PHYSOR 2004 -The Physics of Fuel Cycles and Advanced Nuclear Systems: Global Developments
Chicago, Illinois, April 25-29, 2004, on CD-ROM, American Nuclear Society, Lagrange Park, IL. (2004)

Development of an Object Oriented Nodal Code using the Refined AFEN Derived from the Method of Component Decomposition

Jae Man Noh, Jae Woon Yoo, and Hyung Kook Joo

Korea Atomic Energy Research Institute
150 Deokjin-dong, Yuseong-gu, Daejeon, Korea 305-353
jmnoh@kaeri.re.kr

In this study, we invented a method of component decomposition to derive the systematic internodal coupled equations of the refined AFEN method and developed an object oriented nodal code to solve the derived coupled equations. The method of component decomposition decomposes the intranodal flux expansion of a nodal method into even and odd components in three dimensions to reduce the large coupled linear system equation into several small single equations. This method requires no additional technique to accelerate the iteration process to solve the internodal coupled equations, since the derived equations can automatically act as the coarse mesh rebalance equations. By utilizing the object oriented programming concepts such as abstraction, encapsulation, inheritance and polymorphism, dynamic memory allocation, and operator overloading, we developed an object oriented nodal code that can facilitate the input/output and the dynamic control of the memories, and can make the maintenance easy.

KEYWORDS: Nodal Method, AFEN, Component Decomposition, Object Oriented

1. Introduction

The object oriented nodal code NODOO (the Object Oriented NOdal) developed in this study uses the refined analytic function expansion nodal method (the refined AFEN)[1] to improve the accuracy. The refinement in this method is performed by adding basis functions to the original flux expansion. The added functions are the products of the analytic functions in a specific direction and the possible combinations of the linear functions in the transverse directions to the direction of the analytic function (e.g., y $\sin(\kappa x)$, z $\sin(\kappa x)$, and yz $\sin(\kappa x)$). They also satisfy the diffusion equation at any point in the node. The additional constraints required by the added terms are the continuity conditions of the flux moments and the current moments at the interfaces. A step function is chosen in this study as the weighting function required in this method, since its physical meaning of a node subdivision is clear. The edge nodal fluxes that are nodal unknowns in the original AFEN method are excluded in this study, because we can obtain a sufficient accuracy by adding the weighted surface flux moments as unknowns. The continued factoring method[2] is adopted to remove the numerical singularity which may be involved in the nodal methods by using the analytic function expansions.

In this study, a method of component decomposition (MCD) is invented to reduce the complexity in deriving the nodal coupling equations in a sophisticated nodal method. This method decomposes the three dimensional intranodal flux expansion into eight components according to its even/odd properties in the x, y, and z directions. This allows us to reduce a large coupled linear system equation into several small independent single equations and

reduce the computational effort for solving them on a computer. The MCD does not require an additional acceleration technique, since the derived component-wise neutron balance equations can act as the coarse mesh rebalance equations.

In contrast to the conventional sequential nodal core analysis codes, the nodal code NODOO is developed by fully utilizing the object oriented programming concepts such as abstraction, encapsulation, inheritance and polymorphism, and operator overloading. NODOO consists of four independent modules of the input/output, energy group structure handling, cross-section treatment, and core geometry handling modules according to their functions in a nodal code. The minimized interfaces couplings between the modules are treated not by direct data transfer but by interface functions. For example, the loops repeating over the neutron energy groups appear only in the energy group treatment module. The input module can read data imbedded sequentially in the prose style text like a newspaper, so that the user may add some comments to the data. The output remains in the conventional style to which the users are accustomed. All the memories in the program are allocated dynamically. The variables for the physical quantities are not defined repeatedly as occasional demands, but the access routes to them are diversified. While similar equations are scattered widely with many 'if' statements in the conventional codes, a single equation appears only once in the NODOO for simple maintenance. The core structure is built in a similar manner to that of the actual core construction and folding, unfolding, duplicating, and projecting the core one- or two- dimensionally can be easily done in the program.

2. The Method of Component Decomposition for The Refined AFEN

2.1 Intanodal Flux Expansion

The method of component Decomposition can be applied not only to the refined AFEN but also to other nodal methods to get simpler internodal coupling equations.

Let us start from the multigroup neutron diffusion equation in a node shown in Fig. 1.

$$-\nabla^2\hat{\phi}(\mathbf{r})+\Lambda\hat{\phi}(\mathbf{r})=0 \tag{1}$$

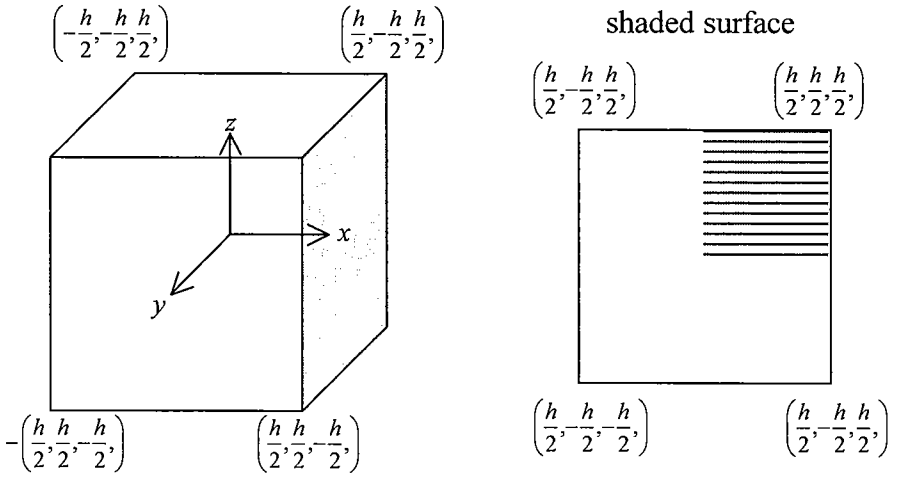

Fig. 1. A three dimensional node

The procedure described here can also be applied to the case that there is a fixed source

distribution or a transient source distribution in the right hand side of Eq. (1).

In spite of the fact that the refined AFEN can use a maximum of 72 basis functions, NODOO uses only 24 basis functions for the intranodal flux expansion.

$$\begin{aligned}\hat{\phi}(x,y,z) = &\cosh\left(\sqrt{\Lambda}x\right)\left(\mathbf{A}^x_{000}+2\frac{y}{h_y}\mathbf{A}^x_{010}+2\frac{z}{h_z}\mathbf{A}^x_{001}+4\frac{y}{h_y}\frac{z}{h_z}\mathbf{A}^x_{011}\right)+\sinh\left(\sqrt{\Lambda}x\right)\left(\mathbf{A}^x_{100}+2\frac{y}{h_y}\mathbf{A}^x_{110}+2\frac{z}{h_z}\mathbf{A}^x_{101}+4\frac{y}{h_y}\frac{z}{h_z}\mathbf{A}^x_{111}\right)\\ &+\cosh\left(\sqrt{\Lambda}y\right)\left(\mathbf{A}^y_{000}+2\frac{z}{h_z}\mathbf{A}^y_{001}+2\frac{x}{h_x}\mathbf{A}^y_{100}+4\frac{z}{h_z}\frac{x}{h_x}\mathbf{A}^y_{101}\right)+\sinh\left(\sqrt{\Lambda}y\right)\left(\mathbf{A}^y_{010}+2\frac{z}{h_z}\mathbf{A}^y_{011}+2\frac{x}{h_x}\mathbf{A}^y_{110}+4\frac{z}{h_z}\frac{x}{h_x}\mathbf{A}^y_{111}\right)\\ &+\cosh\left(\sqrt{\Lambda}z\right)\left(\mathbf{A}^z_{000}+2\frac{x}{h_x}\mathbf{A}^z_{100}+2\frac{y}{h_y}\mathbf{A}^z_{010}+4\frac{x}{h_x}\frac{y}{h_y}\mathbf{A}^z_{110}\right)+\sinh\left(\sqrt{\Lambda}z\right)\left(\mathbf{A}^z_{001}+2\frac{x}{h_x}\mathbf{A}^z_{101}+2\frac{y}{h_y}\mathbf{A}^z_{011}+4\frac{x}{h_x}\frac{y}{h_y}\mathbf{A}^z_{111}\right)\end{aligned} \tag{2}$$

The matrix functions appearing in this equation can be evaluated by the matrix function theory[3]. The superscript for an expansion coefficient means the direction for the hyperbolic function and the subscripts stand for the even/odd properties of the basis function, respectively in the x, y, and z directions. For example $\mathbf{A}^z_{100}$ means a coefficient of a basis function that is odd and even in the x and y directions, respectively and has an even hyperbolic function in the z direction.

The 24 coefficients in Eq. (2) are expressed in terms of 6 surface fluxes and 18 weighted surface flux moments. For example, the flux and the weighted flux moments on the right surface perpendicular to the x direction shown in Fig. 1 are defined by

$$\tilde{\hat{\phi}}^x_{1jk} = \frac{1}{h_y h_z}\int_{-\frac{h_y}{2}}^{\frac{h_y}{2}}\int_{-\frac{h_z}{2}}^{\frac{h_z}{2}} \mathrm{w_j}(y)\mathrm{w_k}(z)\,\hat{\phi}(\frac{h_x}{2},y,z)dzdy\,. \tag{3}$$

Here, 1 in the subscript 1jk means the right surface in the x direction (The left surface will have 0) and j and k mean that it is weighted by even or odd functions in the x and y directions, respectively. Because we are using a step function as a weighting function, $\mathrm{w_i}(u)$ becomes

$$\mathrm{w}_0(u)=1, \qquad \mathrm{w}_1(u)=\begin{cases}1 & \text{for } u\geq 0\\ -1 & \text{for } u<0\end{cases} \tag{4}$$

This equation becomes the surface flux when both j and k are zero, and becomes the weighted surface flux moments when either j or k is one. Since the surface flux can also be considered as a weighted surface flux weighted by a unit function, the surface flux and the weighted flux moments, hereafter, are called without distinction as the weighted surface fluxes.

The weighted surface currents corresponding to Eq. (3) are defined as

$$\tilde{\mathbf{J}}^x_{1jk} = \frac{\mathbf{D}}{h_y h_z}\int_{-\frac{h_y}{2}}^{\frac{h_y}{2}}\int_{-\frac{h_z}{2}}^{\frac{h_z}{2}} w_j(y)w_k(z)\frac{\partial}{\partial x}\hat{\phi}(x,y,z)dzdy\Big|_{x=\frac{h_x}{2}}\,. \tag{5}$$

If the weighted surface fluxes and currents are defined by the quantities weighted with a step function in the transverse directions as in Eqs. (3) and (5), these quantities can be expressed by a linear combination of the four average quantities averaged simply over four quadrants divided by the y and z axes. For example, the average flux of the first quadrant at the right surface in the x direction (the hatched quadrant in Fig. 1) is expressed in

$$\frac{1}{4}\left(\tilde{\hat{\phi}}^{x}_{100}+\tilde{\hat{\phi}}^{x}_{101}+\tilde{\hat{\phi}}^{x}_{110}+\tilde{\hat{\phi}}^{x}_{111}\right). \tag{6}$$

Therefore, the weighted surface quantities are mathematically equivalent to the quantities defined in the four quadrants on the surface.

The MCD decomposes the flux expansion into eight components according to their even/odd properties in the x, y, and z directions. For example the 100 component which is odd in the x direction and even in the y and z directions is given by

$$\begin{aligned}\hat{\phi}_{100}(x,y,z)=\frac{1}{8}\{&\hat{\phi}(x,y,z)-\hat{\phi}(-x,y,z)+\hat{\phi}(x,-y,z)+\hat{\phi}(x,y,-z)\\&-\hat{\phi}(-x,-y,z)+\hat{\phi}(x,-y,-z)-\hat{\phi}(-x,y,-z)-\hat{\phi}(-x,-y,-z)\}\\&=\sinh\left(\sqrt{\Lambda}x\right)\mathbf{A}^{x}_{100}+2\frac{x}{h_x}\cosh\left(\sqrt{\Lambda}y\right)\mathbf{A}^{y}_{100}+2\frac{x}{h_x}\cosh\left(\sqrt{\Lambda}z\right)\mathbf{A}^{z}_{100}\end{aligned} \tag{7}$$

This decomposition makes it possible to reduce the original 24×24 linear system finally to 24 independent equations and 8 individual auxiliary equations.

To determine the three coefficients in Eq. (7), we introduce the weighted average flux for the component as follows

$$\bar{\phi}_{ijk}=\frac{1}{h_x h_y h_z}\int_{-\frac{h_x}{2}}^{\frac{h_x}{2}}\int_{-\frac{h_y}{2}}^{\frac{h_y}{2}}\int_{-\frac{h_z}{2}}^{\frac{h_z}{2}} \mathrm{w_i}(x)\mathrm{w_j}(y)\mathrm{w_k}(z)\,\hat{\phi}(x,y,z)dzdydx\,, \tag{8}$$

Note that the conventional node average flux is the weighted average flux for component 000. The weighted average flux for component 100 which we chose as our practical example becomes

$$\begin{aligned}\bar{\phi}_{100}&=\frac{1}{h_x h_y h_z}\int_{-\frac{h_x}{2}}^{\frac{h_x}{2}}\int_{-\frac{h_y}{2}}^{\frac{h_y}{2}}\int_{-\frac{h_z}{2}}^{\frac{h_z}{2}} \mathrm{w}_1(x)\mathrm{w}_0(y)\mathrm{w}_0(z)\,\hat{\phi}(x,y,z)dzdydx\\&=\frac{4}{\sqrt{\Lambda}h_x}\sinh^2\left(\frac{\sqrt{\Lambda}h_x}{4}\right)\mathbf{A}^{x}_{100}+\frac{1}{2}\left\{\frac{2}{\sqrt{\Lambda}h_y}\sinh\left(\frac{\sqrt{\Lambda}h_y}{2}\right)\mathbf{A}^{y}_{100}+\frac{2}{\sqrt{\Lambda}h_z}\sinh\left(\frac{\sqrt{\Lambda}h_z}{2}\right)\mathbf{A}^{z}_{100}\right\}.\end{aligned} \tag{9}$$

By assuming the same arguments that we discussed for the weighted surface fluxes, we can easily understand that the 8 weighted component average fluxes are mathematically equivalent to the 8 octant average fluxes averaged over the 8 octants divided by the x, y, and z axes.

The weighted component- and direction-wise fluxes are defined from the definitions of the weighted surface fluxes.

$$\hat{\boldsymbol{\omega}}^{x}_{100}=\frac{1}{2}\left(\tilde{\hat{\phi}}^{x}_{100}-\tilde{\hat{\phi}}^{x}_{000}\right)-2\bar{\phi}_{100}=\left\{\sinh\left(\frac{\sqrt{\Lambda}h_x}{2}\right)-\frac{4}{\sqrt{\Lambda}h_x}2\sinh^2\left(\frac{\sqrt{\Lambda}h_x}{4}\right)\right\}\mathbf{A}^{x}_{100}\,, \tag{10}$$

$$\hat{\boldsymbol{\omega}}^{u}_{100}=\frac{1}{2}\left(\tilde{\hat{\phi}}^{u}_{110}+\tilde{\hat{\phi}}^{u}_{100}\right)-\bar{\phi}_{100}=\frac{1}{2}\left\{\cosh\left(\frac{\sqrt{\Lambda}h_u}{2}\right)-\frac{2}{\sqrt{\Lambda}h_u}\sinh\left(\frac{\sqrt{\Lambda}h_u}{2}\right)\right\}\mathbf{A}^{u}_{100}\ \text{ for u = y,z.} \tag{11}$$

Noting that each equation contains only one coefficient, we can see that the original 24×24

linear system is decomposed into 24 independent equations.

2.2 Internodal Coupled Equations

To obtain the equation that couples the weighted component- and direction-wise fluxes and currents, we define the weighted component- and direction-wise currents as follows,

$$\boldsymbol{\eta}^{x}_{100}=\frac{1}{2}\left(\tilde{\tilde{\mathbf{J}}}^{x}_{100}-\tilde{\tilde{\mathbf{J}}}^{x}_{000}\right)+\frac{4\mathbf{D}}{h_x}\bar{\boldsymbol{\phi}}_{100}=-\frac{2\mathbf{D}}{h_x}\left\{\frac{\sqrt{\boldsymbol{\Lambda}}h_x}{2}\cosh(\frac{\sqrt{\boldsymbol{\Lambda}}h_x}{2})-\frac{4}{\sqrt{\boldsymbol{\Lambda}}h_x}2\sinh^2(\frac{\sqrt{\boldsymbol{\Lambda}}h_x}{4})\right\}\mathbf{A}^{x}_{100},$$

$$\boldsymbol{\eta}^{u}_{100}=\frac{1}{2}\left(\tilde{\tilde{\mathbf{J}}}^{u}_{110}+\tilde{\tilde{\mathbf{J}}}^{u}_{100}\right)=-\frac{\mathbf{D}}{h_u}\frac{\sqrt{\boldsymbol{\Lambda}}h_u}{2}\sinh(\frac{\sqrt{\boldsymbol{\Lambda}}h_u}{2})\mathbf{A}^{u}_{100}\quad\text{for u = y,z.} \tag{12}$$

(13)

Eliminating the coefficients, we can obtain the equation which couples the weighted component- and direction-wise fluxes and currents as follows,

$$\boldsymbol{\eta}^{x}_{100}=\boldsymbol{\tau}_{x1}\,\hat{\boldsymbol{\omega}}^{x}_{100}, \tag{14}$$

$$\boldsymbol{\eta}^{u}_{100}=\boldsymbol{\tau}_{u0}\hat{\boldsymbol{\omega}}^{u}_{100}\quad\text{for u = } y,z. \tag{15}$$

Here, for u=x, y, and z

$$\boldsymbol{\tau}_{u0}=\frac{\mathbf{D}}{h_u}\frac{\boldsymbol{\Lambda}_u(\boldsymbol{\rho}_{u1}+1)}{2\boldsymbol{\rho}_{u1}}, \tag{16}$$

$$\boldsymbol{\tau}_{u1}=-\frac{2\mathbf{D}}{h_u}\left\{\frac{\boldsymbol{\rho}_{u1}}{(\boldsymbol{\rho}_{u1}+1)\boldsymbol{\rho}_{u2}}+1\right\}, \tag{17}$$

$$\boldsymbol{\rho}_{ui}=\frac{2i}{\sqrt{\boldsymbol{\Lambda}}h_u}\tanh(\frac{\sqrt{\boldsymbol{\Lambda}}h_u}{2i})\text{-}1\quad\text{for i}=1,2 \tag{18}$$

$$\boldsymbol{\Lambda}_u=\boldsymbol{\Lambda}h_u^2. \tag{19}$$

Note that he coefficients of the component- and direction-wise current equations do not have any dependencies on the transverse directions. The MCD reduces the complexity and the number of the coefficients required in a nodal method.

All the nodal coupled equations have not been obtained yet. The weighted component-wise average flux introduced in the early stage of the MCD requires an additional equation to be solved. This equation can be obtained by integrating the diffusion equation (1) with the corresponding weighting function over the node volume.

$$\frac{\mathbf{D}}{h_xh_yh_z}\int_{-\frac{h_x}{2}}^{\frac{h_x}{2}}\int_{-\frac{h_y}{2}}^{\frac{h_y}{2}}\int_{-\frac{h_z}{2}}^{\frac{h_z}{2}}w_1(x)w_0(y)w_0(z)\left\{-\nabla^2\hat{\boldsymbol{\phi}}(x,y,z)+\boldsymbol{\Lambda}\hat{\boldsymbol{\phi}}(x,y,z)\right\}dzdydx$$

$$=\frac{\boldsymbol{\tau}_{x2}}{h_x}\hat{\boldsymbol{\omega}}^{x}_{100}+\frac{\boldsymbol{\tau}_{y0}}{h_y}\hat{\boldsymbol{\omega}}^{y}_{100}+\frac{\boldsymbol{\tau}_{z0}}{h_z}\hat{\boldsymbol{\omega}}^{z}_{100}\text{-}\boldsymbol{\Lambda}\bar{\boldsymbol{\phi}}_{100}=\mathbf{0} \tag{20}$$

where,

$$\boldsymbol{\tau}_{u2}=\frac{\mathbf{D}}{h_u}\frac{\boldsymbol{\Lambda}_u(\boldsymbol{\rho}_{u2}+1)}{2\boldsymbol{\rho}_{u2}}. \tag{21}$$

Again, we can see that the 8 component-wise average flux equations are mathematically

equivalent to the octant neutron balance equations for the 8 octants divided by the x, y, and z axes. Therefore, it can be finally concluded that the constraints we applied to constrain the intrannodal flux expansion are the quadrant flux and current continuity conditions for all the quadrants of the six surfaces and the octant neutron balance conditions for the eight octants of the node.

Since NODOO uses the response matrix method where the surface partial currents are unknowns, the component- and direction-wise fluxes and currents in the Eqs. (14), (15), and (20) are transformed into the incoming and outgoing partial currents that are defined as follows.

$$\tilde{\mathbf{j}}^{x}_{jk+} = \frac{1}{h_y h_z}\int_{-\frac{h_y}{2}}^{\frac{h_y}{2}}\int_{-\frac{h_z}{2}}^{\frac{h_z}{2}} w_j(y)w_k(z)\left\{+\frac{\mathbf{D}}{2}\frac{\partial}{\partial x}\hat{\phi}(x,y,z)+\frac{1}{4}\hat{\phi}(x,y,z)\right\}dzdy\Bigg|_{x=\frac{h_x}{2}} = +\frac{\tilde{\mathbf{J}}^{x}_{1jk}}{2}+\frac{\tilde{\hat{\phi}}^{x}_{1jk}}{4} \tag{22}$$

$$\tilde{\mathbf{j}}^{x}_{jk-} = \frac{1}{h_y h_z}\int_{-\frac{h_y}{2}}^{\frac{h_y}{2}}\int_{-\frac{h_z}{2}}^{\frac{h_z}{2}} w_j(y)w_k(z)\left\{-\frac{\mathbf{D}}{2}\frac{\partial}{\partial x}\hat{\phi}(x,y,z)+\frac{1}{4}\hat{\phi}(x,y,z)\right\}dzdy\Bigg|_{x=\frac{h_x}{2}} = -\frac{\tilde{\mathbf{J}}^{x}_{1jk}}{2}+\frac{\tilde{\hat{\phi}}^{x}_{1jk}}{4} \tag{23}$$

Here, the subscripts + and – mean the incoming and outgoing directions, respectively. The component- and direction-wise partial currents are defined in a consistent way to the cases of the component- and direction-wise fluxes and currents.

$$\tilde{\xi}^{x}_{100\pm} = \frac{\tilde{\mathbf{j}}^{x}_{100\pm} - \tilde{\mathbf{j}}^{x}_{000\pm}}{2} - \left(\frac{1}{2}\mp 2\frac{\mathbf{D}}{h_x}\right)\bar{\phi}_{100} \tag{24}$$

$$\tilde{\xi}^{u}_{100\pm} = \frac{\tilde{\mathbf{j}}^{u}_{10\pm} + \tilde{\mathbf{j}}^{u}_{00\pm}}{2} - \frac{\bar{\phi}_{100}}{4} \quad \text{for u = y, z} \tag{25}$$

Then, the relationships between the component- and direction-wise fluxes, currents and partial currents become

$$\hat{\omega}^{u}_{100} = 2\left(\tilde{\xi}^{u}_{100+} + \tilde{\xi}^{u}_{100-}\right) \quad \text{for u = x, y, z} \tag{26}$$

$$\eta^{u}_{100} = \tilde{\xi}^{u}_{100+} - \tilde{\xi}^{u}_{100-} \quad \text{for u = x, y, z} \tag{27}$$

Substituting these two equations into Eqs. (14) and (15), we can get the equations for the weighted component- and direction-wise outgoing partial currents.

$$\tilde{\xi}^{x}_{100-} = -\tilde{\xi}^{x}_{100+} + \varsigma_{x1}\tilde{\xi}^{x}_{100+} \tag{28}$$

$$\tilde{\xi}^{u}_{100-} = -\tilde{\xi}^{u}_{100+} + \varsigma_{u0}\tilde{\xi}^{u}_{100+} \quad \text{for u = y, z} \tag{29}$$

Here,

$$\varsigma_{ui} = 2\left(1-2\tau_{ui}\right)^{-1} \quad \text{u = x, y, z, i = 0, 1} \tag{30}$$

Substituting Eqs. (26), (28), and (29) into Eq. (20), finally we can get the component-wise neutron balance equation that is expressed by only the component- and direction-wise incoming currents.

$$\frac{2\varsigma_{x2}}{h_x}\tilde{\xi}^{x}_{100+} + \frac{2\left(\varsigma_{y0}-2\right)}{h_y}\tilde{\xi}^{x}_{100+} + \frac{2\left(\varsigma_{z0}-2\right)}{h_z}\tilde{\xi}^{x}_{100+} - \Lambda\bar{\phi}_{100} = 0, \tag{31}$$

where,

$$\varsigma_{x2} = \frac{\tau_{x2}\varsigma_{x1}}{h_x} \tag{32}$$

By comparing the coupled equations derived here with those in the reference [1], we can expect time savings due to their simpler forms when we program them in a nodal code and solve them on a computer.

Deriving the nodal coupling equations by the method of component decomposition is not only very simple but also very systematic component by component procedure. We used only the 100 component here but we showed equivalently the derivations for the other components. The equations for the other components can be simply obtained by changing the indices on the unknowns and coefficients in the equation for the 100 component according the even/odd properties of those components. They can be handled by a single loop in a program which sweeps over all the eight components and three directions.

2.3 Numerical Singularity Removal and Acceleration

The refined AFEN may also have the numerical singularity like the other analytic nodal methods. The continued factoring method in references [1,4,5] is adopted to remove this numerical singularity.

The advantage in using the MCD is again well borne out by the acceleration. As explained before, the component-wise neutron balance equation for the 000 component means the whole neutron balance equation over the node volume. Therefore, solving the equations for this component has the result of solving the coarse mesh rebalance equations over all the 32 unknowns in the node. For the same reason, solving the equations for the components 100, 010, and 001 is equivalent to solving the coarse mesh rebalance equations for the half nodes divided by the x, y, or z axis, respectively. If we solve unevenly the equations according to their importance (The degree of oddness may be a measure of importance.) we can achieve an acceleration as a natural result.

The following energy group condensing acceleration may be added to this natural acceleration to maximize the acceleration effect.

$$\bar{\xi}^x_{100-} = -\bar{\xi}^x_{100+} + \varsigma_{x100}\bar{\xi}^x_{100+} \tag{33}$$

where,

$$\varsigma_{x100} = \frac{\sum_{g\in G}\varsigma_{x1g}\tilde{\xi}^x_{100g+}}{\bar{\xi}^x_{100+}} \tag{34}$$

$$\bar{\xi}^x_{100+} = \sum_{g\in G}\tilde{\xi}^x_{100g+} \tag{35}$$

Note that the condensed equation has the same form as that of the original equation.

3. Development of Object Oriented Nodal Code

In this study, the nodal code NODOO is developed which is not a sequential program but an object oriented program. The objects to achieve in developing this code are 1) excellent accuracy, 2) simplicity in maintenance, 3) user convenience, 4) easy extension, and 5)

efficient use of resources. The requirements to be kept in developing it are 1) strict modulization according to the function and structure, 2) dynamic memory allocation, 3) prevention of double definitions for the physical quantities, 4) feasibility of the core fold, unfold, and lower dimensional projections, 5) simulation of the real core construction, 6) free input style not bounded to the formats, and 7) full utilization of the object oriented programming concepts such as abstraction, encapsulation, inheritance and polymorphism, and operator overloading.

NODOO consists of four independent modules of the input/output, energy group structure handling, cross-section treatment, and core geometry handling modules according to their functions in a nodal core. The minimized interfaces couplings between the modules are treated not by direct data transfer but by interface functions. In the future, the capabilities to handle the depletion, the control rod movement, the fuel reloading, and the operation support activities will be added.

3.1 Input/Output Module

The input/output module consists of the 'mifstream' and 'mofstream' classes that are derived publicly from the original 'ifstream' and 'ofstream' classes in C++. Therefore, they can do what the 'ifstream' and 'ofstream' classes can do. The 'mifstream' can read a text line, an integer, or a real variable from an input file. The blank, comma, slash, and tab may be used to discriminate between the variables. Since it can skip the comment lines during the reading of the numbers, it can read data imbedded sequentially in the prose style text like a newspaper. Therefore, the user may add some comments explaining the data in his input file. No errors occur during the skipping of the comments, because all the error bits in 'rdstate' remain good during skipping. The multiple assignment of a single value expressed as '16 * 3.14' can also be handled. A group of input data divided by a digit appended to a string 'card' is recognized as a card input. If 'mifstream' meets, e.g., 'card2' during the reading of the data, it recognizes that the current input card ended and card2 starts. It sets the 'eocbit' to 1 just in the same way that the original 'ifstream' sets the 'eofbit' to 1 at the end of a file. Therefore, the 'eoc' condition may be handled in the same way as handling the 'eof' condition. Currently, there are three input cards defined in NODOO composed of a title card, a cross-section dada card, and a core geometry data card. If it has read a complete set of an input card and there remain lines unread, it skips the remaining lines until it meets the next card. Using this function, the user can attach their trial input to the normal input without any problems. The 'mifstream' automatically echoes each data it reads successfully on an echo file that is a private member of the 'mifstream' class. With this function, the user may convert their long input with comments into a compact input or may check whether the program has read their input successfully as they intended.

The output remains in the conventional style as much possible, to which the users are accustomed. Since most of the important classes define their own << and >> operators, the user can easily read and print the information of those classes. For example, we can print the information of an object 'a' of the 'assembly' class using a simple statement 'cout << a'.

3.2 Energy Group Structure Treatment Module

This module handles all the tasks involved in the vectors, matrices, and multidimensional arrays with given sizes. The loops sweeping over the energy groups appear only in this module not in the other modules. All the classes in this module are defined internally as one-dimensional arrays which were dynamically allocated, but they can be accessed in the

maximum four multidimensional form like a[i][j][k][l] in the other modules. Since all the operations involved in the vectors and matrices are overloaded in this module, it can perform the operation such as 'a += 1.0 + b * c' regardless of the types of 'a', 'b', and 'c'. It can calculate the eigen-systems of a matrix and handle the operations involved in a matrix with complex conjugates without any additional complexities, storing the real parts and imaginary parts in different columns of the matrix. Therefore, it can solve the multigroup AFEN equations where the complex eigen-values are involved.

The other task this module completes is the dynamic memory allocation for the other modules. For example, the 'assembly' class in the core geometry handling module is defined a class derived publicly from the 'vector' class of the 'rod' class. This can be done because the classes defined in this module are all template classes that may be applied to any type of class. Using this capability, we can allocate any size of multidimensional array of any type. The operations originally defined for the vectors and matrices can also be applied to the derived arrays. Therefore, we can do any of four arithmetic operations on the rods in an assembly or on the assemblies in a core. This enables the fold, unfold, or lower dimensional projection of a core to be done very simply.

The classes in this module will become the base classes for depletion in the future. If we define depletion as the operations for the vectors and matrices with the order of the number of the nuclides, we can utilize the full capabilities of this module.

The vectors or matrices defined in this module have their own << and >> operators to facilitate easy reading and printing.

3.3 Cross-section Treatment Module

The classes related to the cross-sections are defined in this module. The 'cross-section' class has private members of a diffusion constant and the other homogenization parameters that are defined as objects of the vectors or matrices in the energy groups. The cross-section 'table' class is a class derived publicly from the 'vector' class of the 'cross-section' class defined in the energy group treatment structure. Therefore, the arithmetic operations on the objects of the 'cross-section' class are possible. Using this function, the coefficients of the refined AFEN are calculated in this module. The 'cross-section' class and the 'table' class also have their own << and >> operators.

3.4 Core Geometry Handling Module

The 'depletion point' class having private members of a 'nuclide' class and a 'flux' class gives the basis of this module. Developing the 'core' class starting from the 'depletion point' class is just the simulation of the real core construction. As mentioned before, the 'rod' class is publicly inherited from the 'vector' class of the 'depletion point' class along the axis. Then, the 'assembly' class is derived from the 'vector' class of the 'rod' class. In the refined AFEN, an assembly is composed of a few rods. Finally, the 'core' class will be made of the assembly class. Most classes in this module are filled with their ancestor's pointers when they are created. As soon as the user specifies the core geometry, they allocate dynamically their memories according to the specified core geometry.

By setting the number of axial layers to 1 or the number of assemblies to 1, we can calculate the axial one dimensional or the radial two dimensional problems, respectively. Keeping the same source list for the lower dimensional calculations facilitates further an easy maintenance. It can handle the varieties of core geometries such as the octant, quadrant, and

half cores and handle the reflective and the rotational symmetries for all the geometry types. It can also handle the core fragments divided along the 45 degree and/or 135 degree lines. Changing from a geometry type to another is embodied with a few lines of cording. By not allocating unnecessary intermediate memories, it prevents the memory from overflow during the changing of the geometries.

This module projects easily the core onto a rod object that is a private member of the core class, utilizing the operations defined in the energy group structure handling module. In the same way, it also projects two-dimensionally the core onto the 'depletion point' objects each of which belongs privately to the 'rod' class. Since the projected cores can act just as independent cores, we can do the one dimensional or two dimensional nodal calculations with using them.

The 'node' class is defined by a special access method to a part of the 'core' class. The refined AFEN equations derived by the method of component decomposition are solved in this class. While it accesses the incoming partial currents from its adjacent nodes, the albedos specified by the user are taken into account in the core boundaries. For simplicity in maintenance, the equations and boundary conditions are reiterated nowhere else but appear only once in this class. While the energy group structure handling module will give the basis for depletion, the actual depletion will take place in this class in the future. Considering the intranodally variant cross-sections depleted at all the depletion points, we will achieve a huge axial node size of the node without losing the accuracy.

4. Calculational Results and Conclusions

To test the NODOO code developed here, a test core extended three dimensionally from the EPRI-9[6] benchmark problem is explored. The active core embedded by the 21cm long bottom and top reflectors is 360cm high. Taking a node size of $21\times21\times30$ cm^3 and the convergence criteria for the errors of both the effective multiplication factor and the flux distribution by 10^{-6}, it requires about a minute for a single step calculation on a Pentium class personal computer. Note that we did not apply the energy group condensing acceleration in this calculation. Even though we have not compared this computing time with those of the other codes, we realize that NODOO is much slower than others. Considering our concern in developing the object oriented nodal code was not the computing time, this result is not a disappointing one but a very much expected one. If we apply the group condensing acceleration and the various graphic capabilities C++ has, it can be used in the actual reactor code design. Owing to its easy extension capability, the features to handle the depletion, the control rod movement, the fuel reloading, and the operation support activities will be added in the future.

The method of component decomposition developed in this study may be applied to other nodal methods to obtain simple internodal coupling equations and to achieve easy acceleration.

Acknowledgements

This study has been carried out under the Long-Term Nuclear R&D Program supported by the Ministry of Science and Technology (MOST) of Korea.

References

1) S. W. Woo, N. Z. Cho, and J. M. Noh, "The Analytic Function Expansion Nodal Method Refined with Transverse Gradient Basis Functions and Interface Flux Moments," Nucl. Sci. Eng., 139, 156 (2001)
2) K. S. SMITH, "An Analytic Nodal Method for Solving the Two-Group Multidimensional, Static and Transient Neutron Diffusion Equation," Nuclear Engineering Thesis, Massachusetts Institute of Technology (1979).
3) J. M. NOH, et al., "A General Approach to Multigroup Extension of the Analytic Function Expansion Nodal Method Based on Matrix Function Theory," Proc. 1996 Joint Intl. Conf. Mathematical Methods and Super Computing for Nuclear Applications, Saratoga Springs, New York, Octorber 6-10, 1997, Vol. 1, p. 144, American Nuclear Society (1997).
4) N. Z. Cho, C. J. Park, and J. M. Noh, "Removal of Numerical Singularity in the Transverse-Integrated Analytic Nodal Method via Continued Factoring," Trans. Am. Nucl. Soc., 86, 361 (2002). .
5) S. W. Woo, N. Z. Cho, and J. M. Noh, "Removal of Numerical Singularity in Analytical Nodal Methods via Continued Factoring," Trans. Am. Nucl. Soc., 84, 205 (2001).
6) H. S. Khalil, "The Applications of Nodal Methods to PWR Analysis," PhD Thesis, Massachusetts Institute of Technology (1983).