should be assembled together so that the final global matrices can suitably represent the physical problem. To construct a global matrix from the element matrices a relationship should be established between row and column locations of the coefficients of the element matrix and the corresponding locations of these coefficients in the global matrix. Figure (3.2) and Table (3.1) illustrates the relationship used in this work. In Figure (3.2), the numbers in brackets represent the element numbers, the numbers in parentheses are written for the first element only and they represent the local numbers of the element. The other numbers are the global node numbers. In Table (3.1), the first row is the element numbers, the first column on the left of the vertical line is the row and column locations in the elements. The columns on the RHS of the vertical line are the corresponding row and column locations in the global matrix. From Table (3.1) and Figure (3.2) one can see that the global location of the coefficient m_{23} in the first element is M_{25} in

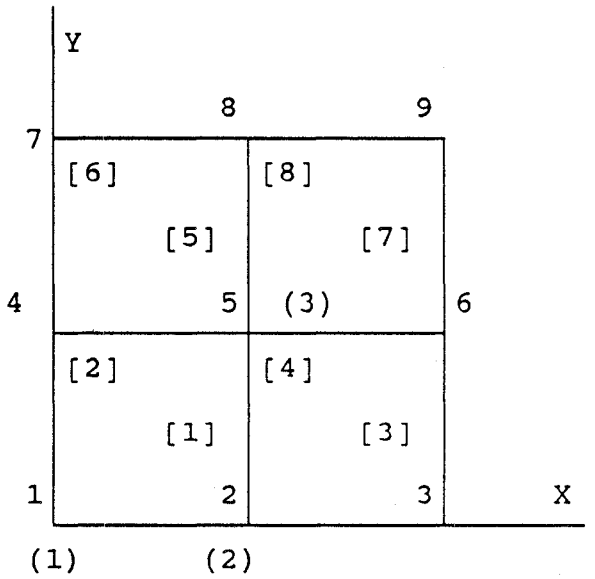

Figure (3.2) Partition of the domain and node numbering

Element # →		1	2	3	4	5	6	7	8
Local Element Node #	1	1	1	2	2	4	4	5	5
	2	2	5	3	6	5	8	6	9
	3	5	4	6	5	8	7	9	8

Table (3.1) Global Nodal Numbers as a function of Element Number and Local Element Node Numbers.

the global matrix.

The above calculations are formulated by the mesh generation subroutine and stored in arrays to use in the construction of global matrices. Once the assembling is carried out, the next step is to solve the following system of equations[8]

$$(A - S - \frac{1}{k}PF^T)[c]=[0] \ , \qquad (3.24)$$

where

$$A = \begin{bmatrix} [A_1] & & & \\ & [A_2] & & \\ & & \ddots & \\ & & & [A_G] \end{bmatrix} \ , \qquad F^T = [\ [F_1]\ [F_2] \ldots [F_G]\ ] \ ,$$

$$S = \begin{bmatrix} [0] & \ldots\ldots & & \\ [S_{21}] & [0] & & \\ \vdots & & \ddots & \\ [S_{G1}] & \ldots [S_{G(G-1)}] & [0] & \end{bmatrix} \ , \qquad P = \begin{bmatrix} [P_1] \\ \vdots \\ [P_G] \end{bmatrix} \ ,$$

$$[c]^T = [\ [c_1], [c_2], \ldots, [c_G]\ ]$$

$$[A_g] = [\ (A_g)_{ij}\ ]$$

$$[F_g] = [\ (F_g)_{ij}\ ]$$

$$[S_{gg'}] = [\ (S_{gg'})_{ij}\ ]$$

$$[P_g] = [\ (p_g)_{ij}\ ] \ ,$$

where

$$(A_g)_{ij} = \sum_{(e)} (D_{g,e}(u,u)_{ij} + \Sigma_{rg,e}(u,u)_{ij})_{(e)}$$

$$(S_{gg'})_{ij} = \sum_{(e)} (\Sigma_{gg',e}(u,u)_{ij})_{(e)}$$

$$(F_g)_{ij} = \sum_{(e)} (\Sigma_{f,g,e}(u,u)_{ij})_{(e)}$$

$$[c_g] = [c_{g,1}, c_{g,2}, \ldots, c_{g,N}] .$$

In the foregoing g is a group index, i and j are respectively the row and column indices.

3.4. NUMERICAL SOLUTION METHODS

In this section, the solution methods for (3.24) will be presented. In general (3.24) is an eigenvalue problem which is solved using a within group solution combined with an outer iteration procedure. To illustrate the methods we consider the two group version of equation (3.24)

$$\mathbf{A}_1\mathbf{c}_1 = \frac{1}{k}\mathbf{f}_1 \qquad (3.25a)$$

$$\mathbf{A}_2\mathbf{c}_2 = \mathbf{S}_{21}\mathbf{c}_1 + \frac{1}{k}\mathbf{f}_2 , \qquad (3.25b)$$

where

$$\mathbf{f}_g = \mathbf{P}_g(\mathbf{F}_1\mathbf{c}_1 + \mathbf{F}_2\mathbf{c}_2) \qquad g=1,2.$$

The within group solution is carried out using either the

Successive Over Relaxation method or the Gaussian Elimination method. The outer iteration is accomplished by the power method which is a conventional computational method. Each of these methods is discussed briefly in the following sections.

Forward power iteration

The forward power iteration is a classic method to determine the eigenvectors corresponding to the largest eigenvalue.

The starting procedure of the iteration is to assume k=1 in (3.25a) and (3.25b), and to choose starting iteration vectors, $\mathbf{x}_{g,0}$, for g=1,2, respectively. Then, it can be shown that as the iteration index, m, approaches infinity

$$\mathbf{x}_{g,m+1} \rightarrow \mathbf{c}_g$$

Having chosen starting iteration vectors, $\mathbf{x}_{g,0}$, one first evaluates

$$\mathbf{f}_{g,0} = \mathbf{P}_g(\mathbf{F}_1\mathbf{x}_{1,0} + \mathbf{F}_2\mathbf{x}_{2,0}) \qquad \text{for } g=1,2. \tag{3.26}$$

The subsequent iteration procedure consists of the following steps:

1. The LHS of the group equations are evaluated to obtain $\mathbf{x}_{i,m+1}$

$$\mathbf{A}_1\mathbf{x}_{1,m+1} = \mathbf{f}_{1,m} \tag{3.27a}$$

$$\mathbf{A}_2\mathbf{x}_{2,m+1} = \mathbf{S}_{21}\mathbf{x}_{1,m+1} + \mathbf{f}_{2,m} \tag{3.27b}$$

$$c_g = \frac{x_{g,l+1}}{(x^T_{g,l} f_{g,l})^{1/2}} \quad . \tag{3.31}$$

In (3.29), W is the weighting vector. There are several options for the choice of weighting vector:

1. The weighting vector can be selected as $W = (f_{1,m+1} + f_{2,m+1})$.
2. $W = x_{1,m+1}$.
3. W can be selected to be a vector with unity elements.

In this study the third option was used since it helps to reduce the number of operations.

Successive Over Relaxation Method

The SOR method is applied by splitting up A_1 and A_2 matrices into lower, diagonal, and upper matrices. In particular, since the matrices A_g are symmetric, the upper matrices are the transposes of corresponding lower matrices.

$$A_g = L_g + D_g + L_g^T \qquad g=1,2 \tag{3.32}$$

Substitution of (3.32) into Eqs. (3.25a) and (3.25b) followed by some manipulation yields

$$c_1^{(n+1)} = M_1(w_1) c_1^{(n)} + G_1(w_1) f^{1(n)} \ , \tag{3.33a}$$

$$c_2^{(n+1)} = M_2(w_2) c_2^{(n)} + G_2(w_2) [f_2^{(n)} + S_{21} c_1^{(n+1)}] \ , \tag{3.33b}$$

where n is the iteration index,

$$M_g(w_g) = (D_g + w_g L_g)^{-1}[(1-w_g)D_g - w_g L_g^T] ,$$

$$G_g(w_g) = w_g(D_g + w_g L_g)^{-1} \qquad g=1,2 ,$$

and the w_g are the optimum acceleration parameters of the related groups. The relationship between w_g and the spectral radius, $r(J_g)$ of the Jacobian matrix, J_g, is

$$w_g = \frac{2}{1 + (1-r^2(J_g))^{1/2}} ,$$

where Jacobian matrix is given by

$$J_g = - D_g^{-1}(L_g + L_g^T) .$$

If we define the error vector as

$$e_g^{(n+1)} = c_g - c_g^{(n+1)} , \tag{3.34}$$

then it can be shown that

$$e_g^{(n+1)} = M(w_g) e_g^{(n)}$$

or that

$$e_g^{(n+1)} = [M(w_g)]^{(n)} e_g(0) . \tag{3.35}$$

From numerical analysis it is known that when the spectral radius of $M(w_g)$ is less than 1, then

$$\lim_{n\to\infty} [M(w_g)]^{(n)} = 0 ,$$

and we say that $M(w_g)$ is a convergent matrix. This condition in turn insures that

$$\lim_{n\to\infty} e_g^{(n)} = 0. \tag{3.36}$$

Multiplying (3.33a) and (3.33b) by $D_g^{-1}(D_g + w_g L_g)$, g=1,2, and rearranging we find that

$$c_1^{(n+1)} = (1-w_1)c_1^{(n)} + w_1 D_1^{-1}[f_1^{(n)} - L_1 c_1^{(n+1)} - L_1^T c_1^{(n)}], \qquad (3.37a)$$

$$c_2^{(n+1)} = (1-w_2)c_2^{(n)} + w_2 D_2^{-1}\{ [f_2^{(n)} + S_{21} c_1^{(n+1)}] - L_2 c_2^{(n+1)} - L_2^T c_2^{(n)} \}. \qquad (3.37b)$$

In practice, w_g is usually calculated only for the first equation and the same value is also used for the second group calculation because the $\mathbf{A}_1$ and $\mathbf{A}_2$ matrices constitute the system matrix **A** they have the same convergence properties.

Gaussian elimination with static condensation[34]

Consider the following matrix equation

$$\mathbf{Ac} = \mathbf{f} \qquad (3.38)$$

In the static condensation procedure, the matrix **A** is partitioned in the form of

$$\begin{bmatrix} \mathbf{A}_{11} & \mathbf{A}_{12} \\ \mathbf{A}_{21} & \mathbf{A}_{22} \end{bmatrix} \begin{bmatrix} \mathbf{c}_1 \\ \mathbf{c}_2 \end{bmatrix} = \begin{bmatrix} \mathbf{f}_1 \\ \mathbf{f}_2 \end{bmatrix},$$

where $\mathbf{A}_{11}$, $\mathbf{A}_{12}$, $\mathbf{A}_{21}$, $\mathbf{A}_{22}$ are (1x1), (1x(N-1)), ((N-1)x1), and ((N-1)x(N-1)) matrices, respectively, and N is the number of equations. Then by using the Gaussian elimination method, the

original matrix equation can be reduced to (N-1) matrix equation of order N-1 having the form

$$\mathbf{A}^*\mathbf{c}^* = \mathbf{f}^* \ , \tag{3.39}$$

where

$$\mathbf{A}^* = \mathbf{A}_{22} - \mathbf{A}_{21}\mathbf{A}_{11}^{-1}\mathbf{A}_{12} \ ,$$

$$\mathbf{f}^* = \mathbf{f}_2 - \mathbf{A}_{21}\mathbf{A}_{11}^{-1}\mathbf{f}_1 \ .$$

This procedure can be repeated by partitioning $\mathbf{A}^*$ in the same way. When the matrix $\mathbf{A}^*$ is reduced to a (1x1) matrix, direct solution is possible for the last unknown c_n. Back substitution may now be applied for the remaining unknown c_i's until one finally determines $\mathbf{c}_1$ using the equation

$$\mathbf{c}_1 = \mathbf{A}_{11}^{-1}\mathbf{f}_1 - \mathbf{A}_{11}^{-1}\mathbf{A}_{12}\mathbf{c}_2$$

Because the group matrix $\mathbf{A}_g$ is symmetric, the number of operations is reduced approximately by half compared to the full matrix case. When a compact storage scheme is incorporated the number of operations can be further reduced.

3.5. NUMERICAL RESULTS

In this section, the programs developed, OPUS-SOR, (which utilizes successive over relaxation method), and OPUS-G, (Gauss elimination method with static condensation) are compared in terms of their computational efficiency. The test cases selected are well documented benchmark problems and

represent reliable tests for the proposed approximation methods in this field. Condition numbers for the test cases were evaluated by using the L_∞ norms of the matrices, hence, they may be considered as upper limit bounds rather than exact values. All of the calculations were performed on the VAX/VMS 3100 machine with double precision arithmetic. The results of each test case is presented in this section, except for the graphical comparison which are tabulated in Appendix C.

Test Case 3.1

The first reactor configuration is a homogeneous reactor with quarter reactor boundary conditions as shown in Figure (3.3). This problem is different from the other two group benchmark problems in a sense that there is a fission yield into the second group as well. The nuclear data for this problem is given in Appendix B under the title "Test Case 3.1". The exact k_{eff} value for this configuration is

$$k_{eff} = 1.465\ 387\ 387$$

Figure (3.4) through (3.6) illustrates how the triangulation was proceeded to evaluate the condition number of the matrices. Triangles with large interior angles were avoided. The number of nodes were increased until the computational difficulty was encountered in the evaluation of the matrix norms. The results are shown in Table (3.2). The

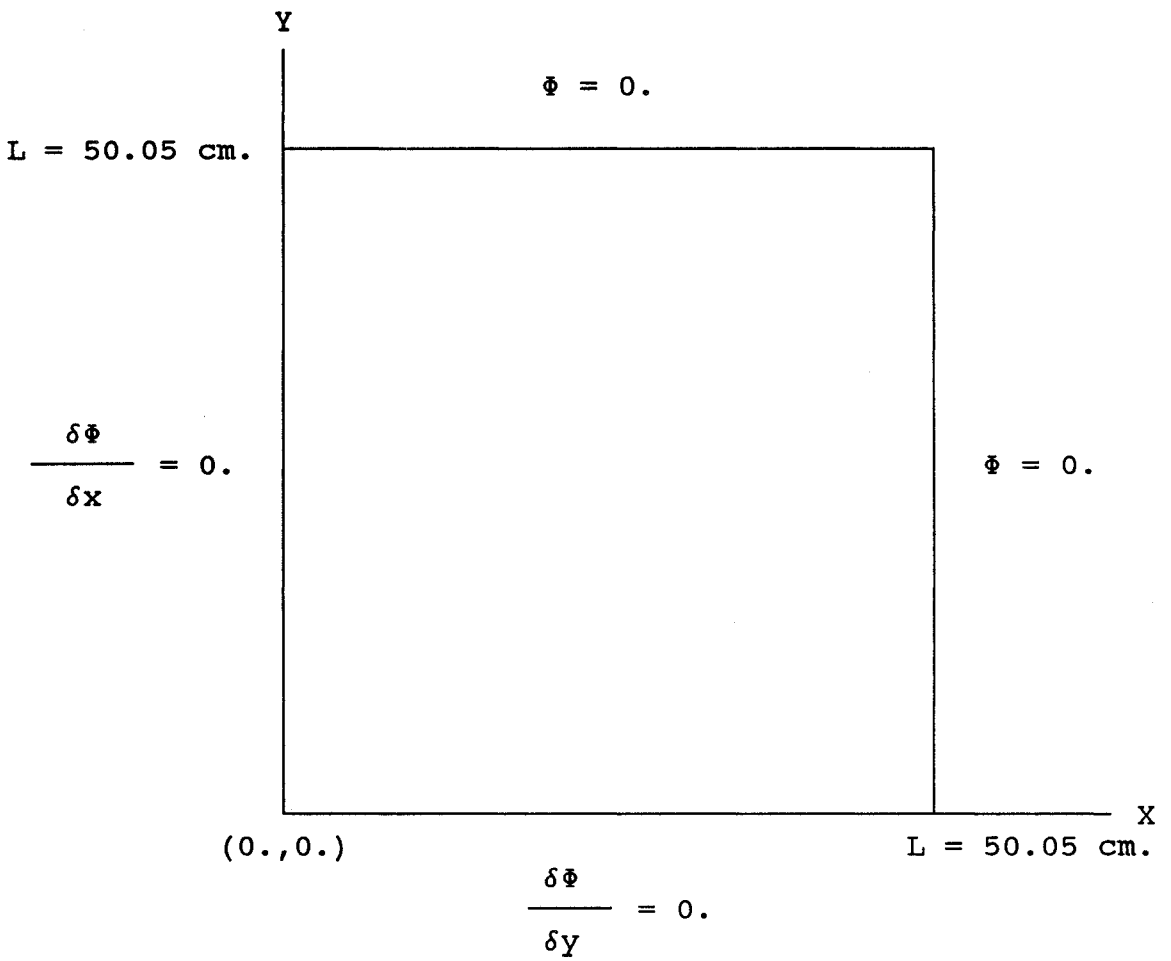

Figure (3.3). Test problem 3.1 (quarter reactor)

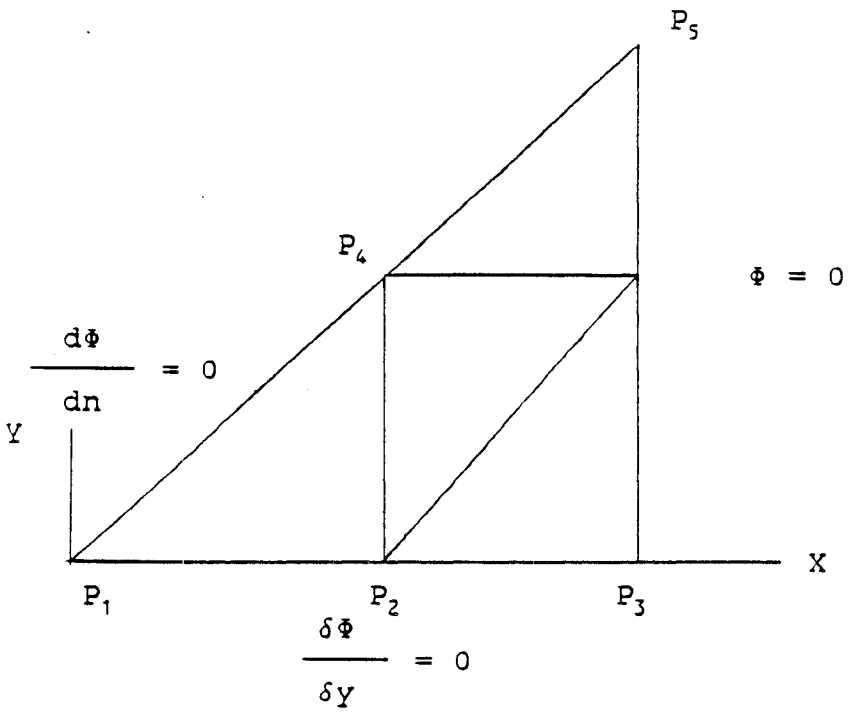

Figure (3.4). 6 node triangulation of the homogeneous type reactor; P_1 : x = 0., y = 0., P_2 : x = 25.025, P_3 : x = 50.05, P_4 : y = 25.025, P_5 : x = 50.05, y = 50.05 (in units of cm.)

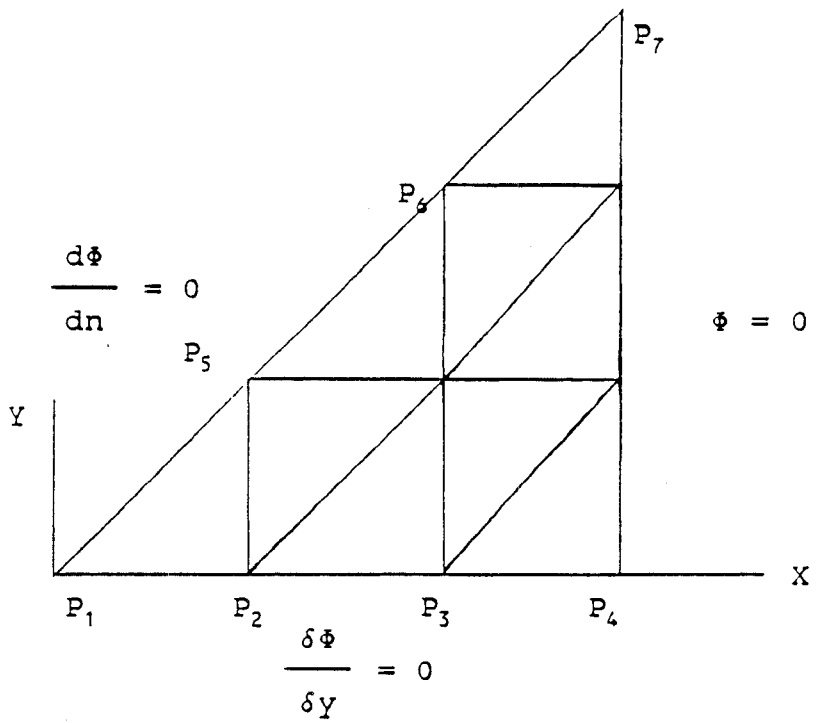

Figure (3.5). 10 node triangulation of the homogeneous type reactor; P_1 : x = 0., y = 0., P_2 : x = 16.683, P_3 : x = 33.367, P_4 : x = 50.05, P_5 : y = 16.683, P_6 : y = 33.367, P_7 : x = 50.05, y = 50.05 (in units of cm.)

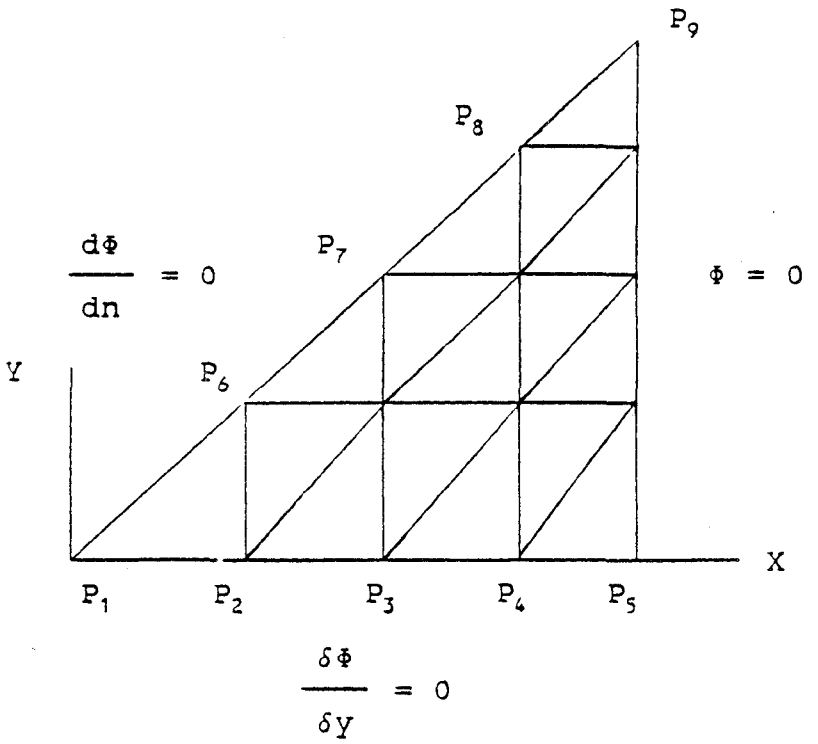

Figure (3.6). 15 node triangulation of the homogeneous type reactor; P_1 : x = 0., y = 0., P_2 : x = 12.5125, P_3 : x = 25.025, P_4 : x = 37.5375, P_5 : x = 50.05, P_6 : y = 12.5125, P_7 : y = 25.025., P_8 : y = 37.5375, P_9 : x = 50.05, y = 50.05 (in units of cm.)

does not change rapidly. Since the reactor is a homogeneous one, a large difference in the size of diagonal elements should not be expected, hence, the values seem to be reasonable and of the correct magnitude.

The variation in the condition number as a function of the number of nodes can be suitably represented by the expression

$$C_f = 0.9350*N^{0.9663}$$

$$C_t = 0.8167*N^{1.1492}$$

where N stands for number of nodes, f and t are indices for fast and thermal groups, and C is the condition number.

k_{eff} value for this problem was calculated by OPUS-G and OPUS-SOR and the results are given in Table (3.3). For each program, a compact storage technique was used. This technique excludes the zero terms during the construction of global matrices. However, the maximum number of columns has to be fixed in the main program, which may still cause some unnecessary allocation of active (in-core) memory. A pointer matrix does the bookkeeping by storing actual column numbers of the original matrix. For a small number of nodes, the combination of the pointer matrix and the compacted global system matrices may take up more space than if the compact storage scheme were not used. Nevertheless, with the increase of number of unknowns, the space saved in active memory is enormous, which in turn, provides appreciable time saving

# of nodes	Fast group	Thermal group
6	0.042E+02	0.050E+02
10	0.108E+02	0.142E+02
15	0.148E+02	0.220E+02
21	0.181E+02	0.285E+02
28	0.223E+02	0.357E+02
36	0.267E+02	0.434E+02

Table (3.2). The condition numbers versus number of nodes for the Test Problem 3.1.

METHOD	# of nodes	k_{eff}	Time(sec.)	Error(%)
OPUS-SOR	25	1.457 832	1	- 0.516
	49	1.461 780	3	- 0.246
	121	1.464 118	6	- 0.087
	441	1.465 043	35	- 0.024
OPUS-G	25	1.459 657	2	- 0.391
	49	1.462 301	4	- 0.211
	121	1.464 177	17	- 0.083
	441	1.465 075	196	- 0.021
Ref.[11]	19	1.453 262	-	- 0.827
	37	1.459 860	-	- 0.377
	61	1.462 247	-	- 0.214
	127	1.465 075	-	- 0.021

Table (3.3). Estimated k_{eff} values for the Test Problem 3.1

in the execution. In spite of the utilization of symmetry in OPUS-G program, the condensation process produced matrices with enough elements for them to be treated as a full matrix, that is, the process requires the use of columns with zero entries which eventually takes up more in-core space than originally expected. Additionally, the destruction of the properties of the elements of matrix requires that matrix should be stored before elimination is started. Although OPUS-SOR was superior in terms of execution time, OPUS-G yielded somewhat better estimates for this case.

Test Case 3.2[7]

In this quarter reactor model, the fuel is surrounded by water as shown in Figure (3.7) and the reactor has a singular point. The two group nuclear data are given in Appendix B. The triangulation procedure for the problem domain was the same as for Test Case 3.1. The results of the calculations are summarized in Table (3.4). For this case, the differences between the values of condition numbers of the group matrices are more distinct. This can be explained by comparing the fast group fuel to moderator nuclear data ratios with thermal group fuel to moderator nuclear data ratios. For example, the ratio of diffusion constants in fast group is $D_F/D_M = 1.25$, while the same ratio for thermal group is 2.67. This property causes the condition number for thermal group to be larger. It shoul be

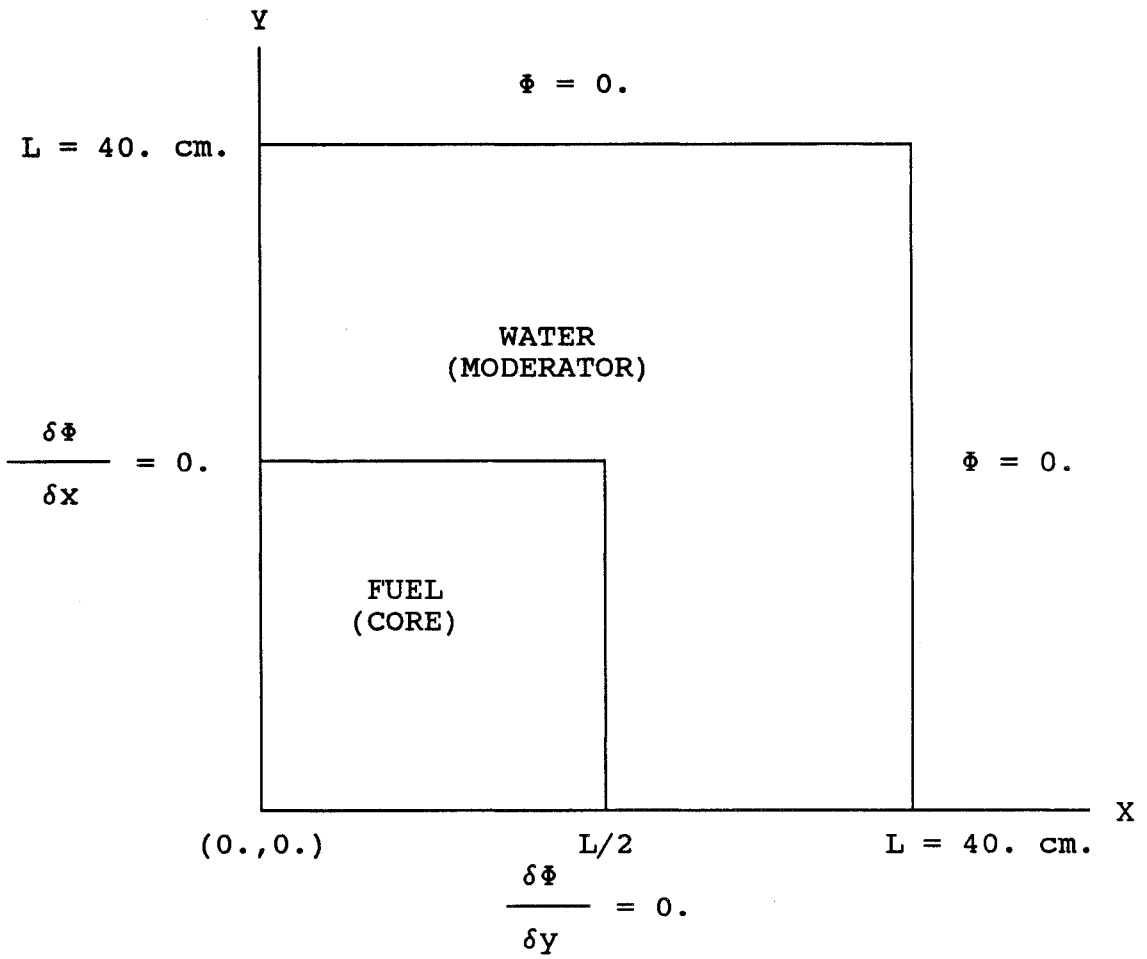

Figure (3.7). Test problem 3.2 (quarter reactor)

# of nodes	Fast group	Thermal group
6	0.056E+02	0.204E+02
10	0.115E+02	0.607E+02
15	0.130E+02	0.455E+02
21	0.191E+02	0.368E+02
28	0.193E+02	0.602E+02
36	0.196E+02	0.762E+02

Table (3.4). The condition numbers versus node numbers for the Test Problem 3.2

recalled that in the formulation used in this study is linear polynomial were used for the trial functions and the partial derivative continuity condition was not applied. Inclusion of this condition can create larger differences in the size of the diagonal elements of the system matrix. As a matter of fact, when hermite polynomials were used in the reference [7], The authors had difficulty around the singular point for this test problem. In their study, they tested several sets of modified polynomials to improve the solution. The same curve fitting procedure as used in Test Case 3.1 produced the following expression for the condition number

$$C_f = 1.965*N^{0.6898}$$

$$C_t = 1.066*N^{0.5235}.$$

CASE	METHOD	k_{eff}	Error(%)	Time(sec)	NODES
L/4	OPUS-G	0.907 948	1.15	3	25
	OPUS-SOR	0.908 715	1.24	1	
	HERMIT2D	0.912 221	1.60	-	
	FINITE DIFR.	0.926 172	3.08	-	
L/6	OPUS-G	0.904 198	0.74	5	49
	OPUS-SOR	0.904 198	0.74	2	
	HERMIT2D	0.905 760	0.91	-	
	FINITE DIFR.	0.917 804	2.25	-	
L/10	OPUS-G	0.900 436	0.32	15	121
	OPUS-SOR	0.900 436	0.32	4	
	HERMIT2D	-	-	-	
	FINITE DIFR.	-	-	-	
L/20	OPUS-G	0.898 291	0.078	184	441
	OPUS-SOR	0.898 292	0.078	20	
	FINITE DIFR.	0.900 493	0.323	-	

Table (3.5). Comparison of k_{eff} calculations for the Test Case 3.2

The benchmark value for Test Case 3.2 is

$$k_{eff} = 0.897\ 590\ 087.$$

The results calculated in reference [7] using a bilinear finite element approximation and finite difference solutions are also included in Table (3.5). For this test case OPUS-G and OPUS-SOR gave nearly identical results.

Theoretical studies predict that for a given mesh size the error bounds for the finite difference, the linear approximation and the bilinear (HERMIT-2D) methods are of the same order. The results of this test case are clearly favor the finite element approximations in terms of the accuracy achieved. Comparison of the L/10 and the L/20 cases shows that finite difference method had 0.32% k_{eff} error for the L/20 case while OPUS-G and OPUS-SOR programs had the same error for the more coarse L/10 case. Also, for a given mesh, the triangular elements used in the OPUS programs appearantly produced more accurate k_{eff} values than the rectangular elements used in HERMIT-2D code.

Test Case 3.3[11]

This test case is a loosely coupled reactor as shown in Figure (3.8). The core region have the same nuclear parameters as in Test Case 3.1, including a fission neutron yield in the thermal group. This configuration provides a severe test due to the very large flux gradients. The fast flux essentially becomes zero within the mid range along the diagonal

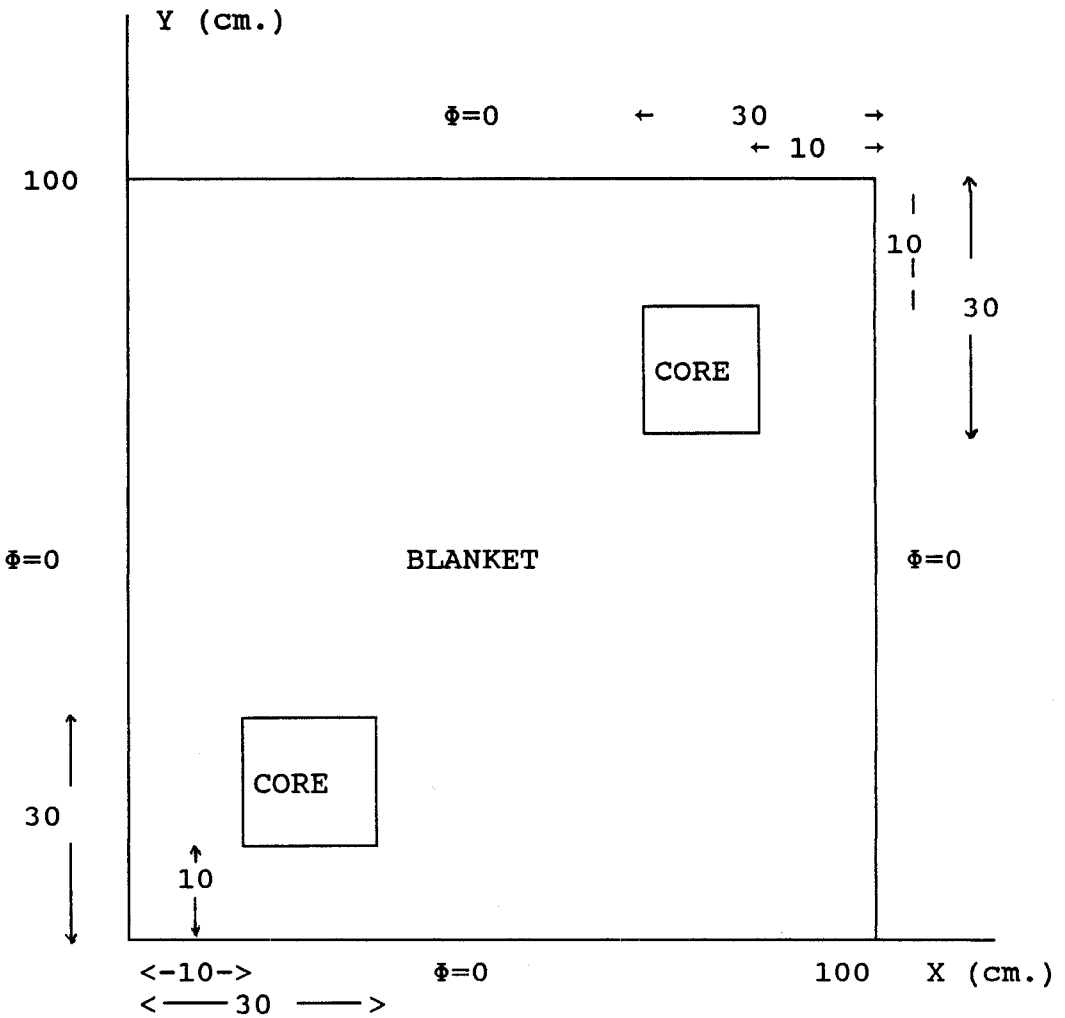

Figure (3.8). Test Case 3.3. Loosely Coupled Reactor (full reactor).

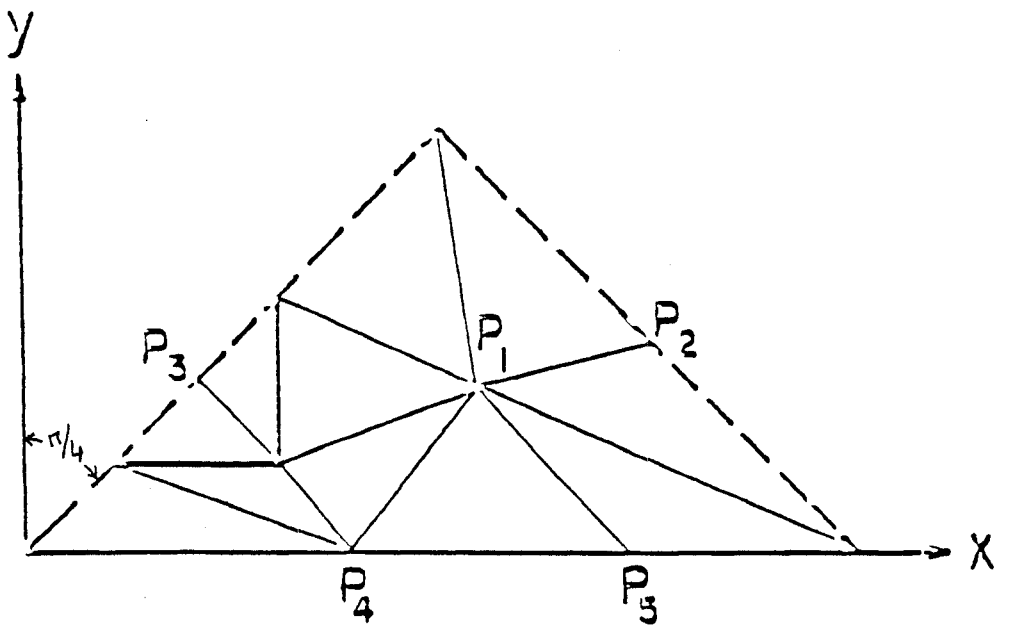

Figure (3.9). 11 node triangulation of the loosely coupled configuration (quarter reactor); P_1: x = 54., y = 20., P_2 : x = 75., y = 25., P_3 : x = 20., y = 20., P_4 : x = 38., y = 0., P_5 : x = 72., y = 0. (in units of cm.)

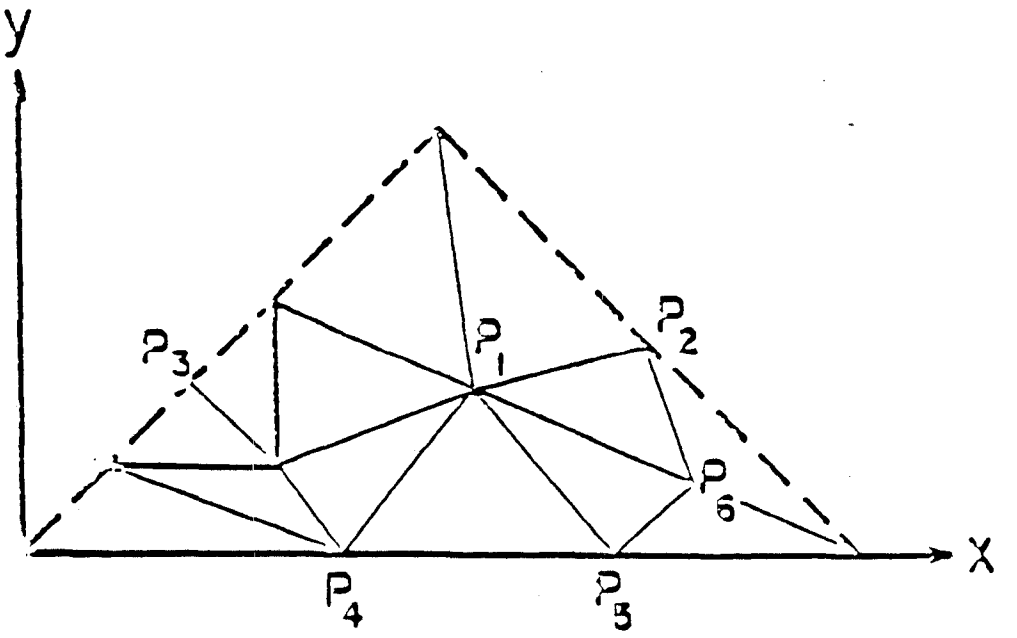

Figure (3.10). 12 node triangulation of the loosely coupled configuration (quarter reactor); the coordinates of P_1 through P_4 are the same as Figure (3.9), P_5 : x = 70., y = 0., P_6 : x = 80., y = 8. (in units of cm.)

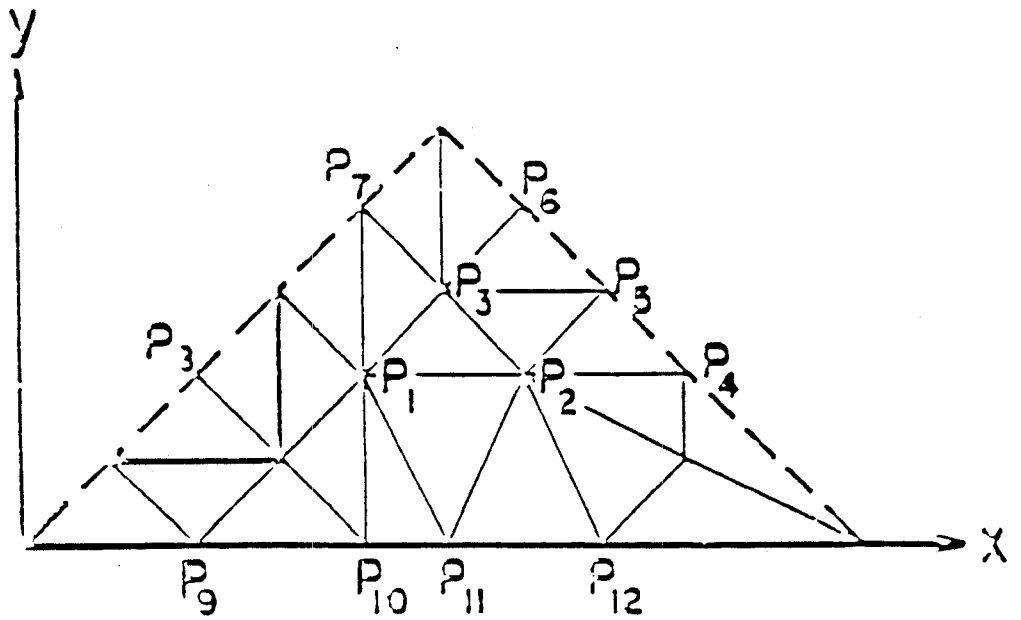

Figure (3.11). 19 node triangulation of the loosely coupled configuration (quarter reactor); P_1 : x = 40., y = 20., P_2 : x = 60., y = 20. P_3 : x = 50., y = 30., P_4 : x = 80., y = 20., P_5 : x = 70., y = 30., P_6 : x = 60., y = 40., P_7 : x = 40., y = 40. P_8 : x = 20., y = 20., P_9 : x = 20., y = 20., P_{10} : x = 40., y = 0., P_{11} : x = 50., y = 0., P_{12} : x = 70., y = 0. (in units of cm.)

connecting the points (0,0) and (100,100) [see Figure C.3, Appendix C]. The condition numbers were evaluated by using 45° symmetry as shown in Figures (3.9) through (3.11) (also see Reference [11]). The results are shown in Table (3.6). For this case the ratio of diffusion constants of the materials in both groups are relatively close in comparison. The regionwise ratio of the material removal cross sections in thermal group is about 3 times larger than the fast group removal x-section ratio. The T/F removal cross section ratios in each region are of the order of 1.44 - 2 while this ratio attains a value nearly 2.7 in the case of test problem 3.2. Despite the very large flux gradient, the condition numbers are not as large as the test case 3.2. The transport mean free path of the fast group for core material is about 8 cm. which is comparable with the dimensions of the core region. This combination produces a large fast neutron leakage fraction from the core region creating a large fast flux gradient. We make this remark to point out that the sharp flux gradient is due to geometrical configuration and it does not necessarily give rise to larger condition numbers. The following curve fitting estimates the fast group condition number with 88% accuracy and thermal condition number with 98% accuracy

$$C_f = 4.5257 * N^{0.3472}$$

$$C_t = 1.7268 * N^{1.511}$$

# of nodes	Fast group	Thermal group
11	0.114E+02	0.065E+02
12	0.108E+02	0.070E+02
19	0.113E+02	0.134E+02
23	0.123E+02	0.250E+02
39	0.178E+02	0.399E+02

Table (3.6). The condition numbers versus node numbers for the Test Problem 3.3

# of nodes	METHOD	k_{eff}	Exec. time(sec)	Error(%)
441	OPUS-G	0.740168	192	1.4
441	OPUS-SOR	0.740087	24	1.41
1151	REF. [11]	0.750682	-	0.0013
2601	OPUS-SOR	0.750532	237	0.021
2661	REF. [11]	0.750692	-	0.

Table (3.7). k_{eff} calculations for test case 3.3

The exact value of k_{eff} is given in Ref.[11] as 0.750692. The number of mesh points required to reach reasonable accuracy is 6 times higher than for the previous cases. For static condensation technique the number of columns of the loss matrix allocated was 44 when 1681 unknown mesh points were involved in the calculation while it remained 10 in the case of SOR method. Still, the results were crude. Because of the large storage requirements combined with the long computational times, no attemp was made to apply OPUS-G to a problem involving a larger number of nodes. Reasonable accuracy was attained using OPUS-SOR when the number of nodes reached 2601. The results are tabulated in Table (3.7). It should be mentioned that the result of reference [11] corresponds to triangular elements with quintic polynomial trial functions.

Test Case 3.4[10]

This case was chosen not only because it is realistic but also it is a case where the diffusion approximation itself fails. The configuration has a black control rod surrounded by fuel and moderator gap as shown in Figure (3.12). The problem has well documented solutions and an accurate PDQ calculation is also available[10]. The nuclear data is listed in Appendix B. The eigenvalue results are given in Table (3.8). OPUS-SOR was modified for this problem because the mesh generation program can only handle quarter or whole reactor boundary

conditions whereas in this case we have zero flux on three sides and zero gradient on one side. In Table (3.8), PDQ is regarded as giving the benchmark result. Except for PDQ, OPUS and FEND, all of the results were obtained using same type of synthesis method. FEND utilizes the finite element method with rectangular elements. In the FEND(A) calculation thermal group in the control rod was treated using diffusion theory. 28 rectangular elements were used in the control rod region. OPUS-SOR(A) also utilized diffusion approximation with 28 triangular elements in the control rod region. In FEND(B), the fast group in the control rod region was treated as a diffusion region while the thermal group is eliminated by using black boundary condition. OPUS(B) codes used the same formulation as FEND(B). This treatment is expected to improve the results somewhat by eliminating the truncation error caused by the diffusion approximation and the results indicate that this correct. FEND(C) took one more step and used a composite structure of triangular and rectangular elements along the control rod boundary. The FEND(C) mesh refinement achieved a significant improvement in the k^{eff} value compared to the results of over FEND(B). In this case the mesh generation program used in OPUS has a drawback. The program does not give us the flexibility to create a refined mesh structure within certain sub-region while keeping the rest of the mesh structure more coarse. Therefore, we stopped at this point and did not attempt a more refined OPUS mesh case.

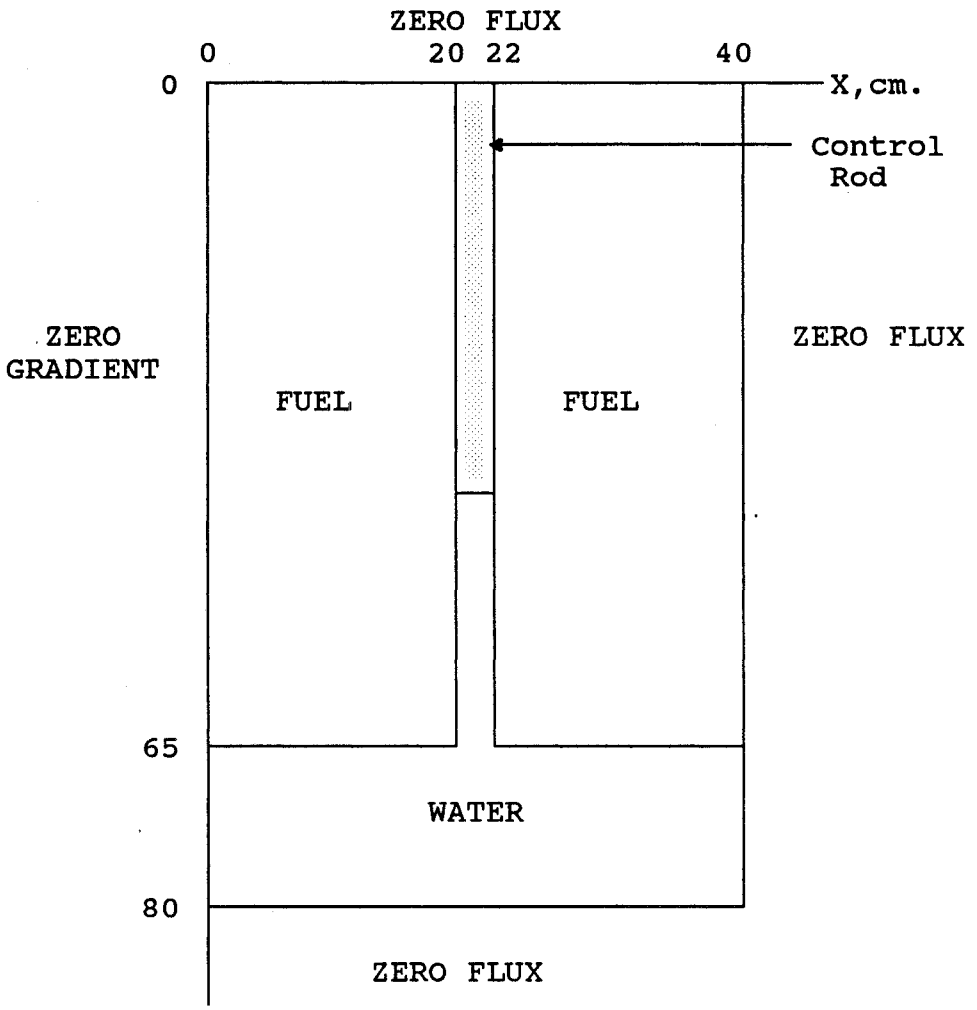

Figure (3.12). Test Case 3.4

METHOD	k_{eff}	% Error
PDQ	1.0171	0
OPUS-SOR(A)	1.0141	-0.2950
OPUS-SOR(B)	1.0164	-0.0688
OPUS-G(B)	1.0164	-0.0688
FEND(A)	1.0139	-0.3146
FEND(B)	1.0160	-0.1081
FEND(C)	1.0174	0.0294
Variational	1.0152	-0.1868
Galerkin	1.0156	-0.1474
Region balance	1.0153	-0.1769
Three spot	1.0282	1.091
Conventional	1.0311	1.377

Table (3.8). Comparison of eigenvalue results for Test Case 3.4. (The other methods are from Ref.[10])

Furthermore, the errors of FEND(C) and OPUS(B) codes are already in the same order. Obviously a good mesh generation program is a must for practical applications to realistic problems

The condition numbers were also calculated for the OPUS(A) case for smaller number of nodes. Table (3.9) shows the results for this problem.

There is one more point which should be brought out. Some 1350 mesh points were involved in the calculation of OPUS(B) case. Execution time was about 12 minutes for OPUS-SOR. When we compare this with the Test Case 3.3 study, 2601 mesh point required 3 minutes 57 seconds of execution time despite of much larger number of unknowns. This shows how the singularities may effect the convergence of an iterative method.

# of nodes	Fast Group	Thermal Group
16	0.176E+02	0.753E+02
24	0.213E+02	0.751E+02
40	0.308E+02	0.620E+02

Table (3.9). Condition Numbers versus Node Numbers for Test Problem 3.4

CHAPTER 4

The purpose of this chapter is to develop a solution method to space-time dependent diffusion equation by employing subdomain collocation method as a means of temporal approximation.

First, the difficulties encountered in the variational formulation are examined. It will be clarified that the Galerkin's method provides the most appropriate formulation. Once the Galerkin procedure is carried out, we are left with the time dependent diffusion equation. Inclusion of precursor equations to the time dependent multi-group equations yields a one-step algorithm, which can be solved by the solution methods discussed in Chapter 3.

4.1. DIFFICULTIES IN THE VARIATIONAL METHOD

In Chapter 2, It was shown that the function Φ was the solution of the static diffusion equation and also made the associated functional stationary. Additionally, We know that the Ritz and Galerkin methods are equivalent for self-adjoint problems.

The concern of this section is to attempt to construct variational formulation for time-dependent problem. The rate of change of neutron density in a nuclear reactor with no delayed neutrons is

$$v^{-1}\frac{\delta\Phi(\mathbf{r},t)}{\delta t} = [\nabla.D(\mathbf{r},t)\nabla - \Sigma_r(\mathbf{r},t)]\Phi(\mathbf{r},t) + q(\mathbf{r},t), \quad (4.1)$$

with the boundary condition on the reactor boundary

$$\Phi(R,t) = 0, \qquad 0 \le t \le T \quad (4.2)$$

and the initial condition

$$\Phi(\mathbf{r},0) = g(\mathbf{r}), \qquad \mathbf{r}\epsilon R \quad (4.3)$$

The process in a nuclear reactor represented by (4.1) is an irreversible (dissipative) process and the nuclear reactor is said to be a nonconservative system. In the variational representation of nonconservative systems, an imaginary system is considered to make both systems conservative and stationary together. This requirement can be satisfied by Roussopolos functional[35]

$$F(\Phi,\Phi^*) = \int_D d\mathbf{r}\int_0^T dt\{\ \Phi^*(\mathbf{r},t)[-v^{-1}\frac{\delta\Phi(\mathbf{r},t)}{\delta t} + \nabla.D(\mathbf{r},t)\nabla\Phi(\mathbf{r},t)$$

$$-\Sigma_r(\mathbf{r},t)\Phi(\mathbf{r},t) + q(\mathbf{r},t)] + q^*(\mathbf{r},t)\Phi(\mathbf{r},t)\ \}, \quad (4.4)$$

where Φ^* is the adjoint flux and q^* is the adjoint source. We expect that when the first variation of (4.4) is made

stationary, its Euler-Lagrange form will give (4.1) and satisfy the conditions (4.2) and (4.3). The first variation of (4.4) is

$$\begin{aligned}\delta F(\Phi,\Phi^*) = \int_D dr \int_0^T dt\{\ & \delta\Phi[\ (v^{-1})^* \frac{\delta\Phi^*}{\delta t} + \nabla\cdot D^*(\mathbf{r},t)\nabla\Phi^*(\mathbf{r},t) \\ & -\Sigma_r^*(\mathbf{r},t)\Phi^*(\mathbf{r},t)+q^*(\mathbf{r},t)\ ] \\ & + \delta\Phi^*[-v^{-1}\frac{\delta\Phi}{\delta t} + \nabla\cdot D(\mathbf{r},t)\nabla\Phi-\Sigma_r(\mathbf{r},t)\Phi(\mathbf{r},t)+q(\mathbf{r},t)]\ \} \\ & -\int_D dr[\ \Phi^*(\mathbf{r},T)v^{-1}\delta\Phi(\mathbf{r},T)-\Phi^*(\mathbf{r},0)v^{-1}\delta\Phi(\mathbf{r},0)\ ] \\ & +\int_0^T dt\{\ \Phi^* D(\mathbf{r},t)\nabla\delta\Phi\cdot\mathbf{n}|_{r=R} - \delta\Phi D^*(\mathbf{r},t)\nabla\Phi^*\cdot\mathbf{n}|_{r=R}\ \} = 0\ ,\end{aligned} \tag{4.5}$$

where $\mathbf{n}$ denotes the outward unit normal to the boundary surface. In the derivation of (4.5), integration by parts was used for the following integrals

$$\int dr\int_0^T dt\Phi^*(\mathbf{r},t)\nabla\cdot D(\mathbf{r},t)\nabla\delta\Phi(\mathbf{r},t) = \int_D dr\int_0^T dt\delta\Phi(\mathbf{r},t)\nabla\cdot D(\mathbf{r},t)\nabla\Phi^*(\mathbf{r},t)$$
$$+\int_0^T dt\{\ \Phi^* D(\mathbf{r},t)\nabla\delta\Phi\cdot\mathbf{n}|_{r=R} - \delta\Phi D^*(\mathbf{r},t)\nabla\Phi^*\cdot\mathbf{n}|_{r=R}\ \}\ ,$$

$$\int dr\int_0^T dt\Phi^*(\mathbf{r},t)v^{-1}\frac{\delta\Phi}{\delta t} = \int dr\{-\int_0^T dt\delta\Phi v^{-1}\frac{\delta\Phi}{\delta t} + \Phi^* v^{-1}\delta\Phi|_0^T\ \}\ .$$

It can be seen from (4.5), that additional restrictions other than (4.2) and (4.3) have to be satisfied in order to set the first variation of the functional equal to zero. In (4.5), it is required that $\delta\Phi$ has to vanish for all $\mathbf{r}$ at t=0

and also on the rector boundary r=R for all $0 \leq t \leq T$. Furthermore, Φ^* must satisfy the equation in the first bracket of (4.5) and must satisfy the following boundary conditions

$$\Phi^*(R,t) = 0 \qquad 0 \leq t \leq T$$

$$\Phi^*(r,T) = 0 \qquad r \epsilon R$$

It is possible to drive a functional whose Euler-Lagrange form gives (4.1) satisfies all the conditions imposed on. Such derivation, clearly, restricts the variation $\delta\Phi$ of Φ. However, the variation of $\delta\Phi$ still has to be continuous for all times $0 \leq t \leq T$. To provide a suitable solution procedure, it may be desirable to develop a variational formalism which admits discontinuous variations of $\delta\Phi$ in space and time (for example, when Φ and Φ^* are the results of step changes in the reactor parameter).

Briefly, we can develop a formalism which overcomes these difficulties[36],[37]. However the formalism requires great deal of effort to solve and the sign of the second variation may depend upon some restrictive conditions. These conditions may mean that the stationary value is not an extrema, but a saddle point. These difficulties persuade us to look for a more convenient procedure.

4.2. THE WEIGHTED RESIDUAL GALERKIN METHOD

Weighted residual methods are techniques for by-passing the variational formulation for time dependent problems. To

apply this method, the neutron dynamics equation is discretized in space variables by the Galerkin method, leading to a system of ordinary differential equations in time.

The neutron dynamics equation coupled with precursor equations is

$$v^{-1}\frac{\delta\Phi}{\delta t} = \nabla\cdot D(\mathbf{r},t)\nabla\Phi(\mathbf{r},t) - R(\mathbf{r},t)\Phi(\mathbf{r},t) + S(\mathbf{r},t)\Phi(\mathbf{r},t)$$

$$+(1-\beta)PF^{T}(\mathbf{r},t)\Phi(\mathbf{r},t) + \sum_{j=1}^{J} d_j\Gamma_j\mu_j(\mathbf{r},t) \ , \qquad (4.6a)$$

$$\frac{\delta\mu_j(\mathbf{r},t)}{\delta t} = \beta_j F^{T}(\mathbf{r},t)\Phi(\mathbf{r},t) - \Gamma_j\mu_j(\mathbf{r},t), \qquad j=1,J \qquad (4.6b)$$

with the conditions

$$\Phi(\mathbf{r},t)\big|_{t=0} = \Phi_0(\mathbf{r}) \qquad (4.7)$$

$$\Phi(\mathbf{R},t) = 0 \qquad (4.8)$$

$$\nabla\cdot D(\mathbf{R},t)\nabla\Phi(\mathbf{R},t) = 0 \ , \qquad (4.9)$$

where

$$D = \begin{bmatrix} D_1(\mathbf{r},t) & & & \\ & \ddots & & \\ & & D_g(\mathbf{r},t) & \\ & & & \ddots \\ & & & & D_G(\mathbf{r},t) \end{bmatrix}$$

$$R = \begin{bmatrix} \Sigma_{r,1}(\mathbf{r},t) & & & \\ & \ddots & & \\ & & \Sigma_{r,g}(\mathbf{r},t) & \\ & & & \ddots \\ & & & & \Sigma_{r,G}(\mathbf{r},t) \end{bmatrix}$$

$$S(\mathbf{r},t) = \begin{bmatrix} 0 & & & & \\ \Sigma_{21} & 0 & & & \\ . & & 0 & & \\ . & . & . & . & 0 \\ \Sigma_{G1} & \Sigma_{G(G-1)} & & & 0 \end{bmatrix}$$

$$v^{-1} = \begin{bmatrix} v_1^{-1} & & & \\ & . & & \\ & & v_g^{-1} & \\ & & & . \\ & & & & v_G^{-1} \end{bmatrix}$$

$$P = [\ p_1\ ,\ p_g\ ,\ p_G\ ]^T$$

$$d_j = [\ d_{j,1}\ ,\ d_{j,g}\ ,\ d_{j,G}\ ]^T$$

$$F^T = [\ (\tau\Sigma_f)_1\ ,\ (\tau\Sigma_f)_g\ ,\ (\tau\Sigma_f)_G\ ]$$

$$\Phi = [\ \Phi_1\ ,\ \Phi_g\ ,\ \Phi_G\ ]^T$$

g =index number of the neutron energy group.

j =index number of the precursor group.

μ_j = density of the $j^{th.}$ precursor (cm^{-3}).

v_g = speed of the neutrons in the $g^{th.}$ group (cm/sec.).

Γ_j = decay constant of the $j^{th.}$ precursor (1/sec.).

$d_{g,j}$ = the delayed neutron spectrum yield in group g, which are emitted by the decay of $j^{th.}$ precursor.

p_g = the fission spectrum yield in group g.

β_j = fractional yield of the $j^{th.}$ precursor by fission.

and the rest of the terms were defined in previous chapters. We define the inner product and the L^2 norm by

$$(w,u) = \int_D w^T u dr \ ,$$

$$\|u\| = (u,u)^{1/2} \ ,$$

and the bilinear form

$$B(w,u) = (\nabla w, D\nabla u) + (w,Ru) \ . \qquad (4.10)$$

Thus the corresponding weak form of the problem becomes

$$(w_f, v^{-1}\frac{\delta u}{\delta t}) + B(w_f,u) = (w_f,Q) \ , \qquad (4.11a)$$

$$(w_{d,j}, \frac{\delta \mu_j}{\delta t}) = \beta_j (w_{d,j}, F^T\Phi) - \sum_{j=1}^{J} (w_{d,j}, \Gamma_j \mu_j) \ , \quad j=1,J \qquad (4.11b)$$

subject to initial condition

$$(w_f,\Phi)_{t=0} = (w_f,\Phi_0) \ ,$$

where

$$(w_f,Q) = (w_f,Su) + (w_f,(1-\beta)F^T u) + \sum_j (w_f, d_j \Gamma_j \mu_j) \ ,$$

w_f= Weighting function for the neutron group equations.

$w_{d,j}$= Weighting function for the j th. precursor equations.

Note that in the definition of bilinear form (4.10) the

surface integral along the domain boundary was omitted since the boundary condition (4.9) is a natural boundary condition. We seek an approximate solution within the element "e" to the neutron flux and precursor density as a linear combination of linear polynomials with time-dependent coefficients

$$\bar{\Phi}_e(\mathbf{r},t) = \sum_{i=1}^{3} u_i(\mathbf{r}) c_i(t), \tag{4.12}$$

$$\bar{\mu}_{j,e}(\mathbf{r},t) = \sum_{i=1}^{3} N_{j,i}(\mathbf{r}) C_{j,i}(t). \tag{4.13}$$

Since $\Phi_e(\mathbf{r},t)$ is a (Gx1) vector whose elements are group fluxes, $u_i(\mathbf{r})$ is (GxG) diagonal matrix and $c_i(t)$ is a (Gx1) vector. The $u(\mathbf{r})$'s are the trial functions as already defined by (3.19). The trial functions, $N_j(\mathbf{r})$, in precursor approximation can be chosen to be the same as the $u(\mathbf{r})$. In this study, following the methodology developed in Ref.[38], the choices for the precursor trial functions and precursor weighting functions are:

$$N_j(\mathbf{r}) = F_0^T u(\mathbf{r}) , \tag{4.14}$$

$$w_{d,j}(\mathbf{r}) = d_j^T u(\mathbf{r}) , \tag{4.15}$$

where the subscript zero refers to an initial unperturbed state. Then, the approximations (4.12) through (4.15) are applied to the weak forms (4.11a) and (4.11b) for each element

$$(u_i, v^{-1}\frac{\delta\bar{\Phi}}{\delta t})_e + B(u_i,\bar{\Phi})_e = (u_i,Q)_e , \tag{4.16}$$

$$(d_j^T u_i, \frac{\delta \bar{\mu}_j}{\delta t})_e = \beta_j (d_j^T u_i, F^T \bar{\Phi})_e - \Sigma (d_j^T u_i, \Gamma_j \bar{\mu_j})_e \ , \qquad (4.17)$$

$$i=1,2,3 \ ,$$

$$1 \leq j \leq J \ ,$$

and the elements are assembled into global matrices

$$V^{-1} \frac{dc(t)}{dt} + A(t)c(t) = Sc(t) + (1-\beta)M(t)c(t) + \sum_{j=1}^{J} \Gamma_j M_{0,j} C_j(t) \ , \qquad (4.18)$$

$$M_{0,j} \frac{dC_j(t)}{dt} = \beta_j M_j(t) c(t) - \Gamma_j M_{0,j} C_j(t) \ ; \quad j=1,6 \ , \qquad (4.19)$$

where

$$A = \begin{bmatrix} [A_1] & & & \\ & \ddots & & \\ & & [A_g] & \\ & & & \ddots \\ & & & & [A_G(t)] \end{bmatrix} \ ,$$

$$V^{-1} = \begin{bmatrix} [V_1^{-1}] & & & \\ & \ddots & & \\ & & [V_g^{-1}] & \\ & & & \ddots \\ & & & & [V_G^{-1}] \end{bmatrix} \ ,$$

$$S = \begin{bmatrix} [0] & \cdots & \\ [S_{21}] & [0] & \\ \cdot & & \\ \cdot & & \cdot \\ [S_{G1}] \cdots & [S_{G(G-1)}] & [0] \end{bmatrix}$$

$$M(t) = \begin{bmatrix} [M_{11}(t)] & \cdots & [M_{1G'}(t)] \\ \cdot & & \\ [M_{g1}(t)] & \cdots & \cdot \\ \cdot & & \\ [M_{G1}(t)] & \cdots & [M_{GG'}(t)] \end{bmatrix}$$

$$M_j(t) = \begin{bmatrix} [M_{j,11}(t)] & \cdots & [M_{j,1G'}(t)] \\ \cdot & & \cdot \\ [M_{j,g1}(t)] & \cdots & [M_{j,gg'}(t)] \\ \cdot & & \cdot \\ [M_{j,G1}(t)] & \cdots & [M_{j,GG'}(t)] \end{bmatrix} ,$$

$$M(t) = PF^T(t)$$

$$M_j(t) = d_j F^T(t)$$

$$M_{j,0} = d_j F_0^T$$

$$F^T(t) = [\, [F_1(t)] \,,\, [F_{g'}(t)] \,,\, [F_{G'}(t)] \,]$$

$$[A_g(t)] = [(A_g(t))_{ij}] \quad ,$$

$$[V_g^{-1}] = [(V_g^{-1})_{ij}] \quad ,$$

$$[S_{gg'}] = [(S_{gg'})_{ij}] \quad ,$$

$$[F_g(t)] = [(F_g(t))_{ij}]$$

$$[M_{gg'}(t)] = [(M_{gg'}(t))_{ij}] \quad ,$$

$$[M_{j,gg'}(t)] = [(M_{j,gg'}(t))_{ij}] \quad ,$$

$$(A_g(t))_{ij} = \sum_e (D_{g,e}(\nabla u, \nabla u) + (u, \Sigma_{r,g,e} u))_e \quad ,$$

$$(V_g^{-1})_{ij} = \sum_e ((u, v_g^{-1} u))_e \quad ,$$

$$(S_{gg'})_{ij} = \sum_e ((u, \Sigma_{gg',e} u))_e \quad ,$$

$$(F_g(t))_{ij} = \sum_e ((u, \Sigma_{f,g,e} u))_e$$

$$(M_{gg'}(t))_{ij} = \sum_e ((u, p_g \Sigma_{f,g',e}(t) u))_e \quad ,$$

$$(M_{j,gg'}(t))_{ij} = \sum_e ((d_{j,g} u, \Sigma_{f,g',e}(t) u))_e \quad ,$$

$$c(t) = [\{c_1(t)\} , \{c_g(t)\} , \{c_G(t)\}]^T \quad ,$$

$$\{c_g(t)\} = \{ c_{g,1}(t), \ldots, c_{g,N}(t) \} \quad ,$$

In the foregoing (,) represents the inner product operation.

4.3. DISCRETIZATION IN TIME BY THE SUBDOMAIN METHOD

To utilize the finite element method fully, trial functions in time domain are applied. In the discretization of (4.18) and (4.19), the coefficients of the space dependent linear trial functions are now represented by time dependent linear polynomials of the form

$$c(t) = (1 - \frac{t - t_i}{\Delta t}) n_i + \frac{t - t_i}{\Delta t} n_{i+1} \quad , \qquad (4.20)$$

and

$$\blacktriangle t = t_{i+1} - t_i \ .$$

In the derivation of discretized equations it will be assumed that only A(t) is a function of time,

$$A(t) = A_0[1+\epsilon f(t)]. \tag{4.21}$$

First, the precursor equation is integrated and substituted into the multigroup equation giving

$$V^{-1}\frac{dc(t)}{dt}+A(t)c(t) = S_0c(t)+(1-\beta)M_0c(t)$$
$$+\sum_{j=1}^{J}\Gamma_jM_{0,j}\{\ C_j(t_i)\exp[-\Gamma_j(t-t_i)]+\beta_j\int_{t_i}^{t}C(t')\exp[-\Gamma_j(t-t')]dt'\}. \tag{4.22}$$

The next step is to partition the time domain into smaller subdomains and to define the following weighting functions

$$w_k(t_n) = \begin{cases}1 \text{ if } t_n \text{ in the } k^{th.} \text{ time domain}\\ 0 \text{ otherwise}\end{cases} \tag{4.23}$$

Then, Eq.(2.25) can be applied to (4.22) for each subdomain $D\epsilon(t_i,t_{i+1})$, making the average value of residual in each subdomain zero. This procedure is called **subdomain method.** The results of integrations give

$$U_{i+1}n_{i+1} = U_in_i + \sum_{j=1}^{J}[\exp(-\Gamma_jt_i) - \exp(-\Gamma_jt_{i+1})]D_{j,i}\ , \tag{4.24}$$

where the matrices are defined by

$$U_{i+1} = V^{-1} + \frac{\Delta t}{2}[A_{i+1} - S_0 - (1-\beta)M_0] - \sum_{j=1}^{J} \Gamma_j \beta_j a_{i+1,j} M_{0,j} + R_{i+1} \ ,$$

$$U_i = V^{-1} + \frac{\Delta t}{2}[A_i - S_0 - (1-\beta)M_0] - \sum_{j=1}^{J} \Gamma_j \beta_j a_{i,j} M_{0,j} + R_i \ ,$$

$$D_{j,s+1} = D_{j,s} + \Gamma_j \beta_j M_{0,j}[b_{j,s+1} n_{s+1} + b_{j,s} n_s] \ ,$$

$$D_{j,1} = \frac{\beta_j}{\Gamma_j} M_{0,j} n(0) \ ,$$

and the coefficients by

$$a_{i+1,j} = \frac{\beta_j}{\Gamma_j}\left\{ \frac{\Delta t}{2} - \frac{1}{\Gamma_j}\left[1 + \frac{1}{\Gamma_j \Delta t}[\exp(-\Gamma_j \Delta t) - 1]\right]\right\} \ ,$$

$$a_{i+1,j} = \frac{\beta_j}{\Gamma_j}\left\{ \frac{\Delta t}{2} + \frac{1}{\Gamma_j}\left[1 + \frac{1}{\Delta t}\left(\Delta t + \frac{1}{\Gamma_j}\right)[\exp(-\Gamma_j \Delta t) - 1]\right]\right\} \ ,$$

$$b_{j,s+1} = \frac{\beta_j}{\Delta t \Gamma_j}\left[\left(\Delta t - \frac{1}{\Gamma_j}\right)\exp(\Gamma_j t_{s+1}) + \frac{1}{\Gamma_j}\exp(\Gamma_j t_s)\right] \ ,$$

$$b_{j,s} = \frac{\beta_j}{\Delta t \Gamma_j}\left[\frac{1}{\Gamma_j}\exp(\Gamma_j t_{s+1}) - \left(\Delta t + \frac{1}{\Gamma_j}\right)\exp(\Gamma_j t_s)\right] \ .$$

Here, R^i and R_{i+1} depends upon the form of (4.21). The benchmark problems used in this study consider the following type of perturbations:

$$A(t) = A_0 + \delta A * t \ , \qquad (4.25)$$

or

$$A(t) = A_0 + \delta A * \mathrm{Sin}(w * t) \ . \qquad (4.26)$$

The use of (4.25) in (4.22) yields

$$R_i = -(\blacktriangle t_i^2/6)*\delta A \ ,$$

$$R_{i+1} = (\blacktriangle t_i^2/3)*\delta A \ .$$

For perturbations of the type shown in (4.26) substitution into (4.22) gives

$$R_i = -\ [\ w*\blacktriangle t_i*\mathrm{Cos}(w*t_i) - \mathrm{Sin}(w*t_{i+1}) + \mathrm{Sin}(w*t_i)\ ]*\delta A/(w^2*\blacktriangle t_i),$$

$$R_{i+1} = [-w*\blacktriangle t_i*\mathrm{Cos}(w*t_{i+1}) + \mathrm{Sin}(w*t_{i+1}) - \mathrm{Sin}(w*t_i)\ ]*\delta A/(w^2*\blacktriangle t_i).$$

4.4. NUMERICAL PROCEDURE

Consider (4.24) in the matrix form for two group calculations

$$\begin{vmatrix} U_{11} & -U_{12} \\ -U_{21} & U_{22} \end{vmatrix} \begin{vmatrix} \Phi_{1,i+1} \\ \Phi_{2,i+1} \end{vmatrix} = \begin{vmatrix} b_{1,i} \\ b_{2,i} \end{vmatrix} \qquad (4.27)$$

where i is the time index. To solve Eq.(4.27), the following procedure was adopted[39]:

$$U_{11}\ \Phi_{1,i+1,n+1} = U_{12}\ \Phi_{2,i+1,n} + b_{1,i}$$

$$U_{22}\ \Phi_{2,i+1,n+1} = U_{21}\ \Phi_{1,i+1,n+1} + b_{2,i}$$

where n represents the iteration step. Since the vector, $b_{1,i}$ is known, one should only start with an initial guess of $\Phi_{2,i+1,0}$ to calculate the $\Phi_{1,i+1,1}$ and then $\Phi_{2,i+1,1}$, respectively, by Gaussian elimination or by the SOR technique. This process

continues until the error is less than a predetermined value. Then the same procedure is followed for the next time step.

4.5. NUMERICAL RESULTS

In this section, several examples of solving the time-dependent two group diffusion equation by the subdomain method are presented. The first three cases are model problems which consider dynamic response of homogeneous-type reactors. The first model problem is a uniform linear perturbation. The other two are localized oscillatory perurbations. These types of perturbations are important in analyzing spontaneous buildup of large power oscillations and also in the calibration of reactor control rods by reactivity oscillation method. The last two problem are for non-model reactor configuration, originally designed by Bettis Atomic Energy Laboratory. The reactor configuration represents a more realistic situation, typical of what might be encountered in a light water breeder reactor.

Test Case 4.1

This is a one-group problem with six groups of precursors. The reactor configuration is shown in Figure (4.1) and the nuclear data for this problem are given in Appendix B. The k_{eff} value was found to be 1.06366 by OPUS-G and 1.06363 by OPUS-SOR using an 121 node finite element model.

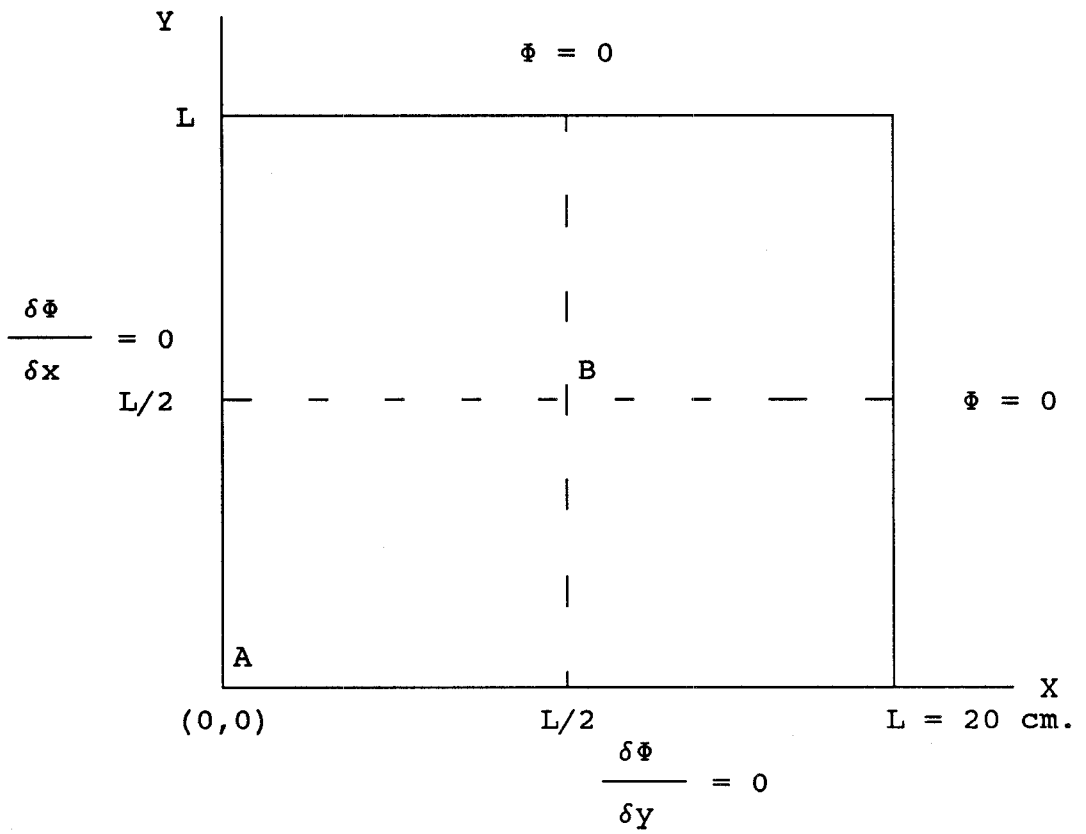

Figure (4.1). Configuration of Test Case I.

▲t	OPUS-SOR		OPUS-G		HERMIT2D	
(sec.)	t=0.1	t=0.5	t=0.1	t=0.5	t=0.1	t=0.5
0.1	1.14070	2.91672	1.13212	3.00475	1.13576	3.02712
0.05	1.14949	3.02383	1.14672	3.05736	1.14844	3.08119
0.01	1.14914	3.10770	1.14413	3.07526	1.14677	3.09877
0.005	1.14934	3.11395	1.14444	3.07185	1.14680	3.09934

Table (4.1). The result of uniform linear perturbation at point A.

▲t	OPUS-SOR		OPUS-G		HERMIT2D	
(sec.)	t=0.1	t=0.5	t=0.1	t=0.5	t=0.1	t=0.5
0.1	0.56598	1.46082	0.56121	1.49101	0.56788	1.51356
0.05	0.57117	1.50871	0.57046	1.51892	0.57422	1.54060
0.01	0.57074	1.54363	0.56839	1.52693	0.57338	1.54939
0.005	0.57070	1.54621	0.56824	1.52729	0.57340	1.54967

Table (4.2). The result of uniform linear perturbation at point B.

The perturbation is induced in the pseudo-critical reactor by changing the absorption cross section uniformly throughout the reactor in a ramp-like manner

$$\Sigma_a(t) = \Sigma_a(0)*(1. - 0.01*t).$$

Table (5.1) compares the results with the Hermite Method proposed by reference [7] at points A and B(cf., Fig. (4.1)) for various time step lengths ▲t. The flux values are normalized by the initial flux value at point A. Since HERMIT2D is not a benchmark we simply compare the results in terms of their closeness. The results are shown in Table (4.1) and Table (4.2). We observe that for larger ▲t, OPUS-SOR yields lower values than the other two programs but when ▲t became smaller OPUS-SOR gave improved results. The consistency of the results as t increases can be observed by checking the ratio of the fluxes at A and B at time t, which should remain unchanged since the perturbation is spatially uniform.

Test Problem 4.2

This is a two group problem without delayed neutrons. The system consists of a uniform fuel(Fig.(4.2)). The nuclear data of this case is given in Appendix B. Using 121 node model, the k_{eff} value is found to be 0.86773 by OPUS-G and 0.86770 by OPUS-SOR programs. The system is perturbed locally in R_2 region (cf., Fig.(4.2)) from the pseudo-critical state by changing the thermal cross section sinusoidally in the form

$$\Sigma_{a,2}(t) = \Sigma_{a,2}(0)*[1. + 0.2*\mathrm{Sin}(\frac{2*\pi}{T}*t)], \quad T = 10^{-3} \text{ sec.}$$

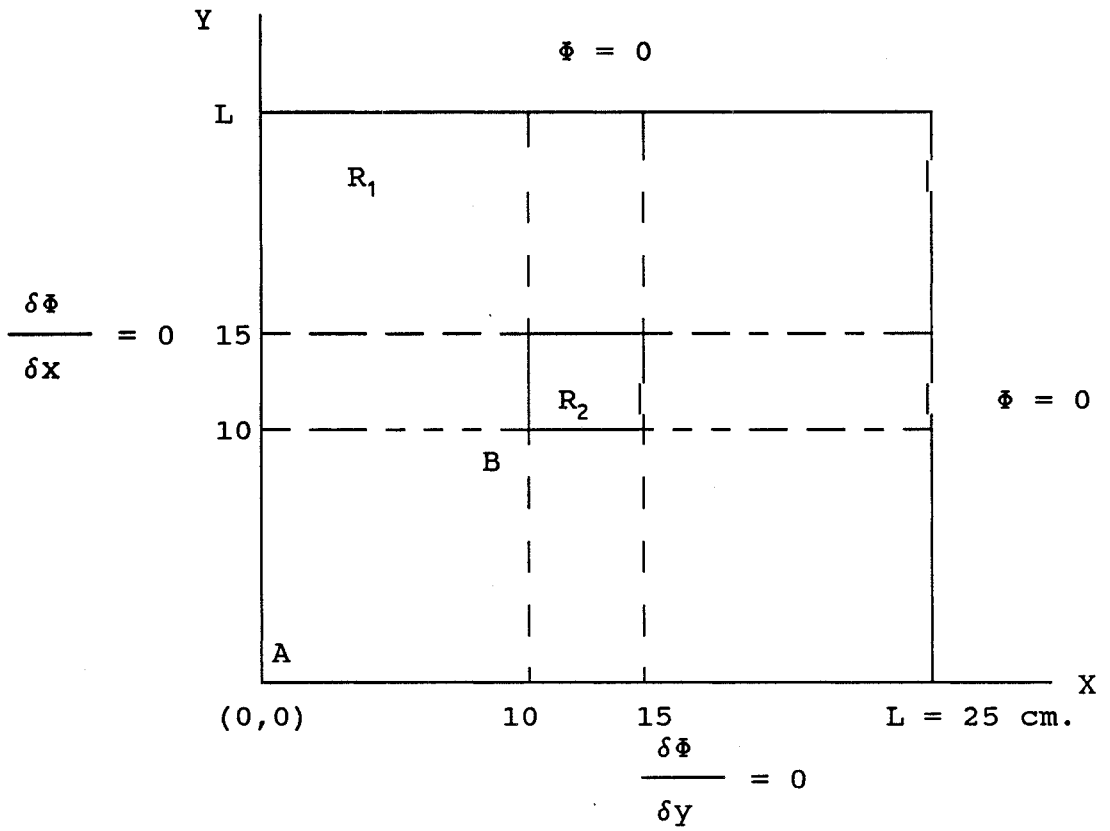

Figure (4.2). Configuration of Test Case 4.2

OPUS-SOR required 6.5 sec average execution time per time step when ▲t = T/4. The total time required was 26 sec for this case. When ▲t = T/20 the average execution time per time step droped to 2.25 second. OPUS-G generally required a much larger execution time than OPUS-SOR. For example, the smallest OPUS-G execution time required was 21 sec per time step for the ▲t = t/20 case.

▲t	METHOD	t=T/4	t=T/2	t=3T/4	t=T
T/4	OPUS-SOR	0.96879	0.87556	0.86260	1.0109
	OPUS-G	0.96869	0.87522	0.86205	0.9953
	HERMIT2D	0.96849	0.87589	0.86531	1.0188
T/8	OPUS-SOR	0.95905	0.85069	0.87791	1.0469
	OPUS-G	0.95889	0.85029	0.87739	1.0460
	HERMIT2D	0.95864	0.85150	0.88171	1.0585
T/16	OPUS-SOR	0.94996	0.84447	0.88683	1.0574
	OPUS-G	0.97517	0.84413	0.88630	1.0566
	HERMIT2D	0.94971	0.84559	0.89122	1.0678
T/20	OPUS-SOR	0.94825	0.82916	0.88877	1.0590
	OPUS-G	0.94807	0.84311	0.88821	1.0581
	HERMIT2D	0.94805	0.84462	0.89323	1.0714

Table (4.3). Test Case 4.2: Local Sinusoidal Perturbation. Thermal Neutron Flux at point A ($T=10^{-3}$ sec).

▲t	METHOD	t=T/4	t=T/2	t=3T/4	t=T
T/4	OPUS-SOR	0.58253	0.49099	0.65198	0.75981
	OPUS-G	0.58250	0.49080	0.65163	0.75926
	HERMIT2D	0.59227	0.50374	0.64698	0.76635
T/8	OPUS-SOR	0.53962	0.51696	0.67011	0.73635
	OPUS-G	0.53958	0.51678	0.66973	0.73580
	HERMIT2D	0.55401	0.52536	0.66985	0.74763
T/16	OPUS-SOR	0.53984	0.52322	0.68225	0.72825
	OPUS-G	0.53980	0.52304	0.68192	0.72768
	HERMIT2D	0.55329	0.52973	0.68189	0.73980
T/20	OPUS-SOR	0.53889	0.52422	0.68454	0.72667
	OPUS-G	0.53882	0.52406	0.68418	0.72613
	HERMIT2D	0.55176	0.53046	0.68423	0.73843

Table (4.4). Test Case 4.2: Local Sinusoidal Perturbation. Thermal Neutron Flux at point B.

Test Case 4.3

This case is similar to Test case 4.2 with the exception that a two-region perturbation in regions R_2 and R_3 is introduced shown in Figure (4.3) and delayed neutron of six groups are included in the problem. The pertinant nuclear data can be found in Appendix B. The thermal group absorption cross section in each of the perturbed regions has the following form

$$\Sigma_{a,2}(t)=\Sigma_{a,2}(0)*[1.-0.1*\mathrm{Sin}(\frac{2\pi t}{T}) \quad , \quad r\epsilon R_2$$

$$\Sigma_{a,2}(t)=\Sigma_{a,2}(0)*[1.+0.1*\mathrm{Sin}(\frac{2\pi t}{T}) \quad , \quad r\epsilon R_3$$

$$T = 10.^{-3}$$

Using 121 node finite element model, the k_{eff} value for this problem was found to be 0.917216 by both OPUS codes. The time step size used for this calculation is $\blacktriangle t=T/8$. The results at points A, B, and C are given in Table (4.5).

t	OPUS-G			OPUS-SOR		
	$\Phi_A(t)$	$\Phi_B(t)$	$\Phi_C(t)$	$\Phi_A(t)$	$\Phi_B(t)$	$\Phi_C(t)$
0.	1.0000	0.43253	0.43253	1.0000	0.43253	0.43253
T/8	1.0001	0.42453	0.44106	1.0001	0.42452	0.44106
T/4	1.0016	0.40301	0.47089	1.0016	0.40300	0.47088
3T/8	1.0059	0.40670	0.47625	1.0059	0.40668	0.47621
T/2	1.0105	0.42322	0.45539	1.0104	0.42319	0.45535
5T/8	1.0122	0.44967	0.42082	1.0121	0.44963	0.42606
3T/4	1.0134	0.47571	0.40900	1.0133	0.47566	0.40895
7T/8	1.0173	0.48153	0.41098	1.0172	0.48146	0.41095
T	1.0214	0.46061	0.42768	1.0213	0.46055	0.42762

Table (4.5). Variation of Thermal Neutron Flux with time for Test Problem 4.3.

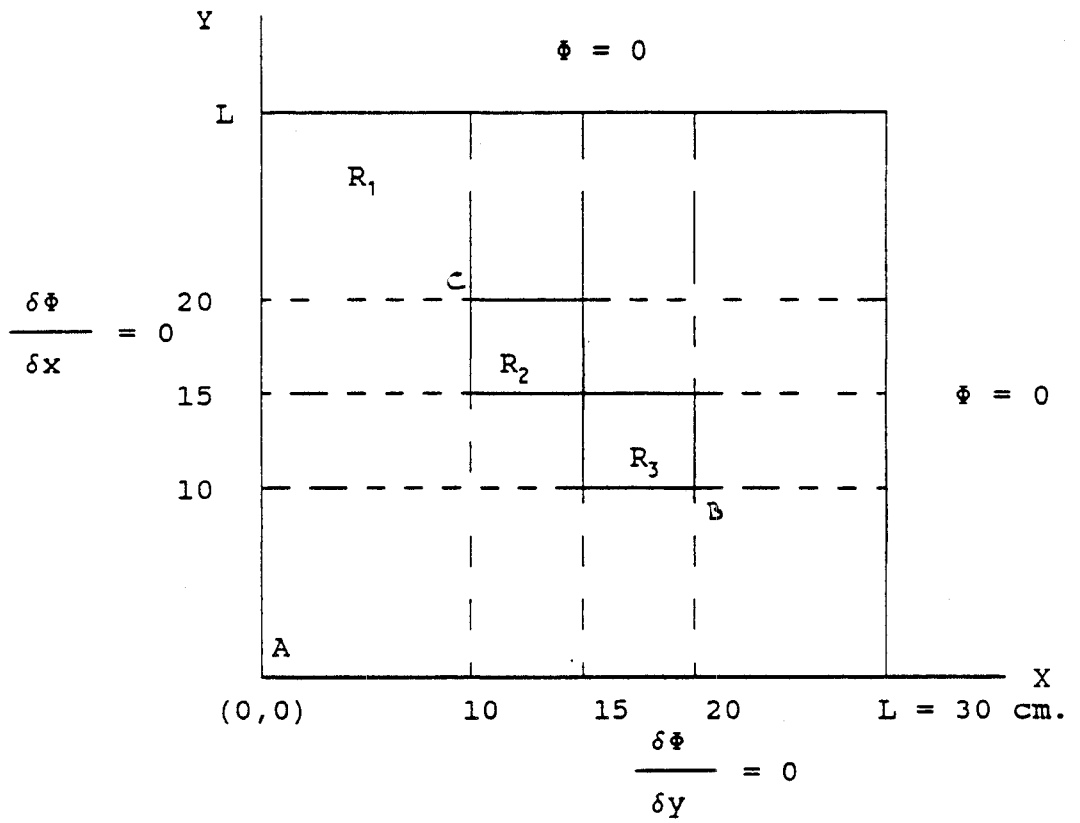

Figure (4.3). Test Problem III

Test Problem 4.4

Test problem 4.4 was originally designed by Bettis Atomic Power Laboratory to test the accuracy of TWIGL code[39],[40]. The reactor consists of a hollow square core surrounded by interior and exterior blankets. The configuration is shown in Figure (4.4). The reactor was divided into 800 subregions in the x and y directions with 8 cm intervals. A positive ramp change is induced by a ramp perturbation of the thermal absorption cross section for 0.2 sec in the four corner seed regions

$$\Sigma_{a,2}(t)=\Sigma_{a,2}(0)-0.0175*t, \quad 0\leq t\leq 0.2 \text{ sec.}$$

For $t>0.2$ sec, $\Sigma_{a,2}$ remains constant at $\Sigma_{a,2}(0.2)$. The perturbation corresponds to positive ramp reactivity change of about 2.5$/sec. over the interval $0 \leq t \leq 0.2$ second. The comparison of the subdomain method was performed against TWIGL, LUMAC, MITKIN, and SADI codes[39]. Among these codes, TWIGL is based on the implicit-difference approach in time using a conventional finite-difference spatial mesh. The other programs are implementations of Alternating-Direction Implicit (ADI) schemes. Tables (4.6) and (4.7) show the results at the center of the reactor and at the center of perturbed regions, respectively. The unperturbed initial flux values at t=0 are all normalized so that the consistency and accuracy of the codes can be easily seen. The results show that the codes utilizing ADI methods require very small time steps compared to those used by the OPUS programs. Only, the TWIGL code seems

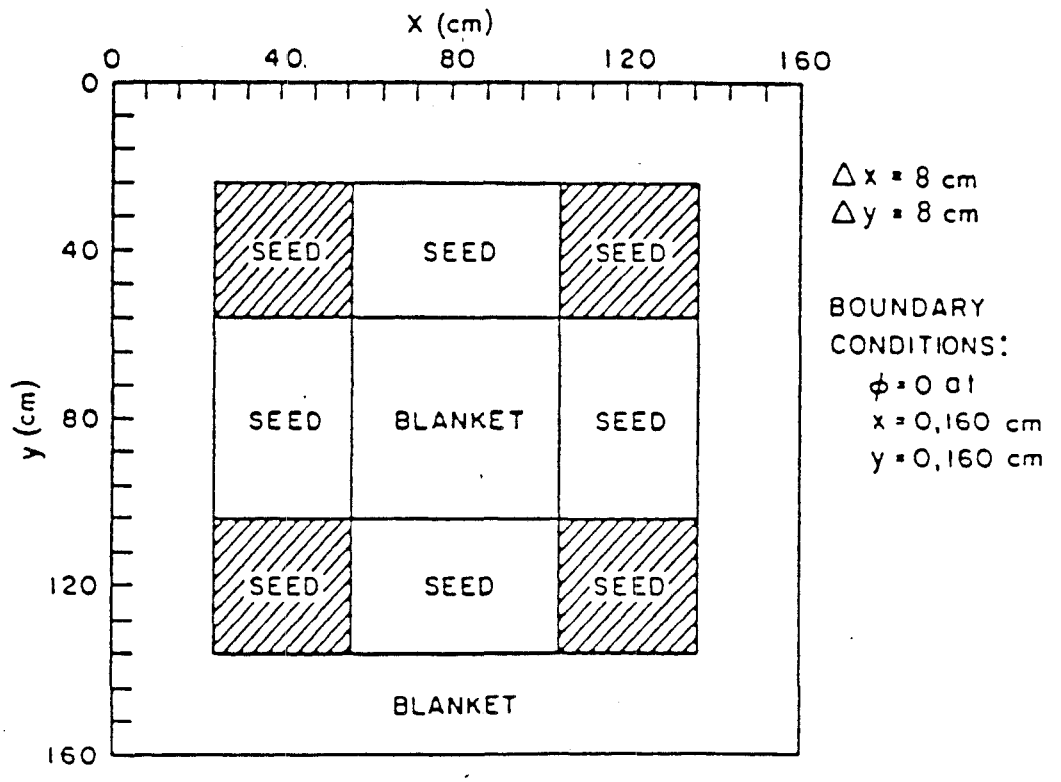

Figure (4.4). Configuration of Test Case IV.

Method	▲t	t (sec.)						
		0.	0.02	0.05	0.1	0.15	0.2	0.3
OPUS-SOR	0.01	16.75	17.37	18.75	21.71	25.89	32.22	34.35
	0.1	16.75	-	-	21.08	-	31.46	-
OPUS-G	0.01	16.75	17.39	18.78	21.73	25.92	32.26	34.38
	0.05	16.75	-	18.56	21.75	25.70	32.04	-
TWIGL	0.01	16.75	-	18.76	21.74	25.96	32.37	34.24
LUMAC	8.0E-4	16.75	-	-	21.73	-	32.49	34.83
MITKIN	0.001	16.75	17.26	18.79	21.75	25.95	32.31	33.57
SADI	2.5E-4	16.75	17.30	18.70	21.66	25.84	32.15	34.29

Table (4.6). Thermal Flux at the Center(11,11) of the Reactor

Time	TWIGL	OPUS-SOR		OPUS-G	
(sec)	▲t=0.01	▲t=0.01	▲t=0.1	▲t=0.01	▲t=0.05
0.	5.390	5.390	5.390	5.390	5.390
0.5	6.146	6.161	-	6.148	6.100
0.1	7.250	7.246	7.027	7.240	7.242
0.15	8.817	8.799	-	8.788	8.709
0.2	11.20	11.15	10.84	11.14	11.05
0.3	11.84	11.71	-	11.70	-

Figure (4.7). Thermal Flux at the Center(6,6) of the Seed

to be competitive. Also included are the results for a time step choice of ▲t=0.05 sec using OPUS-G and ▲t=0.1 sec using OPUS-SOR. The maximum deviation of the results for the ▲t=0.1 sec OPUS-SOR case compared to the ▲t=0.01 sec OPUS cases is about 2.7% and the method is still stable with the larger time step.

Test Case 4.5

Test problem 4.5 has the same reactor configuration as Test Case 4.4. However, for this problem positive step change of 50 cents is inserted at time zero by reducing the thermal capture cross section in the seed region. The results at the center of core are given in Table (4.8) and the comparable results from the other programs are available for comparison. The results of this study at the center of perturbed region is also included in Table (4.9). The time step choice of ▲t=0.005 sec is made for OPUS-G to demonstrate that the method still retains reasonable accuracy even for relatively large steps. For OPUS-SOR, the result for ▲t=0.0025 sec is included for comparison. The stability tests also showed that the subdomain method convergence is oscillatory when the choice ▲t=0.1 sec is made and the results are in error by about 37%.

Method	▲t	t (sec.)						
		0.	0.02	0.03	0.04	0.05	0.1	0.2
OPUS-SOR	0.0025	16.75	31.93	33.00	33.34	33.46	33.64	33.93
OPUS-G	0.005	16.75	32.28	33.23	33.48	33.55	33.65	33.92
TWIGL	0.001	16.75	30.78	32.40	33.15	33.54	34.01	34.31
LUMAC	8.0E-5	16.75	31.48	33.06	-	-	-	-
MITKIN	2.0E-4	16.75	31.50	33.13	33.97	34.63	-	-
SADI	2.5E-4	16.75	33.21	33.94	33.90	33.88	34.03	34.33

Figure (4.8). Thermal Flux at the Center (11,11) of Core.

Time (sec.)	OPUS-SOR ▲t=0.0025	OPUS-G ▲t=0.005
0.	5.39	5.39
0.02	10.95	10.86
0.03	11.24	11.24
0.04	11.36	11.38
0.05	11.49	11.44
0.1	11.62	11.56
0.2	11.72	11.70

Figure (4.9). Thermal Flux at the Center (6,6) of Perturbed Region.

CHAPTER 5

5.1. DISCUSSION AND CONCLUSIONS

A truly useful numerical technique must treat difficult practical problems successfully. A successful application requires knowledge of the inherent limitations of the numerical technique and knowledge of the likelihood of the errors when the technique is incorporated with a solution algorithm.

The application of the finite element method produces sparse matrices in which the location of the non-zero elements depends on the node numbering scheme. This situation is different from the one arising in the finite difference method where the matrices are tridiagonal with a regular structure. Additionally, the finite element method may yield matrices with large condition numbers. It is well known that when the condition number is large, a small change in right hand side vector or the system matrix can cause a relatively large change in the solution. In particular, the rate of convergence of an iterative solution method may be adversely effected

because of numerical instability.

The main objective of Chapter 3 was to investigate the likelihood of solution errors by evaluating the condition numbers for different type of cases. The results of Section 5 of Chapter 3 show that the condition numbers remain in the order of 10^2 even in the case of the test problem with a control rod, where the diffusion theory itself fails and introduces additional truncation error because of stronger singularities. This is in contrast to the cases encountered in some other fields. For example, in structural analysis, the condition number can be on the order of 10^5-10^7 for problems having a similar number of nodes to those used in this study[41]. This is a good indication that the matrices do not require preconditioning to reduce the size of condition number, nor the adaption of an iterative refinement scheme. The eigenvalue results of the Gauss and SOR methods show that without the application of preconditioning and iterative refinement methods, the results are in good agreement and confirm the predictions based upon condition numbers.

The results of application of the subdomain method were presented in Section 5 of Chapter 4. For several test cases, the subdomain method produced accurate solutions in close agreement with results of benchmark calculations carried out by the other methods. The most distinguishing characteristic of the subdomain method results is that they were obtained using relatively large time steps for problems with

oscillatory and ramp reactivity insertions where other methods require much smaller time steps. The method is notably superior over the methods used in TWIGL, LUMAC, MITKIN, and SADI programs in terms of using larger time steps. Even when $\Delta t=0.1$ sec is chosen, the subdomain method yields accurate results without any evidence of oscillatory convergence. This choice is 125 times larger than LUMAC's and 10 times higher than TWIGL's time step for similar problems. The satisfactory results produced by such long time steps is apparently the result of additional R_i and R_{i+1} terms in Eq.(4.24). In the case of a step reactivity insertion, smaller time intervals are required due to the rate of reactivity insertion, and any computational method must use smaller time steps to suitably represent the response of the system for such sudden changes. For the step insertion case, TWIGL and OPUS require similar length time steps and they both are superior to LUMAC, MITKIN, SADI methods for which accurate results can only be produced with the use of much smaller time steps.

Although the computational results of OPUS-SOR and OPUS-G are almost same, OPUS-G proved to be much slower for the time dependent calculations than OPUS-SOR. OPUS-G uses the static condensation algorithm. This algorithm is generally preferred in the cases with internal nodes. Internal nodes can be excluded when the matrix is appropriately collapsed. For eigenvalue calculations or for time-dependent calculations with time-independent coefficients, once the matrix is

condensed, the following outer iterations can be performed on the collapsed matrix. In these cases, the matrix only needs to be collapsed once and then all of the succeeding calculations are carried out on the reduced matrix. However, the problem involves time-dependent coefficients, the collapsing procedure must be followed after each time step. In this situation, OPUS-G, because of the increase number of matrix manipulation operations, becomes computationally inefficient compared to OPUS-SOR.

In general, the finite element method provides analytical treatment of space-time dependent problems which was difficult to achieve until recent years. The subdomain method in conjunction with the finite element method enables us to utilize larger time steps for the problems with time-dependent coefficients more accurately. Therefore, the method is appeared to be good choice to handle more realistic problems such as non-linear cases. Particularly, in the case of time-independent coefficients, the subdomain method is reduced to Padé(1,1) or Crank-Nicolson approximations. Hence, it may be considered as a more general approach.

It has been observed[42] that when simple trial functions are used, a good choice of an weighted residual method becomes more important in obtaining an accurate solution. In this investigation we have used a linear polynomial approximation and we have demonstrated the accuracy of the subdomain method. Increasing the degree of the polynomials, clearly, will result

in further accuracy but it remains an open question as to whether such a development would actually be worthwhile in view of programming effort versus gain in the accuracy. The method can be extended to non-linear problems by incorporating the major sources of feedback effects . Increase in the accuracy of 1-D load follow operation analysis[43] can also be achieved by utilizing bicubic type polynomials. Inclusion of 2D cylindrical coordinates to the OPUS codes in principal should require only a subroutine addition and the extension to 3D applications should also be straightforward.

REFERENCES

1. Courant, R., "Variational Methods for the Solution Problems of Equilibrium and Vibrations", Bull. Am. Math. Soc., 49, 1-23, (1943).
2. Turner, M.J., Clough, R.W., Martin, H.C., Topp, L.J., J. Aero. Sci., 23, 805, (1956).
3. Argyris, J.H., "Energy Theorems and Structural Analysis," Butterworth, London (1960).
4. Whiteman, J.R., "A Bibliography for Finite Elements," Academic Press, London and New York (1975).
5. Norrie, D.H. and de Vries, G., "Finite Element Bibliography," Plenum, New York (1976).
6. Buslik, A.J., "A Variational Principle for the Neutron Diffusion Equation Using Discontinuous Trial Functions," WAPD-T-1885, Bettis Atomic Power Laboratory (1965).
7. Kang, Chang Mu, "Finite Element Methods for Reactor Analysis," Nuclear Science and Engineering, **51**, 456 - 495 (1973).
8. Ohnishi, T., "Application of Finite Element Solution Technique to Neutron Diffusion and Transport Equations," Proceedings, Conf. on New Developments in Reactor

Mathematics and Applications, CONF-710302, Vol II, 273, Idaho Falls (1971).

9. Semenza, L.A., Lewis, E.E., Rossow, E.C., "A Finite Element Treatment of Neutron Diffusion," Trans. Am.Nucl. Soc., 14, 200 (1971).

10. Semenza, L.A., Lewis, E.E., and Rossow, E.C., "The Application of the Finite Element Method to the Multigroup Neutron Diffusion Equation," Nucl. Sci. Eng., 47, 302, (1972).

11. Kaper, Hans G., Leaf, Gary K., and Lindeman, Arthur J., "Application of Finite Element Methods in Reactor Mathematics; Numerical Solution of the Neutron Diffusion Equation," ANL-7925, Argonne National Laboratory (1972).

12. Kaper, H.G., Leaf, G.K., Lindemann, A.J.,"A Timing Comparison for Some High Order Finte Element Approximation Procedures and a Low Order Finite Difference Approximation Proedure for the Numerical Solution of the Multigroup Neutron Diffusion Equation," Nucl. Sci. Eng., 49, 27 (1972).

13. Kavenoky, A., "Neptune a Modular System for the Calculation of Light Water Reactors", in Computational Methods in Nuclear Engineering, CONF 750413, Vol. 2., 27, Charleston (1975).

14. Kavenoky, A., Lautard, J.J., "The Neutron Kinetics and Thermal Hydraulic Transient Computation Module of Neptune System CRONOS", Proceedings of the Topical Meeting on

Advances in Reactor Physics and Core Thermal Hydraulics, Kiamesha Lake, September (1982).

15. Kavenoky, A., Lautard, J.J., "A Finite Element Depletion Diffusion Calculation Method with Space Dependent Cross Sections", Nucl. Sci. Eng., 64, 563-575 (1977).

16. Schmidt, F.A.R., Franke, H.P., "Power Reactor Calculations with the Finite Element Program FEM 2D," Nucl. Sci. Eng., 56, 431 (1975).

17. Franke, H.P., Sapper, F., Schmidt, F.A.R., Atomkernenergie, 26, 158, (1975).California, Berkeley (1982).

18. Özgener, Bilge, "Application of the Finite Element Method To the Three Dimensional Multigroup Diffusion Equations," Technical University of Istanbul, Nuclear Energy Institute, Istanbul, Turkey (1988).

19. Hennart, J.P., Nucl. Sci. Eng., 64, 875-901 (1977).

20. Meade, Daniel, "Collocation Methods for Space Time Nuclear Reactor Dynamics", Ph.D. Dissertation, University of California, Berkeley (1982).

21. Kantorovich, L. V., Knylov, V. I., "Approximate Methods of Higher Analysis," Wiley (Interscience), New York (1959)

22. Biezeno, C. B., Koch, J. J., "Over een Nieuene Methode ter Berekening van Vlokke Platen met Toepassing Openkele voor de Tecniek Belangrijke Belastingsgevallen," Ing. Grav. 38, 25-36 (1923).

23. Biezeno, C. B., "Over een Vereenvoudiging en Over een Uitbreiding van de Methode van Ritze, "Christiaan Huygens 3, 69 (1923-1924).

24. Biezeno, C. B., "Graphical and Numerical Methods for Solving stress problems," Proceeding First Intern. Congr. Appl. Mech., pp 3-17, Delft (1924).

25. Courant, R., "Remark on ' Weighted Averages of the Residual' in Discussion Following Biezeno's Paper," Ibid.

26. Crandall, S. H., "Engineering Analysis," McGraw Hill, 1956, AMR 12(1959), Rev. 1122.

27. Brittan, R. O., "Some Problems in the Safety of Fast Reactors," ANL-5577 (1956).

28. Kaganova, J. J., "Numerical Solution of the One-group Space Independent Reactor Kinetics Equations for Neutron Density given the Excess Reactivity," ANL-6132 (Feb. 1960)

29. Yosida, K., "Functional Analysis," Springer, Berlin (1968).

30. Wilkinson, J. H., "The Algebraic Eigenvalue Problem," Clarendon Press, Oxford (1965).

31. Bers, L., Schechter, M., "Elliptic Equations," in Partial Differential Equations by Bers, L., Schechter, M., and John, F., Lectures in Applied Mathematics, Vol. III, Interscience Publishers (1964).

32. Segerlind, Larry J., "Applied Finite Element Analysis," John Wiley and Sons, 2 nd. edition, New York (1984).

33. Yuan, Y., "Finite Element Solution in Two Dimensional Transport Equation," Dissertation, Northwestern University, Evanston, Illinois (1976).

34. Zienkiewicz, O. C., "The Finite Element Method in Engineering Science," McGraw Hill, London, England (1971).

35. Roussopolos, P., Compte Rendus, 236, 1858 (1953).

36. Stacey, Weston M., "Space-Time Nuclear Reactor Kinetics," Academic Press, New York and London (1969).

37. Adalioglu, Ulvi, "Variational calculation and Synthesis Methods in Reactor Physics," ÇNAEM-NMB-33, Çekmece Nucleer Arastirma Merkezi, Istanbul, Turkey.

38. Kaplan, S., Marlowe, O. J., Bewick, J., "Application of Synthesis Techniques to Problems Involving Time Dependence," Nuclear Science and Engineering, 18, pp 163 (1964).

39. Hageman, L. A., Yasinsky, J. B.,"Comparison of Alternating-Direction Time-Differencing Methods with Other Implicit Methods for the Solution of the Neutron Group-Diffusion Equations," Nuclear Science and Engineering, 38, 8-32 (1969)

40. Wight, A. L., Hansen, K. F., Ferguson, D. R., "Application of Alternating-Direction Implicit Methods to the Space-Dependent Kinetics Equations," Nuclear Science and Engineering, 44, 239-251 (1971).

41. Schwarz, H. R., "Finite Element Methods," Series of

"Computational Mathematics and Applications", Series Editor: Whiteman, J. R., Academic Press (1988).

42. Fuller, Edward L., "Weighted-Residual Methods in Space Dependent Reactor Dynamics," Argonne National Laboratory, ANL-7565 (1969).

43. Park, Won Seok, "Optimization of the Nuclear Design of Light Gray Control Rods for PWR Load-Following Operations," Ph.D. Thesis, Department of Mechanical, Industrial and Nuclear Engineering, University of Cincinnati (1990).

APPENDIX A:

COMPUTER PROGRAM DESCRIPTIONS, SAMPLE INPUTS, AND LISTINGS

PROGRAM DESCRIPTION

OPUS codes were developed to simulate dynamic response of the nuclear reactors in 2D cartesian geometry. Since its design purpose is to apply the subdomain technique and to analyze its accuracy, it is considered to be a prototype model. However, its current features allows us to simulate many of the realistic problems for pre-analyses purpose. Its features and limitations are as follows:

1. It can handle maximum two neutron energy group in conjunction with up to 6 group of delayed neutrons. As opposed to many two group codes it includes the cases where there is a contribution from fission event to the thermal group as well.
2. Eigenvalue calculation can be performed prior to dynamic analysis or initial flux can be read in.
3. The following type of insertions can be handled through the perturbation of thermal group absorption cross section:
    a. Linear ramp reactivity insertion.
    b. Sinusoidal reactivity insertions.
    c. Step reactivity insertion.
    d. Prompt jump approximation.
4. It is assumed that delayed neutrons make contribution only to fast group.
5. Its calculation is performed for zero power cases. To simulate the response at certain power levels initial flux

should be provided.

A.1. Description of the Subroutines

MAIN : Reads the input data. Generates adjustable arrays and calls the subroutines.

MESH : Generates node coordinates and material property index array, assigns the nodes to the related elements. Generates boundary node information for the use of BOUND1 routine. This routine has a read statement and the dimension of the vectors should be changed in the case of increase in the dimension of adjustable arrays in MAIN.

INSPECTOR : Checks the most common input errors. This is especially important in the case of manual input of the coordinates and node numbers corresponding to triangular elements.

NULL : Initialize the matrices. Prepares the pointer matrices IPOINT in the case of OPUS-SOR and IPOINT and IBOOK in the case of OPUS-G for the future use in the routine STORE.

ELEM : Creates element matrices to assemble them in the routine STORE and includes the vacuum boundary condition if it is required.

STORE : Assembles the global matrices according to compact storage technique.

BOUND1 : Applies the penalty method described in Chapter III to the loss matrices.

EIGEN1 : It performs eigenvalue calculations. This routine calls OUTER, BOUND2, in both OPUS codes. It calls OMEG and VECITR in the case of OPUS-SOR and calls GAUSS in OPUS-G.

OUTER : It re-estimates the right hand side vector for the fast group and updates the scattering term for the thermal group at the beginning of each group calculation. This routine is only used for the eigenvalue calculation.

BOUND2 : Applies the penalty method to the right hand side group vectors at the beginning of each group calculation.

GAUSS : Solves the group equations according to static condensation technique. This routine is only used in OPUS-G.

OMEG : Calculates the acceleration parameter for the use of VECITR routine. The dimension of the matrix and vectors used in this routine should be changed in the case of dimension change of adjustable arrays in MAIN. This routine is only used in OPUS-SOR.

VECITR : Solves the group equations by the Successive Over Relaxation Technique. This routine is only used in OPUS-SOR.

COMB : This routine is used in time-dependent calculation. It adds the velocity matrices to the loss matrices of related groups.

BLOCK DATA : This is a block data and not a subroutine. Initial values of 6 group delayed neutron parameters and the speed of fast and thermal group neutrons are stored in this data library.

SOLVE : This routine is the main routine of time-dependent

calculation. It reads initial flux if required and calls CALC, INEST, SOURCE, BOUND2 routines in both OPUS codes, and calls GAUSS or VECITR routines in OPUS-G and OPUS-SOR codes, respectively.

CALC : This routine is called at the beginning of each time step. At t=0 it includes R terms to the right hand side and left hand side of (4.24). At each t_i it includes insertion rates corresponding to Δt time interval. Calculates precursor terms. Prepares the equation to the solution for related time step.

INEST : Performs the initial estimate of group fluxes at the beginning of each time step. It calls routines BOUND2, and GAUSS or VECITR.

SOURCE : This is only used in time-dependent calculations. It performs similar operations as the routine OUTER does.

A.2. Input Description

Inputs are format free for easy entry. Remarks are made for maximum input values. If there is a difference usage between OPUS-G and OPUS-SOR, it is indicated.

Line 1: ISKP,NDYN,NT1

ISKP : 0 (to call mesh generation program MESH. If this choice is made MESH generates boundary nodes assuming that the values of fluxes at these nodes are zero)

not 0 (to enter coordinates and node numbering manually). If this choice is made boundary nodes should be entered manually. It allows to impose any value to boundary nodes and to impose vacuum (or non-reentrant boundary conditions).

NDYN : 0 (for eigenvalue calculation only)

1 (for eigenvalue and time dependent calculations)

2 (for time-dependent calculation only)

NT1 : Only used by OPUS-G. The number of columns to allocate for the loss matrix.

If ISKP is not zero the following reads

Line 2 : NN,NE,NCOEF,NDBC,NG

If ISKP = 0 , Line 2 reads

Line 2 : NCOEF,NG

NN : number of nodes in the system (maximum 1681).

NE : number of elements in the system (maximum 3200).

NCOEF : number of materials in the system (maximum 5).

NDBC : number of sides on which no-reentrant boundary will be imposed (0 if none). Maximum 164.

NG : number of neutron energy groups (max. 2)

Line 3 through Line (3+NCOEF*NG): Fuel (or core) must be the first material to enter.

Reading procedure : do i=1,NCOEF

do j=1,NG

D(i,j),RCS(i,j),FCS(i,j),SCS(i,j),XG(i,j),DSIG(i,j)

D(i,j) : Diffusion constant for group j of the i th. material. In unit of cm.

RCS(i,j) : $\Sigma_a(i,j) + \Sigma_s(i,j)$. Removal cross section for group j of the i th. material (1/cm).

FCS(i,j) : Fission cross section for group j of the i th. material (1/cm).

SCS(i,j) : Scattering cross section for group j of the i th. material (1/cm).

XG(i,j) : Fission yield to group j of the i th. material.

DSIG(i,j) : The perturbation constant δA in Eq. (4.25) or (4.26). Enter 0 if NDYN = 0

Line (4+NCOEF*NG): This line may require different type of entries depending upon the assigned value for ISKP

If(ISKP.ne.0), the following is assumed to be read:

XC(1),XC(2),..,XC(i),...,XC(NN)

YC(1),YC(2),..,YC(i),...,YC(NN)

XC(i) : x coordinates of nodes. It should start from the first node i=1 and continue up to i=NN in sequential order.

YC(i) : y coordinates of nodes. It should start from the first node and continue up to i=NN.

If(ISKP.eq.0) the subroutine MESH reads the following

NY,NH

NY : number of lines to be generated parallel to x axis

NH : 0 if four sides have $\Phi = 0$ boundary condition.

1 if the sides 2 and 3 have $\Phi = 0$ boundary condition as shown in Figure (A.1).

If(ISKP.ne.0), Line (4+NCOEF*NG) is followed by :

Line (5+NCOEF*NG) : If(NDBC.ne.0), following is read otherwise go to if(NDYN.ge.1) conditional reading:

IDBC(i,1),IDBC(i,2),DBC(i)

. , . , .

IDBC(NDBC,1),IDBC(NDBC,2),DBC(NDBC)

IDBC(i,1) : The number of element where one of its side has non-reentrant boundary condition.

IDBC(i,2) : The side of i th. element on which non-reentrant boundary condition is to be imposed. This is illustrated by Figure (3.1) in Chapter 3. Input:

1 = Bottom side of the triangle

2 = Right side of the triangle

3 = Left side of the triangle

DBC(i) : The length of the side of the triangle.

If(ISKP.eq.0), Line (4+NCOEF*NG) is followed by :

Line (5+NCOEF*NG) :

Reading procedure: do i=1,ny

NX(i),XF(i),YF(i),XL(i),YL(i)

continue

NX(i) : the number of mesh regions along the line i

XF(i) : starting x coordinate of line i. If i=1 then XF(1) should be 0.

YF(i) : starting y coordinate of line i. If i=1 then YF(1) should be 0.

XL(i) : the end point x coordinate of line i.

YL(i) : the end point y coordinate of line i. If i=1 then YL(1) should be 0.

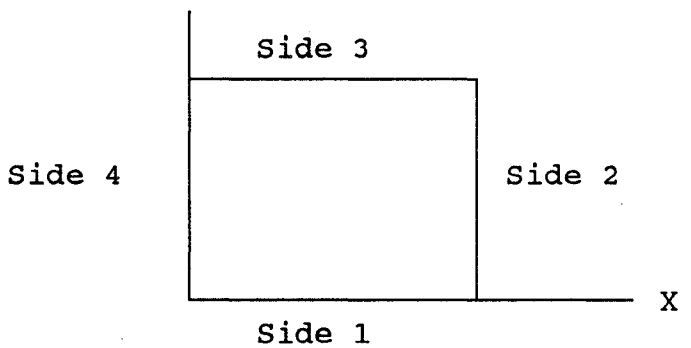

Figure (A.1). Number assignment of the sides

Line(6+NCOEF*NG) through Line (6+NCOEF*NG+(NY-1))

Reading procedure: do i=1,ny-1

(MID(i,j),j=1,NX(i))

continue

MID(i,j) : Material identification numbers. The numbers should be consistent with the readings of Line 3 through Line(3+NCOEF*NG).

If(NDYN.ge.1), The Last line is read

The Line (7+NCOEF*NG+(NY-1)): DELT,NPREC,FIN,VEL,EIGO

DELT : Δt time intervals.

NPREC : the number of precursor group (maximum 6)

FIN : Duration of insertion at which program is terminated.

VEL : 1. if the insertion is not prompt-jump approximation.

0. if the insertion is prompt-jump approximation.

EIGO : k_{eff} value (not $1/k_{eff}$) if NDYN=2, otherwise enter 1.

The Last Line: ININ, PERIOD

ININ : If ININ = 0 Ramp Reactivity Insertion.

If ININ = 1 Oscillatory Ramp.

If ININ = 2 Prompt-Jump Approximation.

PERIOD : $2*\pi*T$ If ININ = 1 [Eq.(4.26)]

0 Otherwise

The end of Input

PS : The kinetic parameters given in Appendix B are included in two separate BLOCK DATA statements. One is commented the other is ready to use. These parameters are not included as input options.

A.3. Sample Inputs

```
SAMPLE INPUT FOR OPUSSOR CODE
TEST CASE I OF CHAPTER 3.


0,0
1,2
2.6800045,5.4577E-02,3.0834E-02,4.07921E-02,0.575,0.
1.5787672,1.4496E-02,2.5200E-02,0.000000000,0.425,0.
7,1
6 0.  0.0000000 50.05 0.
6 0.  8.3416667 50.05 8.3416667
6 0. 16.683333  50.05 16.683333
6 0. 25.025000  50.05 25.025
6 0. 33.366667  50.05 33.3666667
6 0. 41.708333  50.05 41.708333
6 0. 50.050000  50.05 50.05
1,1,1,1,1,1
1,1,1,1,1,1
1,1,1,1,1,1
1,1,1,1,1,1
1,1,1,1,1,1
1,1,1,1,1,1
```

```
SAMPLE INPUT FOR OPUSSOR CODE
TEST CASE IV OF CHAPTER 4.
0,1
3,2
1.4 0.0200 0.007 0.01 1. 0.
0.4 0.1500 0.200 0.00 0. -0.175d-01
1.4 0.0200 0.007 0.01 1. 0.
0.4 0.1500 0.200 0.00 0. 0.
1.3 0.0180 0.003 0.01 1. 0.
0.5 0.0500 0.060 0.00 0. 0.
21,0
20  0.   0.  160.   0.
20  0.   8.  160.   8.
20  0.  16.  160.  16.
20  0.  24.  160.  24.
20  0.  32.  160.  32.
20  0.  40.  160.  40.
20  0.  48.  160.  48.
20  0.  56.  160.  56.
20  0.  64.  160.  64.
20  0.  72.  160.  72.
20  0.  80.  160.  80.
20  0.  88.  160.  88.
20  0.  96.  160.  96.
20  0. 104.  160. 104.
20  0. 112.  160. 112.
20  0. 120.  160. 120.
20  0. 128.  160. 128.
20  0. 136.  160. 136.
20  0. 144.  160. 144.
20  0. 152.  160. 152.
20  0. 160.  160. 160.
3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3
3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3
3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3
3,3,3,1,1,1,1,2,2,2,2,2,2,1,1,1,1,3,3,3
3,3,3,1,1,1,1,2,2,2,2,2,2,1,1,1,1,3,3,3
3,3,3,1,1,1,1,2,2,2,2,2,2,1,1,1,1,3,3,3
3,3,3,1,1,1,1,2,2,2,2,2,2,1,1,1,1,3,3,3
3,3,3,2,2,2,2,3,3,3,3,3,3,2,2,2,2,3,3,3
3,3,3,2,2,2,2,3,3,3,3,3,3,2,2,2,2,3,3,3
3,3,3,2,2,2,2,3,3,3,3,3,3,2,2,2,2,3,3,3
3,3,3,2,2,2,2,3,3,3,3,3,3,2,2,2,2,3,3,3
3,3,3,2,2,2,2,3,3,3,3,3,3,2,2,2,2,3,3,3
3,3,3,1,1,1,1,2,2,2,2,2,2,1,1,1,1,3,3,3
3,3,3,1,1,1,1,2,2,2,2,2,2,1,1,1,1,3,3,3
3,3,3,1,1,1,1,2,2,2,2,2,2,1,1,1,1,3,3,3
3,3,3,1,1,1,1,2,2,2,2,2,2,1,1,1,1,3,3,3
3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3
3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3
3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3
0.05,1,0.2,1.,1
0,0.
```

```
SAMPLE INPUT FOR OPUSG CODE
TEST CASE II OF CHAPTER 3


1,0,14
9,7,2,0,2
1.5,0.0623,0.0,0.06,1.,0.
0.4,0.2,0.218,0.0,0.,0.
1.2,0.101,0.0,0.1,1.,0.
0.15,0.02,0.0,0.0,0.,0.
0.,10.,10.,20.,20.,30.,40.,40.,40.
0.,0.,10.,20.,0.,0.,0.,10.,40.
1,1,1,2,3
2,1,2,4,3
3,1,2,5,4
4,2,5,6,4
5,2,6,8,4
6,2,6,7,8
7,2,8,9,4
3
7,0.,8,0.,9,0.
```

SAMPLE INPUT FOR OPUSG CODE
TEST CASE II OF CHAPTER 4

```
0,1,14
2,2
1.5 0.0623 0. 0.06 1. 0.
0.4 0.2 0.218 0. 0. 0.
1.5 0.0623 0. 0.06 1. 0.
0.4 0.2 0.218 0. 0. 0.04
11,1
10,0.,0.,25.,0.
10,0.,2.5,25.,2.5
10,0.,5.,25.,5.
10,0.,7.5,25.,7.5
10,0.,10.,25.,10.
10,0.,12.5,25.,12.5
10,0.,15.,25.,15.
10,0.,17.5,25.,17.5
10,0.,20.,25.,20.
10,0.,22.5,25.,22.5
10,0.,25.,25.,25.
1,1,1,1,1,1,1,1,1,1
1,1,1,1,1,1,1,1,1,1
1,1,1,1,1,1,1,1,1,1
1,1,1,1,1,1,1,1,1,1
1,1,1,1,2,2,1,1,1,1
1,1,1,1,2,2,1,1,1,1
1,1,1,1,1,1,1,1,1,1
1,1,1,1,1,1,1,1,1,1
1,1,1,1,1,1,1,1,1,1
1,1,1,1,1,1,1,1,1,1
0.00025,0,0.001,1.,1.
1,6283.1853
```

```
c----------          THE OPUSG.FOR----------c
c  is a FORTRAN code to solve both static and time-space
c  dependent coupled 2D - 2G neutron diffusion equation by the
c  finite element method ( linear polynomial approximation in time
c  and space). Gauss elimination with static condensation method is
c  utilized as the calculation algorithm.
c  The precursor equations are integrated and the Subdomain Method
c  for the multigroup equations is utilized in the
c  approximation of time dependency.  Written by Haluk UTKU.
c-----------------------------------------------------------------c
c  COMMON/BLK1/
c  nh = The number of sides on which essential or reflective boundary
c       conditions is to be applied. if 0 all sides have zero flux
c       if 1 side 2 and 3 have zero flux side 1 and 4 have reflective
c       boundary conditions.
c                      Y   Side 3
c
c
c
c              Side 4                  Side 2
c
c                                              X
c                      Side 1
c
c  mntl() = stores material identification numbers
c  nel(m,n) = stores element numbers(=m) and its node numbers(=n)
c            node numbering i,j,k should be counter clockwise in the
c            case of manual entry.
c  xc(),yc()  = stores x and y coordinates of nodes(must be in
c             numerical sequence  relative to the numbers in the case
c             of manual input.
c  mid(i,j) = image matrix of material identifacitions.
c  COMMON/BLK2/
c  xg(id,kg)  = fission yield to energy group kg of material(id)
c  fcs(id,kg)  = fission cross section of material in energy group kg
c  dsig(id,kg)  = perturbation constant introduced in material id
c  in energy group kg.( see Eq.(4.25) and (4.26). ONLY thermal group
c  COMMON/BLK3/
c  de() = diffusion constant
c  rcse() = removal cross section (absortion plus out scattering)
c  fcse() = fission cross section
c  scse()  = scattering cross section
c  ds() = perturbation constant
c  xge() = fission yield
c  COMMON/BLK4/
```

```
c  idbc(i,1)= element number with a derivative boundary condition
c  idbc(i,2)= side of the element with a derivative boundary
c  dbc()= (length of the side)
c  ndbc  = number of non-reentrant boundary conditions.
c  COMMON/BLK5/
c  plam() = array storing the decay constants of precursors
c  bet()  = array storing the fraction of contributions from delayed
c           neutrons.
c  v() = neutron velocities in group 1 and 2
c  beta = average of fractional contributions of delayed neutrons
c  COMMON/BLK6/
c  nn = total number of nodes in the system
c  ne = total number of elements in the system
c  nt = total number of columns allocated for the right hand side
c       matrices
c  nt1 = total number of columns allocated for the left hand side
c        group loss matrices
c  ng = number of neutron energy groups
c  eigo = previous eigenvalue estimate
c  delt = time steps
c  ndyn = number of pecursor groups
c  iskp = condition to choose manual input or MESH generation.
c  COMMON/BLK7/
c  ib() = array stroing node numbers having a boundary value.
c  bv() = array storing known flux values.
c  COMMON/BLK8/
c  eu11(kg,3,3)  = the kg group loss matrix of a triangular element
c  x(),y() = coordinates of triangular element.
c  v11(,) = scattering matrix of an element.
c  evel1(kg,3,3)  = element velocity matrices of related groups.
c  eu12(kg,3,3)  = element fission matrices of related groups.
c  tdab(,) = element perturbation matrix.
c  DIMENSION d(,),rcs(,),scs(,)
c  d(,) = matrix storing diffusion constants of related materials.
c  rcs(,) = matrix storing removal cross sections of related
c           materials.
c  scs(,) = matrix storing scattering cross sections of related
c           materials.
c  DIMENSION ick( ),arr( ),marr( ),flux( ),lnode( )
c  ick() = array used in error checking.
c  arr() = adjustable array storing global matrices
c  marr() = adjustable array storing pointer matrices
c  flux() = adjustable array storing flux and/or sorce vector arrays
c  lnode() = pointer array storing element node numbers
```

```
c  DIMENSION rins( )
c  rins() = adjustable array storing global perturbation matrix
c  DIMENSION dlnm( )
c  dlnm() = adjustable array storing infrmation related with
c           precursors.

c-------------------------------------------------------------------------c
      PROGRAM OPUSG
      IMPLICIT REAL*8(a-h,o-z)
      COMMON/BLK1/nh,mntl(3200),nel(3200,3),xc(1681),
     & yc(1681),mid(40,40)
      COMMON/BLK2/xg(5,2),fcs(5,2),dsig(5,2)
      COMMON/BLK3/de(2),rcse(2),fcse(2),scse(2),ds(2),xge(2)
      COMMON/BLK4/idbc(164,2),dbc(164),ndbc
      COMMON/BLK5/plam(6),bet(6),v(2),beta
      COMMON/BLK6/nn,ne,nt,nt1,ng,eigo,delt,ndyn,iskp
      COMMON/BLK7/ib(164),bv(164),la
      COMMON/BLK8/eu11(2,3,3),x(3),y(3),v11(3,3),evel1(2,3,3),
     &          evel2(2,3,3),eu12(2,3,3),tdab(3,3)
      DIMENSION d(5,2),rcs(5,2),scs(5,2)
      DIMENSION ick(1681),arr(267780),marr(90775),flux(13449),lnode(3)
      DIMENSION rins(16811)
      DIMENSION dlnm(13449)

c..... maximum number of columns to be stored: nmax - very important -
c.....  max. # of nodes  = nn = 1681 = (41x41)
c.....  ng = 2 (# of energy groups)

      nmax = 10
      nt = nmax
c-->  nt1 = 44 (max value for input)
c.....
c If OPUS.DAT is not on the same directory the system
c sets 'error code' equal to a negative number.

      open(5,file='opusg.dat',iostat=ju,err=777,status='old')
      open(6,file='opusg.out',status='unknown')
 777  print*,'error code=',mod(ju,256)

      read(5,*) iskp,ndyn,nt1
      if(iskp.eq.0) go to 1
      read(5,*) nn,ne,ncoef,ndbc,ng
      write(6,103) nn,ne,ncoef,ndbc,ng
      goto 2
```

```
1    read(5,*) ncoef,ng
     write(6,100) ncoef,ng

2    write(6,101)
     do 5 i=1,ncoef
     do 5 j=1,ng
     read(5,*) d(i,j),rcs(i,j),fcs(i,j),scs(i,j),xg(i,j),dsig(i,j)
     write(6,102) i,j,d(i,j),rcs(i,j),fcs(i,j),scs(i,j),xg(i,j)
5    continue
     if(iskp.eq.0) then
     CALL MESH
     goto 20
     endif
     write(6,105)
     read(5,*) (xc(i),i=1,nn)
     read(5,*) (yc(i),i=1,nn)
     write(6,106) (i, xc(i), yc(i),i=1,nn)

c..... hardcopy  of the element nodal data .....

     write(6,107)
     nid=0
       do 9 kk=1,ne
     read(5,*) n,mntl(kk),(nel(n,i),i=1,3)
     if((n-1).ne.nid) write(6,108) n
     nid=n
9    write(6,109) n,mntl(kk),(nel(n,i),i=1,3)

     read(5,*) la
     if(la.le.0) goto 20
209  read(5,*) (ib(i),bv(i),i=1,la)

20   continue

c..... hardcopy  of the derivative boundary conditions .....

     if(ndbc.eq.0) goto 14
     write(6,110)
     do 12 i=1,ndbc
     read(5,*) idbc(i,1),idbc(i,2),dbc(i)
12   write(6,*) idbc(i,1),idbc(i,2),dbc(i)
14   continue

     if(ndyn.ge.1) then
```

```
      read(5,*) delt,nprec,fin,vel,eigo
      read(5,*) inin,period
      endif

c.....  generation of adjustable arrays .....

      ngnn = ng*nn
      nnmax = nn*nmax
      ngnnm1 = ng*nn*nt1
      ngnnm = ng*nnmax
      na1 = 1
      na2 = na1 + ngnnm1
      na3 = na2 + ngnnm
      na4 = na3 + ngnnm
      na5 = na4 + ngnnm
      na6 = na5 + nnmax
      na7 = na6 + nn
      me1 = 1
      me2 = me1 + ngnn
      me3 = me2 + ngnn
      me4 = me3 + ngnn
      ka1 = 1
      ka2 = ka1 + nn*nprec
      ia1 = 1
      ia2 = ia1 + nn*nt


      CALL INSPECTOR(ick,lnode)
      CALL NULL(arr(na1),arr(na2),arr(na3),
     & arr(na4),arr(na5),rins(na1),marr(ia1),marr(ia2))

      do 25 kk=1,ne

      do 22 i=1,3
      lnode(i)=nel(kk,i)
      j=lnode(i)
      x(i)=xc(j)
   22 y(i)=yc(j)
c....
      ii=mntl(kk)
c....
      do 23 ig=1,ng
      de(ig)=d(ii,ig)
      rcse(ig)=rcs(ii,ig)
```

```
      fcse(ig)=fcs(ii,ig)
      scse(ig)=scs(ii,ig)
      xge(ig)=xg(ii,ig)
      ds(ig) = dsig(ii,ig)
   23 continue

c...  calculation of element matrices ...

      CALL ELEM(kk,vel)

c...  store element matrices in global matrices....

      CALL STORE(arr(na1),lnode,arr(na2)
     & ,marr(ia1),marr(ia2),arr(na3),arr(na4),arr(na5),rins(na1))

   25 continue

      CALL BOUND1(arr(na1))

c  if ndyn=2  initialflux is read in

      if(ndyn.eq.2) goto 30

      CALL EIGEN(flux(me1),flux(me2),flux(me3)
     & ,arr(na1),arr(na4),arr(na5),marr(ia1),marr(ia2),ick)

c  if ndyn  = 0 no time dependent  calculation

   30 if(ndyn.eq.0) goto 900

c..... start calling routines for transient calculations

      CALL COMB(marr(ia1),arr(na1),arr(na2)
     & ,arr(na3))

c  max loop  = 500 in arr(na7)

      loop  = fin/delt + 1.1
      npre  = nprec
      if(npre.eq.0) npre  = 1

      CALL SOLVE(loop,arr(na1),marr(ia1),
     & marr(ia2),arr(na2),flux(me1),flux(me2),flux(me3),
     & flux(me4),arr(na4),rins(na1),dlnm(ka1),dlnm(ka2),arr(na5),
```

```
     & arr(na7),ick,nprec,npre,inin,period,arr(na6))


c..... output formats .....

  100  format(1x/10x,'NCOEF=',i2,2x,'NO. OF GROUPS=',i2,2x
     + ,17h# OF BOUND.COND.,i3)
  103  format(1x/10x,'NN =',i5/10x,'NE =',i5/10x,'NCOEF=',i2,
     + 2x,'NDBC='i3,2x,'NO. OF GROUPS=',i2)
  101  format(/10x,'EQUATIONCOEFFICIENTS'/3x,'MATERIAL',2x,
     + 'GROUP',2x,9hDIF.CONS,2x,'REMOVAL',4x,'FIS.X SEC.',
     +  1x,'SCAT. X SEC.',2x,'FISS. YIELD')
  102  format(7x,i2,4x,i2,2x,5d12.5)
  105  format(//10x,'NODAL COORDINATES'/10x,'NODE',7x,1hX,14x,1hY)
  106  format(10x,i4,2d15.5)
  107  format(//10x,'ELEMENT DATA'/16x,
     + 'NE',4x,'MNTL',4x,'NODE NUMBERS')
  108  format(10x,7hELEMENT,I4,16HNOT IN SEQUENCE)
  109  format(15x,i3,5x,i3,2x,4i4)
  110  format(//10x,34hDERIVATIVEBOUNDARYCONDITIONDATA/10x,
     + 7hELEMENT,4x,4hSIDE,7x,2hSL)

  900  stop
       end
c------------------------- end of the subroutine -|

       SUBROUTINE ELEM(kk,vel)

       IMPLICIT REAL*8(a-h,o-z)
       COMMON/BLK2/xg(5,2),fcs(5,2),dsig(5,2)
       COMMON/BLK3/de(2),rcse(2),fcse(2),scse(2),ds(2),xge(2)
       COMMON/BLK4/idbc(164,2),dbc(164),ndbc
       COMMON/BLK5/plam(6),bet(6),v(2),beta
       COMMON/BLK6/nn,ne,nt,nt1,ng,eigo,delt,ndyn,iskp
       COMMON/BLK7/ib(164),bv(164),la
       COMMON/BLK8/eu11(2,3,3),x(3),y(3),v11(3,3),evel1(2,3,3),
     &         evel2(2,3,3),eu12(2,3,3),tdab(3,3)
       DIMENSION b(3),c(3)
c-----------------------------------------
       b(1)=y(2)-y(3)
       b(2)=y(3)-y(1)
       b(3)=y(1)-y(2)
       c(1)=x(3)-x(2)
       c(2)=x(1)-x(3)
```

```
      c(3)=x(2)-x(1)
      ar2=x(2)*y(3)+x(3)*y(1)+x(1)*y(2)-x(2)*y(1)
     +         -x(3)*y(2)-x(1)*y(3)
      if(abs(ar2).lt.0.0001) goto 5
      dia = ar2/6.
      fa = dia/4.
      do 20 i=1,3
      do 20 j=1,3

      a = 1.0
c     ab = 0.
      if(i.eq.j) a = 2.
c     if(i.eq.j) ab = 1.
      gt=( b(i)*b(j) + c(i)*c(j) )/(2.*ar2)
      fact = a * fa
c     diag = ab * dia
c
      do 15 kg = 1,ng
      kgg = kg
      if(ng.eq.1) kgg = 2
      eu11(kg,i,j) = de(kg) * gt + rcse(kg) * fact
      eu12(kg,i,j) = fcse(kg) * fact
      if(ndyn.eq.0) goto 15
c     evel1(kg,i,j) = vel * diag/v(kgg)
      evel1(kg,i,j) = vel * fact/v(kgg)
   15 continue
      v11(i,j) = scse(1) * fact
      if(ndyn.eq.0) goto 20
      tdab(i,j) = ds(ng) * fact
c
   20 continue

      if(ndbc.eq.0) goto 7

c THE FOLLOWINGcan be used in the case of non-reentrant
c boundary condition for the thermal group
c if it has to be used MAIN program should read boundary numbers
c and SUBROUTINEMESH has to be modified not to create bundary nodes


      do 11 i=1,ndbc
      if(idbc(i,1).ne.kk) goto 11
      j=idbc(i,2)
      k=j+1
```

```
      if(j.eq.3) k=1
      eu11(2,j,j)=eu11(2,j,j)+dbc(i)/(3.*2.)
      eu11(2,j,k)=eu11(2,j,k)+dbc(i)/(6.*2.)
      eu11(2,k,j)=eu11(2,j,k)
      eu11(2,k,k)=eu11(2,k,k)+dbc(i)/(3.*2.)
   11 continue
    7  continue

   35 return

    5 write(6,6) kk
    6 format(//10x,19hthe area of element,i4,
     + 20h is less than 0.0001/10x,21h the node numbers are,
     + 19h in the wrong order/10x,20hexecution terminated)
      stop
      end
c--------------------------- end of the subroutine -|

      SUBROUTINE STORE(u11,lnode,b11,ipoint,ibook,
     & vel1,u12,scm,tdep)

      IMPLICIT REAL*8(a-h,o-z)
      COMMON/BLK2/xg(5,2),fcs(5,2),dsig(5,2)
      COMMON/BLK6/nn,ne,nt,nt1,ng,eigo,delt,ndyn,iskp
      COMMON/BLK8/eu11(2,3,3),x(3),y(3),v11(3,3),evel1(2,3,3),
     &      evel2(2,3,3),eu12(2,3,3),tdab(3,3)
      DIMENSION u11(ng,nn,nt1),lnode(3),b11(ng,nn,nt)
      DIMENSION ipoint(nn,nt),vel1(ng,nn,nt),u12(ng,nn,nt)
      DIMENSION tdep(nn,nt),scm(nn,nt),ibook(nn,nt1)

c   nt = max number of columns to be stored
c   nnpe = # of nodes per element
c-----------------------------------------------------
      nnpe = 3

c..... store element matrices .....
c..... first row .....

      i = 0
      do 50 jj=1,nnpe
      irow = lnode(jj)
      i = i + 1

c..... then columns .....
```

```
      ik = 0
      do 55 kj= 1,nnpe
      icol =lnode(kj)
      ik = ik + 1

c  skip if term below diagonal

      if(icol-irow)70,60,60
 60    continue

c.....  search  ibook for column no.

      do 75 m=1,nt
      if(ibook(irow,m)-icol)80,90,80
 80    if(ibook(irow,m))85,85,75
 75    continue

c.... found a blank now store icol .....

 85    ibook(irow,m)=icol

c.....  now store element matrix .....

 90          continue
      do 91 kg =1,ng
      u11(kg,irow,m) = u11(kg,irow,m) + eu11(kg,i,ik)
 91   continue

c   the followings are stored globaly since
c  they constitute the r.h.s. vector ....

c .....  search  ipoint for column no. .....

 70   continue
      do 95 mm=1,nt
      if(ipoint(irow,mm)-icol)100,105,100
 100  if(ipoint(irow,mm))110,110,95
 95   continue

c.....  found a blank now store icol .....

 110  ipoint(irow,mm) = icol

c.....  now store element stiffness  matrix .....
```

```
  105  do 106 kg = 1,ng
       u12(kg,irow,mm) = u12(kg,irow,mm) + eu12(kg,i,ik)
       if(ndyn.eq.0) goto 106
       b11(kg,irow,mm) = b11(kg,irow,mm) + eu11(kg,i,ik)
       vel1(kg,irow,mm) = vel1(kg,irow,mm) + evel1(kg,i,ik)
  106  continue
       scm(irow,mm) = scm(irow,mm) + v11(i,ik)
       if(ndyn.eq.0) goto 55
       tdep(irow,mm) = tdep(irow,mm) + tdab(i,ik)

c..... end loop on columns .....

   55  continue

c..... end loop on rows .....

   50  continue

       return
       end

c------------------------- end of the subroutine -|

       SUBROUTINE BOUND1(u11)

       IMPLICIT REAL*8(a-h,o-z)
       COMMON/BLK6/nn,ne,nt,nt1,ng,eigo,delt,ndyn,iskp
       COMMON/BLK7/ib(164),bv(164),la
       DIMENSION u11(ng,nn,nt1)

c      apply known nodal flux values:
c      ib = node number on which bound. condition will be imposed.
c      bv = known boundary flux value
c-----------------------------------------------------

   15        do 20 n=1,la
         irow=ib(n)
       do 20 kg =1,ng
          u11(kg,irow,1) = u11(kg,irow,1) * 10.**15
   20  continue
       return
       end
```

```
c------------------------- end of the subroutine -|

      SUBROUTINE BOUND2(kg,u11,qs1)

      IMPLICIT REAL*8(a-h,o-z)
      COMMON/BLK6/nn,ne,nt,nt1,ng,eigo,delt,ndyn,iskp
      COMMON/BLK7/ib(164),bv(164),la
      DIMENSION u11(ng,nn,nt1),qs1(ng,nn)

c     apply known nodal flux values:
c     ib = node number on which bound. condition will be imposed.
c     bv = known boundary flux value
c-----------------------------------------------------

  15        do 20 n=1,la
        irow=ib(n)
  17    qs1(kg,irow)  = u11(kg,irow,1)*bv(n)
  20  continue
      return
      end

c------------------------- end of the subroutine -|

      SUBROUTINE EIGEN(fli1,qs1,qs1m,u11,u12
     & ,scm,ipoint,ibook,imet)

      IMPLICIT REAL*8(a-h,o-z)
      COMMON/BLK2/xg(5,2),fcs(5,2),dsig(5,2)
      COMMON/BLK3/de(2),rcse(2),fcse(2),scse(2),ds(2),xge(2)
      COMMON/BLK6/nn,ne,nt,nt1,ng,eigo,delt,ndyn,iskp
      COMMON/BLK7/ib(164),bv(164),la
      DIMENSION fli1(ng,nn),qs1(ng,nn),qs1m(ng,nn)
      DIMENSION ipoint(nn,nt),ibook(nn,nt1)

      do 3 n =1,nn
      do 2 m = 1,nt
      if(ipoint(n,m).ne.0) goto 2
      ipoint(n,1) = m-1
      go to 1
  2   continue
  1   continue
  3    continue
```

```
      eigo=1.
      scale=sqrt(real(nn))

c..... assign  unities to initialfluxes .....

      do 4 kg =1,ng
      do 4 i=1,nn
      fli1(kg,i)=1./scale
    4 continue
      den = 1.

      xgs=(xg(1,2)/xg(1,1))**2.
      itr = 0          ! iteration index

      CALL OUTER(1,u12,scm,fli1,qs1,ipoint)

   16 do 7 n = 1,nn
    7 qs1m(1,n) = qs1(1,n)


      do 18 kg =1,ng
      if(kg.eq.1) goto 10
      CALLOUTER(2,u12,scm,fli1,qs1,ipoint)
   10 CALL BOUND2(kg,u11,qs1)
      CALL GAUSS(kg,u11,qs1,fli1,ibook,imet)
   18 continue

      CALL OUTER(1,u12,scm,fli1,qs1,ipoint)

      xnum = 0.
      den = 0.
      do 25 n =1,nn
      if(ng.eq.2) goto 27
      xnum = xnum + qs1(ng,n)
      den = den + qs1m(ng,n)
      goto 25
   27 xnum = xnum + qs1(1,n) + qs1(1,n)*xg(1,2)/xg(1,1)
      den = den + qs1m(1,n) + qs1m(1,n)*xg(1,2)/xg(1,1)
   25 continue

      eign = eigo*(xnum/den)
      rel = abs(eigo/eign - 1.)
      if(rel.lt.1.d-06) goto 30
      eigo = eign
```

```
      den = dsqrt(den)
      itr = itr + 1
      goto 16

   30 write(6,*) 'eign',eign,' itr',itr
      write(6,101)
  101 format(//10X,21HCALCULATEDQUANTITIES/12X,
     + 21HNODAL VALUES FOR FLUX)
      do 32 i =1,nn
   32 write(6,102) i,(fli1(kg,i),kg=1,ng)
  102 format(12X,i4,d14.5,d14.5)
      return
      end

c------------------------- end of the subroutine -|

      SUBROUTINE GAUSS(kg,u11,qs1,fli1,ibook,imet)

      IMPLICIT REAL*8(a-h,o-z)
      COMMON/BLK6/nn,ne,nt,nt1,ng,eigo,delt,ndyn,iskp
      DIMENSION u11(ng,nn,nt1),qs1(ng,nn),fli1(ng,nn)
      DIMENSION ibook(nn,nt1),imet(nn)
      DIMENSION aux(1681,45),jbook(1681,44)


      do 125 ic=1,nn
      do 125 ir=1,nt1
      aux(ic,ir+1)=u11(kg,ic,ir)
      jbook(ic,ir)=ibook(ic,ir)
  125 continue

      do 126 ic=1,nn
  126 aux(ic,1)=qs1(kg,ic)

      do 5 ic=1,nn
    5 imet(ic)=0.

      do 15 m=1,nt1
   15 imet(m)=ibook(1,m)
      do 34 n=2,nn
      do 28 m=1,nt1
      if(imet(m)-n+1)20,28,20
   20 do 24 l=1,nt1
      if(ibook(n,l))23,26,23
```

```
 23   if(ibook(n,l)-imet(m))24,28,24
 24    continue
      write(6,101) n
101    format(37h allowable space  exceeded  in matrices,'row  #=',i4)
      stop
 26    ibook(n,l)=imet(m)
 28   continue
 30   do 32 m=1,nt1
 32    imet(m)=ibook(n,m)
 34    continue
 36    neq=iabs(nn)
      neqm=neq-1


c  loop on equations:

      do 52 i=1,neqm
c
c  modify r.h.s. vector
c

      qs1(kg,i)  = qs1(kg,i)/u11(kg,i,1)

c
c  loop on row to be eliminated
c
      do 46 m =2,nt1
      in =ibook(i,m)
      if(in)365,48,365
c
c  seek appropriate rows
c
 365    do 38 n =1,nt1
      ia = ibook(in,n)
      if(ia)37,40,37
  37  imet(ia) = n
  38  continue
  40  continue

      temp  = u11(kg,i,m)/u11(kg,i,1)

c
c  loop on columns  to be eliminated
c
```

```
      do 42 n =1,nt1
      ia = ibook(i,n)
      if(ia)45,44,45
   45 if(ia-in)42,41,41
   41 im = imet(ia)
c
c     modify term of matrix
c     note that had the matrix not been symmetric
c     then u11(i,n) would have been u11(n,i) and u11(i,m)
c     would have been u11(m,i)
c
      u11(kg,in,im) = u11(kg,in,im) - temp*u11(kg,i,n)
   42  continue
c
c  modify source vector
c
   44  qs1(kg,in) = qs1(kg,in) - qs1(kg,i) * u11(kg,i,m)
   46  continue
   48 continue
c
c  reset row for back-substituion
c
      do 50 m =2,nt1
      u11(kg,i,m) = u11(kg,i,m)/u11(kg,i,1)
   50 continue
   52 continue
      qs1(kg,neq) = qs1(kg,neq)/u11(kg,neq,1)
c
c  back-substitute
c
      do 56 ib =1,neqm
      i = neq -ib
      do 54 m =2,nt1
      j = ibook(i,m)
      if(j)53,56,53
   53 qs1(kg,i) = qs1(kg,i) - u11(kg,i,m)*qs1(kg,j)
      fli1(kg,i) = qs1(kg,i)
   54 continue
   56 continue


   70 do 150 ic=1,nn
      do 150 ir=1,nt1
      u11(kg,ic,ir)=aux(ic,ir+1)
```

```
      ibook(ic,ir)=jbook(ic,ir)
  150 continue

      do 160 ic=1,nn
  160 qs1(kg,ic)=aux(ic,1)

      return
      end

c------------------------- end of the subroutine -|

      SUBROUTINE OUTER(kg,u12,scm,fli1,qs1,ipoint)

      IMPLICIT REAL*8(a-h,o-z)
      COMMON/BLK2/xg(5,2),fcs(5,2),dsig(5,2)
      COMMON/BLK6/nn,ne,nt,nt1,ng,eigo,delt,ndyn,iskp
      DIMENSION u12(ng,nn,nt),scm(nn,nt),fli1(ng,nn),qs1(ng,nn)
      DIMENSION ipoint(nn,nt)

      kgg = 2
      if(ng.eq.1.or.kg.eq.2) kgg = 1
      do 5 n =1,nn
      add1 = 0.
      num = ipoint(n,1)
      do 6 m =1,num
      l = ipoint(n,m)
      if(m.eq.1) l = n
      if(kg.eq.2) go to 7
      if(ng.eq.1) then
      add1 = add1 + u12(kg,n,m)*fli1(kg,l)
      elseif(ng.eq.2) then
      add1 = add1 + u12(kgg,n,m)*fli1(kgg,l) + u12(kg,n,m)*fli1(kg,l)
      endif
      goto 6
    7 add1 = add1 + scm(n,m)*fli1(kgg,l)
    6 continue
      if(kg.eq.1) then
      qs1(kg,n) = xg(1,1)*add1/eigo
      elseif(kg.eq.2) then
      qs1(kg,n) = add1 + qs1(kgg,n)*xg(1,2)/xg(1,1)
      endif
    5 continue
      return
      end
```

```
c------------------------- end of the subroutine -|

      SUBROUTINE COMB(ipoint,u11,b11,vel1)

      IMPLICIT REAL*8(a-h,o-z)
      COMMON/BLK2/xg(5,2),fcs(5,2),dsig(5,2)
      COMMON/BLK5/plam(6),bet(6),v(2),beta
      COMMON/BLK6/nn,ne,nt,nt1,ng,eigo,delt,ndyn,iskp
      COMMON/BLK7/ib(164),bv(164),la
      DIMENSION u11(ng,nn,nt1),ipoint(nn,nt),
     &          b11(ng,nn,nt),vel1(ng,nn,nt)

c  ib = node numbers,  bv = known flux value
c-------------------------------------------------------

      do 50 n =1,nn
      do 80 m = 1,nt
      if(ipoint(n,m).ne.0) goto 80
      ipoint(n,1) = m-1
      go to 20
   80  continue
   20  continue
   50  continue

      delth = delt/2.
      do 72 kg = 1,ng
      do 71 i=1,nn
      jj = 0
      num = ipoint(i,1)
      do 71 j=1,num
      b11(kg,i,j) = vel1(kg,i,j) - delth*b11(kg,i,j)
      l = ipoint(i,j)
      if(j.eq.1) l = i
      if(l.lt.i)goto 71
      jj = jj + 1
      u11(kg,i,jj) = vel1(kg,i,j) + delth * u11(kg,i,jj)
   71 continue
   72 continue

      return
      end
c------------------------- end of the subroutine -|

c the following is for time-dependent Test problems I,II,III
```

```
c       BLOCK DATA

c       IMPLICIT REAL*8(a-h,o-z)
c       COMMON/BLK5/plam(6),bet(6),v(2),beta
c       DATA plam /0.0127,0.0317,0.115,0.311,1.4,3.87     /
c       DATA bet/0.000285,0.0015975,0.00141,0.0030525,0.00096,0.000195/
c       DATA beta,v  /0.0075,1.E08,2.2E05/
c       END

c the following is for Test problem IV and V

        BLOCK DATA

        IMPLICIT REAL*8(a-h,o-z)
        COMMON/BLK5/plam(6),bet(6),v(2),beta
        DATA plam / 8.d-02,0.,0.,0.,0.,0.    /
        DATA bet/ 0.0075,0.,0.,0.,0.,0.    /
        DATA beta,v  /0.0075,1.E08,2.2E05/
        END

c--------------------------- end of the block data -|

        SUBROUTINE SOLVE(loop,u11,ipoint,ibook,
     & b11,fli1,qs1,fli,flim,u12,tdep,c,pv,scm,t,ick,nprec,npre
     & ,inin,period,sumcj)

        IMPLICIT REAL*8(a-h,o-z)
        COMMON/BLK5/plam(6),bet(6),v(2),beta
        COMMON/BLK6/nn,ne,nt,nt1,ng,eigo,delt,ndyn,iskp
        DIMENSION u11(ng,nn,nt1),ipoint(nn,nt),b11(ng,nn,nt)
     &          ,u12(ng,nn,nt),ibook(nn,nt1)
        DIMENSION fli1(ng,nn),qs1(ng,nn),flim(ng,nn),fli(ng,nn),t(loop)

        if(ndyn.ne.2) goto 6
        do 5 i = 1,nn
        read(5,102)  jj,(fli1(kg,i),kg=1,ng)
    5   continue
    6   do 10 i =1,nn
        do 10 kg=1,ng
        flim(kg,i) = fli1(kg,i)
   10   fli(kg,i) = flim(kg,i)

        do 12 ka = 1,ng
        do 12 kb = 1,nn
```

```
      do 12 kc = 1,nt
      u12(ka,kb,kc) = u12(ka,kb,kc)/eigo
   12 continue

c... loop .....

      alp1 = 0.
      alp2 = 0.
      t(1) = 0.

      do 100 nc=2,loop
      t(nc) = t(nc - 1) + delt
      itr = 0

      CALL CALC(fli,flim,ipoint,nprec,npre,sumcj,qs1
     & ,u11,u12,b11,scm,tdep,c,pv,t,nc,loop,alp1,alp2,
     & inin,period)

      if(nc.lt.3) goto 80
      do 20 kg=1,ng
      do 20 i=1,nn
      flim(kg,i)=fli(kg,i)
   20 continue

   80 if(ng.eq.1) goto 50
      CALL INEST(nc,fli1,flim,fli,u11,scm,b11,qs1
     & ,ipoint,ibook,ick)

   50 do 95 kg = 1,ng

      if(ng.eq.1) goto 51
      CALL SOURCE(kg,u12,fli1,qs1,pv,scm,ipoint,alp2)
   51 CALL BOUND2(kg,u11,qs1)
      if(kg.gt.1) goto 52

   52 CALL GAUSS(kg,u11,qs1,fli1,ibook,ick)

   95 continue
      itr = itr + 1
      if(ng.eq.1) goto 114
      den = 0.
      xnum = 0.
      do 110 i = 1,nn
      xnum = xnum + ( fli1(ng,i)- fli(ng,i))**2
```

```
      den = den + fli(ng,i)**2
  110 continue
      xnum = sqrt(xnum)
      den = sqrt(den)
      div = xnum/den
  114 do 115 kg=1,ng
      do 115 i=1,nn
      fli(kg,i) = fli1(kg,i)
  115 continue
      if(div.lt.1.e-6) goto 116
      goto 50
  116 write(6,*) '# of outer iter. reuired for the following',itr
      write(6,101) t(nc)
      do 70 i =1,nn
   70 write(6,102) i,(fli1(kg,i),kg=1,ng)
  100   continue
  101 format(//10X,21HCALCULATEDQUANTITIES/12X,
     + 29HNODAL VALUES FOR  FLUX AT  TIME,1x,d9.3,1x,4Hsec.)
  102 format(12X,i4,d14.5,d14.5)
      return
      end

c------------------------- end of the subroutine -|

      SUBROUTINE INEST(nc,fli1,flim,fli,u11,scm,b11,qs1
     & ,ipoint,ibook,ick)

      IMPLICIT REAL*8(a-h,o-z)
      COMMON/BLK6/nn,ne,nt,nt1,ng,eigo,delt,ndyn,iskp
      DIMENSION scm(nn,nt),b11(ng,nn,nt),fli1(ng,nn),flim(ng,nn)
      DIMENSION fli(ng,nn),qs1(ng,nn),ipoint(nn,nt)

      do 5 i =1,nn
      sum = 0.
      num = ipoint(i,1)
      do 10 j =1,num .
      l = ipoint(i,j)
      if(j.eq.1) l = i
      if(nc.eq.2) goto 7
      favg = fli(1,l)+ 1.46*( fli(1,l)- flim(1,l))
      goto 8
    7 favg = flim(1,l)
    8 sum = sum + scm(i,j)*favg*delt + b11(2,i,j)*flim(2,l)
   10 continue
```

```
   5  qs1(2,i) = sum

c     choose initialvector

      do 15 i =1,nn
      fli1(2,i) = flim(2,i)
  15  continue
      CALL BOUND2(kg,u11,qs1)

c     calculate initialestimate of ' fli1(2,i),i=1,nn '

      CALL GAUSS(2,u11,qs1,fli1,ibook,ick)
      do 20 i =1,nn
  20  fli(2,i) = fli1(2,i)
      return
      end

c--------------------------- end of the subroutine -|

      SUBROUTINESOURCE(kg,u12,fli1,qs1,
     & pv,scm,ipoint,alp2)

      IMPLICITREAL*8(a-h,o-z)
      COMMON/BLK5/plam(6),bet(6),v(2),beta
      COMMON/BLK6/nn,ne,nt,nt1,ng,eigo,delt,ndyn,iskp
      COMMON/BLK7/ib(164),bv(164),la
      DIMENSION u12(ng,nn,nt),scm(nn,nt),fli1(ng,nn)
      DIMENSION qs1(ng,nn),pv(ng,nn),ipoint(nn,nt)

      if(kg.eq.2) go to 30
      kgg = 2
      alpa = alp2
      go to 35
  30  kgg = 1
      alpa = delt/2.
  35  continue
      do 45 i=1,nn
      add = 0.
      num = ipoint(i,1)
      do 40 ji =1,num
      l = ipoint(i,ji)
      if(ji.eq.1) l = i
      if(kg.eq.2) go to 41
      add = add + u12(kgg,i,ji)*fli1(kgg,l)
```

```
      go to 40
   41 add = add + scm(i,ji)*fli1(kgg,l)
   40 continue
      qs1(kg,i) = pv(kg,i) + add*alpa
   45 continue
      return
      end

c------------------------- end of the subroutine -|

      SUBROUTINE CALC(fli,flim,ipoint,nprec,npre,sumcj
     & ,qs1,u11,u12,b11,scm,tdep,c,pv,t,nc,
     & loop,alp1,alp2,inin,period)

      IMPLICIT REAL*8(a-h,o-z)
      COMMON/BLK2/xg(5,2),fcs(5,2),dsig(5,2)
      COMMON/BLK5/plam(6),bet(6),v(2),beta
      COMMON/BLK6/nn,ne,nt,nt1,ng,eigo,delt,ndyn,iskp
      COMMON/BLK7/ib(164),bv(164),la
      DIMENSION u11(ng,nn,nt1),b11(ng,nn,nt),c(nn,npre),pv(ng,nn)
      DIMENSION flim(ng,nn),fli(ng,nn),ipoint(nn,nt)
      DIMENSION u12(ng,nn,nt),scm(nn,nt),tdep(nn,nt),t(loop)
      DIMENSION qs1(ng,nn),sumcj(nn)
      DIMENSION exj0(6), exj1(6),s1(6),s2(6)
      DOUBLE PRECISION exj0,exj1,s1,s2


      nc1 = nc - 1
      nc2 = nc - 2
    3 if(nc.ne.2) go to 9
      delth = delt/2.
      sum1 = delth
      sum2 = delth
      if(nprec.eq.0) go to 51
      do 11 j=1,nprec
      plamj = 1./plam(j)
      plamj2 = plamj**2
      exj0(j) = dexp(-plam(j)*delt)
      sum1 = sum1 + bet(j)*( plamj + (exj0(j)-1.)*
     & ( plamj+plamj2/delt ) )
      sum2 = sum2 + bet(j)*( - plamj -(exj0(j)-1.)*plamj2/delt )
   11 continue
   51 alp1 = sum1
      alp2 = sum2
```

```
      if(ng.gt.1) goto 24
      do 23 i=1,nn
      jj = 0
      num = ipoint(i,1)
      do 23 ji = 1,num
      l = ipoint(i,ji)
      if(ji.eq.1) l = i
      if(l.lt.i)goto 23
      jj = jj + 1
      u11(ng,i,jj) = u11(ng,i,jj) - alp2*u12(ng,i,ji)
23    continue

      go to 9
24    continue
      if(fcs(1,1).eq.0.) goto 9
      do 26 i=1,nn
      jj = 0
      num = ipoint(i,1)
      do 26 ji = 1,num
      l = ipoint(i,ji)
      if(ji.eq.1) l = i
      if(l.lt.i)goto 26
      jj = jj + 1
      u11(1,i,jj) = u11(1,i,jj) - alp2*u12(1,i,ji)
26    continue

 9    if(nprec.eq.0) go to 52
      do 10 j=1,nprec
      plamj = 1./plam(j)
      plamj2 = plamj**2
      ex1 = dexp( plam(j)*t(nc1) )
      ex2 = dexp( plam(j)*t(nc) )
      exj1(j) = 1./ex1 - 1./ex2
      if(nc.le.2) go to 10
      ex0 = dexp( plam(j)*t(nc2) )
      s1(j) = plamj2*ex1 - (delt/plam(j) + plamj2)*ex0
      s2(j) = (delt/plam(j) - plamj2)*ex1 + plamj2*ex0
10    continue

      if(ng.gt.1.and.fcs(1,1).ne.0.)  go to 55
      do 30 i=1,nn
      do 30 j=1,nprec
      if(nc.lt.3) go to 31
      c(i,j) = c(i,j) + (flim(ng,i)*s1(j) +
```

```
     & fli(ng,i)*s2(j))*bet(j)/delt
       go to 30
  31   c(i,j) = flim(ng,i)*bet(j)/plam(j)
  30   continue
       goto 52
  55   continue

       do 62 i=1,nn
       num = ipoint(i,1)
       topl1 = 0.
       topl2 = 0.
       do 64 ji = 1,num
       l = ipoint(i,ji)
       if(ji.eq.1) l = i
       if(nc.lt.3) goto 65
       topl1 = topl1 + u12(1,i,ji)*flim(1,l)+ u12(2,i,ji)*flim(2,l)
       topl2 = topl2 + u12(1,i,ji)*fli(1,l)+ u12(2,i,ji)*fli(2,l)
       goto 64
  65   topl1 = topl1 + u12(1,i,ji)*flim(1,l)
       topl2 = topl2 + u12(2,i,ji)*flim(2,l)
  64   continue
       do 62 j=1,nprec
       if(nc.lt.3) goto 68
       c(i,j) = c(i,j) + (topl1*s1(j) + topl2*s2(j))*bet(j)/delt
       goto 62
  68   c(i,j) = (topl1 + topl2)*bet(j)/plam(j)
  62   continue
       do 70 i = 1,nn
       sumcj(i) = 0.
       do 70 j = 1,nprec
  70   sumcj(i) = sumcj(i) + exj1(j)*c(i,j)

c..    'inin' indicates ramp or sinusoidal or step change

  52   continue
       if(inin.eq.1) then
       go to 12
       elseif(inin.eq.2) then
       goto 27
       endif
       if(nc.ge.3) go to 14
       con1 = delt**2/6.
       con2 = delt**2/3.
       go to 13
```

```
   14  con1  = delth*delt
       con2  = con1
       go to 13
   12  freq = period

c  period is 'sin(period*t)'

       z1  = freq * t(nc1)
       z2  = freq * t(nc)
       sin1  = dsin(z1)
       sin2  = dsin(z2)
       cos1  = dcos(z1)
       cos2  = dcos(z2)
       fdelt = freq*delt
       fi = fdelt * freq
       sind  = sin2 - sin1
       if(nc.ge.3) go to 17
       con1  = (fdelt*cos1 -sind)/fi
       con2  = -(fdelt*cos2 - sind)/fi
       go to 13
   17  z0 = freq*t(nc2)
       sin0  = dsin(z0)
       cos0  = dcos(z0)
       sins  = -sin2 + 2.*sin1 - sin0
       sinb  = sin1 - sin0
       con1  = (fdelt*(cos1 - cos0)  + sins)/fi + sinb*delth
       con2  = -(fdelt*(cos2 - cos1)  + sins)/fi + sinb*delth

   13  continue
       do 20 ki=1,nn
       jj = 0
       num  = ipoint(ki,1)
       do 20 kj = 1,num
       b11(ng,ki,kj)  = b11(ng,ki,kj) - con1*tdep(ki,kj)
       l = ipoint(ki,kj)
       if(kj.eq.1) l = ki
       if(l.lt.ki)goto 20
       jj = jj + 1
       u11(ng,ki,jj)  = u11(ng,ki,jj) + con2*tdep(ki,kj)
   20  continue

   27  do 50 kg  = 1,ng
       kkg  = 2
       if(ng.eq.1.or.kg.eq.2)  kkg  = 1
```

```
      do 50 i=1,nn
      num = ipoint(i,1)
      sum3 = 0.
      do 40 ji = 1,num
      l = ipoint(i,ji)
      if(ji.eq.1) l = i
      if(kg.eq.2) goto 41
      sum4 = 0.
      if(nprec.eq.0) go to 53
      if(ng.gt.1.and.fcs(1,1).ne.0.)  go to 53
      do 35 j=1,nprec
 35   sum4 = sum4 + exj1(j)*u12(kkg,i,ji)*c(l,j)
 53   sum3 = sum3 + sum4+ alp1*u12(kkg,i,ji)*fli(kkg,l)
  &   + b11(kg,i,ji)*fli(kg,l)
      if(ng.gt.1.and.fcs(1,1).ne.0.)  sum3 = sum3 +
  &   alp1*u12(kg,i,ji)*fli(kg,l)
      go to 40
 41   sum3 = sum3 + scm(i,ji)*fli(kkg,l)*delth
  &   + b11(kg,i,ji)*fli(kg,l)
 40   continue
      if(ng.gt.1) then
      pv(kg,i) = sum3
      else
      qs1(kg,i) = sum3
      endif
 50   continue
      if(ng.gt.1.and.fcs(1,1).ne.0.and.nprec.ne.0)    then
      do 190 i=1,nn
 190  pv(1,i) = pv(1,i) + sumcj(i)
      endif

 200  return
      end

c------------------------- end of the subroutine -|
c------------  subroutine mesh  ------------------------------c
c   wf= weighting factor
c   intervals along a generating line will become:
c   progressively shorter if wf<1.
c   stay equal            if wf=1.(default)
c   progressively longer  if wf>1.
c   nx(i)= the number of intervals for each generating line
c   xf(i),yf(i),xl(i),yl(i)=the coordinates of the two end points
c   mid(i,j)= material identification number
```

```
c--------------------          ----------------------------c
      SUBROUTINE MESH

      IMPLICIT REAL*8(a-h,o-z)
      COMMON/BLK1/nh,mntl(3200),nel(3200,3),xc(1681),
     & yc(1681),mid(40,40)
      COMMON/BLK6/nn,ne,nt,nt1,ng,eigo,delt,ndyn,iskp
      COMMON/BLK7/ib(164),bv(164),la
      DIMENSION nx(41),yl(41),xf(41),yf(41),nod(4),sum1(42),xl(41)
c-----------------------------------------------------------------
      wf = 1.
      read(5,*) ny,nh
      write(6,1001) ny
      do 100 i=1,ny
        read(5,*) nx(i),xf(i),yf(i),xl(i),yl(i)
        write(6,1000) nx(i),xf(i),yf(i),xl(i),yl(i)
 100   continue
      nym = ny - 1
      do 110 i=1,nym
 110  read(5,*) (mid(i,j),j=1,nx(i))
      write(6,1429)
      write(6,1431)
      n = 0
      nn = 0
      do 350 i=1,ny
      nxi = nx(i)+1
      sum1(1) = 0.0
      sum1(2) = 1.0
      sum = 1.0
      if(nxi-2) 190,291,190
 190    do 250 k=3,nxi
      sum1(k)= sum1(k-1)*wf
      sum = sum + sum1(k)
 250    continue
 291    continue
      x=xf(i)
      y=yf(i)
     do 300 j=1,nxi
      n = n + 1
      x = (xl(i)-xf(i))*sum1(j)/sum + x
      y=(yl(i)-yf(i))*sum1(j)/sum + y
      xc(nn+j) = x
      yc(nn + j)=y
```

```
      write(6,1430) n,xc(j+nn),yc(j+nn)
300     continue
      nn = nn + nxi
350   continue
      n = 0
      nsum = 0
      nyi = ny - 1
      write(6,1432)
      do 600 i=1,nyi
      nxi = nx(i)
      do 500 j=1,nxi
      if(j-nxi)379,371,379
371   if(nx(i+1)-nx(i))380,379,401
379   nod(1) = j + nsum
      nod(2) = nod(1) + 1
      nod(3) = nod(2) + nxi + 1
      nod(4) = nod(3) - 1
      goto 412
380   nod(1) = nod(2)
      nod(2) = nod(1) + 1
      nod(4) = 0
      goto 412
401   nod(1) = j + nsum
      nod(2) = nod(1) + 1
      nod(3) = nod(2) + nxi + 1
      nod(4) = nod(3) - 1
      n = n + 1
      mntl(n) = mid(i,j)
      write(6,1470) n,nod(1),nod(2),nod(3),mntl(n)
      nel(n,1) = nod(1)
      nel(n,2) = nod(2)
      nel(n,3) = nod(3)
      n = n + 1
      mntl(n) = mid(i,j)
      write(6,1470) n,nod(1),nod(3),nod(4),mntl(n)
      nel(n,1) = nod(1)
      nel(n,2) = nod(3)
      nel(n,3) = nod(4)
      nod(1) = nod(2)
      nod(2) = nod(3) + 1
      nod(4)=0
412   n = n+1
      mntl(n) = mid(i,j)
      write(6,1470) n,nod(1),nod(2),nod(3),mntl(n)
```

```
      nel(n,1) = nod(1)
      nel(n,2) = nod(2)
      nel(n,3) = nod(3)
      if(nod(4)) 433,434,433
  433 n = n + 1
      mntl(n) = mid(i,j)
      write(6,1470) n,nod(1),nod(3),nod(4),mntl(n)
      nel(n,1) = nod(1)
      nel(n,2) = nod(3)
      nel(n,3) = nod(4)
  434   continue
  500   continue
      nsum = nsum + nxi + 1
  600   continue
      ne = n

c...  ki will have the node numbers which have zero boundary conditions
c..   number of boundary conditions = la
c...  if nh=0 Impose zero boundary conditions on 4 sides
c...  if nh=1 Impose zero boundary conditions on 2.nd. and 3.rd. side.

      if(iskp.ne.0) return
      npp = 0
      la = 0
      if(nh.ne.0) goto 10
      npp = nx(1)
      do 5 i = 1,npp
      ib(i) = i
    5 continue
      la = npp

   10 is = 1
      ki = 0
      nj = 1
      if(nh.eq.0) is = 2
      do 15 j=1,ny-1
      la = la + 1
      ki = ki + nx(j) + 1
      ib(npp + nj ) = ki
      nj = nj + is
   15 continue
      ki = ki + 1

      if(nh.ne.0) goto 25
```

```
      njj = 2
      it = 0
      ib(npp + 2) = npp + 2
      do 20 i = 2,ny-1
      la = la + 1
      it = ib(npp + njj - 1) + 1
      ib(npp + njj) =  it
      njj = njj + is
   20 continue

   25 if(nh.eq.0) nj = nj - 1
      do 30 i=1,nx(ny) + 1
      la = la + 1
      ib(npp + nj) = ki
      nj = nj + 1
      ki = ki + 1
   30 continue
 1000   format(i5,5x,4f10.5)
 1001   format(/,4x,'INPUT DATA FOR MESH GENERATION'/,i5)
 1429   format(/,4x,'GENERATED COORDINATES & NODE NUMBERS')
 1431   format(/,4x,' NODAL POINT X CO-ORDINATEY CO-ORDINATE')
 1430   format(7x,i4,2f16.8)
 1432   format(/,5x,'ELMENT NO  NODAL POINTS OF THE ELEMENTS',
     & ' MATERIAL')
 1470         format(3x,i8,5x,3i8,3x,i8)
      return
      end
c------------------------- end of the subroutine -|

      SUBROUTINE INSPECTOR(ick,lnode)

      IMPLICIT REAL*8(a-h,o-z)
      COMMON/BLK1/nh,mntl(3200),nel(3200,3),xc(1681),
     & yc(1681),mid(40,40)
      COMMON/BLK6/nn,ne,nt,nt1,ng,eigo,delt,ndyn,iskp
      DIMENSION ick(nn),lnode(3)

      do 3 i=1,nn
    3 ick(i)=0

c.. check if any node nuber exceeds  nn .....

      do 7 i=1,ne
      do 5 j=1,3
```

```
      k=nel(i,j)
      ick(k)=1
    5 if(k.gt.nn) write(6,100) j,i,nn
    7 continue

c..... check if all node numbers through nn are included

      do 9 i=1,nn
      if(ick(i).eq.0)write(6,101) i
    9 ick(i) = 0

c... calculation of the bandwith

      inbw=0
      nbw=0
      do 15 kk=1,ne
      do 13 i=1,3
   13 lnode(i)= nel(kk,i)
      lk=2
      do 11 i=1,lk
      ij=i+1
      do 11 j=ij,3
      nb=iabs(lnode(i)-lnode(j))
      if(nb.eq.0) write(6,102) kk
      if(nb.le.nbw) goto 11
      inbw=kk
      nbw=nb
   11 continue
   15 continue
      nbw=nbw+1
      write(6,103) nbw,inbw

  100 format(/10x,4hNODE,i4,11h OF ELEMENT,i4,13hEXCEEDSNN =,i4)
  101 format(/10x,4hNODE,i4,15h DOES NOT EXIST)
  102 format(/10x,7hELEMENT,i3,13hHASTWO NODES/
     + 10x,25hWITH THE SAME NODE NUMBER)
  103 format(//10x,12hBANDWITH IS,i4,11h IN ELEMENT,i4)
      return
      end
c-------------------------- end of the subroutine -|

      SUBROUTINENULL(u11,b11,vel1,u12,scm,tdep,ipoint,ibook)

      IMPLICITREAL*8(a-h,o-z)
```

```
      COMMON/BLK6/nn,ne,nt,nt1,ng,eigo,delt,ndyn,iskp
      DIMENSION u11(ng,nn,nt1),b11(ng,nn,nt)
     1 ,ipoint(nn,nt),vel1(ng,nn,nt),u12(ng,nn,nt)
      DIMENSION tdep(nn,nt),scm(nn,nt),ibook(nn,nt1)

      do 5 il=1,nn
      ipoint(il,1) = il
      ibook(il,1) = il
      do 5 jl=1,nt
      do 4 kg = 1,ng
      u11(kg,il,jl) = 0.
      b11(kg,il,jl) = 0.
      vel1(kg,il,jl) = 0.
      u12(kg,il,jl) = 0.
    4 continue
      tdep(il,jl) = 0.
      scm(il,jl) = 0.
    5 continue
      return
      end
c------------------------- end of the subroutine -|
```

```
c---------------------  OPUSSOR.FOR ------------------------- c
c  is a FORTRAN code to solve both static and time-space
c  dependent coupled 2D - 2G neutron diffusion equation by the
c  finite element method ( linear polynomial approximation in time
c  and space). Succesive over relaxation method is utilized as the
c  calculation algorithm.
c  The precursor equations are integrated and the Subdomain Method
c  for the multigroup equations is utilized in the
c  approximation of time dependency.  Written by Haluk UTKU.
c-----------------------------------------------------------------c
c  COMMON/BLK1/
c  nh = The number of sides on which essential or reflective boundary
c       conditions is to be applied. if 0 all sides have zero flux
c       if 1 side 2 and 3 have zero flux side 1 and 4 have reflective
c       boundary conditions.
c                      Y   Side 3
c                      |
c                      |
c                      |
c            Side 4    |          Side 2
c                      |
c                      |
c                      |--------> X
c                         Side 1
c
c  mntl() = stores material identification numbers
c  nel(m,n) = stores element numbers(=m) and its node numbers(=n)
c             node numbering i,j,k should be counter clockwise in the
c             case of manual entry.
c  xc(),yc()  = stores x and y coordinates of nodes(must be in
c             numerical sequence relative to the numbers in the case
c             of manual input.
c  mid(i,j) = image matrix of material identifacitions.
c  COMMON/BLK2/
c  xg(id,kg)  = fission yield to energy group kg of material(id)
c  fcs(id,kg)  = fission cross section of material in energy group kg
c  dsig(id,kg)  = perturbation constant introduced in material id
c  in energy group kg.( see Eq.(4.25) and (4.26)). ONLY thermal group
c  COMMON/BLK3/
c  de() = diffusion constant
c  rcse() = removal cross section (absortion plus out scattering)
c  fcse() = fission cross section
c  scse()  = scattering cross section
c  ds() = perturbation constant
c  xge()  = fission yield
c  COMMON/BLK4/
```

```
c  idbc(i,1)= element number with a derivative boundary  condition
c  idbc(i,2)= side  of the element with a derivative boundary
c  dbc()= (length of the side)
c  ndbc  = number of non-reentrant boundary  conditions.
c  COMMON/BLK5/
c  plam() = array storing the decay  constants  of precursors
c  bet()  = array storing the fraction of contributions from delayed
c           neutrons.
c  v() = neutron velocities in group  1 and 2
c  beta  = average  of fractional contributions of delayed  neutrons
c  COMMON/BLK6/
c  nn  = total number of nodes  in the system
c  ne  = total number of elements  in the system
c  nt  = total number of columns  allocated
c  ng  = number of neutron energy  groups
c  eigo  = previous  eigenvalue  estimate
c  delt  = time steps
c  ndyn  = number of pecursor  groups
c  iskp  = condition to choose  manual input or MESH  generation.
c  COMMON/BLK7/
c  ib() = array stroing node numbers  having a boundary  value.
c  bv() = array storing known  flux values.
c  COMMON/BLK8/
c  eu11(kg,3,3)  = the kg group  loss  matrix of a triangular element
c  x(),y() = coordinates  of triangular element.
c  v11(,)  = scattering  matrix of an element.
c  evel1(kg,3,3)  = element velocity matrices  of related groups.
c  eu12(kg,3,3)  = element fission matrices  of related groups.
c  tdab(,)  = element perturbation matrix.
c  DIMENSION d(,),rcs(,),scs(,)
c  d(,)  = matrix storing diffusion constants  of related materials.
c  rcs(,)  = matrix storing removal cross  sections  of related
c           materials.
c  scs(,)  = matrix storing scattering  cross  sections  of related
c           materials.
c  DIMENSION ick( ),arr( ),marr( ),flux( ),lnode( )
c  ick() = array used  in error checking.
c  arr() = adjustable  array storing global matrices
c  marr() = adjustable  array storing pointer matrices
c  flux() = adjustable  array storing flux and/or sorce  vector arrays
c  lnode() = pointer array storing element node numbers
c  DIMENSION rins( )
c  rins() = adjustable  array storing global perturbation matrix
c  DIMENSION dlnm( )
```

```
c  dlnm() = adjustable array storing infrmation related with
c           precursors.

c-----------------------------------------------------------------c
      PROGRAM OPUSSOR
      IMPLICIT REAL*8(a-h,o-z)
      COMMON/BLK1/nh,mntl(3200),nel(3200,3),xc(1681),
     & yc(1681),mid(40,40)
      COMMON/BLK2/xg(5,2),fcs(5,2),dsig(5,2)
      COMMON/BLK3/de(2),rcse(2),fcse(2),scse(2),ds(2),xge(2)
      COMMON/BLK4/idbc(164,2),dbc(164),ndbc
      COMMON/BLK5/plam(6),bet(6),v(2),beta
      COMMON/BLK6/nn,ne,nt,ng,eigo,delt,ndyn,iskp
      COMMON/BLK7/ib(164),bv(164),la
      COMMON/BLK8/eu11(2,3,3),x(3),y(3),v11(3,3),evel1(2,3,3),
     &          eu12(2,3,3),tdab(3,3)
      DIMENSION d(5,2),rcs(5,2),scs(5,2)
      DIMENSION ick(1681),arr(153972),marr(16811),flux(13449),lnode(3)
      DIMENSION rins(16811)
      DIMENSION dlnm(13449)

c..... maximum number of columns to be stored: nmax - very important -
c.....  max. # of nodes  = nn = 1681= (41x41)
c.....  ng  = 2 (# of energy groups)

      nmax = 10
      nt = nmax
c.....
c If OPUS.DAT is not in the same directory, sytem produces
c negative number for 'error code'

      open(5,file='opus.dat',iostat=ju,err=777,status='old')
      open(6,file='opus.out',status='unknown')
 777  print*,'error code=',mod(ju,256)

       read(5,*) iskp,ndyn
       if(iskp.eq.0) go to 1
      read(5,*) nn,ne,ncoef,ndbc,ng
      write(6,103) nn,ne,ncoef,ndbc,ng
      goto 2
   1   read(5,*) ncoef,ng
      write(6,100) ncoef,ng

   2  write(6,101)
```

```
      do 5 i=1,ncoef
      do 5 j=1,ng
      read(5,*) d(i,j),rcs(i,j),fcs(i,j),scs(i,j),xg(i,j),dsig(i,j)
      write(6,102) i,j,d(i,j),rcs(i,j),fcs(i,j),scs(i,j),xg(i,j)
 5    continue
      if(iskp.eq.0) then
      CALL MESH
      goto 20
      endif
      write(6,105)
      read(5,*) (xc(i),i=1,nn)
      read(5,*) (yc(i),i=1,nn)
      write(6,106) (i, xc(i), yc(i),i=1,nn)

c..... hardcopy of the element nodal data .....

      write(6,107)
      nid=0
        do 9 kk=1,ne
      read(5,*) n,mntl(kk),(nel(n,i),i=1,3)
      if((n-1).ne.nid) write(6,108) n
      nid=n
 9    write(6,109) n,mntl(kk),(nel(n,i),i=1,3)

      read(5,*) la
      if(la.le.0) goto 20
 209  read(5,*) (ib(i),bv(i),i=1,la)

 20   continue

c..... hardcopy of the derivative boundary conditions .....

      if(ndbc.eq.0) goto 14
      write(6,110)
      do 12 i=1,ndbc
      read(5,*) idbc(i,1),idbc(i,2),dbc(i)
 12   write(6,*) idbc(i,1),idbc(i,2),dbc(i)
 14   continue

      if(ndyn.ge.1) then
      read(5,*) delt,nprec,fin,vel,eigo
      read(5,*) inin,period
      endif
```

```
c.....  generation of adjustable arrays  .....

        ngnn  = ng*nn
        nnmax  = nn*nmax
        ngnnm  = ng*nnmax
        na1  = 1
        na2  = na1 + ngnnm
        na3  = na2 + ngnnm
        na4  = na3 + ngnnm
        na5  = na4 + ngnnm
        na6  = na5 + nnmax
        na7  = na6 + nn
        me1  = 1
        me2  = me1 + ngnn
        me3  = me2 + ngnn
        me4  = me3 + ngnn
        ka1  = 1
        ka2  = ka1 + nn*nprec


        CALL INSPECTOR(ick,lnode)
        CALL NULL(arr(na1),arr(na2),marr(na1),
     &  arr(na3),arr(na4),arr(na5),rins(na1))

        do 25 kk=1,ne

        do 22 i=1,3
        lnode(i)=nel(kk,i)
        j=lnode(i)
        x(i)=xc(j)
   22   y(i)=yc(j)
c....
        ii=mntl(kk)
c....
        do 23 ig=1,ng
        de(ig)=d(ii,ig)
        rcse(ig)=rcs(ii,ig)
        fcse(ig)=fcs(ii,ig)
        scse(ig)=scs(ii,ig)
        xge(ig)=xg(ii,ig)
        ds(ig)  =  dsig(ii,ig)
   23   continue

c...  calculation of element matrices ...
```

```
      CALL ELEM(kk,vel)

c...  store element matrices in global matrices....

      CALL STORE(arr(na1),lnode,arr(na2)
     & ,marr(na1),arr(na3),arr(na4),arr(na5),rins(na1))

   25 continue

      CALL BOUND1(arr(na1))

c if ndyn=2 initialflux is read in

      if(ndyn.eq.2) goto 30

      CALL EIGEN(flux(me1),flux(me2),flux(me3)
     & ,arr(na1),arr(na4),arr(na5),marr(na1))

c if ndyn = 0 only eigenvalue calculation

   30 if(ndyn.eq.0) goto 900

c..... start calling routines for transient calculations

      CALL COMB(arr(na1),arr(na2),arr(na3))

      loop = fin/delt + 1.1
      npre = nprec
      if(npre.eq.0) npre = 1

     CALL SOLVE(loop,arr(na1),marr(na1),arr(na2)
     & ,flux(me1),flux(me2),flux(me3),flux(me4),arr(na4)
     & ,rins(na1),dlnm(ka1),dlnm(ka2),arr(na5),arr(na7)
     & ,nprec,npre,inin,period,arr(na6))


c..... output formats .....

  100 format(1x/10x,'NCOEF=',i2,2x,'NO.  OF  GROUPS=',i2,2x
     + ,17h# OF BOUND. COND.,i3)
  103 format(1x/10x,'NN  =',i5/10x,'NE  =',i5/10x,'NCOEF=',i2,
     + 2x,'NDBC='i3,2x,'NO.  OF  GROUPS=',i2)
  101 format(/10x,'EQUATION COEFFICIENTS'/3x,'MATERIAL',2x,
     + 'GROUP',2x,9hDIF.CONS,2x,'REMOVAL',4x,'FIS.X SEC.',
```

```
     +   1x,'SCAT. X SEC.',2x,'FISS.   YIELD')
  102  format(7x,i2,4x,i2,2x,5d12.5)
  105  format(//10x,'NODAL COORDINATES'/10x,'NODE',7x,1hX,14x,1hY)
  106  format(10x,i4,2d15.5)
  107  format(//10x,'ELEMENT DATA'/16x,
     + 'NE',4x,'MNTL',4x,'NODE NUMBERS')
  108  format(10x,7hELEMENT,I4,16H NOT IN SEQUENCE)
  109  format(15x,i3,5x,i3,2x,4i4)
  110  format(//10x,34hDERIVATIVEBOUNDARYCONDITION DATA/10x,
     + 7hELEMENT,4x,4hSIDE,7x,2hLS)

  900  stop
       end
c------------------------- end of the subroutine -|

       SUBROUTINE ELEM(kk,vel)

       IMPLICIT REAL*8(a-h,o-z)
       COMMON/BLK2/xg(5,2),fcs(5,2),dsig(5,2)
       COMMON/BLK3/de(2),rcse(2),fcse(2),scse(2),ds(2),xge(2)
       COMMON/BLK4/idbc(164,2),dbc(164),ndbc
       COMMON/BLK5/plam(6),bet(6),v(2),beta
       COMMON/BLK6/nn,ne,nt,ng,eigo,delt,ndyn,iskp
       COMMON/BLK7/ib(164),bv(164),la
       COMMON/BLK8/eu11(2,3,3),x(3),y(3),v11(3,3),evel1(2,3,3),
     &         eu12(2,3,3),tdab(3,3)
       DIMENSION b(3),c(3)
c---------------------------------------
       b(1)=y(2)-y(3)
       b(2)=y(3)-y(1)
       b(3)=y(1)-y(2)
       c(1)=x(3)-x(2)
       c(2)=x(1)-x(3)
       c(3)=x(2)-x(1)
      ar2=x(2)*y(3)+x(3)*y(1)+x(1)*y(2)-x(2)*y(1)
     +         -x(3)*y(2)-x(1)*y(3)
       if(abs(ar2).lt.0.0001) goto 5
       dia = ar2/6.
       fa = dia/4.
       do 20 i=1,3
       do 20 j=1,3

       a = 1.0
c      ab = 0.
```

```
      if(i.eq.j) a = 2.
c     if(i.eq.j) ab = 1.
      gt=( b(i)*b(j) + c(i)*c(j) )/(2.*ar2)
      fact = a * fa
c     diag = ab * dia

      do 15 kg = 1,ng
      kgg = kg
      if(ng.eq.1) kgg = 2
       eu11(kg,i,j) = de(kg) * gt + rcse(kg) * fact
      eu12(kg,i,j) = fcse(kg) * fact
      if(ndyn.eq.0) goto 15
c     evel1(kg,i,j) = vel * diag/v(kgg)
      evel1(kg,i,j) = vel * fact/v(kgg)
   15 continue
       v11(i,j) = scse(1) * fact
      if(ndyn.eq.0) goto 20
      tdab(i,j) = ds(ng) * fact
c
   20 continue

      if(ndbc.eq.0) goto 7

c THE FOLLOWINGcan be used in the case of non-reentrant
c boundary condition for the thermal group
c if it has to be used MAIN program should read boundary numbers
c and SUBROUTINEMESH has to be modified not to create bundary nodes

      do 11 i=1,ndbc
      if(idbc(i,1).ne.kk) goto 11
      j=idbc(i,2)
      k=j+1
      if(j.eq.3) k=1
      eu11(2,j,j)=eu11(2,j,j)+dbc(i)/(3.*2.)
      eu11(2,j,k)=eu11(2,j,k)+dbc(i)/(6.*2.)
      eu11(2,k,j)=eu11(2,j,k)
      eu11(2,k,k)=eu11(2,k,k)+dbc(i)/(3.*2.)
   11 continue
    7  continue

   35 return

    5 write(6,6) kk
    6 format(//10x,19hthe area of element,i4,
```

```
     + 20h is less than 0.0001/10x,21h  the node numbers are,
     + 19h in the wrong order/10x,20hexecution  terminated)
       stop
       end
c------------------------- end of the subroutine -|

       SUBROUTINE STORE(u11,lnode,b11,ipoint,vel1,u12,scm,tdep)

       IMPLICIT REAL*8(a-h,o-z)
       COMMON/BLK2/xg(5,2),fcs(5,2),dsig(5,2)
       COMMON/BLK6/nn,ne,nt,ng,eigo,delt,ndyn,iskp
       COMMON/BLK8/eu11(2,3,3),x(3),y(3),v11(3,3),evel1(2,3,3),
     &         eu12(2,3,3),tdab(3,3)
       DIMENSION u11(ng,nn,nt),lnode(3),b11(ng,nn,nt)
       DIMENSION ipoint(nn,nt),vel1(ng,nn,nt),u12(ng,nn,nt)
       DIMENSION tdep(nn,nt),scm(nn,nt)

C   nt = max number of columns to be stored
C   nnpe = # of nodes per element
c---------------------------------------
       nnpe = 3

c..... store element matrices .....
c..... first row .....

       i = 0
       do 50 jj=1,nnpe
       irow = lnode(jj)

       i = i+1


c..... then columns .....

       ik = 0
       do 55 kj= 1,nnpe
       icol =lnode(kj)
       ik = ik + 1

c ..... search ipoint for column no. .....

       do 95 mm=1,nt
       if(ipoint(irow,mm)-icol)100,105,100
  100  if(ipoint(irow,mm))110,110,95
```

```
 95   continue

c..... found a blank now store icol .....

 110  ipoint(irow,mm) = icol

c..... now store element stiffness matrix .....

 105  do 106 kg = 1,ng
      u12(kg,irow,mm) = u12(kg,irow,mm) + eu12(kg,i,ik)
      u11(kg,irow,mm) = u11(kg,irow,mm) + eu11(kg,i,ik)
      if(ndyn.eq.0) goto 106              ! only eigenvalue search
      b11(kg,irow,mm) = b11(kg,irow,mm) + eu11(kg,i,ik)
      vel1(kg,irow,mm) = vel1(kg,irow,mm) + evel1(kg,i,ik)
 106  continue
      scm(irow,mm) = scm(irow,mm) + v11(i,ik)
      if(ndyn.eq.0) goto 55
      tdep(irow,mm) = tdep(irow,mm) + tdab(i,ik)

c..... end loop on columns .....

 55   continue

c..... end loop on rows .....

 50   continue

      return
      end

c------------------------- end of the subroutine -|

      SUBROUTINE BOUND1(u11)

      IMPLICIT REAL*8(a-h,o-z)
      COMMON/BLK6/nn,ne,nt,ng,eigo,delt,ndyn,iskp
      COMMON/BLK7/ib(164),bv(164),la
      DIMENSION u11(ng,nn,nt)

c     apply known nodal flux values:
c     ib = node number on which bound. condition will be imposed.
c     bv = known boundary flux value
c-------------------------------------------------------
```

```
 15     do 20 n=1,la
        irow=ib(n)
       do 20 kg =1,ng
          u11(kg,irow,1)  = u11(kg,irow,1)  * 10.**15
 20    continue
       return
       end

c------------------------- end of the subroutine -|

       SUBROUTINE BOUND2(kg,u11,qs1)

       IMPLICIT REAL*8(a-h,o-z)
       COMMON/BLK6/nn,ne,nt,ng,eigo,delt,ndyn,iskp
       COMMON/BLK7/ib(164),bv(164),la
       DIMENSION u11(ng,nn,nt),qs1(ng,nn)

c      apply known nodal flux values:
c      ib = node number on which bound. condition will be imposed.
c      bv = known boundary flux value
c-------------------------------------------------------

 15          do 20 n=1,la
        irow=ib(n)
 17     qs1(kg,irow)  = u11(kg,irow,1)*bv(n)
 20    continue
       return
       end

c------------------------- end of the subroutine -|

       SUBROUTINE EIGEN(fli1,qs1,qs1m,u11,u12
     & ,scm,ipoint)

       IMPLICIT REAL*8(a-h,o-z)
       COMMON/BLK2/xg(5,2),fcs(5,2),dsig(5,2)
       COMMON/BLK3/de(2),rcse(2),fcse(2),scse(2),ds(2),xge(2)
       COMMON/BLK6/nn,ne,nt,ng,eigo,delt,ndyn,iskp
       COMMON/BLK7/ib(164),bv(164),la
       DIMENSION fli1(ng,nn),qs1(ng,nn),qs1m(ng,nn)
       DIMENSION ipoint(nn,nt)

        do 3 n =1,nn
        do 2 m = 1,nt
```

```
      if(ipoint(n,m).ne.0) goto 2
      ipoint(n,1) = m-1
      go to 1
    2 continue
    1 continue
    3  continue

      omeg  = 0.
      eigo=1.
      scale=sqrt(real(nn))


c..... assign  unities to initialfluxes .....

      do 4 kg =1,ng
      do 4 i=1,nn
      fli1(kg,i)=1./scale
    4 continue
      den = 1.

      xgs=(xg(1,2)/xg(1,1))**2.

c     itr is the iteration index

      itr = 0

      CALL OUTER(1,u12,scm,fli1,qs1,ipoint)

   16 do 7 n = 1,nn
    7 qs1m(1,n)  = qs1(1,n)


      do 18 kg =1,ng
      if(kg.eq.1) goto 10
      CALL OUTER(2,u12,scm,fli1,qs1,ipoint)
   10 CALL BOUND2(kg,u11,qs1)
      if(kg.eq.2) goto 12
      if(omeg.eq.0.) CALL OMEGA(u11,ipoint,omeg)
   12 CALL VECITR(kg,omeg,u11,qs1,fli1,ipoint)
   18 continue

      CALL OUTER(1,u12,scm,fli1,qs1,ipoint)

      xnum  = 0.
```

```
      den = 0.
      do 25 n =1,nn
      if(ng.eq.2) goto 27
      xnum = xnum + qs1(ng,n)
      den = den + qs1m(ng,n)
      goto 25
   27 xnum = xnum + qs1(1,n) + qs1(1,n)*xg(1,2)/xg(1,1)
      den = den + qs1m(1,n) + qs1m(1,n)*xg(1,2)/xg(1,1)

   25 continue
      eign = eigo*(xnum/den)
      rel = abs(eigo/eign - 1.)
      if(rel.lt.1.d-06) goto 30
      eigo = eign
      den = dsqrt(den)
      itr = itr + 1
      goto 16

   30 continue
      write(6,101)
  101 format(//10X,21HCALCULATEDQUANTITIES/12X,
     + 21HNODAL VALUES FOR FLUX)
      do 32 i =1,nn
   32 write(6,102) i,(fli1(kg,i),kg=1,ng)
  102 format(12X,i4,d14.5,d14.5)
      return
      end

c------------------------- end of the subroutine -|

      SUBROUTINE VECITR(kg,omeg,u11,qs1,fli1,ipoint)

      IMPLICIT REAL*8(a-h,o-z)
      COMMON/BLK6/nn,ne,nt,ng,eigo,delt,ndyn,iskp
      DIMENSION u11(ng,nn,nt),qs1(ng,nn),fli1(ng,nn)
      DIMENSION ipoint(nn,nt)

      tole = 1.0d-03
      if(ng.lt.2) tole = 1.0d-05

      do 20 nlo=1,40
      err =0.
      do 15 n =1,nn
      r = fli1(kg,n)
```

```
      fx = qs1(kg,n)
      num = ipoint(n,1)
      do 10 m=2,num
      l = ipoint(n,m)
   10 fx = fx - u11(kg,n,m)*fli1(kg,l)

c.... fx is total unbalance of r.h.s. dx is the change

      dx = fx / u11(kg,n,1) - fli1(kg,n)
      fli1(kg,n) = fli1(kg,n) + omeg*dx

c  The following if statement is optional
c     if(fli1(kg,n).lt.1.d-15) fli1(kg,n) = 0.

c... rm & err are convergence parameters

      rm = dabs( fli1(kg,n) - r )
      if(err.lt.rm)err = rm

   15 continue
      if(err.lt.tole)goto 22
   20 continue
   22 return
      end

c------------------------- end of the subroutine -|

      SUBROUTINE OUTER(kg,u12,scm,fli1,qs1,ipoint)

      IMPLICIT REAL*8(a-h,o-z)
      COMMON/BLK2/xg(5,2),fcs(5,2),dsig(5,2)
      COMMON/BLK6/nn,ne,nt,ng,eigo,delt,ndyn,iskp
      DIMENSION u12(ng,nn,nt),scm(nn,nt),fli1(ng,nn),qs1(ng,nn)
      DIMENSION ipoint(nn,nt)

      kgg = 2
      if(ng.eq.1.or.kg.eq.2) kgg = 1
      do 5 n =1,nn
      add1 = 0.
      num = ipoint(n,1)
      do 6 m =1,num
      l = ipoint(n,m)
      if(m.eq.1) l = n
      if(kg.eq.2) go to 7
```

```
      if(ng.eq.1) then
      add1 = add1 + u12(kg,n,m)*fli1(kg,l)
      elseif(ng.eq.2) then
      add1 = add1 + u12(kgg,n,m)*fli1(kgg,l) + u12(kg,n,m)*fli1(kg,l)
      endif
      goto 6
    7 add1 = add1 + scm(n,m)*fli1(kgg,l)
    6 continue
      if(kg.eq.1) then
      qs1(kg,n) = xg(1,1)*add1/eigo
      elseif(kg.eq.2) then
      qs1(kg,n) = add1 + qs1(kgg,n)*xg(1,2)/xg(1,1)
      endif
    5 continue
      return
      end

c------------------------- end of the subroutine -|

      SUBROUTINE COMB(u11,b11,vel1)

      IMPLICIT REAL*8(a-h,o-z)
      COMMON/BLK2/xg(5,2),fcs(5,2),dsig(5,2)
      COMMON/BLK5/plam(6),bet(6),v(2),beta
      COMMON/BLK6/nn,ne,nt,ng,eigo,delt,ndyn,iskp
      COMMON/BLK7/ib(164),bv(164),la
      DIMENSION u11(ng,nn,nt),ipoint(nn,nt),
     &          b11(ng,nn,nt),vel1(ng,nn,nt)

c-------------------------------------------------------

      do 50 n =1,nn
      do 80 m = 1,nt
      if(ipoint(n,m).ne.0) goto 80
      ipoint(n,1) = m-1
      go to 20
   80  continue
   20  continue
   50  continue

      delth = delt/2.

      do 72 kg = 1,ng
      do 71 i=1,nn
```

```
      num = ipoint(i,1)
      do 70 j=1,num

      u11(kg,i,j) = vel1(kg,i,j) + delth * u11(kg,i,j)
      b11(kg,i,j) = vel1(kg,i,j) - delth * b11(kg,i,j)

 70    continue
 71   continue
 72   continue

      return
      end

c------------------------- end of the subroutine -|

c Data for test problems I,II,III

c       BLOCK DATA
c
c       IMPLICIT REAL*8(a-h,o-z)
c       COMMON/BLK5/plam(6),bet(6),v(2),beta
c       DATA plam / 0.127d-01,3.17d-02,1.15d-01,
c     &         3.11d-01,0.14d+01,0.387d+01    /
c       DATA bet/ 0.285d-03,0.15975d-02,0.141d-02,
c     &         0.30525d-02,0.96d-03,0.195d-03    /
c       DATA beta,v /0.0075,1.E08,2.2E05/
c       END

c Data for test problems IV and V.

      BLOCK DATA

      IMPLICIT REAL*8(a-h,o-z)
      COMMON/BLK5/plam(6),bet(6),v(2),beta
      DATA plam / 8.d-02,0.,0.,0.,0.,0.    /
      DATA bet/ 7.5d-03,0.,0.,0.,0.,0.    /
      DATA beta,v /0.0075,1.E08,2.2E05/
      END


c------------------------- end of the block data -|

      SUBROUTINE SOLVE(loop,u11,ipoint,b11,fli1,qs1
     & ,fli,flim,u12,tdep,c,pv,scm,t,nprec,npre
```

```
     & ,inin,period,sumcj)

        IMPLICIT REAL*8(a-h,o-z)
        COMMON/BLK2/xg(5,2),fcs(5,2),dsig(5,2)
        COMMON/BLK5/plam(6),bet(6),v(2),beta
        COMMON/BLK6/nn,ne,nt,ng,eigo,delt,ndyn,iskp
        DIMENSION u11(ng,nn,nt),ipoint(nn,nt),b11(ng,nn,nt)
     &          ,u12(ng,nn,nt)
        DIMENSION fli1(ng,nn),qs1(ng,nn),flim(ng,nn),fli(ng,nn),t(loop)

        if(ndyn.ne.2) goto 6
        do 5 i = 1,nn
         read(5,102) jj,(fli1(kg,i),kg=1,ng)
     5  continue
     6  do 10 i =1,nn
        do 10 kg=1,ng
        flim(kg,i) = fli1(kg,i)
    10  fli(kg,i) = flim(kg,i)

        do 12 ka = 1,ng
        do 12 kb = 1,nn
        do 12 kc = 1,nt
        u12(ka,kb,kc) = u12(ka,kb,kc)/eigo
    12  continue

        omeg = 0.

c... loop .....

        alp1 = 0.
        alp2 = 0.
        t(1) = 0.

        do 100 nc=2,loop
        t(nc) = t(nc - 1) + delt
        itr = 0

        CALL CALC(fli,flim,ipoint,nprec,npre,sumcj,qs1
     & ,u11,u12,b11,scm,tdep,c,pv,t,nc,loop,alp1,alp2,inin,period)

        if(nc.lt.3) goto 80
        do 20 kg=1,ng
        do 20 i=1,nn
        flim(kg,i)=fli(kg,i)
```

```
20   continue

80   if(ng.eq.1) goto 50
     ome = omeg
     if(ome.eq.0.) ome =1.45
     CALL INEST(nc,fli1,flim,fli,u11,scm,b11,qs1
    & ,ipoint,ome)

50   do 95 kg = 1,ng

     if(ng.eq.1) goto 51
     CALL SOURCE(kg,u12,fli1,qs1,pv,scm,ipoint,alp2)
51   CALL BOUND2(kg,u11,qs1)
     if(kg.gt.1) goto 52
     if(omeg.eq.0.) then
     CALL OMEGA(u11,ipoint,omeg)
     endif

52   CALL VECITR(kg,omeg,u11,qs1,fli1,ipoint)

95   continue
     itr = itr + 1
     if(ng.eq.1) goto 114
     den = 0.
     xnum = 0.
     do 110 i = 1,nn
     xnum = xnum + ( fli1(ng,i) - fli(ng,i) )**2
     den = den + fli(ng,i)**2
110  continue
     xnum = sqrt(xnum)
     den = sqrt(den)
     div = xnum/den
114  do 115 kg=1,ng
     do 115 i=1,nn
     fli(kg,i) = fli1(kg,i)
115  continue
     if(div.lt.1.e-6) goto 116
     goto 50
116  write(6,*) '# of outer iter. required for the following=',itr
     write(6,101) t(nc)
     do 70 i =1,nn
 70  write(6,102) i,(fli1(kg,i),kg=1,ng)
100   continue
101  format(//10X,21HCALCULATEDQUANTITIES/12X,
```

```
     + 29HNODAL VALUES FOR  FLUX AT  TIME,1x,d9.3,1x,4Hsec.)
   102 format(12X,i4,d14.5,d14.5)
       return
       end

c------------------------- end of the subroutine -|

       SUBROUTINE INEST(nc,fli1,flim,fli,u11,scm,b11,qs1
     & ,ipoint,ome)

       IMPLICIT REAL*8(a-h,o-z)
       COMMON/BLK6/nn,ne,nt,ng,eigo,delt,ndyn,iskp
       DIMENSION scm(nn,nt),b11(ng,nn,nt),fli1(ng,nn),flim(ng,nn)
       DIMENSION fli(ng,nn),qs1(ng,nn),ipoint(nn,nt)

       do 5 i =1,nn
       sum  = 0.
       num  = ipoint(i,1)
       do 10 j =1,num
       l = ipoint(i,j)
       if(j.eq.1) l = i
       if(nc.eq.2) goto 7
       favg  = fli(1,l)+ ome*( fli(1,l)- flim(1,l))
       goto 8
     7 favg  = flim(1,l)
     8 sum  = sum  + scm(i,j)*favg*delt  + b11(2,i,j)*flim(2,l)
    10 continue
     5 qs1(2,i)  = sum

c      choose  initialvector  to start SOR iteration method

       do 15 i =1,nn
       fli1(2,i) = flim(2,i)
    15 continue
       CALL BOUND2(2,u11,qs1)

c      calculate initialestimate of ' fli1(2,i), i=1,nn '

       CALL VECITR(2,ome,u11,qs1,fli1,ipoint)
       do 20 i =1,nn
    20 fli(2,i) = fli1(2,i)
       return
       end
```

```
c------------------------- end of the subroutine -|

      SUBROUTINE SOURCE(kg,u12,fli1,qs1,
     & pv,scm,ipoint,alp2)

      IMPLICIT REAL*8(a-h,o-z)
      COMMON/BLK5/plam(6),bet(6),v(2),beta
      COMMON/BLK6/nn,ne,nt,ng,eigo,delt,ndyn,iskp
      COMMON/BLK7/ib(164),bv(164),la
      DIMENSION u12(ng,nn,nt),scm(nn,nt),fli1(ng,nn)
      DIMENSION qs1(ng,nn),pv(ng,nn),ipoint(nn,nt)

      if(kg.eq.2) go to 30
      kgg = 2
      alpa = alp2
      go to 35
 30   kgg = 1
      alpa = delt/2.
 35   continue
      do 45 i=1,nn
      add = 0.
      num = ipoint(i,1)
      do 40 ji =1,num
      l = ipoint(i,ji)
      if(ji.eq.1) l = i
      if(kg.eq.2) go to 41
      add = add + u12(kgg,i,ji)*fli1(kgg,l)
      go to 40
 41   add = add + scm(i,ji)*fli1(kgg,l)
 40   continue
      qs1(kg,i) = pv(kg,i) + add*alpa
 45   continue
      return
      end

c------------------------- end of the subroutine -|

      SUBROUTINE CALC(fli,flim,ipoint,nprec,npre,sumcj
     & ,qs1,u11,u12,b11,scm,tdep,c,pv,t,nc,loop,alp1,alp2,inin,period)

      IMPLICIT REAL*8(a-h,o-z)
      COMMON/BLK2/xg(5,2),fcs(5,2),dsig(5,2)
      COMMON/BLK5/plam(6),bet(6),v(2),beta
      COMMON/BLK6/nn,ne,nt,ng,eigo,delt,ndyn,iskp
```

```
      COMMON/BLK7/ib(164),bv(164),la
      DIMENSION u11(ng,nn,nt),b11(ng,nn,nt),c(nn,npre),pv(ng,nn)
      DIMENSION flim(ng,nn),fli(ng,nn),ipoint(nn,nt)
      DIMENSION u12(ng,nn,nt),scm(nn,nt),tdep(nn,nt),t(loop)
      DIMENSION qs1(ng,nn),sumcj(nn)
      DIMENSION exj0(6), exj1(6),s1(6),s2(6)

      nc1 = nc - 1
      nc2 = nc - 2
 3    if(nc.ne.2) go to 9
      delth = delt/2.
      sum1 = delth
      sum2 = delth
      if(nprec.eq.0) go to 51
      do 11 j=1,nprec
      plamj = 1./plam(j)
      plamj2 = plamj**2
      exj0(j) = dexp(-plam(j)*delt)
      sum1 = sum1 + bet(j)*( plamj + (exj0(j)-1.)*
     & ( plamj+plamj2/delt ) )
      sum2 = sum2 + bet(j)*( - plamj -(exj0(j)-1.)*plamj2/delt )
11    continue
51    alp1 = sum1
      alp2 = sum2

      if(ng.gt.1) goto 24
      do 23 i=1,nn
      num = ipoint(i,1)
      do 23 ji = 1,num
      l = ipoint(i,ji)
      if(ji.eq.1)l = i
      u11(ng,i,ji) = u11(ng,i,ji) - alp2*u12(ng,i,ji)
23    continue
      go to 9
24    if(fcs(1,1).eq.0.) goto 9
      do 26 i=1,nn
      num = ipoint(i,1)
      do 26 ji = 1,num
      l = ipoint(i,ji)
      if(ji.eq.1)l = i
      u11(1,i,ji) = u11(1,i,ji) - alp2*u12(1,i,ji)
26    continue

 9    if(nprec.eq.0) go to 52
```

```
      do 10 j=1,nprec
      plamj = 1./plam(j)
      plamj2 = plamj**2
      ex1 = dexp( plam(j)*t(nc1) )
      ex2 = dexp( plam(j)*t(nc) )
      exj1(j) = 1./ex1 - 1./ex2
      if(nc.le.2) go to 10
      ex0 = dexp( plam(j)*t(nc2) )
      s1(j) = plamj2*ex1 - (delt/plam(j) + plamj2)*ex0
      s2(j) = (delt/plam(j)- plamj2)*ex1 + plamj2*ex0
10    continue

      if(ng.gt.1.and.fcs(1,1).ne.0.)  go to 55
      do 30 i=1,nn
      do 30 j=1,nprec
      if(nc.lt.3) go to 31
      c(i,j) = c(i,j) + (flim(ng,i)*s1(j) +
     & fli(ng,i)*s2(j))*bet(j)/delt
      go to 30
31    c(i,j) = flim(ng,i)*bet(j)/plam(j)
30    continue
      go to 52
55    do 62 i=1,nn
      num = ipoint(i,1)
      topl1 = 0.
      topl2 = 0.
      do 64 ji = 1,num
      l = ipoint(i,ji)
      if(ji.eq.1) l = i
      if(nc.lt.3) goto 65
      topl1 = topl1 + u12(1,i,ji)*flim(1,l)+ u12(2,i,ji)*flim(2,l)
      topl2 = topl2 + u12(1,i,ji)*fli(1,l)+ u12(2,i,ji)*fli(2,l)
      goto 64
65    topl1 = topl1 + u12(1,i,ji)*flim(1,l)
      topl2 = topl2 + u12(2,i,ji)*flim(2,l)
64    continue
      do 62 j=1,nprec
      if(nc.lt.3) goto 68
      c(i,j) = c(i,j) + (topl1*s1(j) + topl2*s2(j))*bet(j)/delt
      goto 62
68    c(i,j) = (topl1 + topl2)*bet(j)/plam(j)
62    continue
      do 70 i = 1,nn
      sumcj(i) = 0.
```

```
      do 70 j = 1,nprec
   70 sumcj(i) = sumcj(i) + exj1(j)*c(i,j)


c..   'inin' indicates ramp or sinusoidal or step change

   52 if(inin.eq.1) go to 12
      if(inin.eq.2) go to 27
      if(nc.ge.3) go to 14
      con1 = delt**2/6.
      con2 = delt**2/3.
      go to 13
   14 con1 = delth*delt
      con2 = con1
      go to 13

c     priod is Sin(period*t)

   12 freq = period
      z1 = freq * t(nc1)
      z2 = freq * t(nc)
      sin1 = dsin(z1)
      sin2 = dsin(z2)
      cos1 = dcos(z1)
      cos2 = dcos(z2)
      fdelt = freq*delt
      fi = fdelt * freq
      sind = sin2 - sin1
      if(nc.ge.3) go to 17
      con1 = (fdelt*cos1 -sind)/fi
      con2 = -(fdelt*cos2 - sind)/fi
      go to 13
   17 z0 = freq*t(nc2)
      sin0 = dsin(z0)
      cos0 = dcos(z0)
      sins = -sin2 + 2.*sin1 - sin0
      sinb = sin1 - sin0
      con1 = (fdelt*(cos1 - cos0) + sins)/fi + sinb*delth
      con2 = -(fdelt*(cos2 - cos1) + sins)/fi + sinb*delth
   13 do 20 i=1,nn
      num = ipoint(i,1)
      do 20 ji = 1,num
      b11(ng,i,ji) = b11(ng,i,ji) - con1*tdep(i,ji)
      u11(ng,i,ji) = u11(ng,i,ji) + con2*tdep(i,ji)
```

```
 20   continue

 27   do 50 kg = 1,ng
      kkg = 2
      if(ng.eq.1.or.kg.eq.2)  kkg = 1
      do 50 i=1,nn
      num = ipoint(i,1)
      sum3 = 0.
      do 40 ji = 1,num
      l = ipoint(i,ji)
      if(ji.eq.1) l = i
      if(kg.eq.2) goto 41
      sum4 = 0.
      if(nprec.eq.0) go to 53
      if(ng.gt.1.and.fcs(1,1).ne.0.)  go to 53
      do 35 j=1,nprec
      sum4 = sum4 + exj1(j)*u12(kkg,i,ji)*c(l,j)
 35   continue
 53   sum3 = sum3 + sum4+ alp1*u12(kkg,i,ji)*fli(kkg,l)
     & + b11(kg,i,ji)*fli(kg,l)
      if(ng.gt.1.and.fcs(1,1).ne.0.)  sum3 = sum3 +
     & alp1*u12(kg,i,ji)*fli(kg,l)
      go to 40
 41   sum3 = sum3 + scm(i,ji)*fli(kkg,l)*delth
     & + b11(kg,i,ji)*fli(kg,l)
 40   continue
      if(ng.gt.1) then
      pv(kg,i) = sum3
      else
      qs1(kg,i) = sum3
      endif
 50   continue
      if(ng.gt.1.and.fcs(1,1).ne.0.and.nprec.ne.0)    then
      do 190 i=1,nn
 190  pv(1,i) = pv(1,i) + sumcj(i)
      endif

 200  return
      end

c------------------------- end of the subroutine -|

c---------- subroutine omega(a,x,n,nim,tol,d) ----------c
c  computes  the highest eigenvalue and corresponding  eigenvector
```

```
c  by the stodolla-vianello method.
c  a = system  matrix, d = eigenvector,  x = auxiliray vector
c  n = order of the system  matrix, nim = maximum number of
c  iteration allowed, tol = tolerance
c  notice that x should originally contain the first trial
c  eigenvector. upon the solution is completed it will contain
c  the actual eigenvector.
c-------------------------------------------------------------c

      SUBROUTINE OMEGA(u11,ipoint,omeg)

      IMPLICIT REAL*8(a-h,o-z)
      COMMON/BLK6/nn,ne,nt,ng,eigo,delt,ndyn,iskp
      DIMENSION u11(ng,nn,nt),ipoint(nn,nt)
      DIMENSION om(1681,10),  x(1681),xx(1681),y(1681)
c-----------------------------------------
      tole=0.001
      iter=0
      nim=90

      do 13 n=1,nn
      om(n,1)=0.
      num = ipoint(n,1)
      do 12 m=2,num
   12 om(n,m) = u11(1,n,m)/u11(1,n,1)
   13 continue

      do 10 i=1,nn
   10 x(i)=1.
    1 do 2 i=1,nn
    2 xx(i)=x(i)
      iter = iter + 1
      do 3 i=1,nn
      y(i)=0.
      num = ipoint(i,1)
      do 3 j=2,num
      l = ipoint(i,j)
    3 y(i) = y(i) + om(i,j)*x(l)
      d=y(1)
      do 4 i=1,nn
    4 x(i)=y(i)/d
      do 5 i=1,nn
      if(abs(xx(i)-x(i))-tole)5,5,6
    5 continue
```

```
      go to 9
    6 if(iter-nim)1,9,9
    9 rad = d
      if(abs(rad).gt.1.) then
      write(6,*) 'SPECTRAL RADIUS IS  > 1.'
      write(6,*) 'ATTEMPT: DEFAULTACCEL. VALUE WILL BE ASSIGNED'
      omeg = 1.8
      goto 15
      endif
      omeg=2./(1.+ dsqrt(1.-rad**2.))
   15 write(6,*) 'OMEG=',omeg
      return
      end

c-------------------------- end of the subroutine -|

c------------ subroutine mesh ----------------------------c
c  wf= weighting factor
c  intervals along a generating line will become:
c  progressively shorter if wf<1.
c  stay equal           if wf=1.(default)
c  progressively longer if wf>1.
c  nx(i)= the number of intervals for each generating line
c  xf(i),yf(i),xl(i),yl(i)=the coordinates of the two end points
c  mid(i,j)= material identification number
c-------------------- ----------------------------c

      SUBROUTINE MESH

      IMPLICIT REAL*8(a-h,o-z)
      COMMON/BLK1/nh,mntl(3200),nel(3200,3),xc(1681),
     & yc(1681),mid(40,40)
      COMMON/BLK6/nn,ne,nt,ng,eigo,delt,ndyn,iskp
      COMMON/BLK7/ib(164),bv(164),la
      DIMENSION nx(41),yl(41),xf(41),yf(41),nod(4),sum1(42),xl(41)
C-----------------------------------------------------------------
      wf = 1.
      read(5,*) ny,nh
      write(6,1001) ny
       do 100 i=1,ny
        read(5,*) nx(i),xf(i),yf(i),xl(i),yl(i)
        write(6,1000) nx(i),xf(i),yf(i),xl(i),yl(i)
  100    continue
      nym = ny - 1
```

```
      do 110 i=1,nym
110   read(5,*) (mid(i,j),j=1,nx(i))
      write(6,1429)
      write(6,1431)
      n = 0
      nn = 0
      do 350 i=1,ny
      nxi = nx(i)+1
      sum1(1) = 0.0
      sum1(2) = 1.0
      sum = 1.0
      if(nxi-2) 190,291,190
190     do 250 k=3,nxi
      sum1(k)= sum1(k-1)*wf
      sum = sum + sum1(k)
250     continue
291     continue
      x=xf(i)
      y=yf(i)
       do 300 j=1,nxi
      n = n + 1
      x = (xl(i)-xf(i))*sum1(j)/sum + x
      y=(yl(i)-yf(i))*sum1(j)/sum + y
      xc(nn+j) = x
      yc(nn + j)=y
      write(6,1430) n,xc(j+nn),yc(j+nn)
300     continue
      nn = nn + nxi
350   continue
      n = 0
      nsum = 0
      nyi = ny - 1
      write(6,1432)
       do 600 i=1,nyi
      nxi = nx(i)
       do 500 j=1,nxi
      if(j-nxi)379,371,379
371   if(nx(i+1)-nx(i))380,379,401
379   nod(1) = j + nsum
      nod(2) = nod(1) + 1
      nod(3) = nod(2) + nxi + 1
      nod(4) = nod(3) - 1
      goto 412
380   nod(1) = nod(2)
```

```
      nod(2) = nod(1) + 1
      nod(4) = 0
      goto 412
  401 nod(1) = j + nsum
      nod(2) = nod(1) + 1
      nod(3) = nod(2) + nxi + 1
      nod(4) = nod(3) - 1
      n = n + 1
      mntl(n) = mid(i,j)
      write(6,1470) n,nod(1),nod(2),nod(3),mntl(n)
      nel(n,1) = nod(1)
      nel(n,2) = nod(2)
      nel(n,3) = nod(3)
      n = n + 1
      mntl(n) = mid(i,j)
      write(6,1470) n,nod(1),nod(3),nod(4),mntl(n)
      nel(n,1) = nod(1)
      nel(n,2) = nod(3)
      nel(n,3) = nod(4)
      nod(1) = nod(2)
      nod(2) = nod(3) + 1
      nod(4)=0
  412 n = n+1
      mntl(n) = mid(i,j)
      write(6,1470) n,nod(1),nod(2),nod(3),mntl(n)
      nel(n,1) = nod(1)
      nel(n,2) = nod(2)
      nel(n,3) = nod(3)
      if(nod(4)) 433,434,433
  433 n = n + 1
      mntl(n) = mid(i,j)
      write(6,1470) n,nod(1),nod(3),nod(4),mntl(n)
      nel(n,1) = nod(1)
      nel(n,2) = nod(3)
      nel(n,3) = nod(4)
  434   continue
  500   continue
      nsum = nsum + nxi + 1
  600  continue
      ne = n

c...  ki will have the node numbers which have zero boundary conditions
c...  number of boundary conditions = la
c...  if nh=0 Impose zero boundary conditions on 4 sides
```

```
c...    if nh=1 Impose zero boundary conditions on 2.nd. and 3.rd. side.

        if(iskp.ne.0) return
        npp = 0
        la = 0
        if(nh.ne.0) goto 10
        npp = nx(1)
        do 5 i = 1,npp
        ib(i) = i
   5    continue
        la = npp

  10    is = 1
        ki = 0
        nj = 1
        if(nh.eq.0) is = 2
        do 15 j=1,ny-1
        la = la + 1
        ki = ki + nx(j) + 1
        ib(npp + nj ) = ki
        nj = nj + is
  15    continue
        ki = ki + 1

        if(nh.ne.0) goto 25
        njj = 2
        it = 0
        ib(npp + 2) = npp + 2
        do 20 i = 2,ny-1
        la = la + 1
        it = ib(npp + njj - 1) + 1
        ib(npp + njj) = it
        njj = njj + is
  20    continue

  25    if(nh.eq.0) nj = nj - 1
        do 30 i=1,nx(ny) + 1
        la = la + 1
        ib(npp + nj) = ki
        nj = nj + 1
        ki = ki + 1
  30    continue
1000      format(i5,5x,4f10.5)
1001      format(/,4x,'INPUT DATA FOR MESH GENERATION'/,i5)
```

```
1429     format(/,4x,'GENERATED COORDINATES & NODE NUMBERS')
1431     format(/,4x,' NODAL POINT X CO-ORDINATEY CO-ORDINATE')
1430     format(7x,i4,2f16.8)
1432     format(/,5x,'ELMENT NO NODAL POINTS OF THE ELEMENTS',
   & ' MATERIAL')
1470     format(3x,i8,5x,3i8,3x,i8)
      return
      end

c------------------------- end of the subroutine -|

      SUBROUTINE INSPECTOR(ick,lnode)

      IMPLICIT REAL*8(a-h,o-z)
      COMMON/BLK1/nh,mntl(3200),nel(3200,3),xc(1681),
   & yc(1681),mid(40,40)
      COMMON/BLK6/nn,ne,nt,ng,eigo,delt,ndyn,iskp
      DIMENSION ick(nn),lnode(3)

      do 3 i=1,nn
   3  ick(i)=0

c.. check if any node nuber exceeds  nn .....

      do 7 i=1,ne
      do 5 j=1,3
      k=nel(i,j)
      ick(k)=1
   5  if(k.gt.nn) write(6,100) j,i,nn
   7  continue

c.....  check if all node numbers through nn are included

      do 9 i=1,nn
   9  if(ick(i).eq.0) write(6,101) i

c...  calculation of the bandwith

      inbw=0
      nbw=0
      do 15 kk=1,ne
      do 13 i=1,3
  13  lnode(i)= nel(kk,i)
      lk=2
```

```
      do 11 i=1,lk
      ij=i+1
      do 11 j=ij,3
      nb=iabs(lnode(i)-lnode(j))
      if(nb.eq.0) write(6,102) kk
      if(nb.le.nbw) goto 11
      inbw=kk
      nbw=nb
   11 continue
   15 continue
      nbw=nbw+1
      write(6,103) nbw,inbw
  100 format(/10x,4hNODE,i4,11h OF  ELEMENT,i4,13hEXCEEDS NN =,i4)
  101 format(/10x,4hNODE,i4,15h DOES  NOT  EXIST)
  102 format(/10x,7hELEMENT,i3,13hHAS TWO  NODES/
     + 10x,25hWITH  THE SAME NODE NUMBER)
  103 format(//10x,12hBANDWITH IS,i4,11h IN  ELEMENT,i4)
      return
      end
c------------------------- end of the subroutine -|

      SUBROUTINE NULL(u11,b11,ipoint,vel1,u12,scm,tdep)

      IMPLICIT REAL*8(a-h,o-z)
      COMMON/BLK6/nn,ne,nt,ng,eigo,delt,ndyn,iskp
      DIMENSION u11(ng,nn,nt),b11(ng,nn,nt)
     1 ,ipoint(nn,nt),vel1(ng,nn,nt),u12(ng,nn,nt)
      DIMENSION tdep(nn,nt),scm(nn,nt)

      do 5 il=1,nn
      ipoint(il,1) = il
      do 5 jl=1,nt
      do 4 kg  = 1,ng
      u11(kg,il,jl) = 0.
      b11(kg,il,jl) = 0.
      vel1(kg,il,jl) = 0.
      u12(kg,il,jl) = 0.
    4 continue
      tdep(il,jl) = 0.
      scm(il,jl) = 0.
    5 continue
      return
      end
c------------------------- end of the subroutine -|
```

APPENDIX B:

NUCLEAR DATA FOR TEST PROBLEMS

Nuclear cross-section data of the Test problems presented in Chapter 3.

The following nuclear data are given in units of $cm.^{-1}$ except Diffusion Coefficient whose unit is cm.

TEST PROBLEM 3.1 :

Material	X-Sects.	Fast Group	Thermal Group
CORE	D_g	2.680004513127601	1.578767222771633
	$\Sigma_{r,g}$	5.457703768309E-02	1.44960876000E-02
	$\tau*\Sigma_{f,g}$	3.08344800000E-02	2.5200000000E-02
	p_g	0.575	0.425
	$\Sigma_{f,g}$	1.23338000000E-02	1.0080000000E-02
	$\Sigma_{gg'}$	4.07920706831E-02	0.

TEST PROBLEM 3.2 :

Material	Cross Sections	Fast Group	Thermal Group
FUEL	D_g	1.5	0.4
	$\Sigma_{r,g}$	0.0623	0.2
	$\tau*\Sigma_{f,g}$	0.0	0.218
	p_g	1.0	0.0
	$\Sigma_{gg'}$	0.06	0.0
MODERATOR	D_g	1.2	0.15
	$\Sigma_{r,g}$	0.101	0.02
	$\tau*\Sigma_{f,g}$	0.0	0.0
	p_g	0.0	0.0
	$\Sigma_{gg'}$	0.1	0.0

TEST CASE 3.3 :

The nuclear data for the CORE section is the same as Test Case 3.1

Material	X-Sects.	Fast Group	Thermal Group
BLANKET	D_g	1.549689314102148	1.094165406735201
	$\Sigma_{r,g}$	1.120250077678E-01	9.00749989600E-03
	$\tau*\Sigma_{f,g}$	5.246412000000E-02	3.42000000000E-04
	p_g	0.575	0.425
	$\Sigma_{f,g}$	2.09856000000E-02	1.3680000000E-04
	$\Sigma_{gg'}$	8.898901317976E-02	0.

TEST CASE 3.4 :

Cross Sections	Fuel	Water	Control Rod
D_1	1.7	1.7	0.5
$\Sigma_{r,1}$	0.016	0.041	0.1
$\tau*\Sigma_{f,1}$	0.	0.	0.
Σ_{12}	0.016	0.041	0.
D_2	0.42	0.23	0.1
$\Sigma_{r,2}$	0.055	0.012	1.5
$\tau*\Sigma_{f,2}$	0.0832	0.	0.
Σ_{21}	0.	0.	0.

Nuclear Data of the Test Problems presented in Chapter 4.

Cross section data for Test Cases 4.1, 4.2, and 4.3 are the same as the Test Case 3.2 of Chapter 3. Kinetic parameters are given below.

Neutron speed in fast group : v_1 = 1.0E+08 cm/sec

Neutron speed in thermal group : v_2 = 2.2E+05 cm/sec.

Delayed Neutron Constants :

Group	β_j	Γ_j (sec^{-1})
1	0.285E-03	0.127E-01
2	0.15975E-02	0.317E-01
3	0.141E-02	0.115
4	0.30525E-02	0.311
5	0.960E-03	1.4
6	0.195E-03	3.87

TEST PROBLEM 4.4 AND 4.5:

	D_1 (cm)	D_2 (cm)	$\Sigma_{r,1}$ (1/cm)	$\Sigma_{r,2}$ (1/cm)	$\tau*\Sigma_{f,1}$ (1/cm)	$\tau*\Sigma_{f,2}$ (1/cm)	Σ_{12} (1/cm)	p_1
SEED	1.4	0.4	0.02	0.15	$0.007/k_{eff}$	$0.2/k_{eff}$	0.01	1.
BLANKET	1.3	0.5	0.018	0.05	$0.003/k_{eff}$	$0.06/k_{eff}$	0.01	1.

$k_{eff} = 0.914193$

Inverse Neutron Speeds : $1/v_1 = 1.E-07$ sec/cm

$1/v_2 = 5.E-06$ sec/cm

Delayed Neutron Parameters: $\Gamma = 0.08$ sec^{-1}

$\beta = 0.0075$

APPENDIX C:

FIGURES FOR TEST PROBLEMS 3.1-3.4

FIGURES FOR TEST PROBLEMS 4.1-4.5

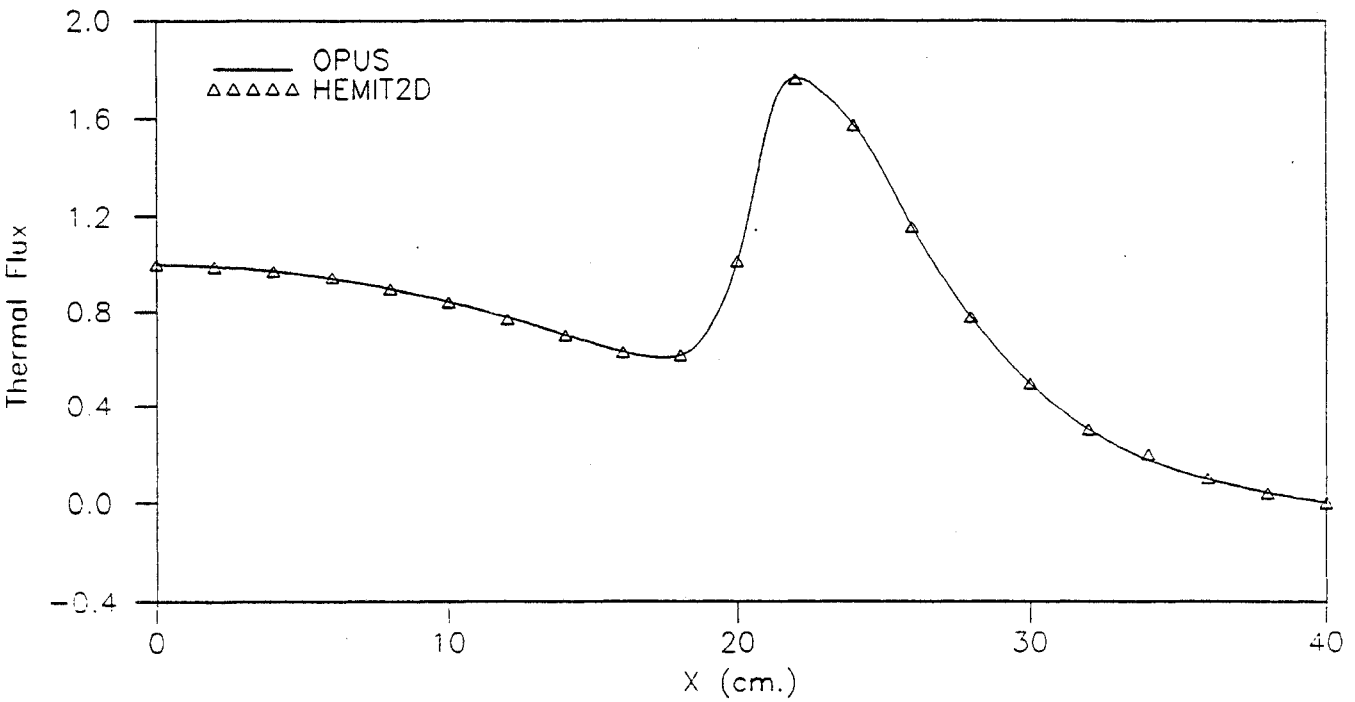

Figure (C.1) Test Case **3.2**
Thermal Neutron Flux at y=0. cm.

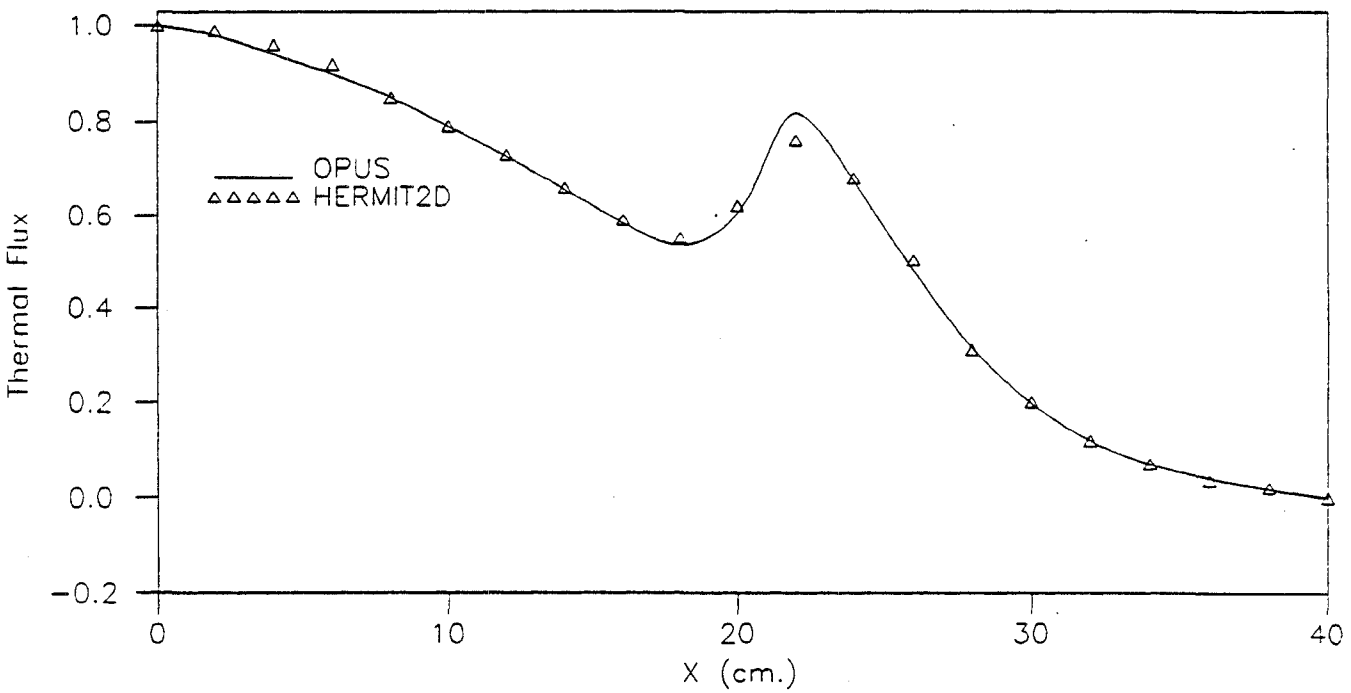

Figure (C.2). Test Case **3.2**
Thermal Neutron Flux at y=20. cm.

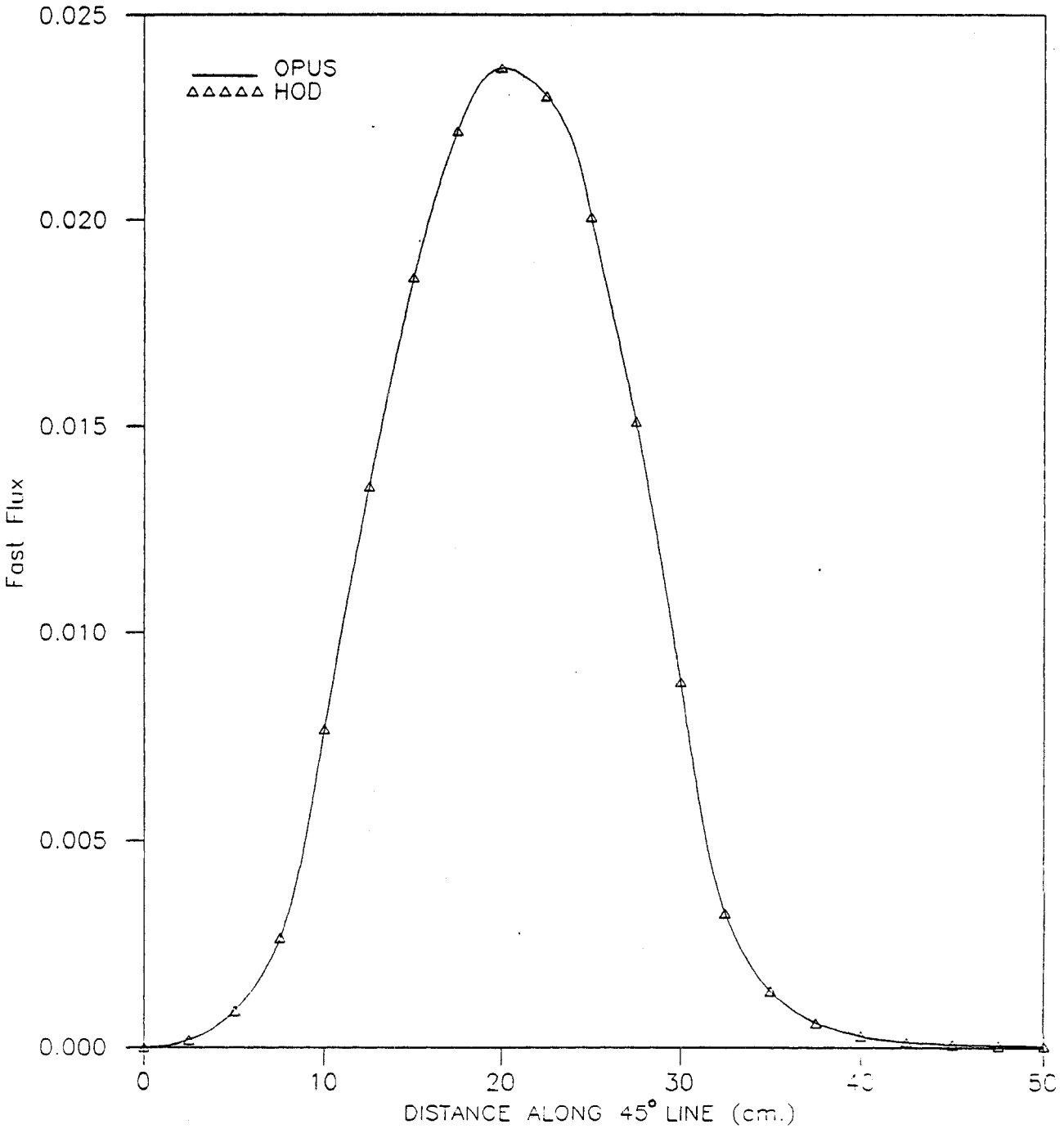

Figure (C.3). Test Case **3.3**
Fast Flux along Diagonal Line
(Half way distance)

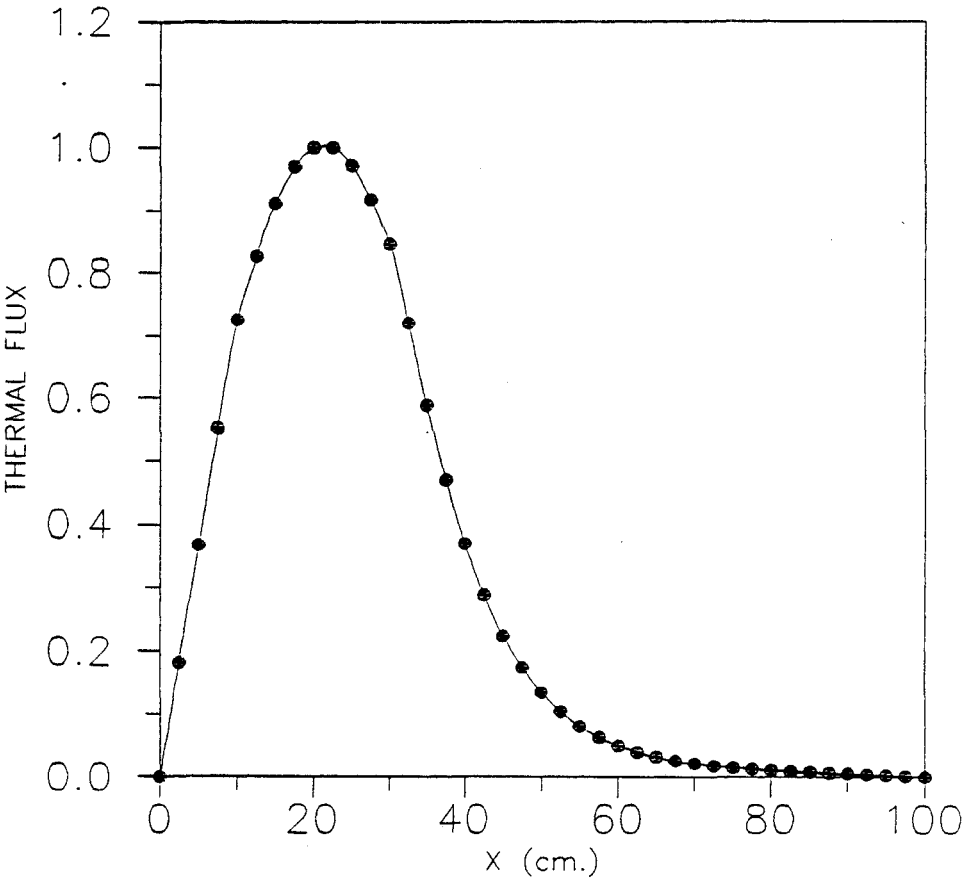

Figure (C.4). Test Case **3.3**
Thermal Flux at y=20 cm.

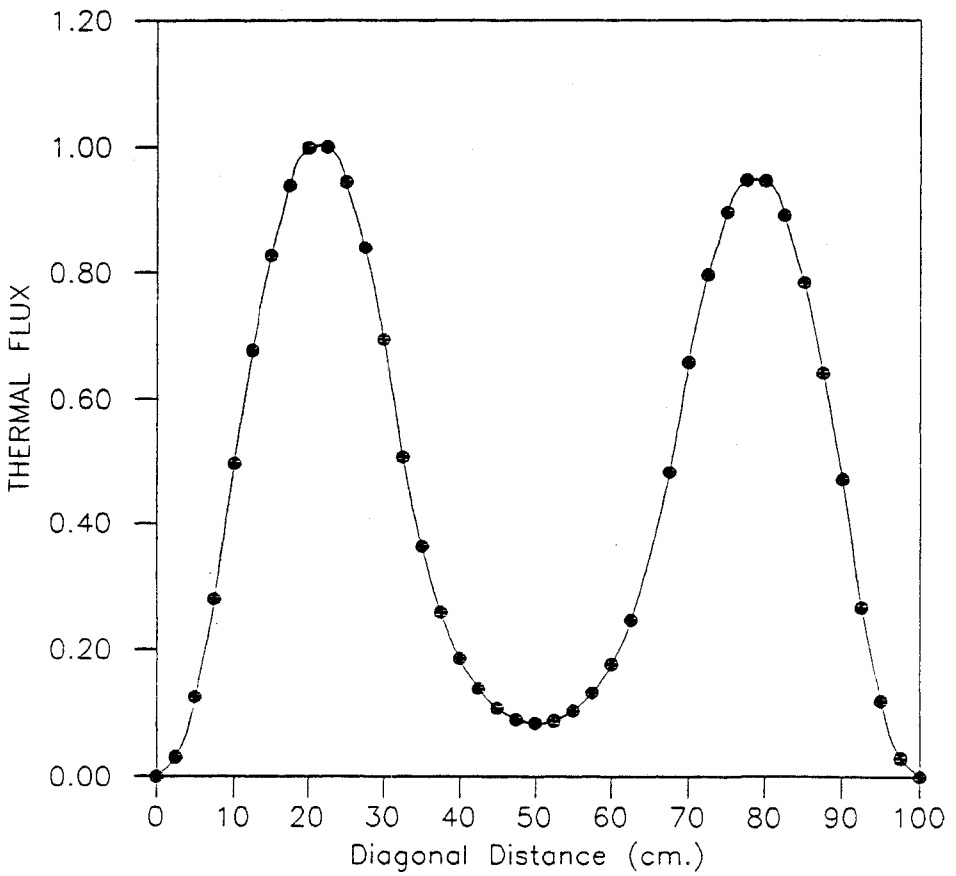

Figure (C.5). Test Case **3.3**
Thermal Flux along diagonal distance

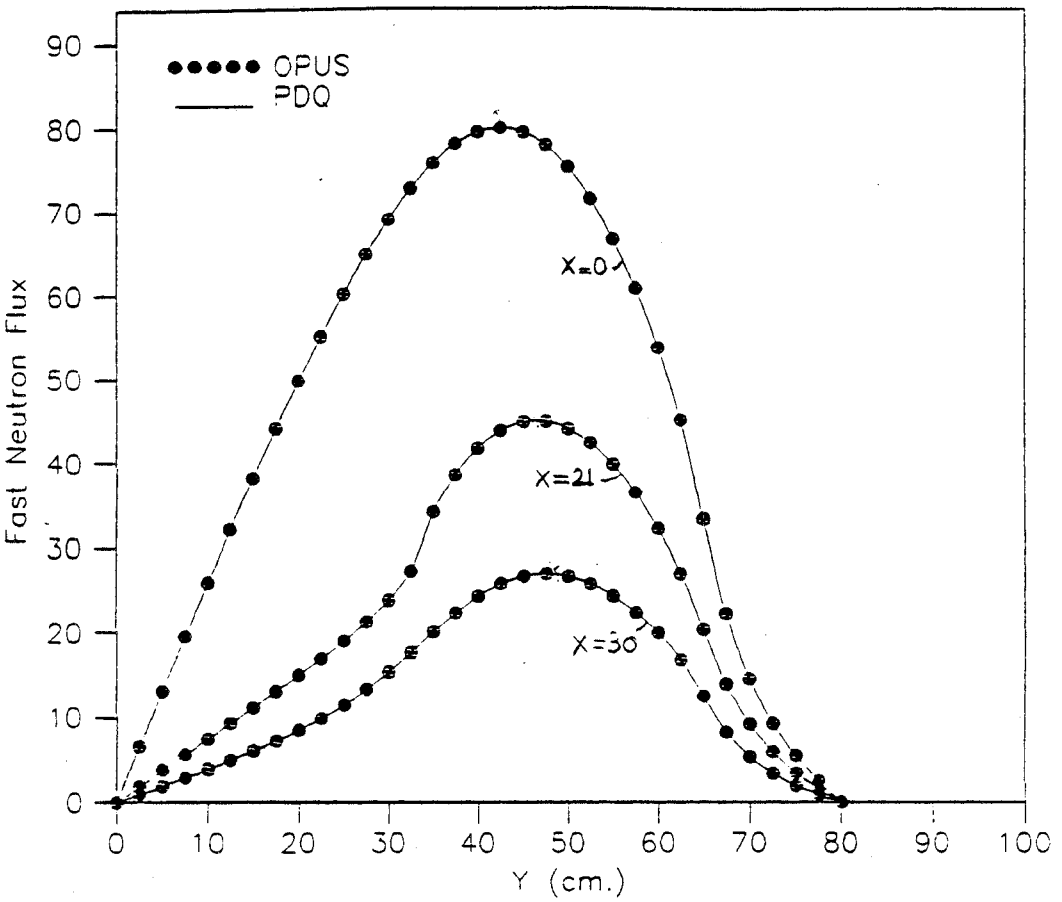

Figure (C.6). Test Case **3.4**
Fast Fluxes at x=0, 21, and 30 cm.

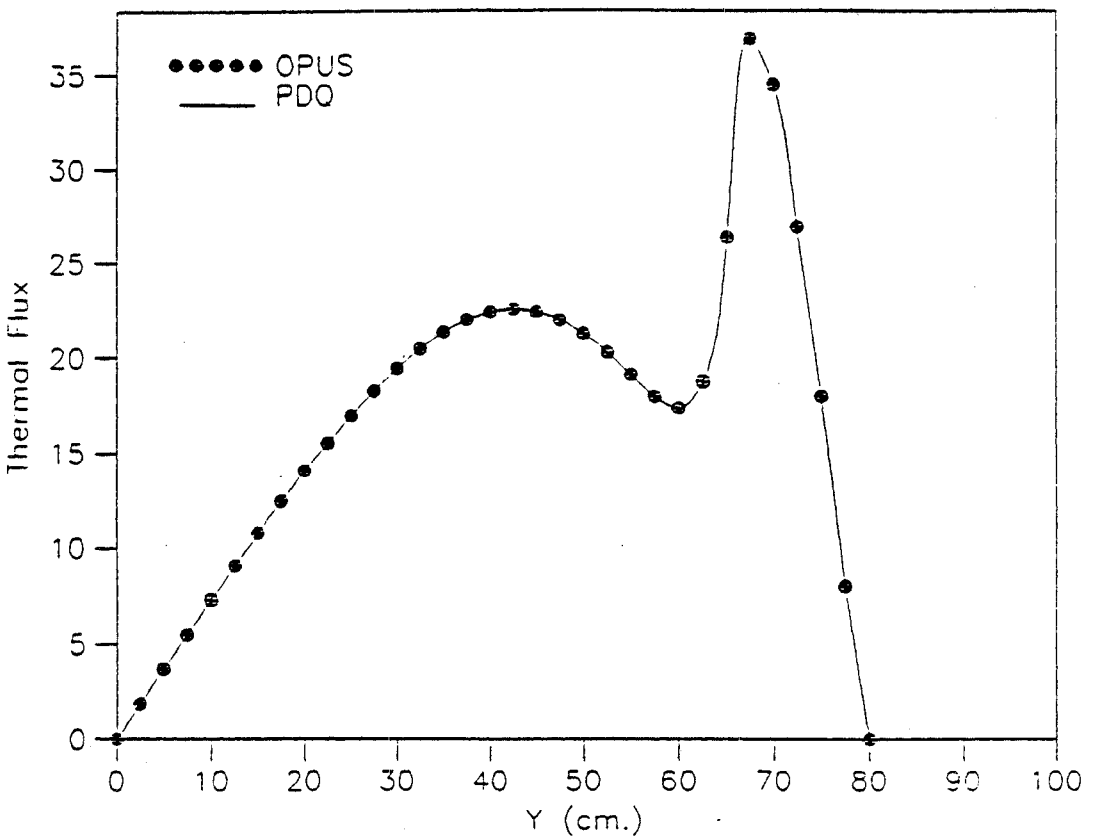

Figure (C.7). Test Case 3.4
Thermal Flux at x=0 cm.

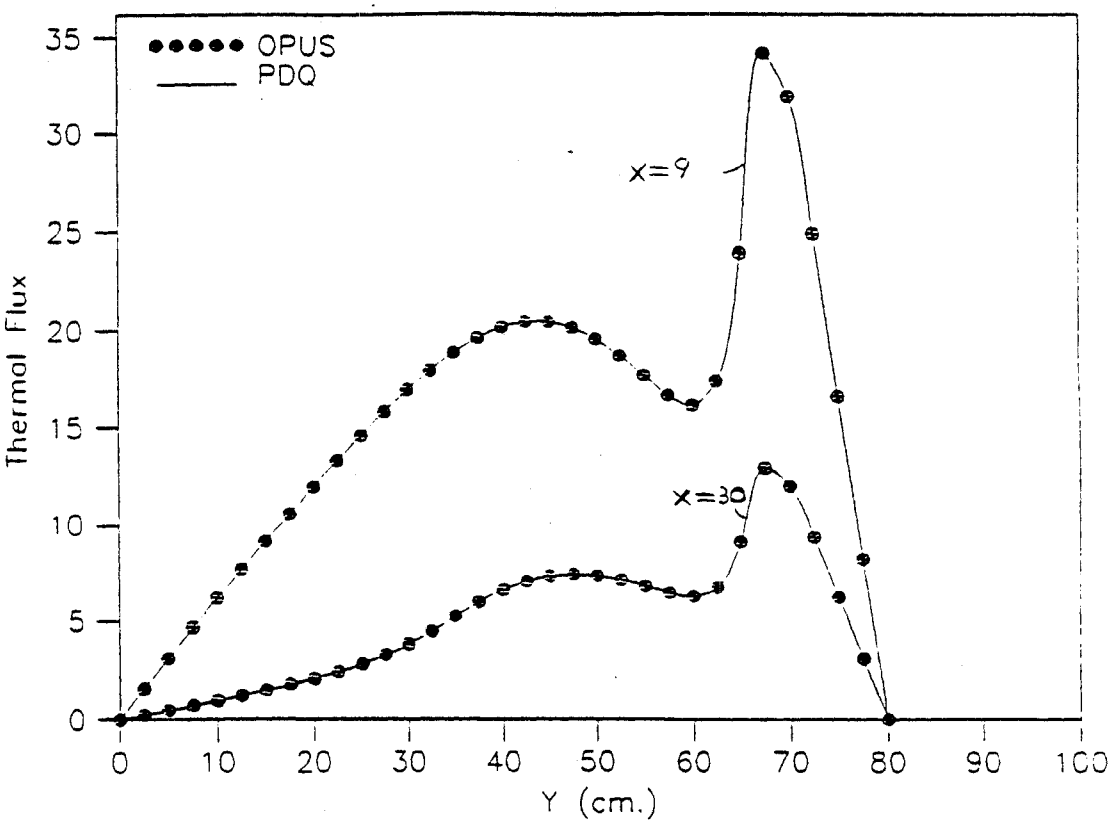

Figure (C.8). Test Case **3.4**
Thermal Flux at x=9 and x=30 cm.

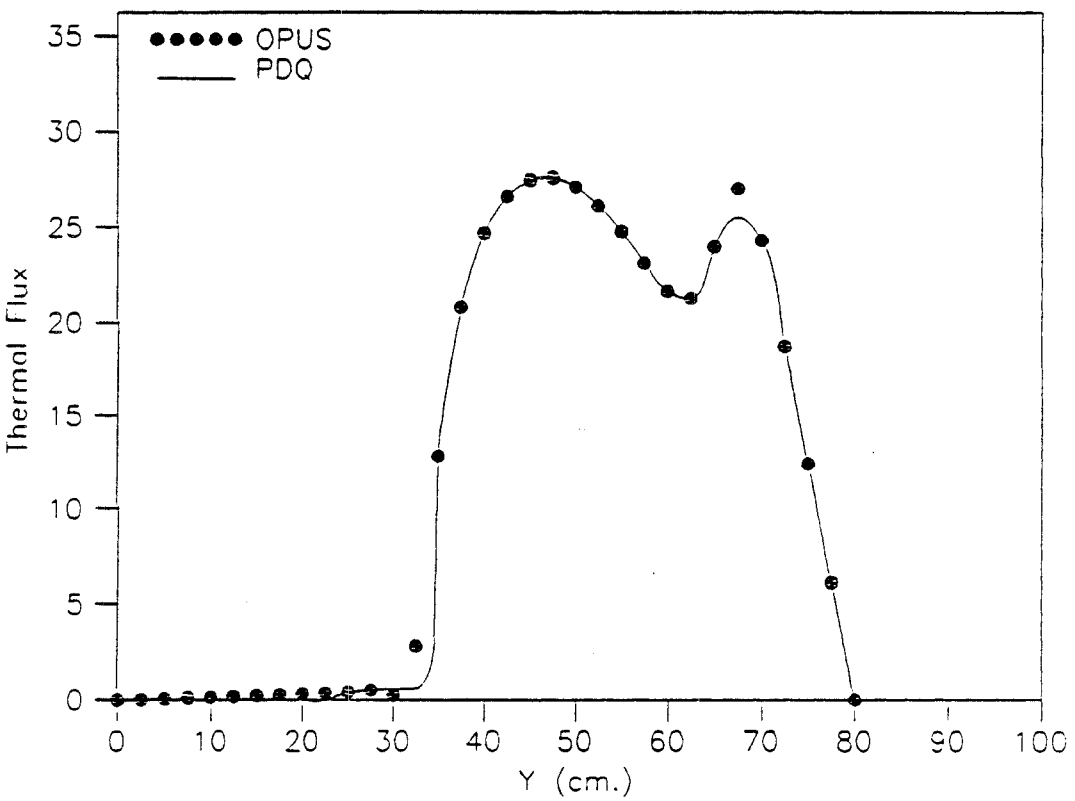

Figure (C.9). Test Case **3.4**
Thermal Flux at x=21 cm.

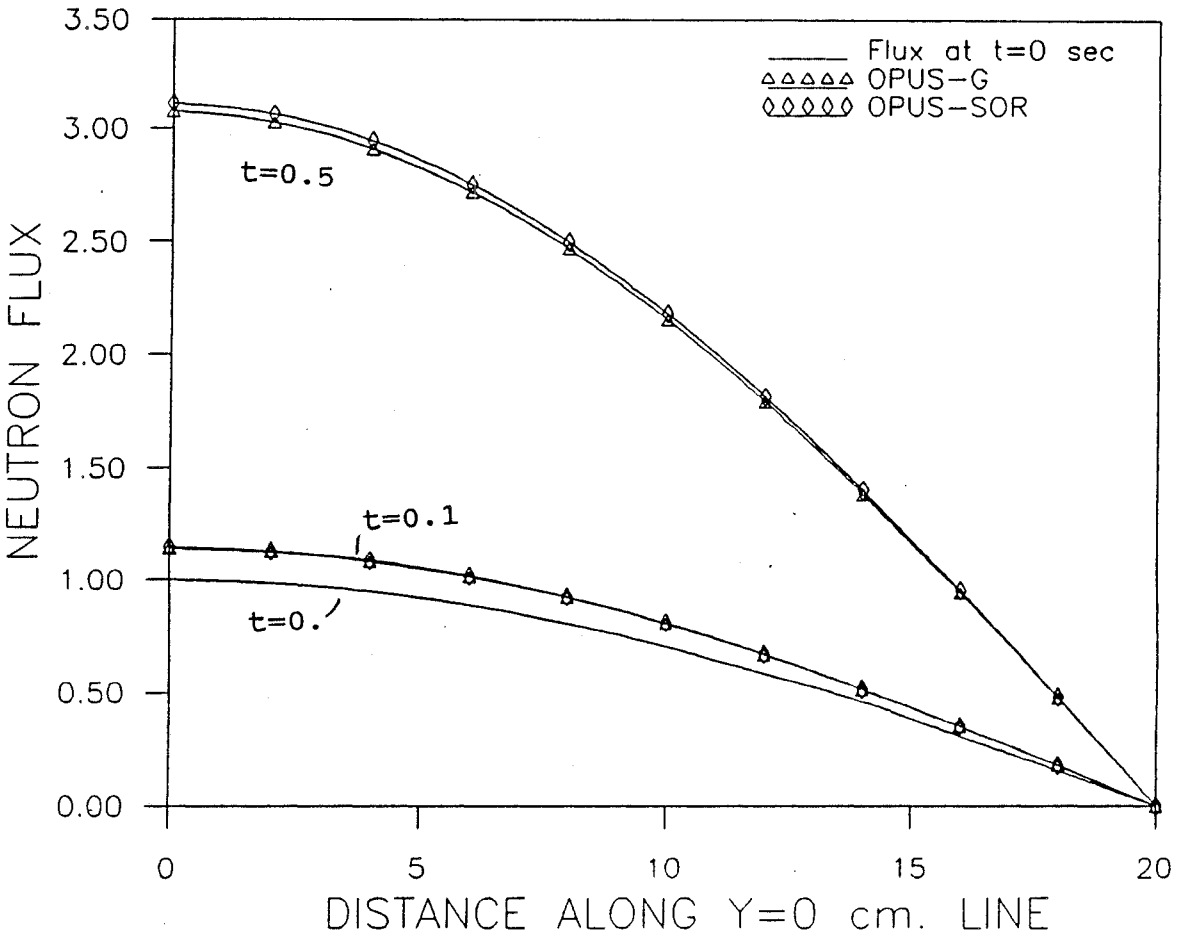

Figure (C.10) Time dependent problem **4.1**

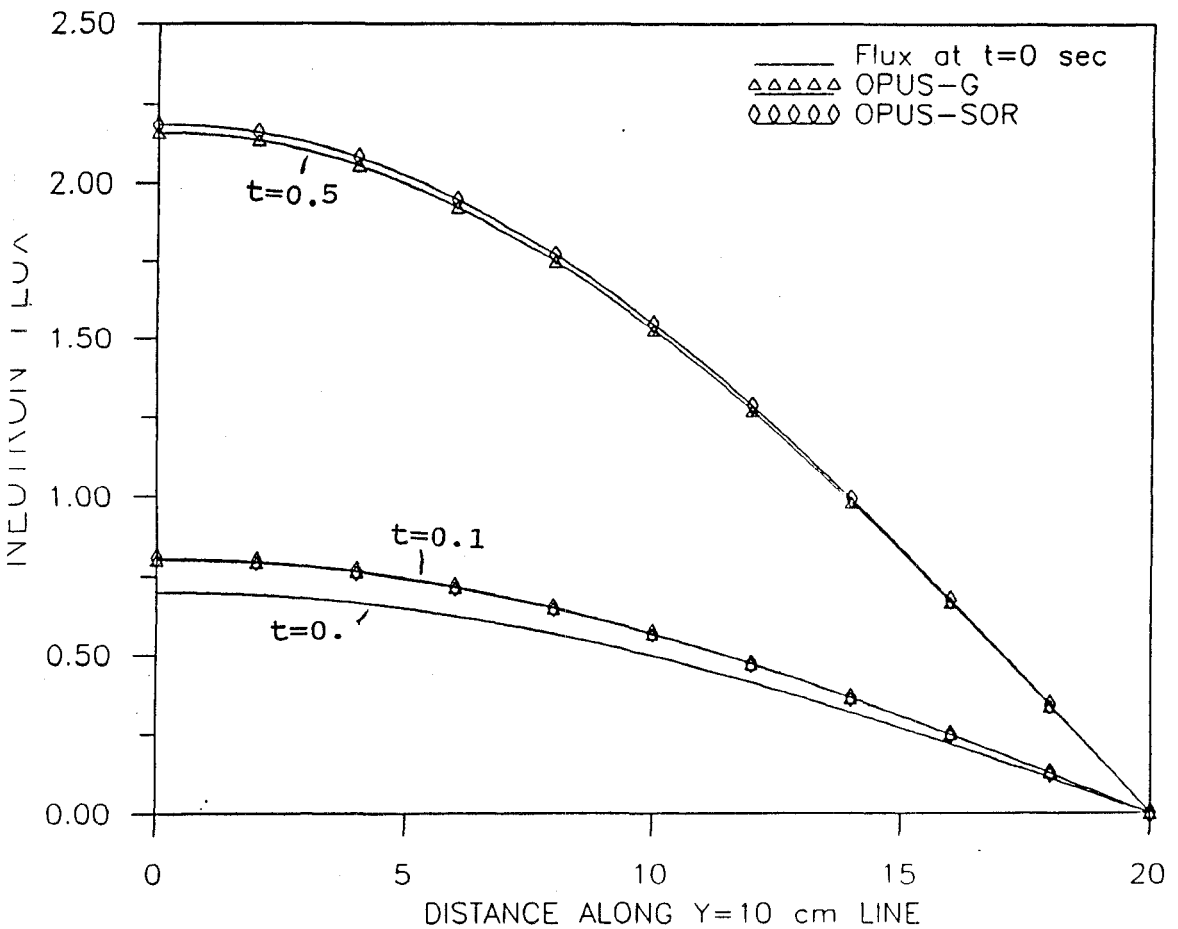

Figure (C.11) Time Dependent Problem **4.1**

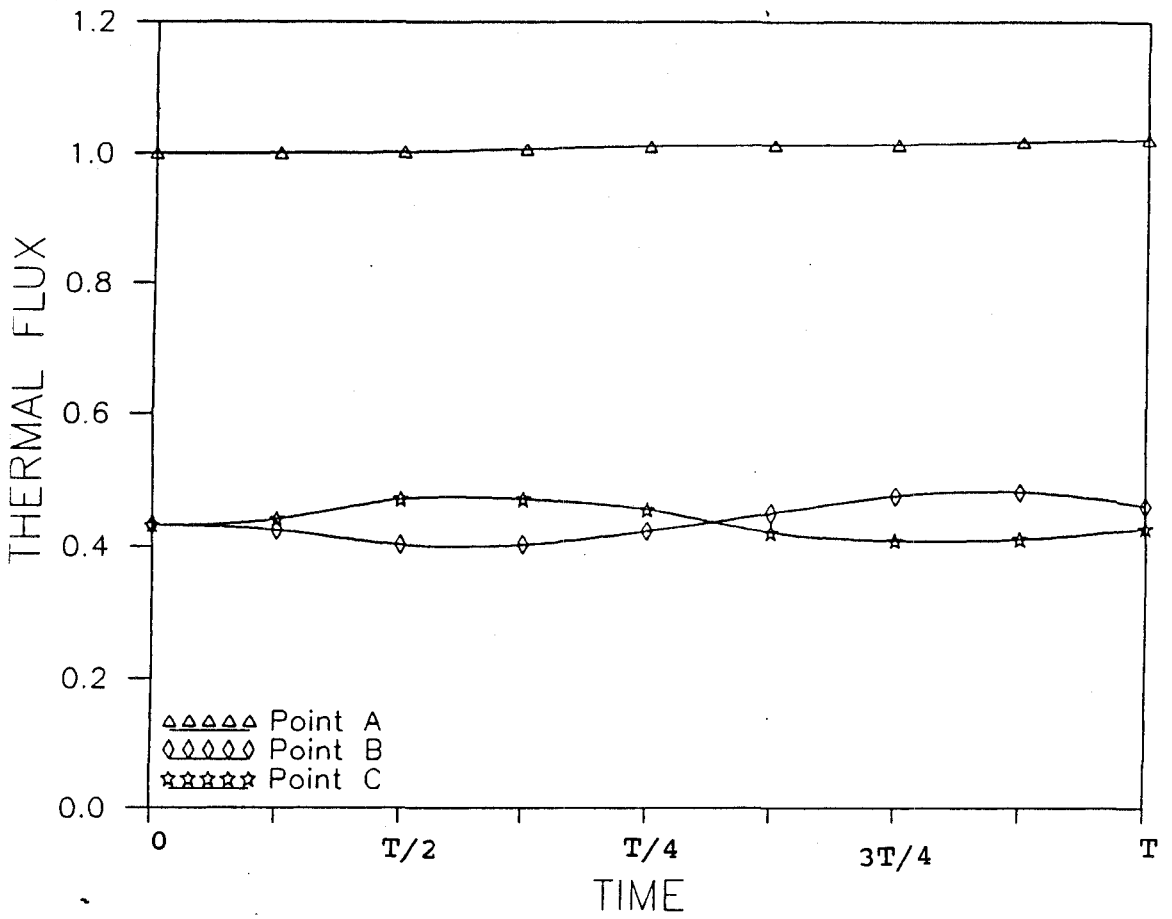

Figure (C.12) Time Dependent Problem **4.3**
Variation of Thermal Flux at Points A, B, C.

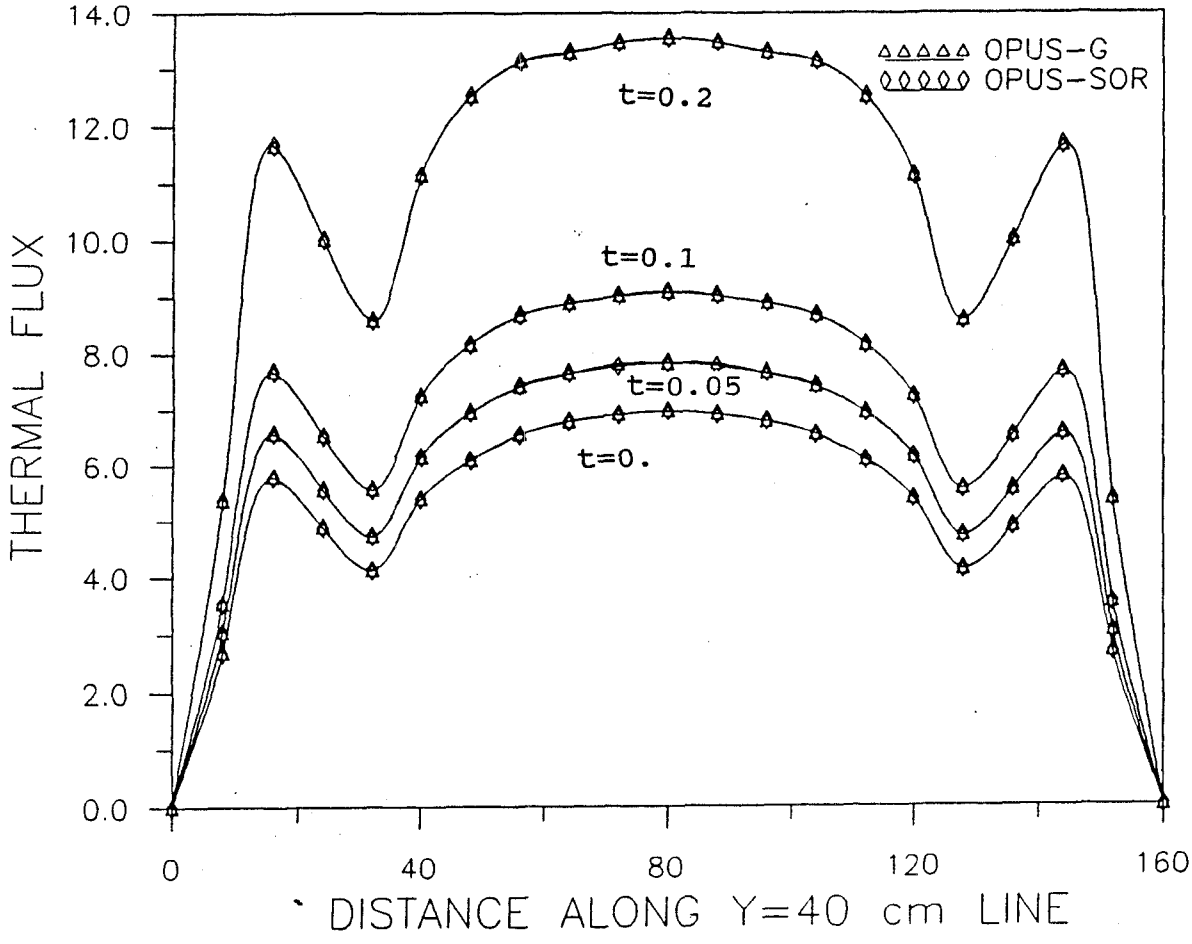

Figure (C.13) Time Dependent Problem **4.4**

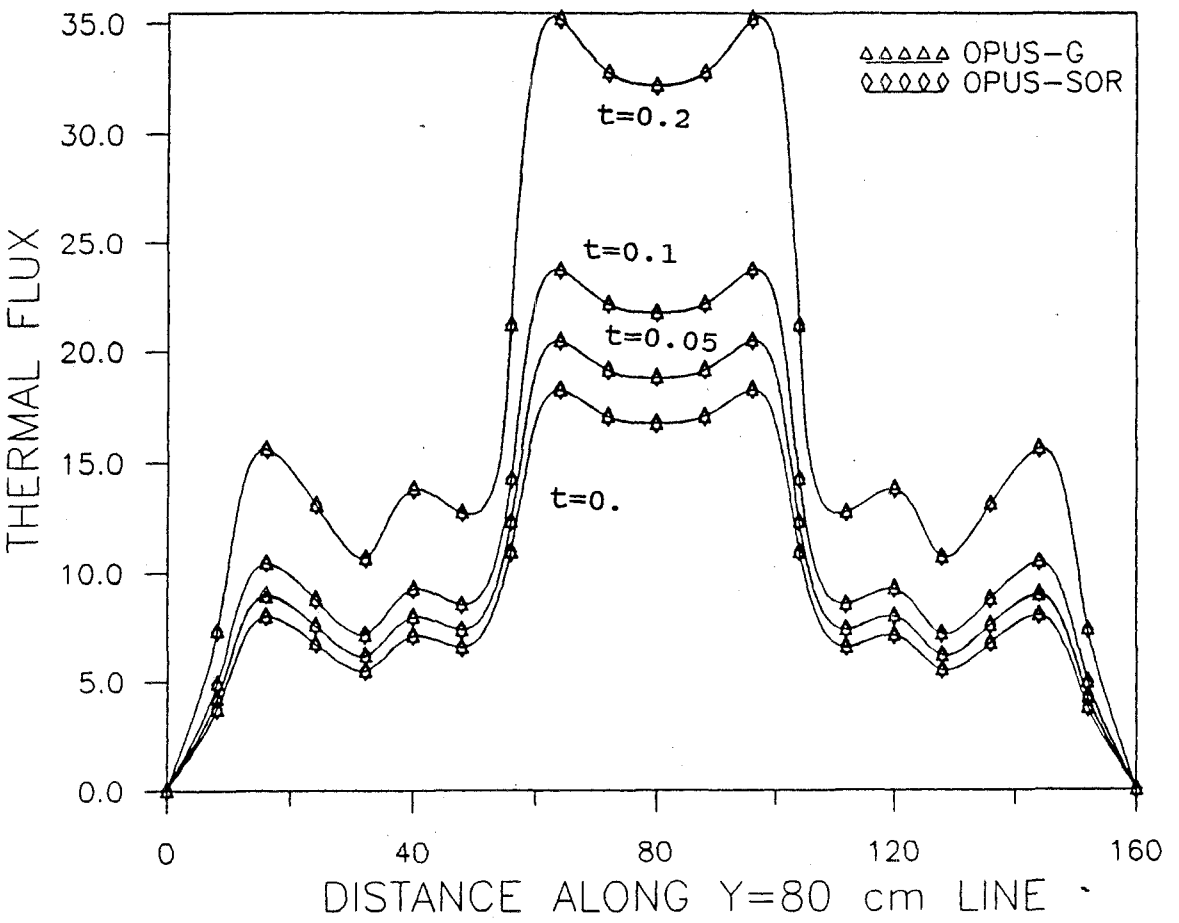

Figure (C.14) Time Dependent Problem **4.4**