12-1996

Improvements in a Hybrid Stochastic/Deterministic Method for Transient Three-Dimensional Neutron Transport

Charles L. Bentley
University of Tennessee - Knoxville

To the Graduate Council:

I am submitting herewith a dissertation written by Charles L. Bentley entitled "Improvements in a Hybrid Stochastic/Deterministic Method for Transient Three-Dimensional Neutron Transport." I have examined the final electronic copy of this dissertation for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Doctor of Philosophy, with a major in Nuclear Engineering.

H. L. Dodds, Major Professor

We have read this dissertation and recommend its acceptance:

L. F. Miller, R. E. Pevey, M. Pace

Accepted for the Council:

Carolyn R. Hodges

Vice Provost and Dean of the Graduate School

(Original signatures are on file with official student records.)

To the Graduate Council:

I am submitting herewith a dissertation written by Charles L. Bentley entitled "Improvements in a Hybrid Stochastic/Deterministic Method for Transient Three-Dimensional Neutron Transport." I have examined the final copy of this dissertation for form and content and recommend that it be accepted in partial fulfillment of the requirement for the degree of Doctor of Philosophy, with a major in Nuclear Engineering.

H.L. Dodds, Major Professor

We have read this dissertation
and recommended its acceptance:

Accepted for the Council:

Associate Vice Chancellor
and Dean of the Graduate School

Improvements in a Hybrid Stochastic/Deterministic Method for Transient, Three-Dimensional Neutron Transport

A Dissertation

Presented for the

Doctor of Philosophy

Degree

The University of Tennessee, Knoxville

Charles L. Bentley

December 1996

Dedication

This dissertation is dedicated to my parents, Jane Ellen Bentley and the late William Dean Bentley, and to my wife, Lisa Lynn Bentley.

Acknowledgements

I would like to thank my advisor, Dr. H.L. Dodds, for his encouragement and guidance. This work could not have been completed without his supervision. I would also like to thank my committee members, Dr. L.F. Miller, Dr. R.E. Pevey and Dr. M. Pace, for their advice and assistance. I also wish to thank the members, past and present, of the Nuclear Criticality Safety Group at the University of Tennessee, Knoxville; particularly, B. Basoglu, R. Demeglio, M. Dunn, S. Goluoglu, K. Norton, L. Paschal, W. Waddell and T. Yamamoto. I also wish to recognize the U.S. Department of Energy and Oak Ridge Institute for Science and Education for the financial support of this work via the Nuclear Engineering/Health Physics Fellowship Program. Finally, I would like to express thanks to the members of my family, Lisa Bentley, Jane Bentley, Jenny Griffith, Danny Griffith and Doris Reynolds.

Abstract

This research develops an improved methodology (and corresponding code) for solving the time-dependent, three-dimensional Boltzmann Transport Equation with explicit representation of delayed neutrons. These improvements are incorporated in a modified version of the code TDKENO, entitled TDKENO-M. Specifically, these improvements are:

1. Incorporate the improved quasistatic methodology into an existing quasistatic framework. Specifically, include the flux shape derivative in the fixed source term instead of being neglected. Also, compute the point kinetics parameters deterministically by their inner product definitions.

2. Incorporate a hierarchy of three different integration time intervals for the numerical solution of the coupled set of ordinary differential equations. The shape function is assumed to vary linearly over the largest time interval. The second largest time interval is used for determining the point kinetics parameters. Finally, the smallest time step is used for solving the point kinetics equations.

3. Apply TDKENO-M to benchmark problems to determine the accuracy of the method. Particularly, TDKENO-M is applied to one- and three-dimensional benchmark problems to evaluate its neutronic capabilities.

4. Combine input requirements into a single input file so that TDKENO-M is less cumbersome to execute.

5. Develop the ability to restart a calculation at an intermediate problem time.

6. Develop a "user-friendly" manual for using TDKENO-M which describes in detail the input requirements as well as the output files, subroutines, modules, and the calculational flow.

Results show that TDKENO-M is quite accurate in comparison with benchmark calculations (less than 2% error in power trace for ANL Benchmark 16-A1) with as much as a factor of 500 reduction in CPU time relative to TDKENO.

Table of Contents

List of Figures

List of Tables

Chapter One

Introduction

1.1 Background

The transient simulation of neutronic systems is of extreme importance for new reactor system design, emergency response planning for inadvertent criticality excursions, and also for the safety analysis of existing systems. The transient neutronic behavior of a system can be modeled using methods ranging in complexity from simple point kinetics algorithms to three-dimensional nodal methods using space-time kinetics.

Point kinetics[1] algorithms were initially used to perform transient reactor calculations. This method ignores the spatial distribution of the neutron flux during the transient. Point kinetics algorithms are usually adequate for modeling small reactors (i.e., a few mean free paths thick) in which the spatial distribution of flux is relatively time-independent. However, the validity of using point kinetics for large reactor systems with loosely coupled regions is questionable and, therefore, the spatial dependence of the flux should be modeled.[1]

To incorporate the spatial dynamics of a reactor system, two methods exist, transport theory and diffusion theory. Diffusion theory, in which the angular dependence is assumed to be linear, is an approximation to transport theory.[2] The time-dependent, three-dimensional Boltzmann transport equation with explicit representation of delayed

neutrons is as follows:

$$\frac{1}{v}\frac{\partial\phi(\bar{r},\bar{\Omega},E,t)}{\partial t} + \bar{\Omega}\cdot\bar{\nabla}\phi + \Sigma_t\phi =$$

$$\iint\Sigma_s(\bar{r};\bar{\Omega}',E'\rightarrow\bar{\Omega},E;t)\,\phi(\bar{r},\bar{\Omega}',E',t)\,d\Omega'dE'$$

$$+ \iint\chi_p(E)\,(1-\beta)\,\nu\Sigma_f(\bar{r},E',t)\,\phi(\bar{r},\bar{\Omega}',E',t)\,d\Omega'dE' \qquad (1)$$

$$+ \sum_j \lambda_j C_j(\bar{r},t)\,\chi_j + Q$$

and

$$\frac{\partial C_j(\bar{r},t)}{\partial t} + \lambda_j C_j = \iint\beta_j\nu\Sigma_f(\bar{r},E',t)\,\phi(\bar{r},\bar{\Omega}',E',t)\,d\Omega'dE' \quad j=1,N \qquad (2)$$

where,

	v	=	particle speed,
	ϕ	=	time-dependent angular flux,
	$\bar{r}$	=	spatial variable,
	E	=	energy,
	$\bar{\Omega}$	=	unit vector describing particle direction,
	t	=	time,
	Σ_t	=	total cross section,
	Σ_s	=	differential scattering cross section,
	Q	=	external source,
	χ_p	=	normalized energy spectrum of prompt neutrons,
	χ_j	=	normalized energy spectrum for precursor group j,
	β	=	delayed neutron fraction,
	ν	=	total number of neutrons emitted per fission,
	Σ_f	=	fission cross section,
	λ_j	=	decay constant for delayed neutron precursor group j,
	C_j	=	density of delayed neutron precursors for group j,
	N	=	number of precursor groups.

Also, the time-dependent three-dimensional diffusion equation with explicit representation of delayed neutrons is given by:

$$\frac{1}{v}\frac{\partial}{\partial t}\phi(\bar{r},E,t) + \Sigma_t\phi = \bar{\nabla}\cdot D(\bar{r},E,t)\bar{\nabla}\phi(\bar{r},E,t)$$
$$+ \int\Sigma_s(\bar{r};E'\rightarrow E;t)\,\phi(\bar{r},E',t)\,dE'$$
$$+ \int\chi_p(E)\,(1-\beta)\,\nu\Sigma_f(\bar{r},E',t)\,\phi(\bar{r},E',t)\,dE' \qquad (3)$$
$$+ \sum_j \lambda_j C_j(\bar{r},t)\,\chi_j + Q$$

and

$$\frac{\partial C_j(\bar{r},t)}{\partial t} + \lambda_j C_j = \int\beta_j\nu\Sigma_f(\bar{r},E',t)\,\phi(\bar{r},E',t)\,dE' \qquad j=1,N \qquad (4)$$

1.2 Previous Work

Due to the complexity of the equations, neither the transport method or the diffusion method can be solved analytically for complicated geometries. However, analytical solutions can be determined for some simple geometries (e.g., homogeneous spheres, rectangular parallelpipeds, etc.) and therefore, numerical techniques are needed to solve the spatial and temporal dependent equations for realistic geometries. Techniques which address the spatial variable include modal, finite element, nodal, and

direct integration.[3] Methods which treat the time variable include direct integration and flux factorization methods. The direct methods use temporal approximation techniques such as fully implicit, fully explicit, Crank-Nicholson, and semi-implicit techniques. Flux factorization methods include exponential transformation, point kinetics, adiabatic, quasistatic and improved quasistatic methods.[4] Note that the adiabatic, quasistatic, and improved quasistatic methods are often generically referred to as the quasistatic method without denoting the intrinsic differences between the three methods.

The modal expansion technique[5,6,7,8] involves the expansion of the solution into specified functions of one or more of the independent variables by using combining coefficients. Thus, the number of independent variables for a problem is reduced. Codes which use modal expansion techniques to solve the time-dependent diffusion equations include one-dimensional RAUMZEIT[9], two-dimensional RADYVAR[10], and three-dimensional SMOKIN.[11]

The nodal method[12,13,14,15] divides the reactor into subregions or nodes. The flux in each node is determined using a single polynomial and the interaction between nodes is included by using coupling coefficients. Diffusion codes which use the nodal method include a one-dimensional code by Alcouffe and Albrecht[16] as well as three-dimensional codes NEM[17], CUBBOX[18], IQSBOX[19], CONQUEST[15], QUANDRY[20], and NIKE.[21]

The most straight forward method is the direct integration technique[22] in which either the transport equations or the diffusion equations are directly integrated. There are several methods for integrating the time variable which range in complexity from fully explicit, in which the flux is an explicit function of variables at the previous time step, to fully implicit, in which the flux depends on variables at the present time step and leads to iterative solutions. In general, direct numerical integration methods are quite time consuming when applied to three-dimensional multigroup problems. Codes which solve the time-dependent diffusion equations using finite difference methods include one-dimensional WIGLE[23], two-dimensional TWIGL[24] and DISCOTHEQUE[25], and three-dimensional 3DKIN[26] and MEKIN.[27] Further, three-dimensional CRONOS[28] uses finite elements. Codes which solve the time-dependent transport equations using direct numerical integration include one dimensional TDA[29] and TIMEX[30], and two-dimensional TRANZIT.[31] TDA is a time-dependent version of the one-dimensional, multigroup discrete ordinates transport code ANISN.[32] TIMEX is a one-dimensional, multigroup transport code which uses a finite element method to treat the spatial variable, and conventional discrete ordinates to treat the angular variable. TRANZIT is a multigroup, time-dependent, two-dimensional discrete ordinates code in cylindrical geometry which uses diamond difference approximation techniques.

Finally, the quasistatic method[33,34] treats the spatial dynamics of a calculation by factoring the flux into an amplitude and a shape function such that the time variation of the shape function is more slowly varying than the time variation of the amplitude

function.[35] This factorization of the flux results in splitting the neutronics equation into two coupled equations; an amplitude equation having the form of the point kinetics equation and a shape equation. The accuracy of the quasistatic method based on diffusion theory has been investigated for a wide range of excursions in fast and thermal systems by comparison with full numerical solutions. The results of these investigations show that the quasistatic method can describe even extreme excursions in both fast and thermal systems very accurately.[4,36,37,38] Deterministic diffusion codes which use the quasistatic method to solve the time-dependent diffusion equations include one-dimensional QX1[38], two-dimensional FX2[39] and TWODQD[40], and three dimensional CERBERUS[41], QUANDRY[20], and CONQUEST.[15] Further, TDKENO[42,43,44] was the first successful attempt at modeling time-dependent transport theory with explicit representation of delayed neutrons using a hybrid stochastic/deterministic method embedded in a quasistatic framework. The quasistatic method is described in more detail in the following chapter of this document.

1.3 Objective

The objective of this work is to develop an improved hybrid/stochastic method for transient, three-dimensional neutron transport. These improvements are incorporated in a modified version of the code TDKENO, entitled TDKENO-M. Specifically, these improvements are:

1. Incorporate the improved quasistatic methodology into an existing quasistatic framework. Specifically, include the flux shape derivative in the fixed source term instead of being neglected. Also, compute the point kinetics parameters deterministically by their inner product definitions.

2. Incorporate a hierarchy of three different integration time intervals for the numerical solution of the coupled set of ordinary differential equations. The shape function is assumed to vary linearly over the largest time interval. The second largest time interval is used for determining the point kinetics parameters. Finally, the smallest time step is used for solving the point kinetics equations.

3. Apply TDKENO-M to benchmark problems to determine the accuracy of the method. Particularly, TDKENO-M is applied to one-dimensional and three-dimensional benchmark problems to evaluate its neutronic capabilities.

4. Combine input requirements into a single input file so that TDKENO-M is less cumbersome to execute.

5. Develop the ability to restart a calculation at an intermediate problem time.

6. Develop a "user-friendly" manual for using TDKENO-M which describes in detail the input requirements as well as the output files, subroutines, modules, and the calculational flow.

TDKENO-M is verified by applying the code to Benchmark Problems 14-A2, 16-A1, 16-A2, 16-A3, 16-A6, and 16-A7 contained in the ANL-7416 Supplement 3 Benchmark Problem Book.[45] Problem 14-A2 is a prompt-critical transient in three dimensions with adiabatic heatup and Doppler feedback. Problems 16-A1, 16-A2, 16-A3, 16-A6, and 16-A7 are one-dimensional, two-group transient problems for which solutions have been determined using the one-dimensional transport codes TDA[29] and TIMEX[30]. Finally, comparisons between TDKENO and TDKENO-M are made whenever possible in order to evaluate the improvements in TDKENO-M.

1.4 Originality and Importance of Work

As mentioned, TDKENO[42,43,44] was the first successful attempt at modeling time dependent, three-dimensional transport theory with explicit representation of delayed neutrons using a hybrid stochastic/deterministic method embedded in a quasistatic framework. Therefore, to the best of my knowledge, TDKENO-M will be the first hybrid stochastic/deterministic code that solves the time dependent, three-dimensional transport equation with explicit representation of delayed neutrons using the improved quasistatic methodology. The method not only incorprates the improved quasistatic methodology, it also involves other improvements (i.e., three time steps instead of two, deterministic rather than stochastic calculation of point kinetics parameters, etc.). These improvements will allow TDKENO-M to serve as a benchmark of comparison for other more approximate and less CPU intensive codes without the prohibitive computing times encountered with TDKENO.

1.5 Scope and Organization

This dissertation contains seven chapters. The first chapter provides background information, a summary of relevent previous work, the objectives of this work as well as a description of the originality and importance of this work.

Chapter Two describes in detail the improved quasistatic methodology. Chapter Three contains information related to the numerical solution algorithms used in this work. Chapter Four describes the transient, three-dimensional transport code, TDKENO-M.

Chapter Five describes the results obtained using TDKENO-M to model various benchmark problems.

Chapter Six describes the sensitivity and parametric studies for the numerical solution algorithms. Finally, Chapter Seven contains the conclusions and suggestions for future work.

Chapter Two

Improved Quasistatic Methodology

2.1 Introduction

The quasistatic approach is a method of treating the spatial dynamics of a transport calculation by factoring the flux into an amplitude and a shape function such that the time variation of the shape function is more slowly varying than the time variation of the amplitude function. This factorization of the flux results in splitting the neutronics equation into two coupled equations; an amplitude equation having the form of the point kinetics equation and a shape equation. The improved quasistatic method replaces the time derivative of the shape function by a first order backward difference approximation thus allowing the improved quasistatic method to be applied more accurately. Note that a complete derivation of the quasistatic equations is given in Appendix A.

The development of the improved quasistatic method begins with the time-dependent Boltzmann transport equation with explicit representation of delayed neutrons as given by equations 1 and 2 on page 2 of this document. Also, the time-independent k_{eff} adjoint form of the transport equation is given by:

$$-\overline{\Omega}\cdot\overline{\nabla}\phi^* + \Sigma_t\phi^* = \iint\Sigma_s(\overline{r};\overline{\Omega},E\rightarrow\overline{\Omega}',E')\,\phi^*(\overline{r},\overline{\Omega}',E')\,d\Omega'dE' + \frac{1}{k_{eff}}\iint\chi(E')\,\nu\Sigma_f(\overline{r},E)\,\phi^*(\overline{r},\overline{\Omega}',E')\,d\Omega'dE' \quad (5)$$

where,

ϕ^* = adjoint flux.

Next, the flux is factored into an amplitude function, P(t), and a shape function, Ψ, such that the flux is given by:

$$\phi(\overline{r},\overline{\Omega},E,t) = P(t)\,\psi(\overline{r},\overline{\Omega},E,t) \quad (6)$$

This factorization requires an additional separation constraint for $t>0$ in order to become a unique definition.[35] There are many conditions that could be used, all of which involve constraining some space-energy integral of $\Psi(r,E,t)$ to a constant value for all $t>0$. The only basic requirement is that $\Psi(r,E,t)$ remain positive and bounded at all points in space (r,E) for all time. According to Henry, the following equation fulfills the requirements for the constraint condition and has the added advantage that it facilitates the transition to the point kinetics formulation:[46,47]

$$\iiint\frac{1}{v}\psi(\overline{r},\overline{\Omega},E,t)\,\phi^*(\overline{r},\overline{\Omega},E)\,d\Omega\,dE\,dV = constant \quad (7)$$

2.2 Flux Amplitude Equation

The flux amplitude equation is determined by combining equations 1, 2, and 5 through 7 and integrating over space, energy and angle to produce the following expression:

$$\frac{dP(t)}{dt} = \frac{\rho(t) - \overline{\beta}(t)}{\Lambda(t)} P(t) + \sum_j \lambda_j C_j(t) + \overline{Q}(t) \tag{8}$$

with the following parameter definitions:

$$\begin{aligned} \rho(t) = & -\frac{1}{F(t)} \iiint (\Sigma_t(\overline{r}, E, t) - \Sigma_t(\overline{r}, E)) \psi(\overline{r}, \overline{\Omega}, E, t) \phi^*(\overline{r}, \overline{\Omega}, E)\, d\Omega\, dE\, dV \\ & + \frac{1}{F(t)} \iiiint \phi^*(\overline{r}, \overline{\Omega}, E) \left[\left(\Sigma_s(\overline{r}; \overline{\Omega}', E' \rightarrow \overline{\Omega}, E; t) - \Sigma_s(\overline{r}; \overline{\Omega}', E' \rightarrow \overline{\Omega}, E) \right) \right. \\ & \left. + \chi_p(E) \left(\nu\Sigma_f(\overline{r}, E', t) - \nu\Sigma_f(\overline{r}, E') \right) \right] \psi(\overline{r}, \overline{\Omega}', E', t)\, d\Omega' dE' d\Omega\, dE\, dV \end{aligned} \tag{9}$$

$$\overline{\beta}(t) = \sum_j \overline{\beta}_j(t) \tag{10}$$

$$\begin{aligned} \overline{\beta}_j(t) = & \frac{1}{F(t)} \iiiint \chi(E)\, \beta_j \nu\Sigma_f(\overline{r}, E') \psi(\overline{r}, \overline{\Omega}', E', t) \\ & \times \phi_0^*(\overline{r}, \overline{\Omega}, E)\, d\Omega' dE' d\Omega\, dE\, dV \end{aligned} \tag{11}$$

$$\Lambda(t) = \frac{1}{F(t)} \iiint \frac{1}{v} \psi(\bar{r}, \bar{\Omega}, E, t) \phi^*(\bar{r}, \bar{\Omega}, E)\, d\Omega\, dE dV \qquad (12)$$

$$C_j(t) = \frac{(1/F(t))}{\Lambda} \iiint \phi^*(\bar{r}, \bar{\Omega}, E)\, C_j(\bar{r}, t)\, \chi_j(E)\, d\Omega\, dE dV \qquad (13)$$

$$\bar{Q}(t) = \frac{(1/F(t))}{\Lambda} \iiint \phi^*(\bar{r}, \bar{\Omega}, E)\, Q(\bar{r}, \bar{\Omega}, E)\, d\Omega\, dE dV \qquad (14)$$

$$F(t) = \iiint\!\!\iint \chi(E)\, \nu\Sigma_f(\bar{r}, E')\, \psi(\bar{r}, \bar{\Omega}', E', t)\, \phi_0^*(\bar{r}, \bar{\Omega}, E)\, d\Omega' dE' d\Omega\, dE dV \qquad (15)$$

where,

ϕ_0^* = steady-state k_{eff} adjoint flux.

Note that $\Lambda(t)$ is called the prompt neutron lifetime in Reference 33, but is called the generation time in this work as well as other references.

2.3 Delayed Neutron Precursor Equations

The delayed neutron precursor equations are obtained by multiplying equation 2 by $\chi_j(E)\phi^*$, integrating over space, energy and angle, and using the separability condition given in equation 6 to yield the following equation for each precursor group:

$$\frac{\partial C_j(t)}{\partial t} + \lambda_j C_j(t) = \frac{\overline{\beta}_j(t)}{\Lambda} P(t) \qquad j=1,N \tag{16}$$

2.4 Shape Equation

The shape equation is obtained by substituting the separability condition (equation 6) into the time-dependent Boltzmann transport equation and dividing by P(t) to yield the following:

$$\begin{aligned}
&\frac{1}{v}\cdot\frac{1}{P(t)}\cdot\frac{\partial P(t)}{\partial t}\cdot\psi(\overline{r},\overline{\Omega},E,t) + \frac{1}{v}\frac{\partial\psi(\overline{r},\overline{\Omega},E,t)}{\partial t} \\
&+ \overline{\Omega}\cdot\overline{\nabla}\psi(\overline{r},\overline{\Omega},E,t) + \Sigma_t(\overline{r},E,t)\,\psi(\overline{r},\overline{\Omega},E,t) \\
&= \int\int\Sigma_s(\overline{r};\overline{\Omega}',E'\rightarrow\overline{\Omega},E;t)\,\psi(\overline{r},\overline{\Omega}',E',t)\,d\Omega'dE' \\
&+ \int\int\chi_p(E)\,(1-\beta)\,\nu\Sigma_f(\overline{r},E',t)\,\psi(\overline{r},\overline{\Omega}',E',t)\,d\Omega'dE' \\
&+ \frac{Q_d(\overline{r},E,t)}{P(t)} + \frac{Q}{P(t)}
\end{aligned} \tag{17}$$

where,

$$\begin{aligned}
Q_d(\overline{r},E,t) &= \int_{-\infty}^{t}\int_{E'}\int_{\overline{\Omega}'}\nu\Sigma_f(\overline{r},E',t)\,P(t')\,\psi(\overline{r},\overline{\Omega}',E',t) \\
&\times\sum_j\beta_j\chi_j\lambda_j e^{-\lambda_j(t-t')}\,d\Omega'dE'dt'
\end{aligned} \tag{18}$$

The quasistatic method, as originally applied in TDKENO, computes Q_d from equation 18 and uses this in equation 17 with $\partial\Psi/\partial t$ set equal to zero. The $[\partial P(t)/\partial t]/P(t)$ term is determined from the solution of the point kinetics equation (equation 8).

The improved quasistatic method, as applied in TDKENO-M, solves the full shape equation (equation 17) by replacing the time derivative of the shape function with a backward difference approximation of first order:

$$\frac{\partial}{\partial t}\psi(\bar{r},\bar{\Omega},E,t) = \frac{\psi(\bar{r},\bar{\Omega},E,t) - \psi(\bar{r},\bar{\Omega},E,t-\Delta t)}{\Delta t} \qquad \textbf{(19)}$$

where,

$t - \Delta t$ = time of the last shape calculation,
Δt = time interval for shape calculations.

Since the shape function is usually slowly varying in time, this approximation is valid over a much larger time range than is acceptable for the total flux. Using the backward difference approximation for the time derivative of the shape function yields the following shape equation:

$$\frac{1}{v}\cdot\frac{1}{P(t)}\cdot\frac{\partial P(t)}{\partial t}\cdot\psi(\bar{r},\bar{\Omega},E,t) + \frac{1}{v}\left(\frac{\psi(\bar{r},\bar{\Omega},E,t) - \psi(\bar{r},\bar{\Omega},E,t-\Delta t)}{\Delta t}\right)$$

$$+ \bar{\Omega}\cdot\bar{\nabla}\psi(\bar{r},\bar{\Omega},E,t) + \Sigma_t(\bar{r},E,t)\,\psi(\bar{r},\bar{\Omega},E,t)$$

$$= \int\int\Sigma_s(\bar{r};\bar{\Omega}',E'\rightarrow\bar{\Omega},E;t)\,\psi(\bar{r},\bar{\Omega}',E',t)\,d\Omega'dE' \qquad \textbf{(20)}$$

$$+ \int\int\chi_p(E)\,(1-\beta)\,\nu\Sigma_f(\bar{r},E',t)\,\psi(\bar{r},\bar{\Omega}',E',t)\,d\Omega'dE'$$

$$+ \frac{Q_d(\bar{r},E,t)}{P(t)} + \frac{Q}{P(t)}$$

which can be rearranged as:

$$\frac{1}{v}\left(\frac{1}{P(t)}\cdot\frac{\partial P(t)}{\partial t} + \frac{1}{\Delta t}\right)\psi(\bar{r},\bar{\Omega},E,t)$$

$$+ \bar{\Omega}\cdot\bar{\nabla}\psi(\bar{r},\bar{\Omega},E,t) + \Sigma_t(\bar{r},E,t)\,\psi(\bar{r},\bar{\Omega},E,t)$$

$$- \int\int\Sigma_s(\bar{r};\bar{\Omega}',E'\rightarrow\bar{\Omega},E;t)\,\psi(\bar{r},\bar{\Omega}',E',t)\,d\Omega'dE' \qquad \textbf{(21)}$$

$$- \int\int\chi_p(E)\,(1-\beta)\,\nu\Sigma_f(\bar{r},E',t)\,\psi(\bar{r},\bar{\Omega}',E',t)\,d\Omega'dE'$$

$$= \frac{Q_d(\bar{r},E,t)}{P(t)} + \frac{Q}{P(t)} + \frac{\psi(\bar{r},\bar{\Omega},E,t-\Delta t)}{v\Delta t}$$

This shape equation is an inhomogeneous partial differential equation. As with the quasistatic method, Q_d is computed from equation 18 and the $[\partial P(t)/\partial t]/P(t)$ term is determined from the solution of the point kinetics equation (equation 8). With the improved quasistatic method, there is now an additional $1/\Delta t$ term as well as an additional source term from the backward difference approximation of the shape derivative which is incorporated as described below.

The right hand side of the backward difference approximation (equation 19) is used to modify the source distribution in KENO V.a for each generation of neutrons. A random selection is made based on the cumulative distribution functions to determine whether each neutron is prompt, delayed, or contributes to the flux shape derivative. If a neutron is selected to be delayed, the starting energy, angle, and position are determined from the delayed source distribution. If the neutron is determined to contribute to the flux shape derivative, the starting parameters are selected based on the source distribution calculated from the numerical integration of the right hand side of the backward difference approximation. Note that prompt neutrons are tracked by KENO V.a without any modifications.

2.5 System Power

The system power can be represented by:

$$\begin{aligned} Power(t) &= \epsilon_f \iiint \Sigma_f(\bar{r}, E)\, \phi(\bar{r}, \bar{\Omega}, E, t)\, d\Omega\, dE dV \\ &= P(t)\, \epsilon_f \iiint \Sigma_f(\bar{r}, E)\, \psi(\bar{r}, \bar{\Omega}, E, t)\, d\Omega\, dE dV \end{aligned} \tag{22}$$

where,

ϵ_f = average energy release per fission.

2.6 Point Kinetics Parameters

The original version of TDKENO determines the point kinetics parameters (reactivity, generation time, etc.) during the random walk using the concept of

tracklength estimation to determine the following contribution to reactivity and generation time at every collision point and at every boundary crossing during the random walk of a Monte Carlo calculation:[42,43,44]

$$\rho = l \times wate \left[-\Delta\Sigma_t(\bar{r}, E', t)\,\phi_0^*(\bar{r}, \bar{\Omega}', E') \right.$$

$$+ \int\int \Delta\Sigma_s(\bar{r}; \bar{\Omega}', E' \rightarrow \bar{\Omega}, E; t)\,\phi_0^*(\bar{r}, \bar{\Omega}, E)\, d\Omega\, dE \qquad (23)$$

$$\left. + \int\int \chi(E)\,\Delta\nu\Sigma_f(\bar{r}, E', t)\,\phi_0^*(\bar{r}, \bar{\Omega}, E)\, d\Omega\, dE \right]$$

and

$$\Lambda = l \times wate \times \frac{1}{v}\phi^*(\bar{r}, \bar{\Omega}, E) \qquad (24)$$

where,

l	=	tracklength
wate	=	neutron weight.

TDKENO-M determines the point kinetics parameters deterministically using their inner product definitions as given by equations 9-15 with an interpolated shape function. This allows the point kinetics parameters to be calculated at times intermediate to the flux shape calculations and, therefore, should provide improved accuracy relative to TDKENO for many transients of interest. This procedure should also be much more efficient than TDKENO since the additional histories required by TDKENO to have a small uncertainty in reactivity is avoided in TDKENO-M due to the fact that the reactivity is computed deterministically.

Chapter Three

Numerical Solution

3.1 Time-step Hierarchy

The shape equation (21), point kinetics equations (8 & 16), normalization constraint equation (7), and inner product definitions (9 through 15) are solved numerically using a hierarchy of three different integration time intervals as shown in Figure 1.

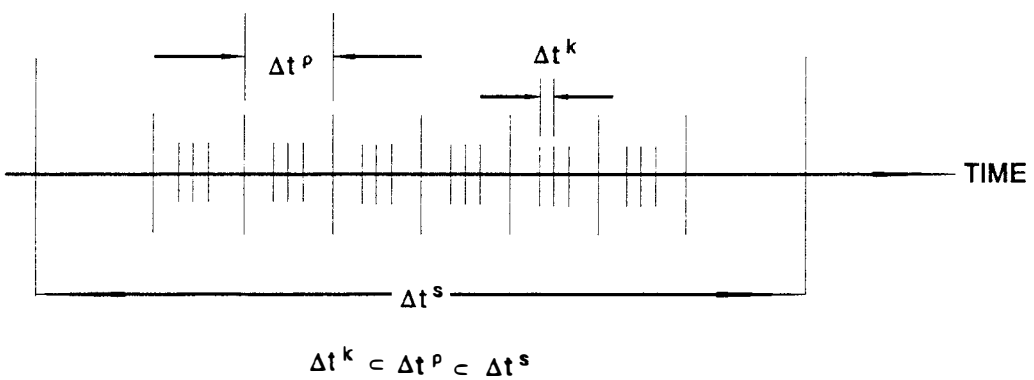

Figure 1: Time-step Hierarchy in the Improved Quasistatic Method

The shape function is assumed to vary linearly over the largest time interval, Δt^s, where for the n^{th} time interval, $\Delta t^s_n = t^s_n - t^s_{n-1}$. The reactivity, delayed neutron fraction, and the generation time are calculated using the Δt^o time intervals with linearly interpolated flux shapes. The Δt^k time intervals are used for solving the point kinetics equations as well as the delayed neutron spatial precursor concentrations with

quadratically (second order) interpolated point kinetics parameters.

3.2 Solution Algorithm

A general description of the solution algorithm for the improved quasistatic method is given below:

1. Assume everything is known at time t^s_{n-1}.

2. Extrapolate the shape function, $\Psi(\bar{r},\bar{\Omega},E,t)$, linearly with respect to time from $\Psi(\bar{r},\bar{\Omega},E,t^s_{n-1})$ to $\Psi(\bar{r},\bar{\Omega},E,t^s_n)$.

3. Evaluate reactivity, generation time, and delayed neutron fraction at the t^o_n points out to t^s_n using equations 9-15 with an interpolated shape function.

4. Interpolate reactivity, generation time and delayed neutron fraction quadratically (second order) between t^s_{n-1} and t^o_1 at the t^k points.

5. Solve the point kinetics equations and update the delayed neutron precursor spatial distribution using LSODE out to t^o_1 at the t^k points using interpolated point kinetics parameters.

6. If the problem considers thermalhydraulic feedback (i.e., Benchmark 14-A2), compute the temperature distribution out to t^o_1 at the t^k points using the adiabatic heatup model.

7. If $t^o_1 < t^s_n$, go to the next t^o_n time point and repeat Steps 4 through 7.

8. It $t^o_n = t^s_n$, calculate a new flux shape.

9. Normalize the flux shape to satisfy the normalization constraint equation.

10. Repeat steps 3 through 7 until $t^o_n = t^s_n$ using a shape function, $\Psi(\bar{r},\bar{\Omega},E,t)$, which is linearly interpolated with respect to time from $\Psi(\bar{r},\bar{\Omega},E,t^s_{n-1})$ to $\Psi(\bar{r},\bar{\Omega},E,t^s_n)$.

11. Go to the next Δt^s time interval and repeat Steps 2 through 10.

Chapter Four

TDKENO-M

4.1 Introduction

TDKENO-M is a Fortran code which solves the time-dependent, three-dimensional Boltzmann transport equation with explicit representation of delayed neutrons as given by equations 1 and 2. The code consists of several large subroutines which perform specific functions as described below:

1. KENO V.a — Modified to perform a fixed source calculation with user-defined angular bining and to explicitly track delayed neutrons based on the distribution function given by equation 18. KENO V.a was also modified to include the flux shape derivative with the fixed source term using a backward difference approximation of first order (equation 19). Note that the modified version of KENO V.a uses both AMPX and Monte Carlo formatted binary cross section libraries.[32]

2. RHO — Determines the point kinetics parameters (reactivity, generation time and effective delayed neutron fraction) from their inner product definitions as given by equations 9 through 15.

3. PTKIN — Solves the point kinetics equations using the Livermore Solver of Ordinary Differential Equations (LSODE).

4. CDELAY — Calculates the delayed neutron precursor spatial distribution (equation 18) as well as the cumulative distribution functions used to incorporate the flux shape derivative.

5. POWER — Determines the power history using equation 22.

Refer to the user's manual as well as the code listing given in Appendix B for greater detail on the individual subroutines as well as the input and output files for a sample problem.

4.2 Calculational Flow

The computational flow for a typical TDKENO-M calculation is given in Figure 2. Initially, a steady-state adjoint calculation is performed using the modified version of KENO V.a. This steady state adjoint flux is used for all subsequent calculations of the point kinetics parameters. Next, a steady-state forward KENO V.a calculation is performed to determine the initial flux shape as well as the effective multiplication factor. Next, subroutines RHO and PTKIN are called to determine the initial values of the point kinetics parameters and the flux amplitude, respectively. In the case of thermalhydraulic feedback (i.e., Benchmark 14-A2), subroutine FEEDBACK is called to calculate the system temperature from the adiabatic heatup model described in Appendix C as well as the system power. Note that the system power calculated by subroutine FEEDBACK is only used intermediately to calculate the system temperature and is not written as output. The subroutine CDELAY is then called by TDKENO-M to determine the delayed neutron source distribution as well as the cumulative distribution functions needed for the implementation of the flux shape derivative for the transient calculations.

TDKENO-M next calls KENO V.a to perform the first flux shape calculation of the perturbed system. Subsequently, subroutines RHO, PTKIN and CDELAY are called again to update the system parameters and the amplitude as described above. This sequence is continued until the desired cutoff time is reached for the transient calculation. Upon the completion of the transient calculation, subroutine POWER is used to determine the power trace for the system.

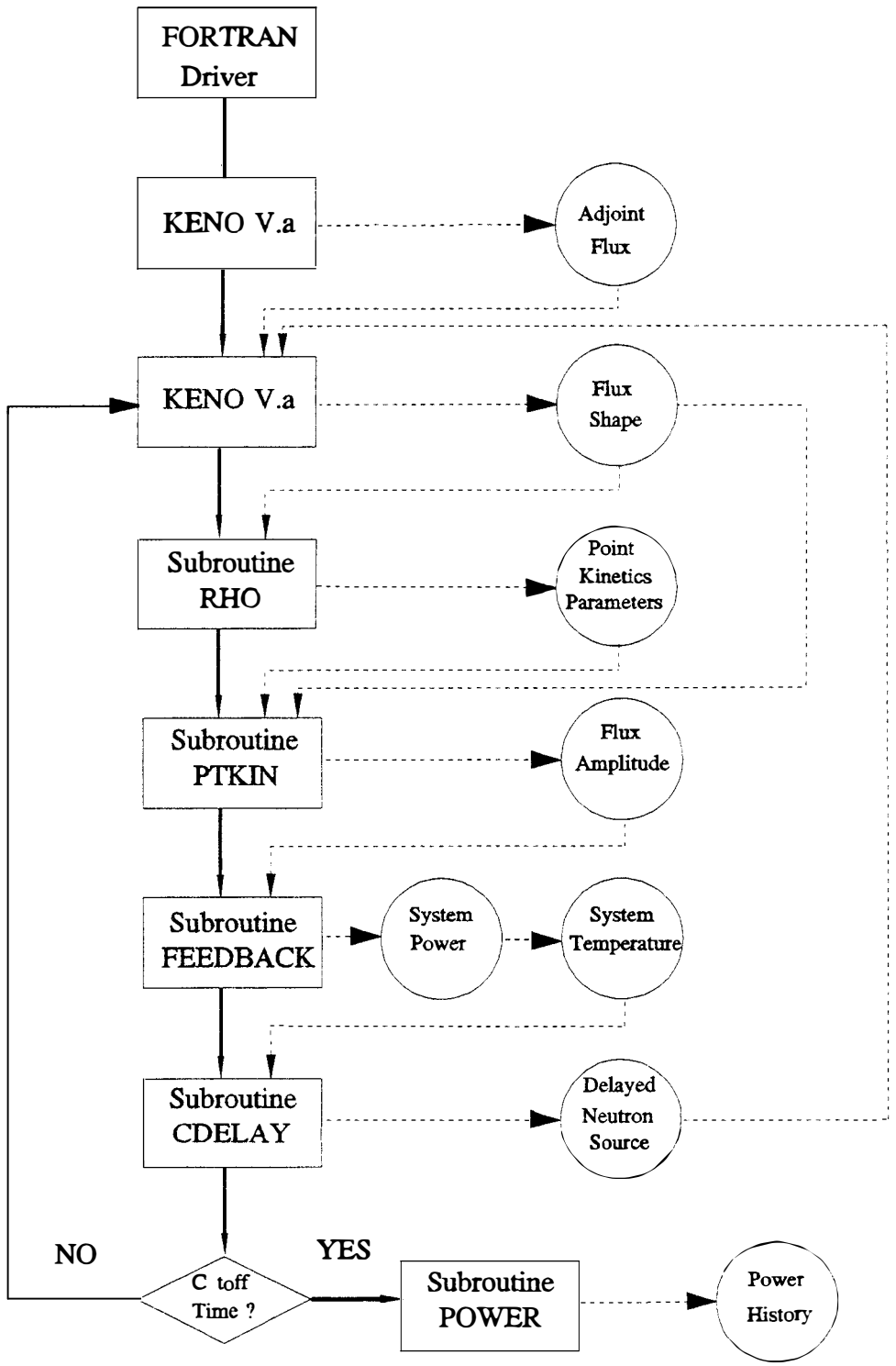

Figure 2: Computational Flow for TDKENO-M

Chapter Five

Results

5.1 Introduction

To verify that TDKENO-M accurately solves the three-dimensional, time-dependent Boltzmann transport equation with explicit representation of delayed neutrons, TDKENO-M was used to model benchmark problems 16-A1, 16-A2, 16-A3, 16-A6[a], 16-A7[a], and 14-A2 in the Argonne National Laboratory Benchmark Problem Book[45] using a Hewlett-Packard 9000 model 730 workstation. Problems 16-A1, 16-A2, 16-A3, 16-A6 and 16-A7 are one-dimensional, two-group transient problems for which solutions have been determined using the one-dimensional transport codes TDA[29] and TIMEX[30]. Problem 14-A2 is a prompt-critical transient in three dimensions with adiabatic heatup and Doppler feedback designed to test codes based on two-group diffusion theory. Note that one-dimensional transport benchmarks and three-dimensional diffusion benchmarks are used in this work since transient, three-dimensional transport benchmarks do not exist. Further, even though the 16A series of benchmark problems are one-dimensional problems, TDKENO-M still performs all calculations in three dimensions. A complete description of the benchmark problems is given in Appendix B.

a Benchmark problems 16-A6 and 16-A7 are unpublished benchmark problems which use the same geometrical configuration and data as Benchmark problem 16-A1.

5.2 Benchmark Problem 16-A1

Problem 16-A1 is a seven region, one-dimensional slab liquid metal fast breeder reactor (LMFBR). The configuration consists of three core regions (zones 2, 4, and 6) containing core material and sodium, two control rod regions (zones 3 and 5) containing control rod material and sodium, and two fuel blanket regions (zones 1 and 7) containing blanket material and sodium. Refer to Figure 3 for a schematic representation of the problem. Two neutron energy groups and six delayed neutron groups are used (Tables 1 and 2). The initiating perturbation is caused by a 5% increase in density of the material in zone 2 and a 5% decrease in density of the material in zone 6 resulting in a delayed supercritical transient.

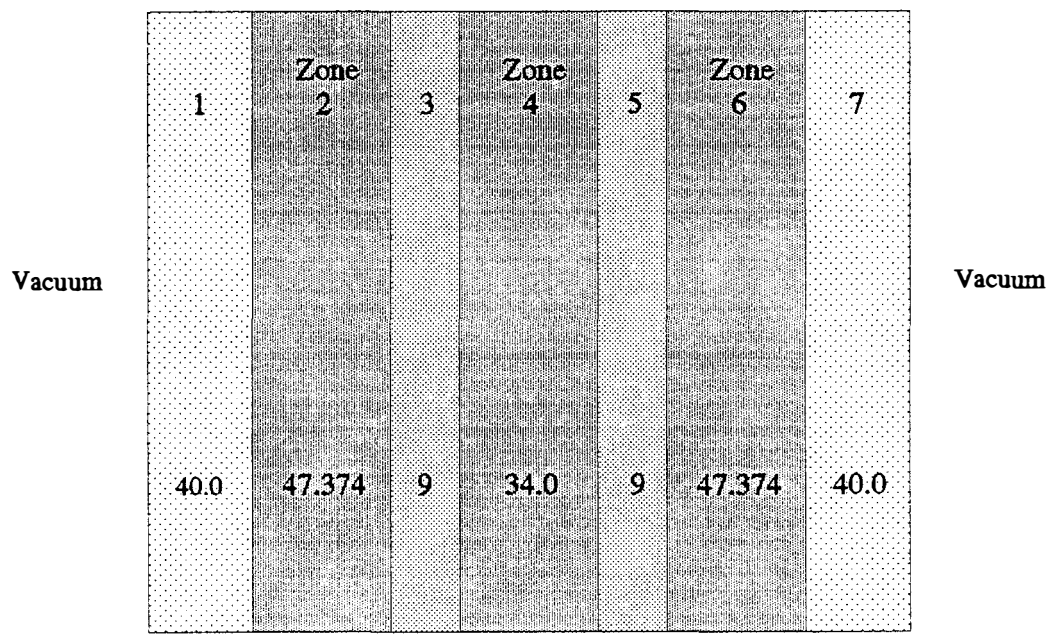

Figure 3: Schematic Diagram of Benchmark Problem 16-A1

Table 1: Benchmark Problem 16-A1 Initial Two-Group Data

Zone	Group	$\nu\Sigma_f^i$	Σ_t^i	$\Sigma_s^{i\to i}$	$\Sigma_s^{i\to j}$
1,7	1	8.3441×10^{-4}	2.411×10^{-1}	2.336×10^{-1}	3.598×10^{-3}
	2	3.2776×10^{-4}	4.172×10^{-1}	4.070×10^{-1}	0.0
2,4,6	1	7.4518×10^{-3}	1.849×10^{-1}	1.777×10^{-1}	2.085×10^{-3}
	2	1.1061×10^{-2}	3.668×10^{-1}	3.537×10^{-1}	0.0
3,5	1	0.0	9.432×10^{-2}	8.571×10^{-2}	1.717×10^{-3}
	2	0.0	1.876×10^{-1}	1.713×10^{-1}	0.0

Table 2: Benchmark Problem 16-A1 Delayed Neutron Parameters[a]

Delayed Neutron Group	Delayed Neutron Fraction, β	Decay Constant, λ
1	8.10×10^{-5}	1.29×10^{-2}
2	6.87×10^{-4}	3.11×10^{-2}
3	6.12×10^{-4}	1.34×10^{-1}
4	1.14×10^{-3}	3.31×10^{-1}
5	5.12×10^{-4}	1.26
6	1.70×10^{-4}	3.21

[a]Prompt and delayed neutron spectra are identical with $\chi_1=1.0$ and $\chi_2=0.0$. Also, $1/v_1 = 1.851\times10^{-9}$ sec/cm and $1/v_2 = 1.088\times10^{-8}$ sec/cm.

Results obtained with TDKENO-M as well as TIMEX and TDA for Benchmark Problem 16-A1 are given in Figures 4-6 and Table 3. Figure 4 is a plot of normalized power versus time (power trace). Figures 5 and 6 are plots of the normalized flux distribution for neutron energy groups one and two, respectively. Table 3 is a tabular representation of the CPU time and the error (as compared to TIMEX and TDA) associated with both TDKENO and TDKENO-M due to a reduction of the total number of histories for each flux shape calculation. As the number of histories decreases, the accuracy of TDKENO-M is much better than TDKENO. This is because TDKENO-M computes reactivity and generation time deterministically rather than using the stochastic method (i.e., boundary crossing estimation) used by TDKENO which requires a huge number of histories to obtain acceptable uncertainties.

The results show that TDKENO-M is quite accurate in comparison with Benchmark Problem 16-A1 (approximately 2% error in power trace) with as much as a factor of 400 reduction in CPU time relative to TDKENO. Note that the results presented were obtained by performing shape calculations at 0.0, 10^{-6}, 10^{-5}, and 10^{-2} seconds. Further, ten intermediate calculations of the point kinetics parameters are performed between each shape calculation and ten flux amplitude calculations are performed between each calculation of the point kinetics parameters. Also, 21 spatial mesh intervals are used for modeling Benchmark Problem 16-A1 as described in Appendix D.

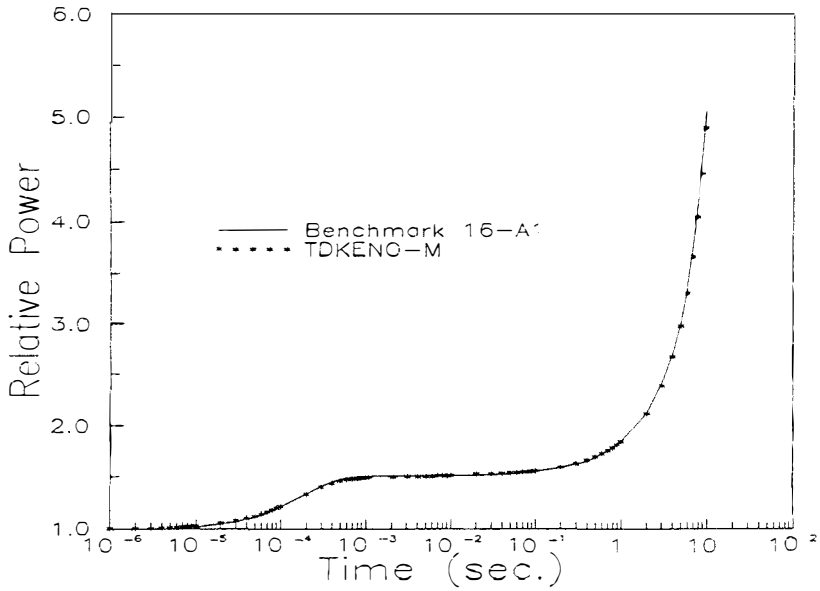

Figure 4: Relative Power vs Time for Benchmark Problem 16-A1

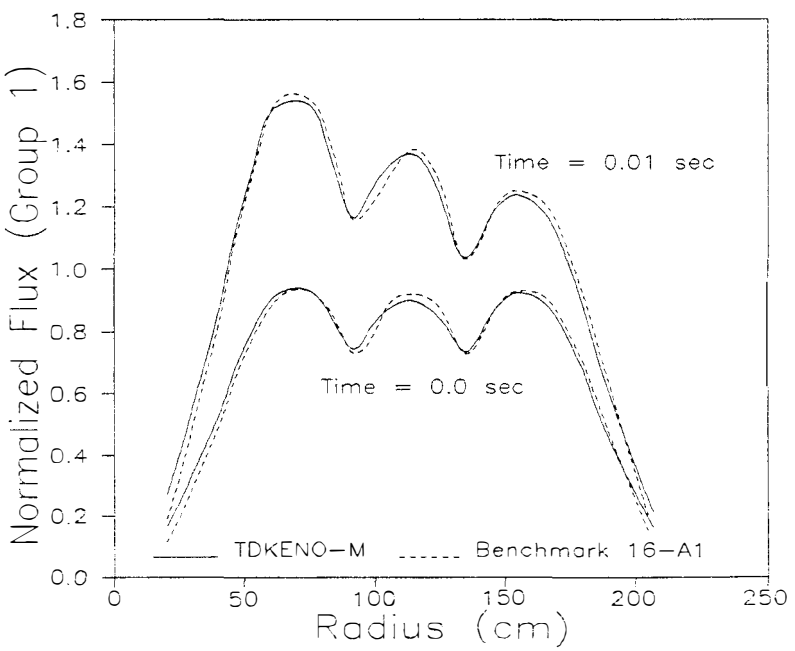

Figure 5: Normalized Group 1 Flux vs Radial Position for Problem 16-A1

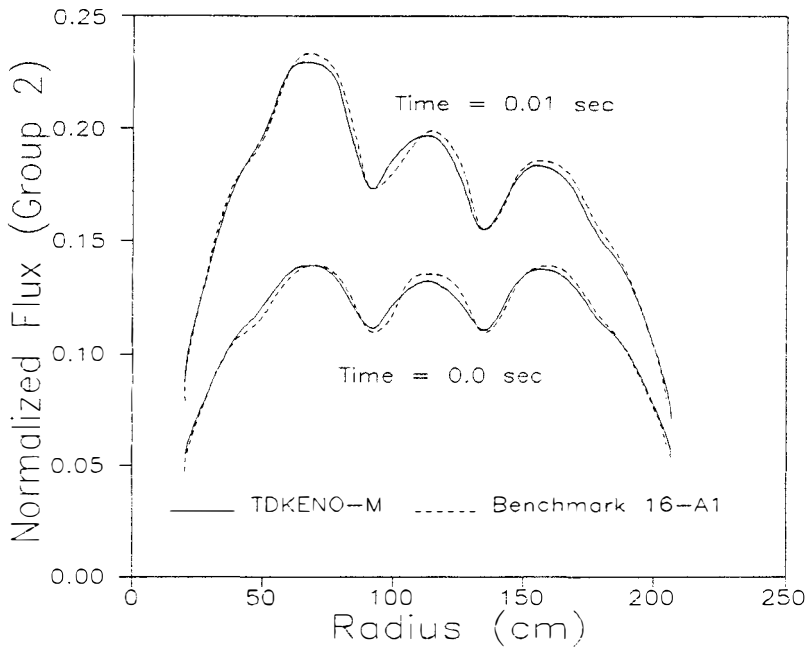

Figure 6: Normalized Group 2 Flux vs Radial Position for Problem 16-A1

Table 3: Results for Benchmark Problem 16-A1

Fraction of Original Histories	CPU Time* (min.)	Average Shape* Deviation (%)	Average Power Deviation (%) TDKENO	Average Power Deviation (%) TDKENO-M
1	2023	0.25	0.41	0.13
1/2	1019	0.51	2.02	0.16
1/4	509	0.72	3.64	0.21
1/10	213	0.86	12.62	0.39
1/100	23.6	2.13	69.23	1.02
1/1000	4.8	7.24	154.3	2.14

*Values of CPU Time and Average Flux Deviation are for both TDKENO and TDKENO-M since these parameters depend primarily on the total number of histories and not the solution methodology.

5.3 Benchmark Problem 16-A2

Problem 16-A2 is identical to Benchmark 16-A1 except for the initiating perturbation which is a 10% increase in density in core zone 2 and a 10% decrease in density of core zone 6 (refer to Figure 3 on page 25). These perturbations result in a prompt supercritical transient.

Results obtained with TDKENO-M as well as TIMEX and TDA for Benchmark Problem 16-A2 are given in Figure 7 and Table 4. Figure 7 is a plot of normalized power versus time (power trace). Table 4 is a tabular representation of the CPU time and the error (as compared to TIMEX and TDA) associated with both TDKENO and TDKENO-M. Similar to Benchmark Problem 16-A1, as the number of histories decreases, the accuracy of TDKENO-M is much better than TDKENO.

Results show that TDKENO-M is quite accurate in comparison with Benchmark Problem 16-A2 (approximately 2.4% error in power trace) with as much as a factor of 700 reduction in CPU time relative to TDKENO. Note that the results presented were obtained by performing flux shape calculations at 0.0, 10^{-4}, and 1.1×10^{-3} seconds. Further, ten intermediate calculations of the point kinetics parameters are performed between each shape calculation and ten flux amplitude calculations are performed between each calculation of the point kinetics parameters. Also, 32 spatial mesh intervals are used for modeling Benchmark Problem 16-A2 as described in Appendix D.

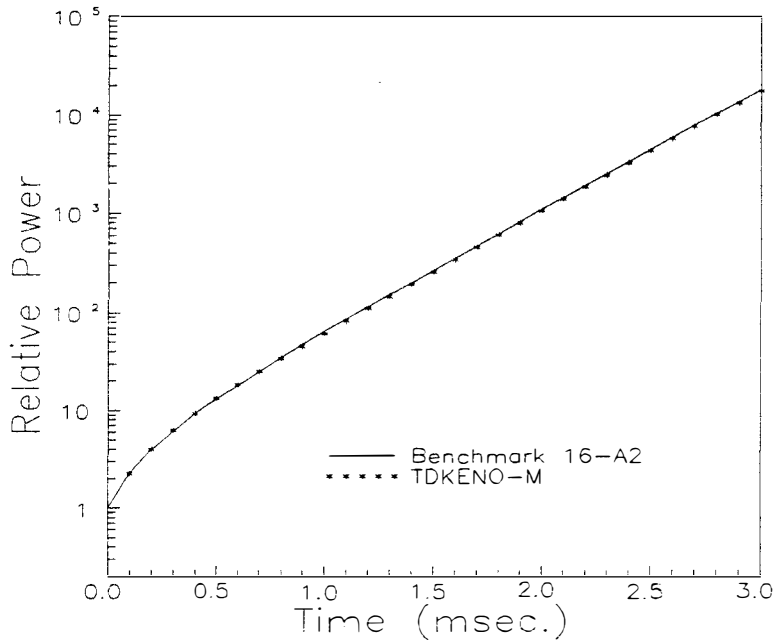

Figure 7: Relative Power vs Time for Benchmark Problem 16-A2

Table 4: Results for Benchmark Problem 16-A2

Fraction of Original Histories	CPU Time* (min.)	Average Shape* Deviation (%)	Average Power Deviation (%) TDKENO	Average Power Deviation (%) TDKENO-M
1	7201	0.23	0.37	0.25
1/2	3604	0.49	1.84	0.32
1/4	1805	0.76	3.96	0.38
1/10	725	0.88	10.73	0.43
1/100	75.4	2.52	58.80	1.38
1/1000	9.75	6.03	136.9	2.43

*Values of CPU Time and Average Flux Deviation are for both TDKENO and TDKENO-M since these parameters depend primarily on the total number of histories and not the solution methodology.

5.4 Benchmark Problem 16-A3

Benchmark Problem 16-A3 simulates the complete ejection of a control rod from the reactor followed by a complete control rod insertion at a different location. Initially, the material in zone 5 is changed from a mixture of sodium and control rod material to complete sodium. Then, at 0.0001 sec, the material in zone 3 is changed from a mixture of sodium and control rod material to 100% control rod material.

Results obtained with TDKENO-M as well as TIMEX and TDA for Benchmark Problem 16-A3 are given in Figure 8 and Table 5. Figure 8 is a plot of normalized power versus time. Table 5 gives the CPU time and the error (as compared to TIMEX and TDA) associated with both TDKENO and TDKENO-M. Results show that TDKENO-M is quite accurate in comparison with Benchmark Problem 16-A3 (approximately 2.5% error in power trace) with as much as a factor of 350 reduction in CPU time relative to TDKENO. The results presented were obtained by performing shape calculations at 0.0, $2x10^{-6}$, $5x10^{-5}$, $9x10^{-5}$, $1x10^{-4}$, $1.01x10^{-4}$ and $1.5x10^{-3}$ seconds using 29 mesh intervals as described in Appendix D.

It should be noted that the power trace discrepancy after the control rod insertion as encountered with TDKENO[44] (see Figure 8) is not present using TDKENO-M. This is due to the fact that the flux shape derivative term is included in TDKENO-M while it is ignored in TDKENO. Thus, the flux shape derivative creates a retardation time which, if not modeled, causes the power trace to be shifted towards larger times.

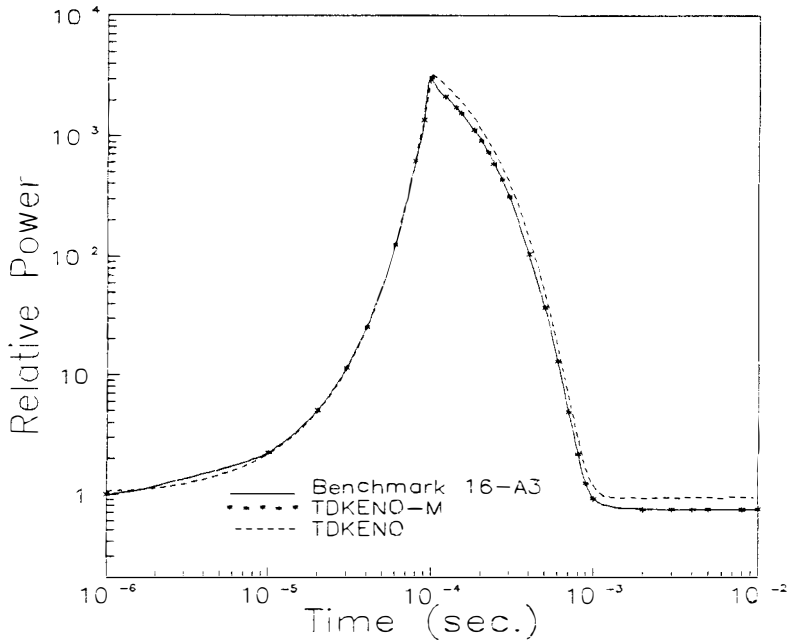

Figure 8: Relative Power vs Time for Benchmark Problem 16-A3

Table 5: Results for Benchmark Problem 16-A3

Fraction of Original Histories	CPU Time* (min.)	Average Shape* Deviation (%)	Average Power Deviation (%) TDKENO	Average Power Deviation (%) TDKENO-M
1	3145	0.32	0.71	0.15
1/2	1575	0.64	3.74	0.18
1/4	789	0.89	4.96	0.24
1/10	319	0.96	15.30	0.45
1/100	35.2	2.53	79.48	1.13
1/1000	8.42	8.82	217.8	2.56

*Values of CPU Time and Average Flux Deviation are for both TDKENO and TDKENO-M since these parameters depend primarily on the total number of histories and not the solution methodology.

5.5 Benchmark Problem 16-A6

Problem 16-A6 simulates both control rod movement and material motion. The transient is initiated by a decrease in control rod material and an increase in sodium in zone 5 resulting in the cross sections for zone 5 at $t \geq 0.0$ seconds given in Table 6. Also, for the time interval between $t = 0.0$ and $t = 0.5$ seconds, the densities of the materials in zones 1, 2, 4, 6, and 7 are changed linearly (i.e., material motion) resulting in final cross sections for each zone at $t = 0.5$ sec as shown in Table 7.

Results obtained with TDKENO-M as well as TIMEX and TDA for Benchmark Problem 16-A6 are given in Figure 9 and Table 8. Figure 9 is a plot of normalized power versus time. Table 8 is a tabular representation of the CPU time and the error (as compared to TIMEX and TDA) associated with both TDKENO and TDKENO-M.

Results show that TDKENO-M is quite accurate in comparison with Benchmark Problem 16-A6 (approximately 2.1% error in power trace) with as much as a factor of 90 reduction in CPU time relative to TDKENO. Note that the results presented were obtained by performing flux shape calculations at 0.0, 10^{-5}, 10^{-2}, 10^{-1}, $2x10^{-1}$, and $5x10^{-1}$ seconds using 36 mesh intervals as described in Appendix D. Further, ten intermediate calculations of the point kinetics parameters are performed between each shape calculation and ten flux amplitude calculations are performed between each calculation of the point kinetics parameters.

Table 6: Two-Group Constants for Step Perturbation in Zone 5

Energy Group	Σ_t^i	$\Sigma_s^{i\to i}$	$\Sigma_s^{i\to j}$
1	9.05399×10^{-2}	8.24499×10^{-2}	1.6550×10^{-3}
2	1.7860×10^{-1}	1.640×10^{-1}	0.0

Table 7: Two-Group Constants for Ramp Perturbations

Zone	Group	$\nu\Sigma_f^i$	Σ_t^i	$\Sigma_s^{i\to i}$	$\Sigma_s^{i\to j}$
1	1	1.71696×10^{-3}	2.62999×10^{-1}	2.54691×10^{-1}	3.8449×10^{-3}
	2	1.63779×10^{-3}	4.60642×10^{-1}	4.48897×10^{-1}	0.0
2	1	6.70661×10^{-3}	1.6641×10^{-1}	1.5994×10^{-1}	1.8765×10^{-3}
	2	9.95507×10^{-3}	3.3012×10^{-1}	3.18349×10^{-1}	0.0
4,6	1	2.98072×10^{-3}	7.39599×10^{-2}	7.10843×10^{-2}	8.3399×10^{-3}
	2	4.42448×10^{-3}	1.4672×10^{-1}	1.41488×10^{-1}	0.0
7	1	9.93014×10^{-3}	4.66791×10^{-1}	4.50559×10^{-1}	6.1429×10^{-3}
	2	1.38292×10^{-2}	8.64919×10^{-1}	8.38759×10^{-1}	0.0

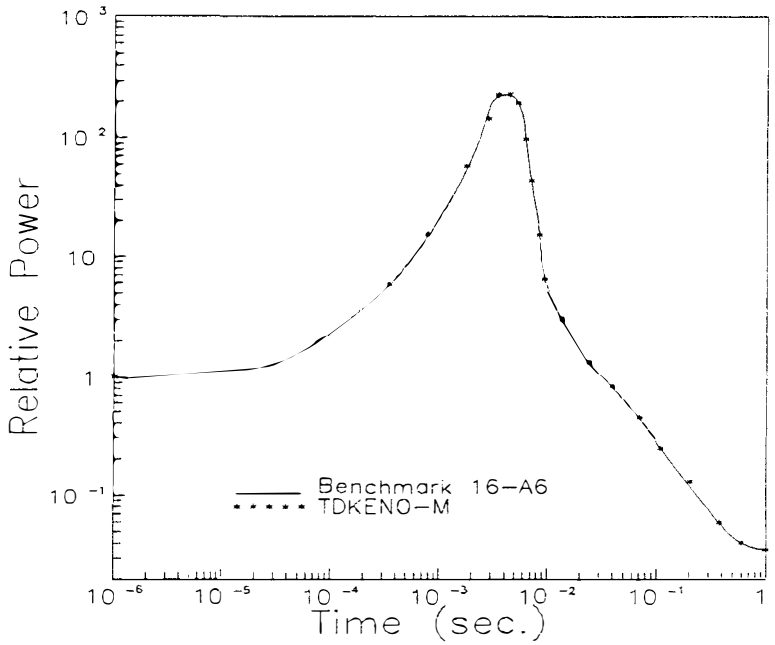

Figure 9: Relative Power vs Time for Benchmark Problem 16-A6

Table 8: Results for Benchmark Problem 16-A6

Fraction of Original Histories	CPU Time* (min.)	Average Shape* Deviation (%)	Average Power Deviation (%) TDKENO	Average Power Deviation (%) TDKENO-M
1	866	0.53	0.62	0.31
1/2	435	1.17	4.49	0.42
1/4	219	1.62	8.69	0.69
1/10	88.2	1.95	39.83	0.93
1/100	9.23	4.58	204.72	2.13

*Values of CPU Time and Average Flux Deviation are for both TDKENO and TDKENO-M since these parameters depend primarily on the total number of histories and not the solution methodology.

5.6 Benchmark Problem 16-A7

Benchmark Problem 16-A7 is identical to Benchmark Problem 16-A6 except that the ramp representing material motion is replaced with a step perturbation at 0.01 seconds. Results obtained with TDKENO-M as well as TIMEX and TDA for Benchmark Problem 16-A7 are given in Figure 10 and Table 9. Figure 10 is a plot of normalized power versus time (power trace). Table 9 is a tabular representation of the CPU time and the error (as compared to TIMEX and TDA) associated with both TDKENO and TDKENO-M. Also, 36 spatial mesh intervals are used for modeling Benchmark Problem 16-A7 as described in Appendix D.

Results show that TDKENO-M is quite accurate in comparison with Benchmark Problem 16-A7 (approximately 3.1% error in the power trace) with as much as a factor of 95 reduction in the CPU time relative to TDKENO. Note that the results presented were obtained by performing flux shape calculations at 0.0, $5x10^{-5}$, $1x10^{-4}$, 10^{-2}, $1.1x10^{-2}$, $2x10^{-2}$, and 10^{-1} seconds using 36 mesh intervals as described in Appendix D. Further, ten intermediate calculations of the point kinetics parameters are performed between each flux shape calculation and ten flux amplitude calculations are performed between each calculation of the point kinetics parameters.

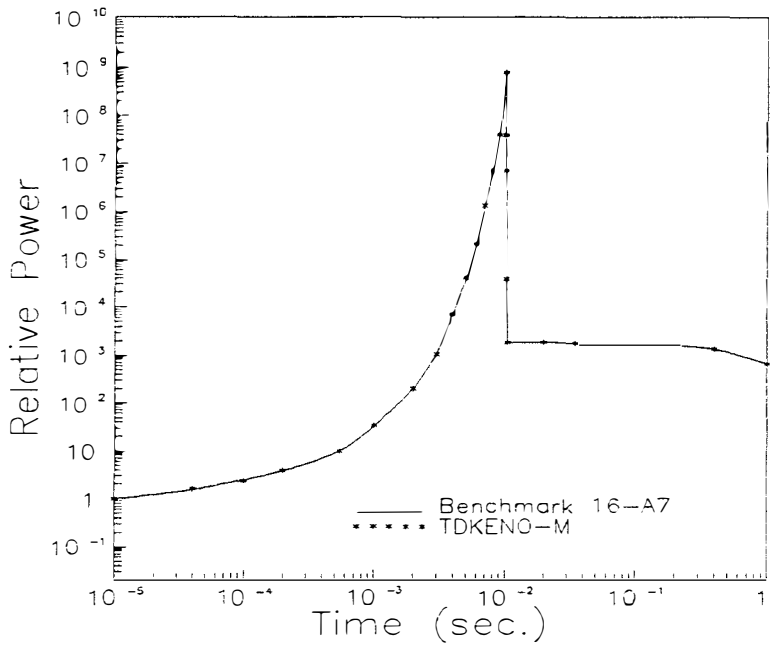

Figure 10: Relative Power vs Time for Benchmark Problem 16-A7

Table 9: Results for Benchmark Problem 16-A7

Fraction of Original Histories	CPU Time* (min.)	Average Shape* Deviation (%)	Average Power Deviation (%) TDKENO	Average Power Deviation (%) TDKENO-M
1	398	0.64	0.66	0.43
1/2	200	1.41	4.80	0.55
1/4	101	1.96	9.25	0.86
1/10	40.5	2.35	97.3	1.29
1/100	4.21	5.53	500.2	3.14

*Values of CPU Time and Average Flux Deviation are for both TDKENO and TDKENO-M since these parameters depend primarily on the total number of histories and not the solution methodology.

5.7 Benchmark Problem 14-A2

Benchmark Problem 14-A2 is a three-dimensional model of a Boiling Water Reactor (BWR) in which the initiating perturbation is the complete withdrawal of a control rod resulting in a super-prompt transient. Refer to Figures 11 and 12 for a schematic representation of the problem as well as material/region assignments. Note that the region denoted by R is the location of the control rod to be withdrawn. Two neutron energy groups and two delayed neutron groups are used for Benchmark Problem 14-A2 (Tables 10 and 11). Finally, the problem includes a feedback model with adiabatic heatup and Doppler feedback as described in Appendix C.

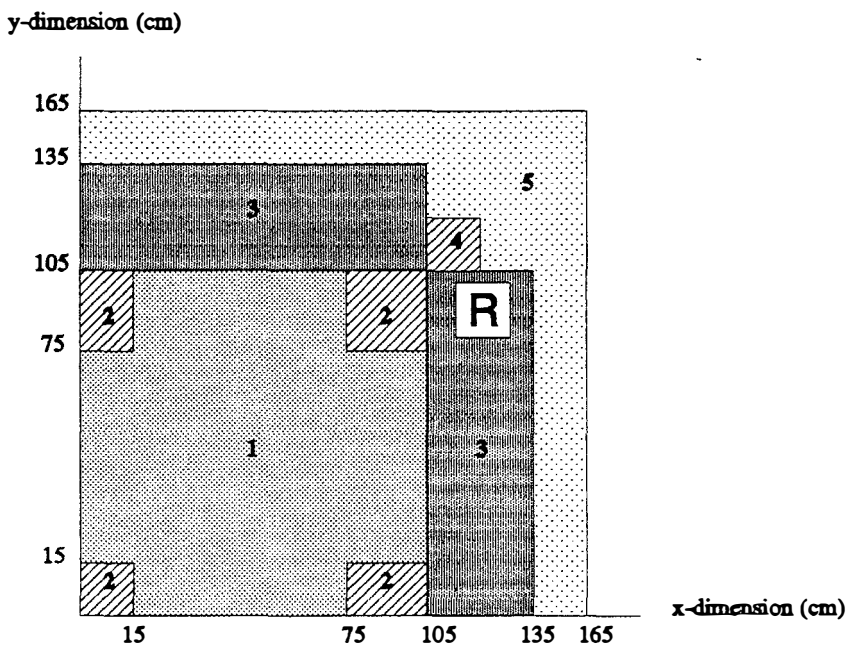

Figure 11: Quadrant of Reactor Horizontal Cross Section for Benchmark 14-A2

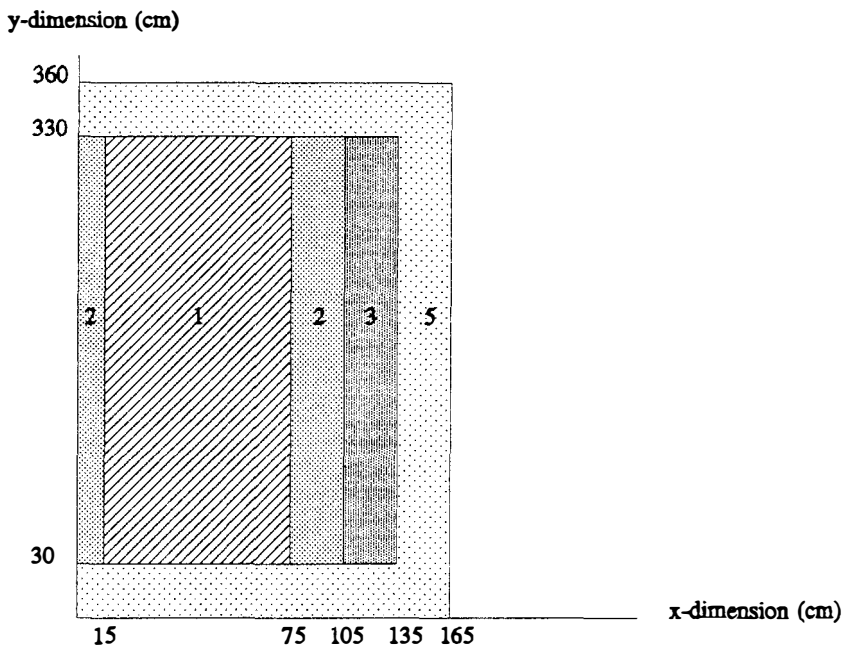

Figure 12: Reactor Vertical Cross Section for Benchmark 14-A2

Table 10: Benchmark Problem 14-A2 Delayed Neutron Parameters[a]

Delayed Neutron Group	Delayed Neutron Fraction, β	Decay Constant, λ
1	5.40×10^{-3}	6.54×10^{-3}
2	1.087×10^{-3}	1.35

[a]Prompt and delayed neutron spectra are identical with $\chi_1 = 1.0$ and $\chi_2 = 0.0$. Also, $1/v_1 = 3.333 \times 10^{-6}$ sec/cm and $1/v_2 = 3.333 \times 10^{-4}$ sec/cm.

Table 11: Benchmark Problem 14-A2 Initial Two-Group Data

Zone	Material	Group	D_i	Σ_a^i	$\nu\Sigma_f^i$	$\Sigma_s^{1\to2}$
1	Fuel 1 with rod	1 2	1.255 0.211	8.252×10^{-3} 1.003×10^{-1}	4.602×10^{-3} 1.091×10^{-1}	2.533×10^{-2}
2	Fuel 2 without rod	1 2	1.268 0.1902	7.181×10^{-3} 7.047×10^{-2}	4.609×10^{-3} 8.675×10^{-2}	2.767×10^{-2}
3	Fuel 2 with rod	1 2	1.259 0.2091	8.002×10^{-3} 8.344×10^{-2}	4.663×10^{-3} 1.021×10^{-1}	2.617×10^{-2}
4	Fuel 2 without rod	1 2	1.259 0.2091	8.002×10^{-3} 7.332×10^{-2}	4.663×10^{-3} 1.021×10^{-1}	2.617×10^{-2}
5	Reflector	1 2	1.257 0.1592	6.034×10^{-4} 1.911×10^{-2}	0.0 0.0	4.754×10^{-2}

Problem 14-A2 was designed for testing diffusion codes; therefore, diffusion coefficients are given rather than total cross sections. The cross sections are obtained from the diffusion coefficients using the following equation:[22]

$$D = \frac{1}{3\Sigma_s(1-\overline{\cos\psi})}$$

where,

D = diffusion coefficient,
Σ_s = scattering cross section,
ψ = scattering angle in the laboratory system.

Isotropic scattering in the laboratory system is assumed which causes the average cosine term to vanish. The total cross section is then obtained from the addition of the scattering and absorption cross sections.

Results obtained with TDKENO-M as well as IQSBOX[48] for Benchmark Problem 14-A2 are given in Figures 13 and 14 and Table 12. Figure 13 is a plot of normalized power versus time (power trace). Figure 14 is a plot of the average core temperature versus time. Table 12 is a tabular representation of the CPU time and the error (as compared to IQSBOX) associated with both TDKENO and TDKENO-M due to a reduction of the total number of histories for each flux shape calculation.

Results show that TDKENO-M provides adequate accuracy in comparison with Benchmark Problem 14-A2 (less than 6.5% error in power trace) with as much as a factor of 9.4 reduction in CPU time relative to TDKENO. Note that the results presented were obtained by performing 30 flux shape calculations. Further, ten intermediate calculations of the point kinetics parameters are performed between each flux shape calculation and ten flux amplitude calculations are performed between each calculation of the point kinetics parameters. Also, 217 spatial mesh intervals are used for both the neutronic and thermalhydraulic modeling of Benchmark Problem 14-A2 as described in Appendix D.

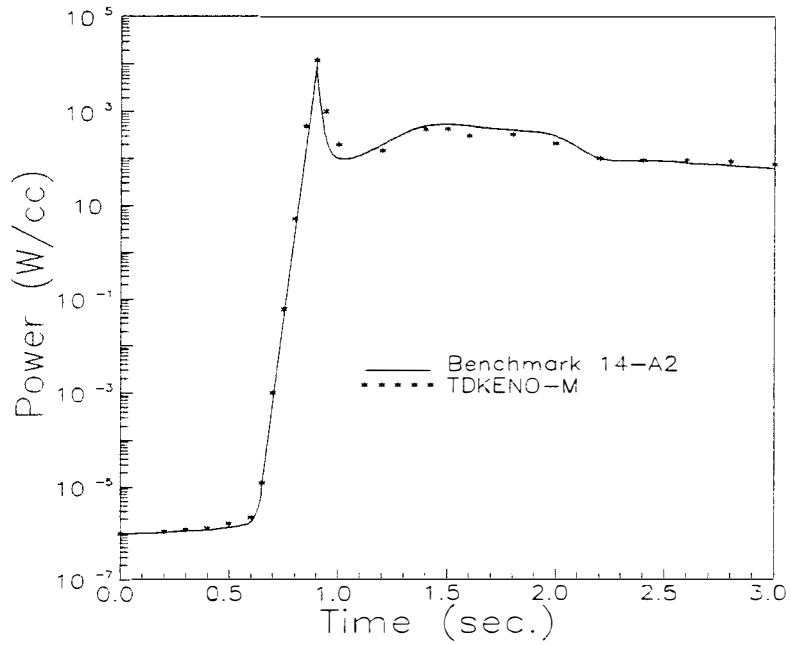

Figure 13: Power vs Time for Benchmark Problem 14-A2

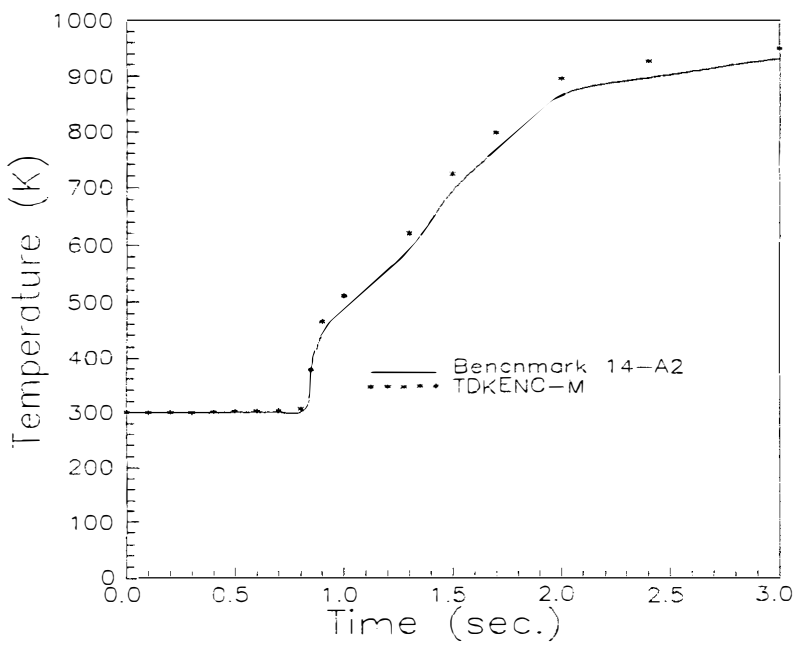

Figure 14: Average Temperature vs Time for Benchmark Problem 14-A2

Table 12: Results for Benchmark Problem 14-A2

Fraction of Original Histories	CPU Time* (min.)	Average Shape* Deviation (%)	Average Power Deviation (%) TDKENO	Average Power Deviation (%) TDKENO-M
1	2339	0.97	29.5	1.64
1/2	1198	2.19	105	1.83
1/4	605	5.45	133	2.20
1/10	249	10.5	285	6.38

*Values of CPU Time and Average Flux Deviation are for both TDKENO and TDKENO-M since these parameters depend primarily on the total number of histories and not the solution methodology.

Chapter Six

Sensitivity Studies

6.1 Introduction

Several of the user-specified input parameters used by TDKENO-M are somewhat arbitrary. Therefore, the sensitivity which these parameters have on the results is investigated in this chapter. Parameters considered in this investigation include 1.) the total number of histories per shape calculation, 2.) the number of shape calculations, and 3.) the number of reactivity and point kinetics time steps. Note that all sensitivity studies presented in this chapter are for Benchmark Problem 16-A1.

6.2 Sensitivity to Total Histories

As described in Chapter 5, as the number of histories decreases, the accuracy of TDKENO-M is much better than TDKENO. This is because TDKENO-M computes reactivity and generation time deterministically rather than using the stochastic method (i.e., boundary crossing estimation) used by TDKENO which requires a huge number of histories to obtain acceptable uncertainties. Table 3 (page 29) is a tabular representation of the CPU time and the error (as compared to TIMEX and TDA) associated with both TDKENO and TDKENO-M due to a reduction of the total number of histories for each flux shape calculation.

Results show that the average power deviation as calculated by TDKENO-M is approximately 2% error in comparison with Benchmark Problem 16-A1 with as much as a factor of 400 reduction in CPU time relative to TDKENO.

6.3 Sensitivity to Number of Shape Calculations

The effect of a reduction in the total number of shape calculations is investigated in this section. Since TDKENO-M determines the point kinetics parameters using their inner product definitions, they can be calculated at times intermediate to the flux shape calculations. The ability to calculate point kinetics parameters at intermediate times should reduce the number of shape calculations for many transients of interest.

Figure 15 is a plot of normalized power versus time for Problem 16-A1 without a shape calculation at 10^{-5} seconds. Results of this calculation show that the power trace calculated by TDKENO-M differs from the benchmark results by approximately 3%. Figure 16 is a power trace for Benchmark Problem 16-A1 without a shape calculation at 10^{-2} seconds. Results of this calculation show that the power trace calculated by TDKENO-M differs from the benchmark results by approximately 26%. The large error for this calculation can be attributed to the errors produced by extrapolating the flux shape from 10^{-5} seconds to 10.0 seconds. Thus, the user should be careful to ensure that the total number of shape calculations as well as the spacing of the flux shape calculations is appropriate for the problem of interest.

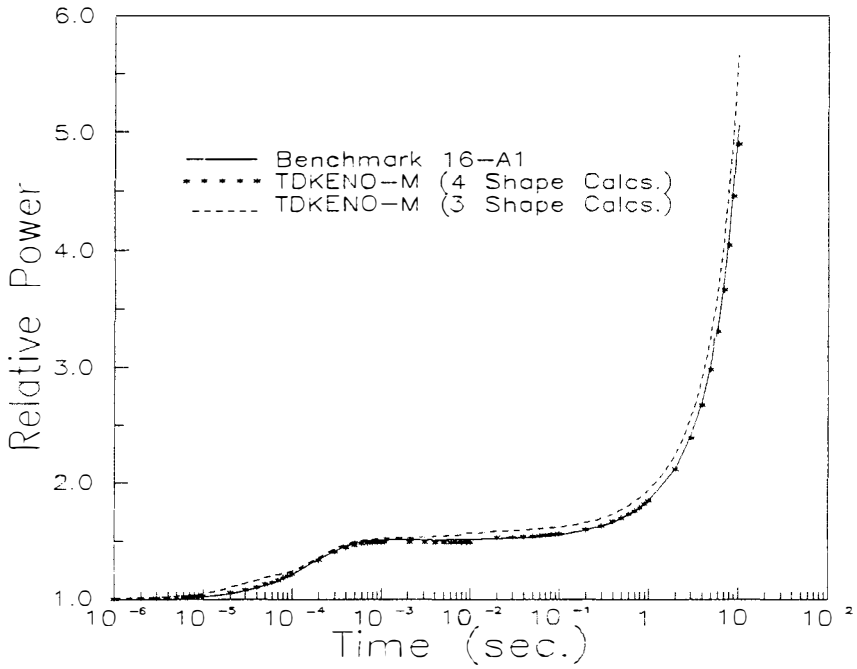

Figure 15: Power Trace for Benchmark Problem 16-A1 without Flux Shape Calculation at 10^{-5} Seconds

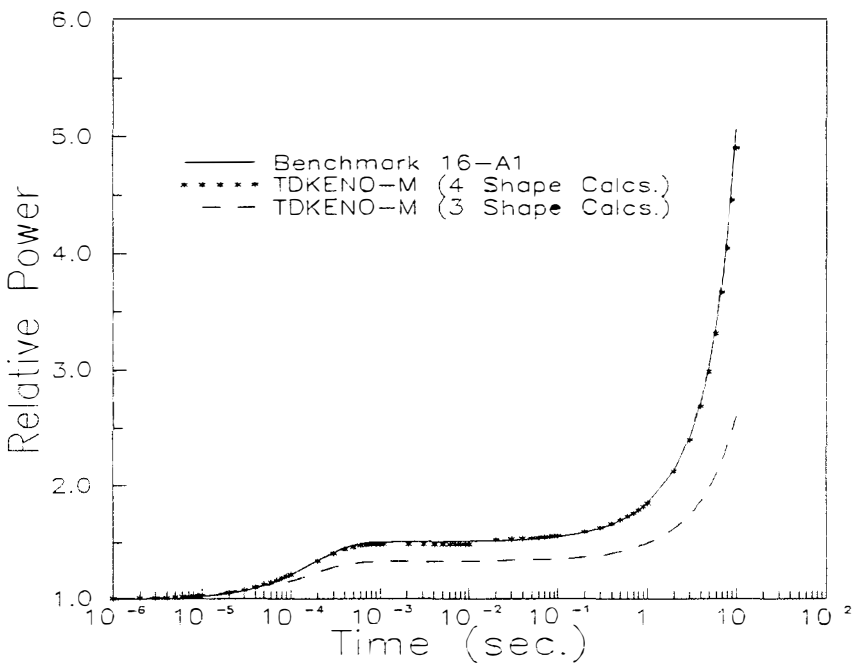

Figure 16: Power Trace for Benchmark Problem 16-A1 without Flux Shape Calculation at 10^{-2} Seconds

6.4 Sensitivity to Time Steps

This section investigates the effect of changing the number of reactivity calculations between shape calculations as well as the number of point kinetics calculations between reactivity calculations. Table 13 is a tabular representation of the number of time steps used between each calculation of flux shape and the point kinetics parameters along with the error associated with each transient calculation.

Results show that the time step spacing is of extreme importance. Particularly, the effect of increasing the number of intermediate reactivity calculations between each shape calculation is of greater importance than increasing the number of amplitude calculations between reactivity calculations. For example, if the number of intermediate amplitude calculations remains fixed at zero while the number of intermediate reactivity calculations is increased from 0 to 100, the average power deviation is reduced from 0.685% to 0.152%. In contrast, if the number of intermediate reactivity calculations remains fixed at 10 while the number of intermediate amplitude calculations is increased from 0 to 100, the average power deviation is only reduced from 0.158 to 0.129.

It should be noted that the number of intermediate calculations of either amplitude or the point kinetics parameters reaches a threshold value after which an increase in the number of intermediate calculations causes only negligible improvement in the average power deviation.

Table 13: Results of Time Step Sensitivity Studies

Number of Intermediate Reactivity Calculations	Number of Intermediate Amplitude Calculations	Average Power Deviation (%)
0	0	0.685
0	10	0.478
0	100	0.452
10	0	0.158
10	10	0.134
10	100	0.129
100	0	0.152
100	10	0.131
100	100	0.129

Chapter Seven

Conclusions and Future Work

7.1 Conclusions

An improved hybrid stochastic/deterministic method for transient, three-dimensional neutron transport has been developed. The method, as incorporated in TDKENO-M, provides a mechanism for benchmarking more approximate and less CPU intensive codes without the prohibitive computing times encountered with TDKENO. Finally, the method has been verified by comparisons with several benchmark problems to ensure that TDKENO-M accurately solves the three-dimensional, time-dependent Boltzmann transport equation with explicit representation of delayed neutrons.

TDKENO-M is verified by applying the code to Benchmark Problems 14-A2, 16-A1, 16-A2, 16-A3, 16-A6, and 16-A7 contained in the ANL-7416 Benchmark Problem Book.[45] Results show that TDKENO-M is quite accurate in comparison with benchmark calculations (less than 2% error in power trace for ANL Benchmark 16-A1) with as much as a factor of 500 reduction in CPU time relative to TDKENO.

In conclusion, the transport code as well as the methodology developed in this work is an extremely important new tool for the transient simulation of neutronic systems. Thus, the user has the ability to model the three-dimensional transient behavior of fissile systems using an exact transport treatment.

7.2 Future Work

The primary area where future work is recommended is with respect to the thermalhydraulic feedback models which are utilized for the systems being modeled. The current version of TDKENO-M considers only simple adiabatic heatup with Doppler broadening of the absorption cross sections. Thus, a better mechanism for including thermalhydraulic feedback would be to use deterministic models for conduction, convection, and radiation of heat for the system as well as any other physical phenomena that affects reactivity.

Work is also recommended in the testing and evaluation of TDKENO-M. Particularly, more testing should be performed to identify the types of problems for which TDKENO-M is best (as well as least) suited. It would also be beneficial to apply TDKENO-M to additional three-dimensional benchmark problems (e.g., problem 17-A2) as further evaluation of the neutronic capabilities of TDKENO-M.

References

References

1. A.F. Henry, Nuclear-Reactor Analysis, MIT Press, Cambridge, MA (1975).

2. J.J. Duderstadt and L.J. Hamilton, Nuclear Reactor Analysis, John Wiley & Sons, New York, NY (1976).

3. J.B. Yasinsky and A.F. Henry, "Some Numerical Experiments Concerning Space-Time Reactor Kinetics Behavior," *Nucl. Sci. Eng.*, **22**, 171-181 (1965).

4. G.I. Bell and S. Glasstone, Nuclear Reactor Theory, Van Nostrand, New York (1970).

5. W.M. Stacey, Jr., Modal Approximations: Theory and Application to Reactor Physics, MIT Press, Cambridge, MA (1967).

6. W.M. Stacey, Jr., "Some Numerical Experiments Concerning Space-Time Reactor Kinetics Behavior," *Nucl. Sci. Eng.*, **22**, 171-181 (1965).

7. S. Kaplan, "Synthesis Methods in Reactor Analysis," *Advan. Nucl. Sci. Tech.*, **3**, 233 (1965).

8. S. Kaplan, O.J. Marlowe and J. Bewick, "Application of Synthesis Techniques to Problems Involving Time Dependence," *Nucl. Sci. Eng.*, **18**, 163 (1964).

9. W.M. Stacey, Jr., "Studies of Spectral Synthesis in Spatially Dependent Fast Reactor Dynamics," *Trans. Am. Nucl. Soc.*, **13**, 619 (1970).

10. G. Kessler, "Space-Dependent Dynamic Behavior of Fast Reactors Using the Time-Discontinuous Synthesis Method," *Trans. Am. Nucl. Soc.*, **11** (1968).

11. J.C. Luxat and G.M. Frescura, *Nucl. Tech.*, **46**, 507 (1979).

12. R.P. Jacqmin, "A Semi-Experimental Nodal Synthesis Method for the On-Line Reconstruction of Three-Dimensional Neutron Flux Shapes and Reactivity," Ph.D. Dissertation, Department of Nuclear Engineering, Massachusetts Institute of Technology (1991).

13. E.L. Wachspress, R.D. Burgess and S. Baron, "Multichannel Flux Synthesis," *Nucl. Sci. Eng.*, **12**, 381-389 (1962).

14. S.J. Cage and F.T. Adler, *Trans. Am. Nucl. Soc.*, **8**, 502 (1965).

15. J.C. Gehin, Ph.D. Dissertation, Department of Nuclear Engineering, Massachusetts Institute of Technology (September 1992).

16. R.E. Alcouffe and R.W. Albrecht, "A Nodal Model for Space-Dependent Nuclear Reactor Kinetics," *Trans. Am. Nucl. Soc.*, **11** (1968).

17. Personal communication, H.L. Dodds, July 1996.

18. S. Langenbuch, W. Maurer and W. Werner, "Coarse Mesh Flux-Expansion Method for the Analysis of Space-Time Effects in Large Light Water Reactor Cores," *Nucl. Sci. Eng.*, **63**, 437 (1977).

19. F. Bennewitz, H. Finneman, and M.R. Wagner, "Higher Order Corrections in Nodal Reactor Calculations," *Trans. Am. Nucl. Soc.*, **22**, 250 (1975).

20. G. Greenman, K. Smith and A.F. Henry, "Recent Advances in Analytical Nodal Method for Static and Transient Reactor Analysis," *Proc. Topl. Mtg. Computational Methods in Nucl Eng.*, Williamsburgh, Virginia (1979).

21. J.E. Morel and J.M. McGhee, "A Three-Dimensional Time-Dependent Unstructured Tetrahedral-Mesh SP_N Method," *Nuc. Sci. Eng.*, **123**, 319 (1996).

22. M.R. Buckner and J.W. Stewart, "Multidimensional Space-Time Nuclear-Reactor Kinetics Studies - Part I: Theoretical," *Nucl. Sci. Eng.*, **59**, 289 (1976).

23. W.R. Caldwell, et. al., "WIGLE - A Program for the solution of the Two-Group Space-Time Diffusion Equations in Slab Geometry," WAPD-TM-416 (1964).

24. J.B. Yasinsky, M. Natelson and L.A. Hageman, "TWIGL - A Program to Solve the Two-Dimensional, Two-Group, Space-Time Neutron Diffusion Equations with Temperature Feedback," WAPD-TM-743, Bettis Atomic Power Laboratory (1968).

25. J.W. Stewart, "Transient DISCOTHEQUE Calculation - Module DISC," DPSTM-500, **7**, Savannah River Laboratory (1972).

26. D.R. Fergurson, Ph.D. Dissertation, MIT, Nuclear Engineering Department, EC Report MIT-3903-4, U.S. Atomic Energy Commission (1971).

27. R.W. Bowring, J.W. Stewart, R.A. Shober and R.N. Sims, "MEKIN: MIT-EPRI Nuclear Reactor Core Kinetics Code," CCM-1, Electric Power Research Institute, RP227 (1975).

28. A. Kavenoky, J. Lautard and P. Reuss, "Modeling of Feedback Effects: Development of the CRONOS Code," *Trans. Am. Nucl. Soc.*, **32**, 721 (1979).

29. W. Engle, Jr., et. al., "One-Dimensional Time Dependent Discrete Ordinates," *Trans. Am. Nucl. Soc.*, **12**, 400 (1969).

30. T.R. Hill and W.H. Reed, "TIMEX: A Time-Dependent Explicit Discrete Ordinates Program for the Solution of Multigroup Transport Equations with Delayed Neutrons," LA-6201-MS, Los Alamos Scientific Laboratory (1976).

31. K.D. Lathrop, R.E. Andersen and F.W. Brinkley, "TRANZIT: A Program for Multigroup Time-Dependent Transport in (ρ,z) Cylinder Geometry," LA-4575, Los Alamos Scientific Laboratory (1970).

32. W. Engle, "A User's Manual for ANISN, A One Dimensional Discrete Ordinates Transport Code with Anisotropic Scattering," K-1693, Computing Technology Center, Oak Ridge Gaseous Diffusion Plant (1967).

33. A.F. Henry, *Nucl. Sci. Eng.*, **3**, 52 (1958).

34. H.L. Dodds, "Accuracy of the Quasistatic Method for Two-Dimensional Thermal Reactor Transients with Feedback," *Nucl. Sci. Eng.*, **59**, 271 (1976).

35. K.O. Ott and R.J. Neuhold, Nuclear Reactor Dynamics, ANS Publications, La Grange Park, IL (1985).

36. K.O. Ott and D.A. Meneley, "Accuracy of the Quasistatic Treatment of Spatial Reactor Kinetics," *Nucl. Sci. Eng.*, **36**, 402 (1969).

37. D.A. Meneley, K. Ott, and E.S. Wiener, "Space-Time Kinetics, the QX1 Code," ANL-7310, Argonne National Laboratory (1967).

38. J.T. Madell, "Quasistatic Treatment of Spatial Phenomena in Reactor Dynamics," *Nucl. Sci. Eng.*, **26**, 563 (1966).

39. D.A. Meneley, et. al., "A Kinetics Model for Fast Reactor Analysis in Two Dimensions," Dynamics of Nuclear Systems, The University of Arizona Press, Tucson, AZ (1972).

40. Personal communication, H.L. Dodds, August 1994.

41. G. Kugler and A.R. Dastur, "Accuracy of the Improved Quasistatic Space-Time Method Checked with Experiment," *Trans. Am. Nucl. Soc.*, **23**, 592 (1976).

42. M.W. Waddell, "A Hybrid Stochastic/Deterministic Method for Transient, Three-Dimensional Neutron Transport," *Trans. Am. Nucl. Soc.*, **66**, 226 (1992).

43. M.W. Waddell and H.L. Dodds, "A Method for Transient, Three-Dimensional Neutron Transport Calculations," *Proc. Conf. Mathematical Methods and Supercomputing in Nuclear Applications*, Karlsruhe, Germany, 633 (1193).

44. M.W. Waddell, "Development of a Transient, Three-Dimensional Neutron Transport Code with Feedback," *Trans. Am. Nucl. Soc.*, **70**, 214 (1994).

45. National Energy Software Center: Benchmark Problem Book, Argonne National Laboratory, ANL-7416, Supplement No. 3, (1985).

46. A.F. Henry, *Nucl. Sci. Eng.*, **36**, 402 (1969).

47. A.F. Henry and N.J. Curlee, *Nucl. Sci. Eng.*, **4**, 727 (1958).

48. F. Bennewitz, H. Finneman, and M.R. Wagner, "Higher Order Corrections in Nodal Reactor Calculations," *Trans. Am. Nucl. Soc.*, **22**, 250 (1975).

49. L. Leithold, The Calculus, Harper & Row, New York (1986).

Appendices

Appendix A: Derivation of Improved Quasistatic Equations

The quasistatic approach is a method of treating the spatial dynamics of the time-dependent Boltzmann transport calculation by factoring the flux into an amplitude and a shape function such that the time variation of the shape function is more slowly varying than the time variation of the amplitude function. This factorization of the flux results in splitting the neutronics equation into two coupled equations; an amplitude equation having the form of the point kinetics equation and a shape equation. The improved quasistatic method replaces the time derivative of the shape function by a first order backward difference approximation thus allowing the improved quasistatic method to be applied more accurately.

The development of the improved quasistatic method begins with the time-dependent Boltzmann transport equation with explicit representation of delayed neutrons:

$$\frac{1}{v}\frac{\partial\phi(\bar{r},\bar{\Omega},E,t)}{\partial t} + \bar{\Omega}\cdot\bar{\nabla}\phi + \Sigma_t\phi = \int\int\Sigma_s(\bar{r};\bar{\Omega}',E'\rightarrow\bar{\Omega},E;t)\phi(\bar{r},\bar{\Omega}',E',t)d\Omega'dE' \tag{a.1}$$

$$+ \int\int\chi_p(E)(1-\beta)\nu\Sigma_f(\bar{r},E',t)\phi(\bar{r},\bar{\Omega}',E',t)d\Omega'dE' + \sum_j\lambda_jC_j(\bar{r},t)\chi_j + Q$$

and

$$\frac{\partial C_j(\bar{r},t)}{\partial t} + \lambda_jC_j = \int\int\beta_j\nu\Sigma_f(\bar{r},E',t)\phi(\bar{r},\bar{\Omega}',E',t)d\Omega'dE' \quad for\ j=1,N \tag{a.2}$$

where,

v	=	particle speed,
ϕ	=	time-dependent angular flux,
$\bar{r}$	=	spatial variable,
E	=	energy,
$\bar{\Omega}$	=	unit vector describing particle direction,
t	=	time,
Σ_t	=	total cross section,
Σ_s	=	differential scattering cross section,
Q	=	external source,
χ_p	=	normalized energy spectrum of prompt neutrons,
χ_j	=	normalized energy spectrum for precursor group j,
β	=	delayed neutron fraction,
ν	=	total number of neutrons emitted per fission,
Σ_f	=	fission cross section,
λ_j	=	decay constant for delayed neutron precursor group j,
C_j	=	density of delayed neutron precursors for group j,
N	=	number of precursor groups.

The time-independent adjoint form of the transport equation is given by:

$$-\bar{\Omega}\cdot\bar{\nabla}\phi^* + \Sigma_t\phi^* = \int\int \Sigma_s(\bar{r};\bar{\Omega},E\rightarrow\bar{\Omega}',E')\phi^*(\bar{r},\bar{\Omega}',E')d\Omega'dE' + \int\int \chi(E')\nu\Sigma_f(\bar{r},E)\phi^*(\bar{r},\bar{\Omega}',E')d\Omega'dE' \quad \textbf{(a.3)}$$

where,

ϕ^*	=	adjoint flux.

Next, the flux is factored into an amplitude function, P(t), and a shape function, Ψ, such that the flux is given by:

$$\phi(\bar{r},\bar{\Omega},E,t) = P(t)\psi(\bar{r},\bar{\Omega},E,t) \quad \textbf{(a.4)}$$

This factorization requires an additional separation condition for $t > 0$ in order to become a unique definition.[35] There are many conditions that could be used, all of which involve constraining some space-energy integral of $\Psi(r,E,t)$ to a constant value for all $t > 0$. The only basic requirement is that $\Psi(r,E,t)$ remain positive and bounded at all points in space (r,E) for all time. According to Henry, the following equation fulfills the requirements for the constraint condition and has the added advantage that it facilitates the transition to the point kinetics formulation:[45,46]

$$\iiint \frac{1}{v}\psi(\bar{r},\bar{\Omega},E,t)\phi^*(\bar{r},\bar{\Omega},E)d\Omega dE dV = constant \qquad \textbf{(a.5)}$$

Substituting the separability assumption given in equation a.4 into the time-dependent Boltzmann transport equation and multiplying by ϕ^* yields the following equation:

$$\begin{aligned}
&\frac{1}{v}\frac{\partial P(t)\psi(\bar{r},\bar{\Omega},E,t)}{\partial t}\phi^* + \bar{\Omega}\cdot\bar{\nabla}[P(t)\psi(\bar{r},\bar{\Omega},E,t)]\phi^* \\
&\quad + \Sigma_t(\bar{r},E,t)P(t)\psi(\bar{r},\bar{\Omega},E,t)\phi^* \\
&= \phi^*\int\int \Sigma_s(\bar{r};\bar{\Omega}',E'\rightarrow\bar{\Omega},E;t)P(t)\psi(\bar{r},\bar{\Omega}',E',t)d\Omega' dE' \qquad \textbf{(a.6)} \\
&+ \phi^*\int\int \chi_p(E)(1-\beta)\nu\Sigma_f(\bar{r},E',t)P(t)\psi(\bar{r},\bar{\Omega}',E',t)d\Omega' dE' \\
&\quad + \phi^*\sum_j \lambda_j C_j(\bar{r},t)\chi_j + \phi^* Q
\end{aligned}$$

Next, the time-independent adjoint form of the transport equation (equation a.3) is multiplied by ϕ and the separability assumption (equation a.4) is again used to yield the following:

$$
\begin{aligned}
&-(\bar{\Omega}\cdot\bar{\nabla}\phi^*)P(t)\psi(\bar{r},\bar{\Omega},E,t) \;+\; \Sigma_t(\bar{r},E)\phi^*P(t)\psi(\bar{r},\bar{\Omega},E,t) \\
&= P(t)\psi(\bar{r},\bar{\Omega},E,t)\int\int\Sigma_s(\bar{r};\bar{\Omega},E\rightarrow\bar{\Omega}',E')\phi^*d\Omega'dE' \\
&+ P(t)\psi(\bar{r},\bar{\Omega},E,t)\int\int\chi(E')\nu\Sigma_f(\bar{r},E)\phi^*d\Omega'dE'
\end{aligned}
\qquad \textbf{(a.7)}
$$

The flux-weighted adjoint equation (equation a.7) is subtracted from the adjoint-weighted forward equation (equation a.6) to give:

$$
\begin{aligned}
&\frac{1}{\nu}\frac{\partial P(t)\psi(\bar{r},\bar{\Omega},E,t)}{\partial t}\phi^* \;+\; \bar{\Omega}\cdot\bar{\nabla}[P(t)\psi(\bar{r},\bar{\Omega},E,t)]\phi^* \\
&+ (\bar{\Omega}\cdot\bar{\nabla}\phi^*)P(t)\psi(\bar{r},\bar{\Omega},E,t) \\
&+ \Sigma_t(\bar{r},E,t)P(t)\psi(\bar{r},\bar{\Omega},E,t)\phi^* \;-\; \Sigma_t(\bar{r},E,t)\phi^*P(t)\psi(\bar{r},\bar{\Omega},E,t) \\
&= \phi^*\int\int\Sigma_s(\bar{r};\bar{\Omega}',E'\rightarrow\bar{\Omega},E;t)P(t)\psi(\bar{r},\bar{\Omega}',E',t)d\Omega'dE' \\
&- P(t)\psi(\bar{r},\bar{\Omega},E,t)\int\int\Sigma_s(\bar{r};\bar{\Omega},E\rightarrow\bar{\Omega}',E')\phi^*(\bar{r},\bar{\Omega}',E')d\Omega'dE' \\
&+ \phi^*\int\int\chi_p(E)(1-\beta)\nu\Sigma_f(\bar{r},E',t)P(t)\psi(\bar{r},\bar{\Omega}',E',t)d\Omega'dE' \\
&- P(t)\psi(\bar{r},\bar{\Omega},E,t)\int\int\chi(E')\nu\Sigma_f(\bar{r},E)\phi^*(\bar{r},\bar{\Omega}',E')d\Omega'dE' \\
&+ \phi^*\sum_j\lambda_jC_j(\bar{r},t)\chi_j \;+\; \phi^*Q
\end{aligned}
\qquad \textbf{(a.8)}
$$

Integration of this equation over space, energy and angle gives:

$$\iiint \frac{1}{v}\frac{\partial P(t)\psi(\bar{r},\bar{\Omega},E,t)}{\partial t}\phi^*(\bar{r},\bar{\Omega},E)d\Omega dE dV$$

$$+ \iiint \bar{\Omega}\cdot\bar{\nabla}P(t)\psi(\bar{r},\bar{\Omega},E,t)\phi^*(\bar{r},\bar{\Omega},E)d\Omega dE dV$$

$$+ \iiint \bar{\Omega}\cdot\bar{\nabla}\phi^*(\bar{r},\bar{\Omega},E)P(t)\psi(\bar{r},\bar{\Omega},E,t)d\Omega dE dV$$

$$+ \iiint (\Sigma_t(\bar{r},E,t)-\Sigma_t(\bar{r},E))P(t)\psi(\bar{r},\bar{\Omega},E,t)\phi^*(\bar{r},\bar{\Omega},E)d\Omega dE dV$$

$$= \iiint \phi^*(\bar{r},\bar{\Omega},E)\iint \Sigma_s(\bar{r};\bar{\Omega}',E'\rightarrow\bar{\Omega},E;t)P(t)\psi(\bar{r},\bar{\Omega}',E',t)d\Omega' dE' d\Omega dE dV \qquad \textbf{(a.9)}$$

$$- \iiint P(t)\psi(\bar{r},\bar{\Omega},E,t)\iint \Sigma_s(\bar{r},\bar{\Omega},E\rightarrow\bar{\Omega}',E')\phi^*(\bar{r},\bar{\Omega}',E')d\Omega' dE' d\Omega dE dV$$

$$+ \iiint \phi^*(\bar{r},\bar{\Omega},E)\iint \chi_p(E)(1-\beta)\nu\Sigma_f(\bar{r},E',t)P(t)\psi(\bar{r},\bar{\Omega}',E',t)d\Omega' dE' d\Omega dE dV$$

$$- \iiint P(t)\psi(\bar{r},\bar{\Omega},E,t)\iint \chi(E')\nu\Sigma_f(\bar{r},E)\phi^*(\bar{r},\bar{\Omega}',E')d\Omega' dE' d\Omega dE dV$$

$$+ \iiint \phi^*(\bar{r},\bar{\Omega},E)\sum_j \lambda_j C_j(\bar{r},t)\chi_j d\Omega dE dV + \iiint \phi^*(\bar{r},\bar{\Omega},E)Q d\Omega dE dV$$

The first term in equation a.9 can be rewritten as follows:

$$\iiint \frac{1}{v}\frac{\partial P(t)\psi(\bar{r},\bar{\Omega},E,t)}{\partial t}\phi^*(\bar{r},\bar{\Omega},E)d\Omega dE dV$$

$$= P(t)\frac{\partial}{\partial t}\iiint \frac{1}{v}\psi(\bar{r},\bar{\Omega},E,t)\phi^*(\bar{r},\bar{\Omega},E)d\Omega dE dV \qquad \textbf{(a.10)}$$

$$+ \iiint \frac{1}{v}\psi(\bar{r},\bar{\Omega},E,t)\frac{\partial P(t)}{\partial t}\phi^*(\bar{r},\bar{\Omega},E)d\Omega dE dV$$

Using the space-energy integral constraint given in equation a.5 it is obvious that the first term on the right hand side of equation a.10 vanishes. Thus, the first term in equation a.9 can be written as:

$$\iiint \frac{1}{v}\frac{\partial P(t)\psi(\bar{r},\bar{\Omega},E,t)}{\partial t}\phi^*(\bar{r},\bar{\Omega},E)d\Omega dE dV$$

$$= \frac{\partial P(t)}{\partial t}\iiint \frac{1}{v}\psi(\bar{r},\bar{\Omega},E,t)\phi^*(\bar{r},\bar{\Omega},E)d\Omega dE dV \qquad \textbf{(a.11)}$$

Since the gradient in the second and third terms of equation a.9 is invariant with respect to the angular variable, those two terms can be respectively simplified as follows:

$$\iiint \bar{\Omega}\cdot\bar{\nabla}\left(P(t)\psi(\bar{r},\bar{\Omega},E,t)\right)\phi^*(\bar{r},\bar{\Omega},E)d\Omega dE dV$$

$$= \iiint \bar{\nabla}\cdot\bar{\Omega}\left(P(t)\psi(\bar{r},\bar{\Omega},E,t)\right)\phi^*(\bar{r},\bar{\Omega},E)d\Omega dE dV \qquad \textbf{(a.12)}$$

and

$$\iiint \bar{\Omega}\cdot\bar{\nabla}\phi^*(\bar{r},\bar{\Omega},E)P(t)\psi(\bar{r},\bar{\Omega},E,t)d\Omega dE dV$$

$$= \iiint \bar{\nabla}\cdot\bar{\Omega}\phi^*(\bar{r},\bar{\Omega},E)P(t)\psi(\bar{r},\bar{\Omega},E,t)d\Omega dE dV \qquad \textbf{(a.13)}$$

Next, the divergence theorem is used to change these volume integrals (equations a.12 and a.13) into surface integrals as follows:[48]

$$\iiint_V \bar{\nabla}\cdot\bar{\Omega}\left(P(t)\psi(\bar{r},\bar{\Omega},E,t)\right)\phi^*(\bar{r},\bar{\Omega},E)\,d\Omega\,dE\,dV$$

$$= \iiint_S \bar{n}\cdot\bar{\Omega}\left(P(t)\psi(\bar{r},\bar{\Omega},E,t)\right)\phi^*(\bar{r},\bar{\Omega},E)\,d\Omega\,dE\,dV \qquad \textbf{(a.14)}$$

and

$$\iiint_V \bar{\nabla}\cdot\bar{\Omega}\,\phi^*(\bar{r},\bar{\Omega},E)P(t)\psi(\bar{r},\bar{\Omega},E,t)\,d\Omega\,dE\,dV$$

$$= \iiint_S \bar{n}\cdot\bar{\Omega}\,\phi^*(\bar{r},\bar{\Omega},E)P(t)\psi(\bar{r},\bar{\Omega},E,t)\,d\Omega\,dE\,dV \qquad \textbf{(a.15)}$$

If the boundary conditions for Ψ and ϕ^* are assumed such that $\Psi\phi^*$ is zero on the surface, which is the usual assumption, then the second and third terms of equation a.9 vanish.

The first two terms on the right hand side of equation a.9 can be rearranged by interchanging the dummy variables of integration and combined to be rewritten as:[48]

$$\iiint \phi^*(\bar{r},\bar{\Omega},E)\iint \Sigma_s(\bar{r};\bar{\Omega}',E'\rightarrow\bar{\Omega},E;t)P(t)\psi(\bar{r},\bar{\Omega}',E',t)d\Omega' dE' d\Omega dE dV$$

$$- \iiint P(t)\psi(\bar{r},\bar{\Omega},E,t)\iint \Sigma_s(\bar{r};\bar{\Omega},E\rightarrow\bar{\Omega}',E')\phi^*(\bar{r},\bar{\Omega}',E')d\Omega' dE' d\Omega dE dV \qquad \textbf{(a.16)}$$

$$= P(t)\iiint\iint \phi^*(\bar{r},\bar{\Omega},E)\left(\Sigma_s(\bar{r};\bar{\Omega}',E'\rightarrow\bar{\Omega},E;t) - \Sigma_s(\bar{r};\bar{\Omega}',E'\rightarrow\bar{\Omega},E)\right)$$

$$\times\ \psi(\bar{r},\bar{\Omega}',E',t)d\Omega' dE' d\Omega dE dV$$

If the assumption is now made that the energy spectrum of prompt neutrons is identical to the energy spectrum of total neutrons (i.e., $\chi = \chi_p$), the two fission terms on the right hand side of equation a.9 (minus the β term) can be combined and rewritten as:

$$\iiint \phi^*(\bar{r},\bar{\Omega},E)\iint \chi_p(E)\nu\Sigma_f(\bar{r},E',t)P(t)\psi(\bar{r},\bar{\Omega}',E',t)d\Omega' dE' d\Omega dE dV$$

$$- \iiint P(t)\psi(\bar{r},\bar{\Omega},E,t)\iint \chi(E')\nu\Sigma_f(\bar{r},E)\phi^*(\bar{r},\bar{\Omega}',E')d\Omega' dE' d\Omega dE dV \qquad \textbf{(a.17)}$$

$$= P(t)\iiint\iint \phi^*(\bar{r},\bar{\Omega},E)\chi_p(E)\left(\nu\Sigma_f(\bar{r},E',t) - \nu\Sigma_f(\bar{r},E')\right)$$

$$\times\ \psi(\bar{r},\bar{\Omega}',E',t)d\Omega' dE' d\Omega dE dV$$

The fission term containing β on the right hand side of equation a.9 can also be written as:

$$- \iiint \phi^*(\bar{r},\bar{\Omega},E) \iint \chi_p(E)\beta\nu\Sigma_f(\bar{r},E',t)P(t)\psi(\bar{r},\bar{\Omega}',E',t)d\Omega' dE' d\Omega dE dV$$

$$= -P(t)\iiint\iiint \phi^*(\bar{r},\bar{\Omega},E)\chi_p(E)\left(\sum_j \beta_j\right)\nu\Sigma_f(\bar{r},E',t) \qquad \textbf{(a.18)}$$

$$\times \psi(\bar{r},\bar{\Omega}',E',t)d\Omega' dE' d\Omega dE dV$$

Substituting equations a.11, a.14, a.15, a.16, a.17, and a.18 into equation a.9 yields the following equation:

$$\frac{\partial P(t)}{\partial t}\iiint \frac{1}{\nu}\psi(\bar{r},\bar{\Omega},E,t)\phi^*(\bar{r},\bar{\Omega},E)d\Omega dE dV$$

$$= -\iiint\left(\Sigma_t(\bar{r},E,t)-\Sigma_t(\bar{r},E)\right)P(t)\psi(\bar{r},\bar{\Omega},E,t)\phi^*(\bar{r},\bar{\Omega},E)d\Omega dE dV$$

$$+ P(t)\iiint\iiint \phi^*(\bar{r},\bar{\Omega},E)\left(\Sigma_s(\bar{r};\bar{\Omega}',E'\rightarrow\bar{\Omega},E;t)-\Sigma_s(\bar{r};\bar{\Omega}',E'\rightarrow\bar{\Omega},E)\right)$$

$$\times \psi(\bar{r},\bar{\Omega}',E',t)d\Omega' dE' d\Omega dE dV$$

$$+ P(t)\iiint\iiint \phi^*(\bar{r},\bar{\Omega},E)\chi_p(E)\left(\nu\Sigma_f(\bar{r},E',t)-\nu\Sigma_f(\bar{r},E')\right) \qquad \textbf{(a.19)}$$

$$\times \psi(\bar{r},\bar{\Omega}',E',t)d\Omega' dE' d\Omega dE dV$$

$$- P(t)\iiint\iiint \phi^*(\bar{r},\bar{\Omega},E)\chi_p(E)\left(\sum_j \beta_j\right)\nu\Sigma_f(\bar{r},E',t)$$

$$\times \psi(\bar{r},\bar{\Omega}',E',t)d\Omega' dE' d\Omega dE dV$$

$$+ \iiint \phi^*(\bar{r},\bar{\Omega},E)\sum_j \lambda_j C_j(\bar{r},t)\chi_j d\Omega dE dV + \iiint \phi^*(\bar{r},\bar{\Omega},E)Q d\Omega dE dV$$

Equation a.19 can be greatly simplified by introducing the following notation:

$$A = \iiint \frac{1}{v} \psi(\bar{r},\bar{\Omega},E,t)\phi^*(\bar{r},\bar{\Omega},E)d\Omega dE dV \tag{a.20}$$

$$\begin{aligned} B = & -\iiint \left(\Sigma_t(\bar{r},E,t) - \Sigma_t(\bar{r},E)\right)\psi(\bar{r},\bar{\Omega},E,t)\phi^*(\bar{r},\bar{\Omega},E)d\Omega dE dV \\ & + \iiiint\int \phi^*(\bar{r},\bar{\Omega},E)\left[\left(\Sigma_s(\bar{r};\bar{\Omega}',E'\rightarrow\bar{\Omega},E;t) - \Sigma_s(\bar{r};\bar{\Omega}',E'\rightarrow\bar{\Omega},E)\right)\right. \\ & \left. + \chi_p(E)\left(\nu\Sigma_f(\bar{r},E',t) - \nu\Sigma_f(\bar{r},E')\right)\right]\psi(\bar{r},\bar{\Omega}',E',t)d\Omega' dE' d\Omega dE dV \end{aligned} \tag{a.21}$$

$$\begin{aligned} C = & \iiiint\int \phi^*(\bar{r},\bar{\Omega},E)\chi_p(E)\left(\sum_j \beta_j\right)\nu\Sigma_f(\bar{r},E',t) \\ & \times \psi(\bar{r},\bar{\Omega}',E',t)d\Omega' dE' d\Omega dE dV \end{aligned} \tag{a.22}$$

Using equations a.20, a.21, and a.22, equation a.19 can be written as:

$$\begin{aligned} \frac{\partial P(t)}{\partial t} = & \frac{B - C}{A}P(t) + \frac{1}{A}\iiint \phi^*(\bar{r},\bar{\Omega},E)\left(\sum_j \lambda_j C_j(\bar{r},t)\chi_j\right)d\Omega dE dV \\ & + \frac{1}{A}\iiint \phi^*(\bar{r},\bar{\Omega},E)Q d\Omega dE dV \end{aligned} \tag{a.23}$$

Next, an arbitrary normalization factor, F(t), is introduced so that the various parameters in the point kinetics equation have physical interpretations. Note that this normalization factor has no effect on the solution of equation a.23 since it always cancels in the numerator and the denominator of each term. Bell & Glasstone suggest that the most appropriate normalization factor is:[4]

$$F(t) = \int\int\int\int\int \chi(E)\nu\Sigma_f(\bar{r},E')\psi(\bar{r},\bar{\Omega}',E',t)\phi_0^*(\bar{r},\bar{\Omega},E)d\Omega' dE' d\Omega dE dV \qquad \textbf{(a.24)}$$

where,

ϕ_0^* = steady-state adjoint flux.

Using this normalization factor, equation a.23 can be rewritten as:

$$\frac{dP(t)}{dt} = \frac{\rho(t)-\bar{\beta}(t)}{\Lambda(t)}P(t) + \sum_j \lambda_j C_j(t) + \bar{Q}(t) \qquad \textbf{(a.25)}$$

with the following parameter definitions:

$$\rho(t) = -\frac{1}{F(t)}\int\int\int\left(\Sigma_t(\bar{r},E,t)-\Sigma_t(\bar{r},E)\right)\psi(\bar{r},\bar{\Omega},E,t)\phi^*(\bar{r},\bar{\Omega},E)d\Omega dE dV$$

$$+\frac{1}{F(t)}\int\int\int\int\int\phi^*(\bar{r},\bar{\Omega},E)[\left(\Sigma_s(\bar{r},\bar{\Omega}',E'\rightarrow\bar{\Omega},E;t)-\Sigma_s(\bar{r},\bar{\Omega}',E'\rightarrow\bar{\Omega},E)\right) \qquad \textbf{(a.26)}$$

$$+\chi_p(\ )\left(\nu\Sigma_f(\bar{r},E',t)-\nu\Sigma_f(\bar{r},E')\right)]\psi(\bar{r},\bar{\Omega}',\ ',t)d\Omega' dE' d\Omega dE dV$$

$$\bar{\beta}(t) = \sum_j \bar{\beta}_j(t) \tag{a.27}$$

$$\bar{\beta}_j(t) = \frac{1}{F(t)} \int\int\int\int\int \chi(E)\beta_j \nu \Sigma_f(\bar{r},E')\psi(\bar{r},\bar{\Omega}',E',t) \times \phi_0^*(\bar{r},\bar{\Omega},E)d\Omega' dE' d\Omega dE dV \tag{a.28}$$

$$\Lambda(t) = \frac{1}{F(t)} \int\int\int \frac{1}{v} \psi(\bar{r},\bar{\Omega},E,t)\phi^*(\bar{r},\bar{\Omega},E)d\Omega dE dV \tag{a.29}$$

$$C_j(t) = \frac{(1/F(t))}{\Lambda} \int\int\int \phi^*(\bar{r},\bar{\Omega},E)C_j(\bar{r},t)\chi_j(E)d\Omega dE dV \tag{a.30}$$

$$\bar{Q}(t) = \frac{(1/F(t))}{\Lambda} \int\int\int \phi^*(\bar{r},\bar{\Omega},E)Q(\bar{r},\bar{\Omega},E)d\Omega dE dV \tag{a.31}$$

The delayed neutron precursor equations are obtained by multiplying equation a.2 by $\chi_j(E)\phi^*$ and using the separability condition given in equation a.4 to yield the following equation for each precursor group:

$$\frac{\partial\left(\chi_j(E)\phi^*(\bar{r},\bar{\Omega},E)C_j(\bar{r},t)\right)}{\partial t} + \lambda_j\chi_j(E)\phi^*(\bar{r},\bar{\Omega},E)C_j(\bar{r},t) = \int\int \chi_j(E)\beta_j \nu \Sigma_f(\bar{r},E',t)P(t)\psi(\bar{r},\bar{\Omega},E,t)\phi^*(\bar{r},\bar{\Omega},E)d\Omega' dE' \tag{a.32}$$

Integrating equation a.32 over space, energy and angle yields the following equation:

$$\iiint \frac{\partial\left(\chi_j(E)\phi^*(\bar{r},\bar{\Omega},E)C_j(\bar{r},t)\right)}{\partial t} d\Omega dE dV$$

$$+ \iiint \lambda_j \chi_j(E)\phi^*(\bar{r},\bar{\Omega},E)C_j(\bar{r},t) d\Omega dE dV \qquad \textbf{(a.33)}$$

$$= \iiiint \chi_j(E)\beta_j \nu \Sigma_f(\bar{r},E',t)P(t)\psi(\bar{r},\bar{\Omega},E,t)$$

$$\times \phi^*(\bar{r},\bar{\Omega},E) d\Omega' dE' d\Omega dE dV$$

Rearranging equation a.30 and substituting into equation a.33 gives:

$$\frac{\partial C_j(t)}{\partial t} + \lambda_j C_j(t) = \frac{P(t)}{\Lambda} \cdot \frac{1}{F(t)} \iiiint \beta_j \nu \Sigma_f(\bar{r},E',t) \qquad \textbf{(a.34)}$$

$$\times \psi(\bar{r},\bar{\Omega}',E',t)\phi^*(\bar{r},\bar{\Omega},E) d\Omega' dE' d\Omega dE dV$$

Using the parameter definition of $\bar{\beta}_j(t)$ (equation a.28), equation a.34 can be written as:

$$\frac{\partial C_j(t)}{\partial t} + \lambda_j C_j(t) = \frac{\bar{\beta}_j(t)}{\Lambda} P(t) \qquad \textbf{(a.35)}$$

The shape equation is obtained by substituting the separability condition (equation a.4) into the time-dependent Boltzmann transport equation and dividing by P(t) to yield the following:

$$\frac{1}{v}\cdot\frac{1}{P(t)}\cdot\frac{\partial P(t)}{\partial t}\cdot\psi(\bar{r},\bar{\Omega},E,t) + \frac{1}{v}\frac{\partial\psi(\bar{r},\bar{\Omega},E,t)}{\partial t} + \bar{\Omega}\cdot\bar{\nabla}\psi(\bar{r},\bar{\Omega},E,t)$$

$$+ \Sigma_t(\bar{r},E,t)\psi(\bar{r},\bar{\Omega},E,t) = \int\int\Sigma_s(\bar{r},\bar{\Omega}',E'\rightarrow\bar{\Omega},E;t)\psi(\bar{r},\bar{\Omega}',E',t)d\Omega'dE' \qquad \textbf{(a.36)}$$

$$+ \int\int\chi_p(E)(1-\beta)\nu\Sigma_f(\bar{r},E',t)\psi(\bar{r},\bar{\Omega}',E',t)d\Omega'dE'$$

$$+ \frac{Q_d(\bar{r},E,t)}{P(t)} + \frac{Q}{P(t)}$$

where,

$$Q_d(\bar{r},E,t) = \int_{-\infty}^{t}\int_{E'}\int_{\bar{\Omega}'}\nu\Sigma_f(\bar{r},E',t)P(t')\psi(\bar{r},\bar{\Omega}',E',t) \qquad \textbf{(a.37)}$$

$$\times\sum_j\beta_j\chi_j\lambda_j e^{-\lambda_j(t-t')}d\Omega'dE'dt'$$

The improved quasistatic method, as applied to TDKENO-M, solves the full shape equation (equation a.36) by replacing the time derivative of the shape function with a backward difference approximation of first order:

$$\frac{\partial}{\partial t}\psi(\bar{r},\bar{\Omega},E,t) = \frac{\psi(\bar{r},\bar{\Omega},E,t) - \psi(\bar{r},\bar{\Omega},E,t-\Delta t)}{\Delta t} \qquad \textbf{(a.38)}$$

where,

t - Δt = time of the last shape calculation,
Δt = time interval for shape calculations.

Since the rate of change of the shape function is determined essentially by a slow variation of system parameters, this approximation is valid over a much larger time range than is acceptable for the total flux. Using the backward difference approximation for the time derivative of the shape function yields the following shape equation:

$$\begin{aligned}
&\frac{1}{v}\cdot\frac{1}{P(t)}\cdot\frac{\partial P(t)}{\partial t}\cdot\psi(\bar{r},\bar{\Omega},E,t) + \frac{1}{v}\left(\frac{\psi(\bar{r},\bar{\Omega},E,t) - \psi(\bar{r},\bar{\Omega},E,t-\Delta t)}{\Delta t}\right) \\
&\quad + \bar{\Omega}\cdot\bar{\nabla}\psi(\bar{r},\bar{\Omega},E,t) + \Sigma_t(\bar{r},E,t)\psi(\bar{r},\bar{\Omega},E,t) \\
&\quad = \int\int \Sigma_s(\bar{r};\bar{\Omega}',E'\rightarrow\bar{\Omega},E;t)\psi(\bar{r},\bar{\Omega}',E',t)d\Omega'dE' \qquad \textbf{(a.39)} \\
&\quad + \int\int \chi_p(E)(1-\beta)v\Sigma_f(\bar{r},E',t)\psi(\bar{r},\bar{\Omega}',E',t)d\Omega'dE' \\
&\quad + \frac{Q_d(\bar{r},E,t)}{P(t)} + \frac{Q}{P(t)}
\end{aligned}$$

Equation a.39 can be rearranged to be written as:

$$\frac{1}{v}\left(\frac{1}{P(t)}\cdot\frac{\partial P(t)}{\partial t} + \frac{1}{\Delta t}\right)\psi(\bar{r},\bar{\Omega},E,t)$$

$$+ \bar{\Omega}\cdot\bar{\nabla}\psi(\bar{r},\bar{\Omega},E,t) + \Sigma_t(\bar{r},E,t)\psi(\bar{r},\bar{\Omega},E,t)$$

$$- \int\int \Sigma_s(\bar{r};\bar{\Omega}',E'\rightarrow\bar{\Omega},E;t)\psi(\bar{r},\bar{\Omega}',E',t)d\Omega'dE' \qquad \textbf{(a.40)}$$

$$- \int\int \chi_p(E)(1-\beta)\nu\Sigma_f(\bar{r},E',t)\psi(\bar{r},\bar{\Omega}',E',t)d\Omega'dE'$$

$$= \frac{Q_d(\bar{r},E,t)}{P(t)} + \frac{Q}{P(t)} + \frac{\psi(\bar{r},\bar{\Omega},E,t-\Delta t)}{v\Delta t}$$

This shape equation is of inhomogeneous partial differential form. As with the quasistatic method, Q_d is computed from equation a.37 and the $[\partial P(t)/\partial t]/P(t)$ term is determined from the solution of the point kinetics equation (equation a.25). With the improved quasistatic method, there is now an additional source term from the backward difference approximation of the shape derivative as well as an additional $1/\Delta t$ term which is incorporated as shown in equation a.40.

Appendix B: TDKENO-M

TDKENO-M Code Listing

```
      program tdkenom
************************************************************************
*                                                                      *
*   TTTTTTTTTT   DDDD    K   K  EEEEE  N     N    OOO    M      M    *
*        T       D   D   K  K   E      NN    N   O   O   MM    MM    *
*        T       D    D  K K    E      N N   N  O     O  M M  M M    *
*        T       D    D  KK     EEEEE  N  N  N  O     O  M  MM  M    *
*        T       D    D  K K    E      N   N N  O     O  M      M    *
*        T       D   D   K  K   E      N    NN   O   O   M      M    *
*        T       DDDD    K   K  EEEEE  N     N    OOO    M      M    *
*                                                                      *
************************************************************************
*                                                                      *
*                              VARIABLES                               *
*                                                                      *
************************************************************************

*                                                                      *
*       a(1)-a(nadjp1)  =  phi angle bins                              *
*       a(nadjp1)-a(i)  =  theta andle bins                            *
*       abeta(i)        =  fraction of delayed neutrons for group i    *
*       abeta2(i)       =  total abeta(i)                              *
*       abeta3(i)       =  variance of abeta(i)                        *
*       adj             =  flag for doing adjoint calculation          *
*       adjfis          =  adjoint flux                                *
*       adjin           =  adjoint in incoming direction               *
*       alamda(i)       =  fraction of delayed neutrons for group i    *
*       albb            =  albedo                                      *
*       amp(i)          =  flux amplitude                              *
*       ampti(i)        =  time of flux amplitude calculation          *
*       amx             =  flag for printing all mixture X-sections    *
*       atol            =  tolerance used by LSODE                     *
*       beta(i)         =  delayed neutron fraction for group i        *
*       betat           =  total delayed neutron fraction              *
*       betatot         =  total delayed neutron fraction              *
*       binu(i)         =  boundaries of phi bins                      *
*       binv(i)         =  boundaries of theta bins                    *
*       c(i)            =  precursor concetration for group i          *
*       c0(i)           =  initial concentration for group i           *
*       ccons           =  constraint integral                         *
*       ccons2          =  total constraint integral                   *
*       ccons3          =  variance of constraint integral             *
*       chi(i,j)        =  fission spectrum                            *
*       conp2b          =  conversion factor to btu/sec                *
*       const           =  constraint constant                         *
*       delmin          =  minimum time step                           *
*       delt            =  time step                                   *
*       deltat          =  time step                                   *
*       dpdt            =  (1/pow)*(dp/dt)                             *
*       eb              =  conversion factor from energy to velocity   *
*       etamin          =  cosine of scattering angle in COM           *
*       exprn()         =  exponential random number                   *
*       fap             =  absorption probability                      *
*       fk(i)           =  pdf for delayed neutron source              *
*       fkig(i)         =  pdf for delayed neutron source              *
*       flux(i)         =  adjoint flux for bin i                      *
*       flux(i,j)       =  flux shape                                  *
*       flux(i,j,1)     =  flux shape                                  *
*       flux(i,j,2)     =  percent deviation flux shape                *
*       flux(i,j,3)     =  variance of flux shape                      *
*       fluxti          =  time of flux shape calculation              *
```

```
*       flx             =  flag for collecting and printing fluxes    *
*       fnap            =  scatter probability                        *
*       fnfp            =  fission probability                        *
*       fs              =  flux shift/(delayed + flux shift)          *
*       fsp             =  p0 angles                                  *
*       gen             =  generation time (also # of generations)    *
*       gfis            =  fission weighted energy group              *
*       iadjfl          =  file identifier for adjoint file           *
*       icexs0          =  file identifier for ft434001               *
*       icon,iconf      =  flag for determining perturbation of X's   *
*       id              =  id number for mixture                      *
*       idelay          =  file identifier for delay.dat              *
*       ifixsc          =  pointer for fixed source data              *
*       iflux           =  file identifier for flux.dat               *
*       iflux0          =  file identifier for flux.out               *
*       ig              =  energy group number (also flags EOF)       *
*       igg             =  flag for gammas                            *
*       igen            =  file identifier for general.dat            *
*       igen            =  generation number (also)                   *
*       igfis           =  energy group of fission                    *
*       ikin            =  file identifier for kinetics.dat           *
*       iline           =  dummy variable to count line numbers       *
*       inn             =  file identifier for power.in               *
*       iopt            =  parameter used by LSODE                    *
*       iout            =  file identifier for output files           *
*       iphi            =  phi bin number                             *
*       isec            =  section number of general.dat file         *
*       isigf           =  file identifier for nusigf.dat             *
*       isort           =  flag used to sort values                   *
*       istate          =  parameter used by LSODE                    *
*       istop           =  flag used for last iteration               *
*       itask           =  parameter used by LSODE                    *
*       itheta          =  theta bin number                           *
*       itime           =  file identifier for timen.dat              *
*       itime0          =  file identifier for time0.dat              *
*       itimen          =  file identifier for timen.dat              *
*       itol            =  tolerance used for LSODE                   *
*       ixsst           =  pointer for steady state X-sections        *
*       kmax            =  number of geometry regions                 *
*       kmax0           =  number of geometry regions                 *
*       knew            =  region number of fixed source neutron      *
*       lamp            =  pointer for flux amplitude                 *
*       lampt           =  pointer for flux amplitude times           *
*       lchi            =  pointer for flux spectrum                  *
*       lcon            =  pointer for constraint constant            *
*       lend            =  pointer for end of array                   *
*       lf              =  pointer for flux shift                     *
*       lfk             =  pointer for flux shift spatial pdf         *
*       lfkig           =  pointer for flux shift pdf                 *
*       lflux           =  pointer for flux shapes                    *
*       lfulxt          =  pointer for flux times                     *
*       lib             =  logical unit number for AMPX library       *
*       lim             =  maximum number of delayed groups           *
*       limmem          =  maximum number of array values             *
*       liw             =  parameter used by LSODE                    *
*       ll              =  lower limit                                *
*       lmt             =  maximum number of mixtures                 *
*       lncor           =  pointer for mixture i.d. numbers           *
*       lnextr          =  number of entries of extra data from EXTRA *
*       lntegr          =  pointer for integrand values of qd         *
*       lpow            =  pointer for flux amplitudes                *
*       lpowt           =  pointer for flux amplitude times           *
*       lrw             =  parameter used by LSODE                    *
*       lqd             =  pointer for delayed neutron source         *
*       lqdk            =  pointer for pdf of delayed neutron source  *
*       lqdkig          =  pointer for pdf of delayed neutron source  *
*       lsigf           =  pointer for nusigf                         *
*       lsigft          =  pointer for nusigf times                   *
*       ltime           =  pointer for time                           *
*       lu              =  upper limit                                *
*       lvel            =  pointer for energy boundaries              *
*       lvol            =  pointer for volumes                        *
*       m               =  mixture id number (also flags crossings)   *
```

```
*       matt            =  number of mixtures in geometry            *
*       mf              =  parameter used by LSODE                   *
*       mix             =  mixture id number                         *
*       mixnum          =  number of mixtures                        *
*       mwa             =  magic word array (2-D scatter transfer)   *
*       n1d             =  number of 1-D X-sections on ICE tape      *
*       n1m             =  length of p0 array for primaries          *
*       nadj            =  flag for adjoint calculations (true)      *
*       nadjph          =  number of phi angle bins                  *
*       nadjth          =  number of theta angle bins                *
*       namp            =  maximum number of times on timen.dat file *
*       ncork(i)        =  mixture id number for region i            *
*       ndg             =  number of delayed neutron groups          *
*       ndglim          =  maximum number of delayed neutron groups  *
*       ndxadj          =  pointer for adjoint flux                  *
*       neq             =  number of equations solved by LSODE       *
*       neut            =  neutron number                            *
*       newbar          =  flag for calculation average nu           *
*       ngp             =  number of energy groups                   *
*       ngp1            =  ngp + 1                                   *
*       ngp0            =  number of energy groups                   *
*       nmat            =  number of cross-section mixtures          *
*       nnamp           =  number of time steps on timen.dat file    *
*       nntmef          =  number of time steps on flux.dat          *
*       nntmes          =  number of time steps on nusigf.dat        *
*       nnpow           =  number of time steps on timen.dat         *
*       np1             =  order of expansion of p-set               *
*       npg             =  neutrons per generation                   *
*       npow            =  maximum number of times on timen.dat file *
*       nrec            =  number of records/section on general.dat  *
*       nsk             =  number of generations omitted             *
*       nskip           =  number of generations skipped             *
*       ntegr           =  maximum number of integrand values        *
*       ntime           =  number of time intervals                  *
*       ntimef          =  maximum number of flux shape calculations *
*       ntimes          =  maximum number of times on nusigf.dat file*
*       nx              =  number of secondary 1-D X-sections        *
*       pi              =  3.14159265                                *
*       plt             =  flag for drawing pictures of problem geo  *
*       pow(i)          =  flux amplitude                            *
*       powini          =  initial power                             *
*       powti(i)        =  time of flux amplitude calculation        *
*       pth             =  particle pathlength                       *
*       qd(i,j)         =  delayed neutron source                    *
*       qdk(i)          =  pdf for delayed neutron source            *
*       qdkig(i,j)      =  pdf for delayed neutron source            *
*       r1              =  prompt/(prompt + delayed)                 *
*       r2              =  flux shift/(prompt + delayed)             *
*       r3              =  prompt/delayed                            *
*       react           =  contribution to reactivity                *
*       react2          =  total reactivity                          *
*       react3          =  variance of reactivity                    *
*       rho             =  reactivity                                *
*       rhotot          =  reactivity                                *
*       rnd             =  hexadecimal random number                 *
*       rtol            =  tolerance used by LSODE                   *
*       sct             =  number of discrete scattering angles      *
*       sigcon          =  contribution to reactivity due to perturb *
*       sigf(i,j,k)     =  nusigf                                    *
*       sigf0           =  unperturbed fission X-section             *
*       sigs            =  scattering X-section                      *
*       sigt            =  total X-section                           *
*       sigt0           =  unperturbed total X-section               *
*       sold            =  beginning point of TRACK                  *
*       ssfiss          =  nusigf*flux at t=0                        *
*       sum,sumf        =  dummy variable used for summation         *
*       t               =  time                                      *
*       tba             =  time for each generation (min)            *
*       tcool1          =  coolant temperature                       *
*       tcool2          =  coolant temperature                       *
*       tfisn           =  average fission X-section                 *
*       tfuel           =  fuel temperature                          *
*       time            =  time                                      *
```

```
*       time2           =  current time                                 *
*       tme             =  execution time (min)                         *
*       tout            =  time at end of an iteration                  *
*       trial           =  convergence parameter                        *
*       tstop           =  time of end of interval                      *
*       twopi           =  2*pi                                         *
*       u,v,w           =  direction cosines                            *
*       vel(i)          =  velocity (energy) boundary for group i       *
*       vinv            =  inverse velocity                             *
*       vol(i)          =  volume of region i                           *
*       wt              =  particle weight                              *
*       wtavg           =  average particle weight                      *
*       wtxlen          =  wt*xlen                                      *
*       x1              =  x-coordinate                                 *
*       x1d             =  extra 1-D X-sections                         *
*       xap             =  flag for printing discrete scatter angles    *
*       xc              =  interpolated precursor concentration         *
*       xcc(i)          =  precursor concentration of group i           *
*       xflux           =  interpolated flux shape                      *
*       xgen,xgent      =  generation time                              *
*       xgen2           =  total generation time                        *
*       xgen3           =  variance of generation time                  *
*       xntegr(i)       =  integrand values of qd                       *
*       xpow            =  interpolated flux amplitude                  *
*       xsc             =  logical unit number for X-section library    *
*       xtime           =  last time on timen.dat                       *
*       y1              =  y-coordinate                                 *
*       y1d(i,j)        =  X-sections                                   *
*       y1d(i,1)        =  total X-sections                             *
*       y(1)            =  normalized flux amplitude                    *
*       y(2)-y(ndgp1)   =  normalized precursor concentrations          *
*       ydot(i)         =  time derivative of y(i)                      *
*       x1              =  z-coordinate                                 *
*       xc              =  interpolated precursor concentration         *
*       xflux           =  interpolated flux shape                      *
*       xlen            =  component of pathlength in x-direction       *
*       xnormc          =  normalizaiton factor for constraint const    *
*       xnormp          =  normalization factor for flux amplitude      *
*       xpow            =  interpolated flux amplitude                  *
*       xsig            =  interpolated nusigf                          *
*       xtra(1)-(i)     =  boundaries of phi bins                       *
*       xtra(i)-(i+j)   =  boundaries of theta bins                     *
*       xtra(ndxadj)    =  adjoint flux                                 *
*                                                                       *
*************************************************************************
*                                                                       *
*                              SUBROUTINES                              *
*                                                                       *
*************************************************************************
*                                                                       *
*       CDELAY          calculates delayed neutron source.              *
*       CONMWA          converts 2-D scatter array to KENO format       *
*       CONRHO          makes contribution to reactivity and            *
*                       generation time at every collision site and     *
*                       every boundary crossing.                        *
*       DATAIN          controls most of data input.  Calls EXTRA.      *
*       EXTRA           inputs angle bins for adjoint case.             *
*       FISFLX          calculates statistics for k-eff, reactivity,    *
*                       generation time, etc.                           *
*       FITFLX          prints fluxes to interface file.                *
*       GUIDE           controls tracking.                              *
*       INITAL          initializes and opens interface files.          *
*       MASTER          controlling subroutine of KENO.                 *
*       NORM            merges and reformats flux output of KENO into   *
*                       flux shape interface file.                      *
*       NSTART          generates the fission source for a generation.  *
*       PRTFLX          outputs fluxes to interface file.               *
*       PTKIN           solves the point kinetics equations.            *
*       POWER           calculates the normalized power from the flux   *
*                       amplitude and shape.  Also, determines the      *
*                       normalized flux distribution.                   *
*       REMVID          removes reaction type id's for 1-D X-sections.  *
*       TRACK           tracks each neutron through the geometry.       *
```

```
*       WWFITF          edits adjoint fluxes.                          *
*       WWINF           reads adjoint fluxes from file.                *
*       WWOUTF          outputs adjoint fluxes to an interface file.   *
*       WWRETR          retrieves steady-state scatter X-sections.     *
*       WWRHO           outputs reactivity to an interface file.       *
*       WWSTAT          calculates statistics for adjoint flux.        *
*       WSTRSU          selects starting position of fixed source n's.*
*       WWTAPE          reads steady-state cross sections.             *
*       WWXSEC          reads steady state cross sections from file.   *
*                                                                      *
************************************************************************
      integer iadj,imeth,iss,inumtfs,irhonum,ipknum
      common/flags/irhonum,ipknum
c
c-----Find initial time of calculation
c
      call jstime(iim)
      diii = iim
c
c-----Make symbolic links to executables
c
      call slinks
c
c-----Create interface files
c
      call interfiles(iadj,imeth,iss,inumtfs,iouttt)
c
      write(iouttt,*)('Adjoint calculation flag =           ',iadj)
      write(iouttt,*)('Forward calculation method flag =    ',imeth)
      write(iouttt,*)('Forward calculation flag =           ',iss)
      write(iouttt,*)('Number of transient calcs =          ',iss)
      write(iouttt,*)('Number of reactivities per shape =   ',irhonum)
      write(iouttt,*)('Number of point kinetics per shape = ',ipknum)
c
c-----Perform steady-state adjoint calculation
c
c     NOTE:  A steady-state adjoint calculation must be performed unless
c            an adjoint file already exists and is saved in the
c            local directory as ft42f001.
c
      if (iadj.eq.1) then
        call adjoint
      end if
c
c-----Perform steady-state forward calculation
c
c     NOTE:  A steady-state forward calculation must be performed unless
c            a forward flux file already exists and is saved in the
c            local directory as flux.dat.
c
      if (iss.eq.1) then
        call ssforward
      end if
c
c-----Extrapolate reactivity and flux amplitude to first transient
c     flux shape calculation
c
      call extrap
c
c-----Perform transient calculations
c
      do i=1,inumtfs
c
c-----Perform shape calculation
c
          call transhape
c
c-----Perform reactivity and point kinetics calculations
c
          call extrap
c
      end do
c
```

```
c-----Calculate the power trace and the normalized flux distribution
c
      call power
c
c-----Find final time of calculation and determine total time
c
      call jstime(ii)
      diim = ii
      dtme = (diii-diim)/6000.0
      write(iouttt,101)dtme
      write(6,101)dtme
c
      close(ioutt)
      close(61)
c
c-----Cleanup the job
c
      call cleanup
c
c----------End TDKENOM program
c
 101  format(' You have successfully completed TDKENO-M in ',f10.5,
     *       ' minutes !!!')
      stop
      end
c
c
      subroutine slinks
      character*60 link1,link2
c
      link1='ln -s $HOME/tdkeno/tdkeno         tdkeno     '
      link2='ln -s $HOME/KENO/load/tdkenoadj   tdkenoadj  '
c
      status=system(link1)
      if (status.ne.0) stop
      status=system(link2)
      if (status.ne.0) stop
      return
      end
c
c
      subroutine interfiles(iadj,imeth,iss,inumtfs,iouttt)
      common/flags/irhonum,ipknum
      dimension beta(20),alamda(20),xcc(20),vol(1000)
      dimension chi(20),ebound(21),ncor(1000),sigfti(20)
      dimension sigf(20,5,20),fluxti(40),in1(7),re1(7)
      integer iadj,imeth,iss,inumtfs,irhonum,ipknum
      character title*60
      character*60 link1,link2,link3
      iouttt = 61
      ntimes = 20
c
      open(41,file='time0.dat',status='new')
      open(47,file='delay.dat',status='new')
      open(49,file='kinetics.dat',status='new')
      open(44,file='rho.out',status='new')
      open(45,file='general.dat',status='new')
      open(54,file='nusigf.dat',status='new')
      open(55,file='flux.dat',status='unknown')
      open(46,file='timen.dat',status='new')
      open(50,file='keno.meth',status='new')
      open(58,file='infile',status='old')
      open(51,file='rho.dat',status='new')
      open(52,file='cdelay.dat',status='new')
      open(53,file='outfile',status='new')
      open(61,file='tmeout',status='new')
c
      link1='cp timen.dat ft46f001      '
      link2='cp delay.dat ft47f001      '
      link3='cp kinetics.dat ft49f001   '
c
      rewind 41
      rewind 47
```

```
      rewind 54
      rewind 44
      rewind 45
      rewind 46
      rewind 55
      rewind 58
      rewind 49
      rewind 50
      rewind 51
      rewind 52
      rewind 53
      rewind 61
c
      read(58,*) iadj,imeth,iss,inumtfs,irhonum,ipknum
      write(50,*) imeth
      read(58,801) kmax,ngp
      read(58,802) dpf2t,f2dpf,d2p,xdelt
      if(iss.eq.1) then
         write(55,801) kmax,ngp
      end if
      write(47,801) kmax,ngp
      write(47,802) dpf2t,f2dpf,d2p,xdelt
      read(58,*) iflag1
      write(44,*) iflag1
      read(58,805) ndg
      read(58,806) (beta(i),i=1,ndg)
      read(58,806) (alamda(i),i=1,ndg)
      write(49,805) ndg
      write(49,806) (beta(i),i=1,ndg)
      write(49,806) (alamda(i),i=1,ndg)
      read(58,803)time,pow,tfuel,tcool1,tcool2
      write(46,803)time,pow,tfuel,tcool1,tcool2
      read(58,810) (xcc(i),i=1,ndg)
      write(46,804) (xcc(i),i=1,ndg)
      read(58,807)kmax0,ngp0,nmat
      write(54,807)kmax0,ngp0,nmat
      if(kmax0 .ne. kmax)then
          write(6,950)kmax,kmax0
          stop
      end if
      if(ngp0 .ne. ngp)then
          write(6,960)ngp,ngp0
          stop
      end if
      read(58,808)(vol(k),k=1,kmax)
      read(58,808)(chi(ig),ig=1,ngp)
      read(58,808)(ebound(ig),ig=1,ngp+1)
      read(58,809)(ncor(k),k=1,kmax)
      write(54,808)(vol(k),k=1,kmax)
      write(54,808)(chi(ig),ig=1,ngp)
      write(54,808)(ebound(ig),ig=1,ngp+1)
      write(54,809)(ncor(k),k=1,kmax)
c
      i2 = 0
  30  continue
c
      i = 0
  40  continue
c
          i = i + 1
          if(i .gt. ntimes)then
              write(*,910)ntimes
              stop
          end if
c
          read(58,808)sigfti(i),
     *              ((sigf(ig,m,i),ig=1,ngp),m=1,nmat)
          write(54,808)sigfti(i),
     *              ((sigf(ig,m,i),ig=1,ngp),m=1,nmat)
          if(sigfti(i) .lt. 0.0)then
              i = i - 1
              go to 50
          end if
```

```
          go to 40
  50  continue
c
c-----just read nusigf, go back for sigf
c
      if(i2 .eq. 0)then
          i2 = 1
          go to 30
      end if
c
      nntmes = i
      if(nntmes .le. 0) then
          write(iout,990)
          stop
      end if
c
      read(58,902)isec,nrec
      write(45,902)isec,nrec
      do 120 ic=1,nrec
          read(58,900)title
          write(45,900)title
  120 continue
c
c-----skip section #2 of general interface file
c
      read(58,902)isec,nrec
      write(45,902)isec,nrec
      read(58,805)(in1(i),i=1,5)
      write(45,805)(in1(i),i=1,5)
      read(58,805)(in1(i),i=1,6)
      write(45,805)(in1(i),i=1,6)
      read(58,*)in4,in5,re4,re5
      write(45,815)in4,in5,re4,re5
      read(58,805)(in1(i),i=1,7)
      write(45,805)(in1(i),i=1,7)
      read(58,805)in2
      write(45,805)in2
      read(58,805)in2
      write(45,805)in2
      read(58,806)(re1(i),i=1,7)
      write(45,806)(re1(i),i=1,7)
      read(58,806)re2
      write(45,806)re2
      read(58,806)(re1(i),i=1,2)
      write(45,806)(re1(i),i=1,2)
      read(58,806)(re1(i),i=1,2)
      write(45,806)(re1(i),i=1,2)
      read(58,806)(re1(i),i=1,4)
      write(45,806)(re1(i),i=1,4)
      read(58,806)(re1(i),i=1,4)
      write(45,806)(re1(i),i=1,4)
      read(58,806)(re1(i),i=1,4)
      write(45,806)(re1(i),i=1,4)
      read(58,806)(re1(i),i=1,4)
      write(45,806)(re1(i),i=1,4)
c
c-----skip section 3 of interface file
c
      read(58,902)isec,nrec
      write(45,902)isec,nrec
      read(58,805)(in1(i),i=1,3)
      write(45,805)(in1(i),i=1,3)
      read(58,805)(in1(i),i=1,7)
      write(45,805)(in1(i),i=1,7)
      read(58,806)(re1(i),i=1,3)
      write(45,806)(re1(i),i=1,3)
      do i=1,3
         read(58,*) in4,in5,re4
         write(45,816) in4,in5,re4
      end do
c
c-----read time data  section #4
c
```

```
      read(58,902)isec,nrec
      write(45,902)isec,nrec
      itimes = nrec
      n=0
  101 continue
      read(58,903)fluxti(n+1)
      write(45,903)fluxti(n+1)
      n=n+1
      nlft=nrec-n
      if(nlft.gt.0)go to 101
c
c-----read time data  section #5
c
  100 continue
      read(58,902)isec,nrec
      write(45,902)isec,nrec
      read(58,902)ntime
      write(45,902)ntime
      do 200 ic=1,ntime
          read(58,903)tstop,delt
          write(45,903)tstop,delt
  200 continue
c
      close(41)
      close(47)
      close(54)
      close(44)
      close(45)
      close(46)
      close(55)
      close(58)
      close(49)
      close(50)
      close(51)
      close(52)
c
      status=system(link1)
      if (status.ne.0) stop
      status=system(link2)
      if (status.ne.0) stop
      status=system(link3)
      if (status.ne.0) stop
c
c
 801  format(10i5)
 802  format(6e11.4)
 803  format(7e11.4)
 810  format(6e11.4)
 804  format(11x,6e11.4)
 805  format(10i5)
 806  format(7e11.4)
 807  format(3i5)
 808  format(6e11.4)
 809  format(12i5)
 815  format(2i5,2e11.4)
 816  format(2i5,1e11.4)
 900  format(a60)
 902  format(15i5)
 903  format(7e11.4)
 904  format(1x,21hend of problem  time=,f7.2)
 910  format(1x,'more times on nusigf.dat than allowed',i11)
 950  format(1x,'kmax not consistent in interface files in norm',2i5)
 960  format(1x,'ngp  not consistent in interface files in norm',2i5)
 990  format(1x,'no data on nusigf.dat')
c
      return
      end
c
c
      subroutine adjoint
      character*60 link2,link3,link4,link5,link6,link7
      character*60 link8,link9,link11,link12
c
```

```
      link2='cp xs ft02f001                  '
      link3='mv input input.forward          '
      link4='cp input.adj input              '
      link5='tdkenoadj                       '
      link6='cat _out* input > output.adj    '
      link7='rm _out* input _P*              '
      link8='mv output.adj     output        '
      link9='mv input.forward  input         '
      link11='rm ft08f001 ft09f001 ft14f001  '
      link12='rm tdkenoadj                   '
c
      status=system(link2)
      if (status.ne.0) stop
      status=system(link3)
      if (status.ne.0) stop
      status=system(link4)
      if (status.ne.0) stop
      status=system(link5)
      if (status.ne.0) stop
      status=system(link6)
      if (status.ne.0) stop
      status=system(link7)
      status=system(link8)
      if (status.ne.0) stop
      status=system(link9)
      if (status.ne.0) stop
      status=system(link11)
      status=system(link12)
c
      return
      end
c
c
      subroutine ssforward
      character*60 link1,link2,link4,link5,link6,link7
      character*60 link8,link9,link10
c
      link9='cp input.for input              '
      link1='cp xspmc ft43f001               '
      link2='cp xsmc ft02f001                '
      link4='cat norm.out >> outfile         '
      link10='rm norm.out                    '
      link5='mv fluxnew.dat flux.dat         '
      link6='mv ft46f001 timen.dat           '
      link7='cp flux.dat flux.ss             '
      link8='mv flux.ss output               '
c
      status=system(link9)
      if (status.ne.0) stop
      status=system(link1)
      if (status.ne.0) stop
      status=system(link2)
      if (status.ne.0) stop
      call kenova
      call norm
      status=system(link4)
      status=system(link10)
      status=system(link5)
      status=system(link6)
      status=system(link7)
      status=system(link8)
c
      return
      end
c
      subroutine transhape
      character*60 link1,link2,link3,link4,link5,link6,link7
      character*60 link8,link9
c
      link9='cp input.for input              '
      link1='mv timen.dat ft46f001           '
      link2='mv delay.dat ft47f001           '
      link3='cat norm.out >> outfile         '
```

```
      link4='rm norm.out                      '
      link5='mv fluxnew.dat flux.dat          '
      link6='cat rho.out >> rho.dat           '
      link7='cat ptkin.out >> outfile         '
      link8='rm ptkin.out                     '
c
      status=system(link1)
      if (status.ne.0) stop
      status=system(link2)
      if (status.ne.0) stop
      call kenova
      call norm
      status=system(link3)
      status=system(link4)
      status=system(link5)
      call rho
      status=system(link6)
      if (status.ne.0) stop
      call ptkin
      status=system(link7)
      if (status.ne.0) stop
      status=system(link8)
      if (status.ne.0) stop
c
      return
      end
c
      subroutine extrap
      character*60 link1,link2,link3,link4,link5,link6,link7
      character*60 link8,link9
c
      link8='cp xsmc ft43f001                 '
      link9='cp xspmc ft02f001                '
      link7='mv timen.dat time0.dat           '
      link1='cat rho.out >> rho.dat           '
      link2='cat ptkin.out >> outfile         '
      link3='rm ptkin.out                     '
      link4='cat delay.dat >> cdelay.dat      '
      link5='cat cdelay.out >> outfile        '
      link6='rm cdelay.out                    '
c
      status=system(link8)
      status=system(link9)
      status=system(link7)
      if (status.ne.0) stop
      call rho
      status=system(link1)
      if (status.ne.0) stop
      call ptkin
      status=system(link2)
      if (status.ne.0) stop
      status=system(link3)
      if (status.ne.0) stop
      call cdelay
      status=system(link4)
      if (status.ne.0) stop
      status=system(link5)
      if (status.ne.0) stop
      status=system(link6)
      if (status.ne.0) stop
c
      return
      end
**********************************************************************
*                                                                    *
*                     SUBROUTINE RHO                                 *
*                                                                    *
**********************************************************************
      subroutine rho
c--------------------------------------------------------------------
c
c     calculate point kinetics parameters including:  reactivity,
c     generation time, effective betas, F(t), precursor densities,
```

```
c     and extraneous source.
c
c     nntmef = no. of time steps on flux.dat
c     nntmes = no. of time steps on nusigf.dat
c     nnamp  = no. of time steps on timen.dat
c     ntimef = max no of flux shape calcs
c     ntimes = max no of times on nusigf.dat file
c     namp   = max no of times on timen.dat file
c     nproca = max no of processes on ampx file
c     irhonum =no. of intermediate reactivity calculations
c     ipknum = no. of intermediate point kinectics calculations
c-----------------------------------------------------------------
c
      common/flags/irhonum,ipknum
      common/unitsrho/iout,isigf,iflux,ikin,iadjoint,icexss,icexsp,
     * itimen,iflux0,igen
      common/paramrho/kmax,ngp,ntimef,ntimes,namp,nadjph,nadjth,
     * nntmef,nntmes,nnamp,nmat,ndg,nadjn,ngp1,iflag
      double precision d
      dimension d(500000)
      data limmem/500000/
      integer nadjn,nadjph,nadjth,iflag
c
c
      ntimef = 40
      ntimes = 20
      namp   = 10000
      nproca = 6
c
      call ofilerho
c
      rewind iflux
      rewind isigf
      rewind ikin
      rewind itimen
      rewind iout
c
      read(iflux,940)kmax,ngp
c     write(iout,*)('kmax from flux file = ',kmax)
c     write(iout,*)('ngp from flux file  = ',ngp)
c
      read(isigf,940)kmax0,ngp0,nmat
c     write(iout,*)
c     write(iout,*)('kmax from nusigf file = ',kmax0)
c     write(iout,*)('ngp from nusigf file  = ',ngp0)
c     write(iout,*)('nmat from nusigf file = ',nmat)
      if(kmax0 .ne. kmax)then
          write(iout,950)kmax,kmax0
          stop
      end if
      if(ngp0 .ne. ngp)then
          write(iout,960)ngp,ngp0
          stop
      end if
c
      read(ikin,941) ndg
c     write(iout,*)
c     write(iout,*)('ndg from kinetics file = ',ndg)
c
c
      read(iout,*) iflag
c     write(6,*)('Iflag = ',iflag)
c
c-----input adjoint data
c
      rewind iadjoint
      read(iadjoint)nadjph,nadjth
c     write(iout,*)('nadjph and nadjth from adjoint file = ',
c    * nadjph,nadjth)
      nadjn=nadjph*nadjth
c
c-----generate array pointers
      ngp1     = ngp + 1
```

```
      lflux    = 1
      lsigf    = lflux + kmax*ngp*ntimef
      lalamda  = lsigf + nmat*ngp*ntimes
      lbeta    = lalamda + ndg
      lfluxti  = lbeta + ndg
      lsigfti  = lfluxti + ntimef
      lncor    = lsigfti + ntimes
      lvol     = lncor + kmax
      lcon     = lvol + kmax
      lbin     = lcon + ntimef
      lvel     = lbin + nadjn
      lc       = lvel + ngp1
      lci      = lc + ndg
      ladflux  = lci + ndg
      lchi     = ladflux + ngp*kmax
      lds      = lchi + ngp
      lids     = lds + nproca
      lxs      = lids + 50
      lys      = lxs + kmax*nmat
      ldp      = lys + nmat*nmat
      lidp     = ldp + nproca
      lxp      = lidp + 50
      lyp      = lxp + kmax*nmat
      ldelx    = lyp + nmat*nmat
      lscats   = ldelx + kmax*nmat
      lscatp   = lscats + nmat*ngp1
      ldelscat = lscatp + nmat*ngp1
      lbetaeff = ldelscat + nmat*ngp1
      lid3s    = lbetaeff + ndg*(irhonum + 1)
      lid3p    = lid3s + nmat*nmat*ngp
      lid4s    = lid3p + nmat*nmat*ngp
      lid4p    = lid4s + nproca
      lrhoti   = lid4p + nproca
      lgen     = lrhoti + (irhonum + 1)
      lrho     = lgen + (irhonum + 1)
      lbetai   = lrho + (irhonum + 1)
      iend     = lbetai + ndg
c     write(6,*)('iend = ',iend)
      if(iend .gt.limmem)then
          write(iout,900)limmem,iend
          stop
      end if
c
      call mastrho(d(lflux),d(lsigf),d(lalamda),d(lbeta),d(lfluxti),
     * d(lsigfti),d(lncor),d(lvol),d(lcon),d(lbin),d(lvel),d(lc),
     * d(lci),d(ladflux),d(lchi),d(lds),d(lids),d(lxs),d(lys),
     * d(ldp),d(lidp),d(lxp),d(lyp),d(ldelx),d(lscats),d(lscatp),
     * d(ldelscat),d(lbetaeff),d(lid3s),d(lid3p),d(lid4s),d(lid4p),
     * d(lrhoti),d(lgen),d(lrho),d(lbetai))
c
      close (iout)
      close (isigf)
      close (iflux)
      close (ikin)
      close (iadjoint)
      close (icexss)
      close (icexsp)
      close (itimen)
      close (iflux0)
      close (igen)
c
      return
  900 format(1x,'insufficient memory',2i10)
  940 format(3i5)
  941 format(2i5)
  942 format(7e11.4)
  950 format(1x,'kmax not consistent in interface files in main',2i5)
  960 format(1x,'ngp not consistent in interface files in main',2i5)
      end
c
c
      subroutine mastrho(flux,sigf,alamda,beta,fluxti,sigfti,ncor,
     * vol,con,bin,vel,c,ci,adflux,chi,ds,ids,xs,ys,dp,idp,xp,
```

```
     * yp,delx,scats,scatp,delscat,betaeff,id3s,id3p,id4s,id4p
     * rhoti,gen,rho,betaeffi)
c-----------------------------------------------------------------
c
c     mastrho is the controlling subroutine
c
c-----------------------------------------------------------------
c
      common/flags/irhonum,ipknum
      common/unitsrho/iout,isigf,iflux,ikin,iadjoint,icexss,icexsp,
     * itimen,iflux0,igen
      common/paramrho/kmax,ngp,ntimef,ntimes,namp,nadjph,nadjth,
     * nntmef,nntmes,nnamp,nmat,ndg,nadjn,ngp1,iflag
      dimension flux(ngp,kmax,ntimef),sigf(ngp,nmat,ntimes)
      dimension alamda(ndg),beta(ndg),betaeffi(ndg)
      dimension fluxti(ntimef),sigfti(ntimes),ncor(kmax)
      dimension vol(kmax),con(ntimef),bin(nadjn),vel(ngp+1)
      dimension c(ndg),ci(ndg),adflux(ngp,kmax),chi(ngp)
      dimension ds(6),ids(50),xs(21,5),ys(5,5),dp(6),idp(50)
      dimension xp(21,5),yp(5,5),delx(5,21),scats(nmat,3)
      dimension scatp(nmat,3),delscat(nmat,3),betaeff(irhonum+1,ndg)
      dimension rho(irhonum+1),gen(irhonum+1),rhoti(irhonum+1)
      integer id3s(10,5),id3p(10,5),id4s(6),id4p(6)
      real time,pow
c
c-----input delayed neutron data
c
      read(ikin,945)(alamda(i),i=1,ndg)
      read(ikin,945)(beta(i),i=1,ndg)
c      write(6,*)('ndg = ',ndg)
c      do 801 i=1,ndg
c        write(6,*)('beta from kinetics file = ',beta(i),
c     *  ' for group ',i)
c  801 continue
c
c-----input precursor densities
c
  101 continue
      read(itimen,901,end=201) time,pow,xdum,xdum,xdum
      read(itimen,902,end=201) (c(i),i=1,ndg)
      go to 101
  201 continue
c     write(iout,*)('time of flux shape = ',time)
c     write(iout,*)('amplitude = ',pow)
      do 802 i=1,ndg
c      write(iout,*)('precursor concentrations = ',c(i),
c     * 'for group',i)
  802 continue
c
c-----input steady state adjoint fluxes and angular bins
c
      call inadj(bin,adflux)
c
c-----input cross sections and calculate deltas
c
      rewind icexss
      rewind icexsp
c-----input unperturbed cross-section data
      read(icexss) ids
c      write(iout,*) (ids(j),j=1,14)
      read(icexss) ds
c      write(iout,*) ds
      do 100 i=1,ids(2)
      read (icexss) ids
c      write(iout,10300) ids(19),(ids(n),n=1,18)
c      write(iout,*) (ids(n),n=20,48),ids(50)
10300 format (i2,4x,18a4)
      read(icexss) (id3s(o,i),xs(2*o-1,i),xs(2*o,i),o=1,ids(28))
      do 110 k=1,ids(28)
c      write(iout,*) ('MT = ',id3s(k,i))
c      write(iout,10400) xs(2*k-1,i),xs(2*k,i)
 110  continue
10400 format ('Group 1= ',E16.8,2x,'Group2= ',E16.8)
```

```
      j=0
      l=ids(23)
      read(icexss) id4s
c      write(iout,*) ('Transfer Matrix Pointers are ',id4s)
      read(icexss) (ys(j,i),j=1,ids(23))
c      write(iout,*) ('Scatter Probs are ',(ys(j,i),j=1,ids(23)))
 100  continue
c-----input perturbed cross-section data
      read(icexsp) idp
c      write(iout,*) (idp(j),j=1,14)
      read(icexsp) dp
c      write(iout,*) dp
      do 103 i=1,idp(2)
      read (icexsp) idp
c      write(iout,10300) idp(19),(idp(n),n=1,18)
c      write(iout,*) (idp(n),n=20,48),idp(50)
      read(icexsp) (id3p(o,i),xp(2*o-1,i),xp(2*o,i),o=1,idp(28))
      do 113 k=1,idp(28)
c      write(iout,*) ('MT = ',id3p(k,i))
c      write(iout,10400) xp(2*k-1,i),xp(2*k,i)
 113  continue
      j=0
      l=idp(23)
      read(icexsp) id4p
c      write(iout,*) ('Transfer Matrix Pointers are ',id4s)
      read(icexsp) (yp(j,i),j=1,idp(23))
c      write(iout,*) ('Scatter Probs are ',(yp(j,i),j=1,idp(23)))
 103  continue
c-----calculate delta cross-sections
c-----note that the cross-sections are loaded as follows:
c     i=1,2 for total cross-section (group 1 and 2)
c     i=3,4 for nu*sigf (group 1 and 2)
c     i=5,6 for chi spectrum (group 1 and 2)
c     i=7,8 for absorption cross-section (group 1 and 2)
c     i=9,10 not used
c     i=11,12 for average nu (group 1 and 2)
c     i=13,14 for total non-absorption cross-section (group 1 and 2)
c-----note that the scatter probabilities are loaded as follows:
c     i=1 for group 1 to group 1
c     i=2 for group 1 to group 2
c     i=3 for group 2 to group 2
      i2times = 2*ids(28)
      do 114 i=1,nmat
        do 115 j=1,i2times
          delx(i,j) = xp(j,i) - xs(j,i)
c          write(iout,*)('Delta cross-section for material ',i,
c     *    ' process ',j,' = ',delx(i,j))
  115 continue
  114 continue
c-----construct transfer matrix
      do 126 i=1,nmat
        do 127 j=1,ngp
            scaterms = (xs(j,i) - xs(6+j,i))
            scatermp = (xp(j,i) - xp(6+j,i))
            if(j.eq.1) then
              do 128 k=1,2
              scats(i,k) = scaterms*ys(k,i)
              scatp(i,k) = scatermp*yp(k,i)
  128 continue
              else
              scats(i,3) = scaterms
              scatp(i,3) = scatermp
            end if
  127 continue
  126 continue
      do 129 i=1,nmat
      do 131 k=1,3
c        write(iout,*)('Unperturbed scatter for material ',i,
c     *  ' process ',k,' = ',scats(i,k))
c        write(iout,*)('Perturbed  scatter for material ',i,
c     *  ' process ',k,' = ',scatp(i,k))
  131 continue
  129 continue
```

```
c-----calculate deltas for transfer matrix
      do 132 i=1,nmat
        do 133 j=1,3
          delscat(i,j) = scatp(i,j) - scats(i,j)
c          write(iout,*)('Delta scatter for material ',i,' process ',
c     *     j,' = ',delscat(i,j))
  133 continue
  132 continue
c
c-----input flux shapes and times
c
      call nfluxrho(flux,rhoww,genww)
c
      call gettimrho(fluxti,itimes)
c
      do 830 i=1,itimes
c      write(iout,999)(fluxti(i))
      do 831 j=1,ngp
        do 832 k=1,kmax
c           write(iout,*)('Flux shape for material ',k,' group ',j,
c    *      ' = ',flux(j,k,i))
  832 continue
  831 continue
  830 continue
  999 format(1x,'Fluxti = ',7e11.4)
c     write(6,*)('Time = ',fluxti(nntmef))
      do 833 i=1,ndg
c      write(iout,*)('Delayed neutron fraction for group ',
c     * i,' = ',beta(i))
  833 continue
c
c-----calculate times of reactivity calculations
c
      if (iflag.eq.-2) then
        nntmefp1 = nntmef + 1
      else
        nntmefp1 = nntmef
      end if
c
      if(nntmef.gt.0) then
        rhoti(irhonum+1) = fluxti(nntmefp1)
        do it=1,irhonum
          rhoti(it) = fluxti(nntmefp1 - 1) +
     *     ((fluxti(nntmefp1) - fluxti(nntmefp1 - 1))/
     *     irhonum)*(it-1)
        end do
      else
        write(iout,*)('Not enough data on flux file!')
        stop
      end if
c     write(iout,*)('Time0 = ',fluxti(nntmef - 1))
      do iw=1,irhonum+1
c       write(6,*)('Rhoti = ',rhoti(iw))
      end do
c
c-----input sigf, volumes, chi, times, velocities, and ncor
c
      call nsigfrho(sigf,sigfti,ncor,vol,chi,vel)
c
      do 834 k=1,kmax
c      write(iout,*)('Volume for region ',k,' = ',vol(k))
  834 continue
      do 835 j=1,ngp
c      write(iout,*)('Chi for energy group ',j,' = ',chi(j))
  835 continue
      do 836 j=1,ngp
c      write(iout,*)('Velocity for energy group ',j,' = ',vel(j))
  836 continue
      do 837 k=1,kmax
c      write(iout,*)('Corresponding index # for region ',k,' = ',ncor(k))
  837 continue
      do 838 i=1,nntmes
c         write(iout,*)('Sigf time = ',sigfti(i))
```

```
          do 839 k=1,nmat
            do 840 j=1,ngp
c             write(iout,*)('Nusigf for material ',k,' group ',j,
c     *       ' = ',sigf(j,k,i))
  840 continue
  839 continue
  838 continue
c
c-----begin iteration over shape time step
c
      do 111 i=1,irhonum+1
c
c-----calculate F(t)
c
      fterm = 0.0
      geni = 0.0
      rhoi = 0.0
      t = 0.0
      do iini = 1,ndg
        betaeffi(iini) = 0.0
      end do
      t = rhoti(i)
c
      call calcf(fterm,flux,sigf,ncor,chi,adflux,vol,t,fluxti)
c     write(6,*)('Rhoti = ',t)
c     write(6,*)('F(t) = ',fterm)
c
c-----calculate generation time
c

      call calclam(adflux,flux,vel,geni,fterm,vol,t,fluxti)
      gen(i) = geni
c     write(6,*)('Gen  = ',geni)
c
c-----calculate reactivity
c
      call calcrho(adflux,flux,fterm,delx,delscat,
     * ncor,chi,rhoi,vol,t,fluxti)
      rho(i) = rhoi
c
c-----calculate effective betas
c
      call calcbeta(flux,sigf,ncor,adflux,chi,
     * beta,betaeffi,vol,fterm,t,fluxti)
c
      betaefft = 0.0
      do 181 ib = 1,ndg
        betaeff(i,ib) = betaeffi(ib)
c       write(6,*)('Betaeff = ',betaeff(i,ib),' i & group = ',i,ib)
        betaefft = betaefft + betaeffi(ib)
 181  continue
c     write(6,*)('Betaefft = ',betaefft)
c
c-----calculate effective precursor densities
c
      call calcpd(c,ci,flux,ncor,adflux,chi,vol,fterm,geni)
c
 111  continue
c
c-----end iteration over shape time step
c
c-----write parameters to interface file
c
      call uploadrho(gen,betaeff,rho,c,rhoti)
c
      return
c
c-----End of master
c
 945  format(6e11.4)
 901  format(5e11.4)
 902  format(11x,6e11.4)
      end
```

```
c
      subroutine ofilerho
      common/unitsrho/iout,isigf,iflux,ikin,iadjoint,icexss,icexsp,
     * itimen,iflux0,igen
c
      iout     = 44
      isigf    = 54
      iflux    = 55
      ikin     = 49
      iadjoint = 42
      icexss   = 43
      icexsp   = 2
      itimen   = 41
      igen     = 45
c
c-----open files
c
      open(44,file='rho.out',status='old',form='formatted')
      open(54,file='nusigf.dat',status='old',form='formatted')
      open(55,file='flux.dat',status='old',form='formatted')
      open(49,file='kinetics.dat',status='old',form='formatted')
      open(42,file='ft42f001',status='old',form='unformatted')
      open(43,file='ft43f001',status='old',form='unformatted')
      open(2,file='ft02f001',status='old',form='unformatted')
      open(41,file='time0.dat',status='old',form='formatted')
      open(45,file='general.dat',status='old',form='formatted')
c
      rewind 44
      rewind 54
      rewind 55
      rewind 49
      rewind 42
      rewind 43
      rewind 2
      rewind 41
      rewind 45
      return
      end
c
      subroutine nfluxrho(flux,rhoww,genww)
c---------------------------------------------------------------------
c
c     input flux shapes
c
c---------------------------------------------------------------------
c
      common/flags/irhonum,ipknum
      common/unitsrho/iout,isigf,iflux,ikin,iadjoint,icexss,icexsp,
     * itimen,iflux0,igen
      common/paramrho/kmax,ngp,ntimef,ntimes,namp,nadjph,nadjth,
     * nntmef,nntmes,nnamp,nmat,ndg,nadjn,ngp1,iflag
      dimension flux(ngp,kmax,ntimef),fluxti(ntimef),beta(ndg)
      dimension con(ntimef)
      integer ig,ix
c
      ix = 0
  40  continue
c
          ip1 = ix + 1
          ip2 = ix + 2
          if(ip2 .gt. ntimef)then
              write(iout,910)ntimef
              stop
          end if
c
          read(iflux,970,end=50)fluxti(ip1),rhoww,genww,con(ip1)
          read(iflux,930,end=50)((flux(ig,k,ip1),ig=1,ngp),k=1,kmax)
          read(iflux,930,end=50)(xdum,j=1,ndg)
          ix = ix + 1
          go to 40
  50  continue
c
      nntmef = ix
```

```
      if(nntmef .le. 0) then
          write(iout,990)
          stop
      end if
c
c-----For first pass, linearly extrapolate the flux
c
      if (iflag.eq.-2) then
        nntmefp1 = nntmef + 1
c       write(6,*)('nntmef = ',nntmef)
        do ig=1,ngp
          do k=1,kmax
            flux(ig,k,nntmefp1) = flux(ig,k,nntmef)
          end do
        end do
      end if
c
c
      return
 910  format(1x,'more times on flux.dat than allowed',i11)
 930  format(6e11.4)
 940  format(i5)
 970  format(4e11.4)
 990  format(1x,'no data on flux.dat')
      end
c
      subroutine nsigfrho(sigf,sigfti,ncor,vol,chi,vel)
c-----------------------------------------------------------------------
c
c     input sigf,  volumes
c
c-----------------------------------------------------------------------
c
      common/unitsrho/iout,isigf,iflux,ikin,iadjoint,icexss,icexsp,
     * itimen,iflux0,igen
      common/paramrho/kmax,ngp,ntimef,ntimes,namp,nadjph,nadjth,
     * nntmef,nntmes,nnamp,nmat,ndg,nadjn,ngp1,iflag
      common/flags/irhonum,ipknum
      dimension sigf(ngp,nmat,ntimes),sigfti(ntimes)
      dimension ncor(kmax),vol(kmax),chi(ngp)
      dimension vel(ngp1)
c
      read(isigf,930)(vol(k),k=1,kmax)
      read(isigf,915)(chi(ig),ig=1,ngp)
      read(isigf,916)(vel(ig),ig=1,ngp1)
c     write(6,*)('ngp1 = ',ngp1)
      read(isigf,950)(ncor(k),k=1,kmax)
c
c
      i2 = 0
  30  continue
c
      i = 0
  40  continue
c
          i = i + 1
          if(i .gt. ntimes)then
              write(iout,910)ntimes
              stop
          end if
c
          read(isigf,930)sigfti(i),
     *             ((sigf(ig,m,i),ig=1,ngp),m=1,nmat)
          if(sigfti(i) .lt. 0.0)then
              i = i - 1
              go to 50
          end if
          go to 40
  50  continue
c
c
c-----just read nusigf, go back for sigf
c
```

```
      if(i2 .eq. 0)then
          i2 = 1
          go to 30
      end if
c
      nntmes = i
      if(nntmes .le. 0) then
          write(iout,990)
          stop
      end if
c
c-----convert energy boundaries to velocities
c
      ig = 0
   80 continue
         ig = ig + 1
         eb = sqrt(vel(ig)*vel(ig+1))
         vel(ig) = 1.3859e+6*sqrt(eb)
         if(ig.lt.ngp) then
             go to 80
         end if
c
      return
  910 format(1x,'more times on nusigf.dat than allowed',i11)
  915 format(2e11.4)
  916 format(3e11.4)
  930 format(6e11.4)
  940 format(2i5)
  950 format(12i5)
  990 format(1x,'no data on nusigf.dat')
      end
c
      subroutine inadj(bin,adflux)
      common/unitsrho/iout,isigf,iflux,ikin,iadjoint,icexss,icexsp,
     * itimen,iflux0,igen
      common/paramrho/kmax,ngp,ntimef,ntimes,namp,nadjph,nadjth,
     * nntmef,nntmes,nnamp,nmat,ndg,nadjn,ngp1,iflag
      common/flags/irhonum,ipknum
      dimension bin(nadjn),adflx(ngp*nadjn,kmax)
      dimension adflux(ngp,kmax)
c
      nadjt1 = nadjth + 1
      nadjp1 = nadjph + 1
      read(iadjoint)(bin(i),i=1,nadjp1)
      do 803 i=1,nadjp1
c      write(iout,*)('phi bins = ',bin(i))
  803 continue
      read(iadjoint)(bin(i+nadjp1),i=1,nadjt1)
      do 804 i=1,nadjt1
c      write(iout,*)('theta bins = ',bin(i+nadjp1))
  804 continue
      do 111 k=1,kmax
        do 112 j=1,ngp
           ipoint=(j-1)*nadjn
           read(iadjoint)(adflx(ipoint+i,k),i=1,nadjn)
  112 continue
  111 continue
      do 805 k=1,kmax
        do 806 j=1,ngp
          do 807 i=1,nadjn
             ipoint=(j-1)*nadjn
c            write(iout,*)('Adjoint flux for material ',k,' group ',
c     *        j,' and bin ',i,' = ',adflx(ipoint+i,k))
  807 continue
  806 continue
  805 continue
      do 810 j=1,ngp
        do 811 k=1,kmax
           adflux(j,k)=0.0
  811 continue
  810 continue
c
      do 116 k=1,kmax
```

```
        do 117 j=1,ngp
            ipoint=(j-1)*nadjn
          do 118 i=1,nadjn
            adflux(j,k)=adflux(j,k)+adflx(i+ipoint,k)
  118 continue
c           write(iout,*)('Adjoint flux for ',k,' group ',
c     *      j,' = ',adflux(j,k))
  117 continue
  116 continue
      return
      end
c
      subroutine calcf(fterm,flux,sigf,ncor,chi,adflux,vol,t,
     * fluxti)
      common/paramrho/kmax,ngp,ntimef,ntimes,namp,nadjph,nadjth,
     * nntmef,nntmes,nnamp,nmat,ndg,nadjn,ngp1,iflag
      common/flags/irhonum,ipknum
      dimension flux(ngp,kmax,nntmef+1)
      dimension sigf(ngp,nmat,nntmes),ncor(kmax)
      dimension term1(kmax),term2(kmax*ngp),chi(ngp)
      dimension adflux(ngp,kmax),vol(kmax),fluxti(nntmef+1)
c-----perform the integral over all other energies first
      do 100 i=1,kmax
        term1(i) = 0.0
        do 110 ig=1,ngp
          d = ncor(i)
          call intflxrho(flux,t,xflux,fluxti,ig,i)
          term1(i) = term1(i) + xflux*
     *               sigf(ig,d,nntmes)
 110  continue
 100  continue
c-----multiply by spectrum and adjoint flux
      do 120 ig=1,ngp
        do 130 i=1,kmax
          l = i + kmax*(ig-1)
          term2(l) = 0.0
          term2(l) = term1(i)*chi(ig)*adflux(ig,i)
 130  continue
 120  continue
c-----perform integration over volume and energy
      do 140 ig=1,ngp
        do 150  i=1,kmax
          l = i + kmax*(ig-1)
          fterm  = fterm  + term2(l)*vol(i)
 150  continue
 140  continue
c-----end calculating F(t)
      return
      end
c
      subroutine calclam(adflux,flux,vel,gen,fterm,vol,
     * t,fluxti)
      common/paramrho/kmax,ngp,ntimef,ntimes,namp,nadjph,nadjth,
     * nntmef,nntmes,nnamp,nmat,ndg,nadjn,ngp1,iflag
      common/flags/irhonum,ipknum
      dimension adflux(ngp,kmax),flux(ngp,kmax,nntmef+1)
      dimension vel(ngp+1),term2(kmax*ngp),vol(kmax)
      dimension fluxti(nntmef+1)
c
c-----multiply forward flux and adjoint flux and velocities
      do 120 ig=1,ngp
        do 130 i=1,kmax
          l = i + kmax*(ig-1)
          call intflxrho(flux,t,xflux,fluxti,ig,i)
          term2(l) = 0.0
          term2(l) = (adflux(ig,i)*xflux)/vel(ig)
 130  continue
 120  continue
c-----perform integration over volume and energy
      do 140 ig=1,ngp
        do 150  i=1,kmax
          l = i + kmax*(ig-1)
          gen  = gen + term2(l)*vol(i)
```

```
 150  continue
 140  continue
      gen = gen/fterm
      return
      end
c
      subroutine calcrho(adflux,flux,fterm,delx,delscat,
     * ncor,chi,rho,vol,t,fluxti)
      common/unitsrho/iout,isigf,iflux,ikin,iadjoint,icexss,icexsp,
     * itimen,iflux0,igen
      common/paramrho/kmax,ngp,ntimef,ntimes,namp,nadjph,nadjth,
     * nntmef,nntmes,nnamp,nmat,ndg,nadjn,ngp1,iflag
      common/flags/irhonum,ipknum
      dimension adflux(ngp,kmax),flux(ngp,kmax,nntmef+1)
      dimension delx(nmat,kmax),delscat(nmat,ngp1+1)
      dimension chi(ngp),vol(kmax),fluxti(nntmef+1)
      dimension term1(kmax*ngp),ncor(kmax),term5(kmax*ngp)
      dimension term2(kmax*ngp),term4(kmax),term6(kmax*ngp)
      do 10 ig=1,ngp
        do 20 i=1,kmax
          d = i + kmax*(ig-1)
          id = ncor(i)
          term1(d) = 0.0
          call intflxrho(flux,t,xflux,fluxti,ig,i)
          term1(d) = delx(id,ig)*adflux(ig,i)*xflux
c         write(iout,*)('Term1= ',term1(d),' id= ',id,' k= ',i)
  20  continue
  10  continue
c
      do 21 i=1,nmat
         delscat(i,4) = 0.0
  21  continue
c
      do 30 ig=1,ngp
        do 40 i=1,kmax
          d = i + kmax*(ig-1)
          term2(d) = 0.0
          id = ncor(i)
          call intflxrho(flux,t,xflux,fluxti,ig,i)
         do 50 ig1=1,ngp
          if(ig.eq.1) then
          b = ig1
          else
          b = ig1 + 2
          end if
          term2(d) = term2(d) +
     *       delscat(id,b)*xflux
c         write(iout,*)('Delscat= ',delscat(id,b),'ig= ',ig,'b=',b)
  50  continue
          term2(d) = term2(d)*adflux(ig,i)
c         write(iout,*)('Term2= ',term2(d),' id= ',id,' k= ',i)
  40  continue
  30  continue
c
      do 60 i=1,kmax
        term4(i) = 0.0
        do 70 ig=1,ngp
          id = ncor(i)
          call intflxrho(flux,t,xflux,fluxti,ig,i)
          c = ig + 2
          term4(i) = term4(i) +
     *     delx(id,c)*xflux
  70  continue
c         write(iout,*)('Term4= ',term4(i),' ig= ',ig,' k= ',i)
  60  continue
c
      do 61 ig=1,ngp
        do 71 i=1,kmax
          d = i + kmax*(ig-1)
          term6(d) = 0.0
          term6(d) = term4(i)*chi(ig)*adflux(ig,i)
c         write(iout,*)('Term6= ',term6(d),' ig= ',ig,' k= ',i)
  71  continue
```

```
  61  continue
c
      do 80 ig=1,ngp
         do 90 i=1,kmax
           d = i + kmax*(ig-1)
           term5(d) = 0.0
           term5(d) = - term1(d) + term2(d) + term6(d)
c          write(iout,*)('Term5= ',term5(d),' ig= ',ig,' k= ',i)
  90  continue
  80  continue
c
      do 140 ig=1,ngp
         do 150  i=1,kmax
           l = i + kmax*(ig-1)
           rho  = rho + term5(l)*vol(i)
 150  continue
 140  continue
      rho = rho/fterm
c     write(6,*)('Rho  = ',rho)
c
      return
      end
c
      subroutine calcbeta(flux,sigf,ncor,adflux,chi,
     * beta,betaeff,vol,fterm,t,fluxti)
      common/unitsrho/iout,isigf,iflux,ikin,iadjoint,icexss,icexsp,
     * itimen,iflux0,igen
      common/paramrho/kmax,ngp,ntimef,ntimes,namp,nadjph,nadjth,
     * nntmef,nntmes,nnamp,nmat,ndg,nadjn,ngp1,iflag
      common/flags/irhonum,ipknum
      dimension term1(kmax),flux(ngp,kmax,nntmef+1)
      dimension sigf(ngp,nmat,nntmes),ncor(kmax)
      dimension adflux(ngp,kmax),chi(ngp),term2(ngp*kmax)
      dimension beta(ndg),betaeff(ndg),vol(kmax)
      dimension fluxti(nntmef+1)
      do 10 id=1,ndg
      do 100 i=1,kmax
         term1(i) = 0.0
         do 110 ig=1,ngp
           d = ncor(i)
           call intflxrho(flux,t,xflux,fluxti,ig,i)
           term1(i) = term1(i) + xflux*
     *               sigf(ig,d,nntmes)
 110  continue
 100  continue
c-----multiply by spectrum and adjoint flux
      do 120 ig=1,ngp
         do 130 i=1,kmax
           l = i + kmax*(ig-1)
           term2(l) = 0.0
           term2(l) = term1(i)*chi(ig)*adflux(ig,i)*beta(id)
 130  continue
 120  continue
c
      do 140 ig=1,ngp
         do 150  i=1,kmax
           l = i + kmax*(ig-1)
           betaeff(id)  = betaeff(id) + term2(l)*vol(i)
 150  continue
 140  continue
      betaeff(id) = betaeff(id)/fterm
c     write(6,*)('Betaeff  = ',betaeff(id),' for group ',id)
c
  10  continue
      return
      end
c
      subroutine calcpd(c,ci,flux,ncor,adflux,chi,vol,fterm,gen)
      common/unitsrho/iout,isigf,iflux,ikin,iadjoint,icexss,icexsp,
     * itimen,iflux0,igen
      common/paramrho/kmax,ngp,ntimef,ntimes,namp,nadjph,nadjth,
     * nntmef,nntmes,nnamp,nmat,ndg,nadjn,ngp1,iflag
      common/flags/irhonum,ipknum
```

```
      dimension flux(ngp,kmax,nntmef+1)
      dimension ncor(kmax)
      dimension adflux(ngp,kmax),chi(ngp),term2(ngp*kmax)
      dimension vol(kmax),c(ndg),ci(ndg)
      do 10 id=1,ndg
c-----multiply by spectrum and adjoint flux
      do 120 ig=1,ngp
        do 130 i=1,kmax
          l = i + kmax*(ig-1)
          term2(l) = 0.0
          term2(l) = chi(ig)*adflux(ig,i)*c(id)
 130  continue
 120  continue
c
      do 140 ig=1,ngp
        do 150  i=1,kmax
          l = i + kmax*(ig-1)
          ci(id)  = ci(id) + term2(l)*vol(i)
 150  continue
 140  continue
      ci(id) = ci(id)/(fterm*gen)
c     write(6,*)('C  = ',c(id),' for group ',id)
c
  10  continue
      return
      end
c
c
      subroutine uploadrho(gen,betaeff,rho,c,rhoti)
      common/unitsrho/iout,isigf,iflux,ikin,iadjoint,icexss,icexsp,
     * itimen,iflux0,igen
      common/paramrho/kmax,ngp,ntimef,ntimes,namp,nadjph,nadjth,
     * nntmef,nntmes,nnamp,nmat,ndg,nadjn,ngp1,iflag
      common/flags/irhonum,ipknum
      dimension betaeff(irhonum+1,ndg),c(ndg),rhoti(irhonum+1)
      dimension gen(irhonum+1),rho(irhonum+1)
      rewind iout
      if (iflag.eq.-1) then
       iflag = -2
      else
       iflag = -1
      end if
      write(iout,*) iflag
      do 100 i=1,irhonum+1
        write(iout,800) rhoti(i),gen(i),rho(i)
        write(iout,810)(c(in),in=1,ndg)
        write(iout,810)(betaeff(i,in),in=1,ndg)
 100  continue
c
 800  format(3e14.7)
 810  format(6e14.7)
      return
      end
c
      subroutine intflxrho(flux,t,x,fluxti,ig,k)
c-----------------------------------------------------------------
c
c     interpolate flux values
c
c-----------------------------------------------------------------
c
      common/paramrho/kmax,ngp,ntimef,ntimes,namp,nadjph,nadjth,
     * nntmef,nntmes,nnamp,nmat,ndg,nadjn,ngp1,iflag
      common/flags/irhonum,ipknum
      dimension flux(ngp,kmax,nntmef+1)
      dimension fluxti(nntmef+1)
c
      if (iflag.eq.-2) then
        nntmefp1 = nntmef + 1
      else
        nntmefp1 = nntmef
      end if
c
```

```
      if(nntmef.le.0)then
        write(iout,*)('Not enough data on flux file!')
        stop
      else
        x=flux(ig,k,nntmefp1-1)+(flux(ig,k,nntmefp1)-
     *    flux(ig,k,nntmefp1-1))*
     *    (t-fluxti(nntmefp1-1))/(fluxti(nntmefp1)-
     *    fluxti(nntmefp1-1))
      end if
      return
      end
c
      subroutine gettimrho(fluxti,itimes)
      common/unitsrho/iout,isigf,iflux,ikin,iadjoint,icexss,icexsp,
     * itimen,iflux0,igen
      common/paramrho/kmax,ngp,ntimef,ntimes,namp,nadjph,nadjth,
     * nntmef,nntmes,nnamp,nmat,ndg,nadjn,ngp1,iflag
      common/flags/irhonum,ipknum
      dimension fluxti(ntimef)
      integer itimes
c-----------------------------------------------------------------
c     gettim reads tstop,delt, tin of general interface file
c-----------------------------------------------------------------
c
c-----initial variables
c
      rewind igen
c
c-----skip  section #1
c
      read(igen,902)isec,nrec
      do 20 ic=1,nrec
          read(igen,900)
   20 continue
c
c-----skip section #2 of general interface file
c
      read(igen,902)isec,nrec
      do 30 ic=1,nrec
          read(igen,902)
   30 continue
c
c-----skip section 3 of interface file
c
      read(igen,902)isec,nrec
      do 50 ic=1,nrec
          read(igen,902)
   50 continue
c
c-----read time data  section #4
c
      read(igen,902)isec,nrec
      itimes = nrec
      n=0
  100 continue
      read(igen,903)fluxti(n+1)
      n=n+1
      nlft=nrec-n
      if(nlft.gt.0)go to 100
c
  900 format(80a1)
  902 format(15i5)
  903 format(7e11.4)
  904 format(1x,21hend of problem  time=,f7.2)
c
      return
      end
c
c
*****************************************************************
*                                                               *
*                     SUBROUTINE PTKIN                          *
*                                                               *
```

```
*****************************************************************
      subroutine ptkin
c-----------------------------------------------------------------
c
c     point kinetics code- no feedback
c
c-----------------------------------------------------------------
      common/flags/irhonum,ipknum
      common/rdata/rho(1000+1),gen(1000+1),pow,time,
     * betat(1000+1),tfuel,
     * tcool1,tcool2,tin,rhon,genn,betatn,iter,
     * tfuel0,tcol10,tcol20,pow0,rhoti(1000+1),
     * betaeff(1000+1,100)
      common/idata/inn,iout,ikin,itime0,ndg,itimen,igen,iflux,irho
      common/betdat/beta(100),alamda(100)
      external f
      double precision y,rtol,atol,tout,dtime,rwork,delt
c
      dimension c(100),c0(100),y(200)
      dimension rwork(500),iwork(100)
      integer iflag,iter
      data itol/1/
      data rtol/1.0d-04/
      data atol/0.0d0/
      data itask/1/
      data istate/1/
      data iopt/0/
      data lrw/500/
      data liw/100/
      data mf/22/
c
c-----assign logical unit #'s and open files
c
      call ofilepk
c
c-----input kinetics data
c
      call nkinpk(iflag,c)
c
c      write(iout,*)('iflag = ',iflag)
       do 313 i=1,irhonum+1
c         write(iout,998)rhoti(i),gen(i),rho(i),betat(i)
c         write(iout,999)(betaeff(i,in),in=1,ndg)
c         write(iout,999)(c(in),in=1,ndg)
 313  continue
 998  format(4e14.7)
 999  format(6e14.7)
c
c-----get power, c's, and last time
c
      call getpowpk(c,c0)
c     write(iout,999)(c0(in),in=1,ndg)
c
c
c-----calculate system equation constants and initial conditions
c
      neq=1+ndg
      call calconpk(c,c0)
c
c-----initialize
c
      rhon   = rho(1)
      genn   = gen(1)
      betatn = betat(1)
      delt   = (rhoti(2) - rhoti(1))/ipknum
      tout   = time + delt
c
c-----load dependent variables into y array and normalize
c
      call dloadpk(y,neq,c,c0)
c
c-----begin iteration over tj points
c
```

```
      do 100 iter=1,irhonum
c
      tk     = rhoti(iter + 1)
      delt   = (rhoti(iter + 1) - rhoti(iter))/ipknum
      tstop  = rhoti(irhonum+1)
      betatn = betat(iter)
c
c      write(iout,*)('iter = ',iter)
c      write(iout,997)time,tstop,delt,tout
c 997  format('Time,tstop,delt,tout = ',4e11.4)
c
c-----integrate
c
  170 continue
c
c-----time must be double precision
c
      dtime = time
c
c
      call lsode(f,neq,y,dtime,tout,itol,rtol,atol,itask,istate,
     * iopt,rwork,lrw,iwork,liw,jac,mf)
c
      time = dtime
      if(istate.lt.0)go to 775
c
c-----unload y array and unnormalize
c
      call unloadpk(y,neq,c)
c
c-----save to time file
c
      call savpowpk(c)
c
c-----interpolate point kinetics parameters
c
      if(time.ge.rhoti(1)) then
         call intrhopk(xnew,time)
         if (xnew.ge.0.0) then
             rhon = xnew
         end if
         call intgenpk(xnew,time)
         if (xnew.ge.0.0) then
            genn = xnew
         end if
      end if
c      write(iout,*)('Time  = ',time,' Reactivity = ',rhon)
c      write(iout,*)('Betatn = ',betatn,' Gen = ',genn)
c      write(iout,*)('Beta  = ',betaeff(iter,ig),ig=1,ndg)
c      write(iout,*)('lamda = ',alamda(ig),ig=1,ndg)
c
      if(time.ge.tstop)go to 700
      if(time.ge.tk)go to 100
      tout = time + delt
      go to 170
c
  100 continue
c
c
  700 continue
c
c
      write(iout,701)tstop
  701 format(1x,'ptkin finished at time=',e11.4)
      write(iout,702)pow
  702 format(5x,'last power=',e11.4)
c
c
      close (iout)
      close (ikin)
      close (itime0)
      close (itimen)
      close (igen)
```

```
      close (iflux)
      close (irho)
c
c
      return
  775 continue
      write(iout,776)istate
  776 format(1x,'error......istate=',i4)
      stop
      end
c
c
      subroutine ofilepk
c-----------------------------------------------------------------
c
c     ofile assigns logical unit # & open files
c
c-----------------------------------------------------------------
      common/idata/inn,iout,ikin,itime0,ndg,itimen,igen,iflux,irho
c
      igen   = 45
      itime0 = 41
      itimen = 46
      iout   = 40
      ikin   = 49
      iflux  = 55
      irho   = 44
c
      open(40,file='ptkin.out',status='unknown',form='formatted')
      open(45,file='general.dat',status='old',form='formatted')
      open(41,file='time0.dat',status='old',form='formatted')
      open(46,file='timen.dat',status='new',form='formatted')
      open(49,file='kinetics.dat',status='old',form='formatted')
      open(55,file='flux.dat',status='old',form='formatted')
      open(44,file='rho.out',status='old',form='formatted')
c
      rewind 40
      rewind 45
      rewind 41
      rewind 46
      rewind 49
      rewind 55
      rewind 44
c
      return
      end
      subroutine nkinpk(iflag,c)
c-----------------------------------------------------------------
c     nkin reads kinetics data
c-----------------------------------------------------------------
c
      common/flags/irhonum,ipknum
      common/betdat/beta(100),alamda(100)
      common/rdata/rho(1000+1),gen(1000+1),pow,time,
     * betat(1000+1),tfuel,
     * tcool1,tcool2,tin,rhon,genn,betatn,iter,
     * tfuel0,tcol10,tcol20,pow0,rhoti(1000+1),
     * betaeff(1000+1,100)
      common/idata/inn,iout,ikin,itime0,ndg,itimen,igen,iflux,irho
      dimension c(100)
      integer iflag
c
c
      read(ikin,802)ndg
      read(ikin,803)(alamda(i),i=1,ndg)
c
c
c-----input effective beta's at latest time step
c
      read(iflux,940)kmax,ngp
c
      rewind irho
      read(irho,*)iflag
```

```
      do 113 i=1,irhonum+1
          read(irho,930)rhoti(i),gen(i),rho(i)
          read(irho,940)(c(in),in=1,ndg)
          read(irho,940)(betaeff(i,j),j=1,ndg)
 113  continue
c
c
c-----calculate total beta
c
      do 101 i=1,irhonum+1
        do 100 j=1,ndg
           betat(i) = betat(i) + betaeff(i,j)
  100 continue
  101 continue
c
 802  format(10i5)
 803  format(7e11.4)
 930  format(3e14.7)
 940  format(6e14.7)
 970  format(4e11.4)
 990  format(1x,'effective betas not found on flux.dat file')
      return
      end
c
      subroutine getpowpk(c,c0)
c-----------------------------------------------------------------
c     getpow reads initial conditions of time file
c-----------------------------------------------------------------
c
      common/rdata/rho(1000+1),gen(1000+1),pow,time,
     * betat(1000+1),tfuel,
     * tcool1,tcool2,tin,rhon,genn,betatn,iter,
     * tfuel0,tcol10,tcol20,pow0,rhoti(1000+1),
     * betaeff(1000+1,100)
      common/flags/irhonum,ipknum
      common/idata/inn,iout,ikin,itime0,ndg,itimen,igen,iflux,irho
      dimension c0(100),c(100),c2(100)
c
c-----read variables at last time and save on new time file
c
      rewind itime0
      rewind itimen
c
  100 continue
      read(itime0,900,end=200)time,pow,tfuel,tcool1,tcool2
      write(itimen,900)time,pow,tfuel,tcool1,tcool2
      read(itime0,907,end=200)(c2(i),i=1,ndg)
      write(itimen,907)(c2(i),i=1,ndg)
      go to 100
c
  200 continue
c
c-----initial variables
c
      if (time.eq.0.0) then
         time = (rhoti(2) - rhoti(1))/irhonum
      end if
      pow0 = pow
      tfuel0 = tfuel
      tcol10 = tcool1
      tcol20 = tcool2
      do 300 i=1,ndg
          c0(i) = c(i)
  300 continue
c
  901 format(1e11.4)
  900 format(7e11.4)
  907 format(11x,6e11.4)
      return
      end
c
c
      subroutine gettim(tstop,delt)
```

```
c-----------------------------------------------------------------
c     gettim reads tstop,delt, tin of general interface file
c-----------------------------------------------------------------
c
      common/idata/inn,iout,ikin,itime0,ndg,itimen,igen,iflux,irho
c
c-----initial variables
c
      rewind igen
c
c-----skip  section #1
c
      read(igen,902)isec,nrec
      do 20 ic=1,nrec
          read(igen,900)
   20 continue
c
c-----skip section #2 of general interface file
c
      read(igen,902)isec,nrec
      do 30 ic=1,nrec
          read(igen,902)
   30 continue
c
c-----skip section 3 of interface file
c
      read(igen,902)isec,nrec
      do 50 ic=1,nrec
          read(igen,902)
   50 continue
c
c-----read t/h data  section #4
c
      read(igen,902)isec,nrec
      n=0
      read(igen,903)tin
      n=n+1
      nlft=nrec-n
      if(nlft.le.0)go to 100
      do 60 ic=1,nlft
          read(igen,902)
   60 continue
c
c-----read time data  section #5
c
  100 continue
      read(igen,902)isec,nrec
      read(igen,902)ntime
      do 200 ic=1,ntime
          read(igen,903)tstop,delt
          if(time.lt.tstop)go to 300
  200 continue
c
c-----end of problem
c
      write(iout,904)time
      return
c
  900 format(80a1)
  902 format(15i5)
  903 format(7e11.4)
  904 format(1x,21hend of problem  time=,f7.2)
c
  300 continue
      return
      end
c
      subroutine calconpk(c,c0)
c-----------------------------------------------------------------
c     calcon calculates equation constants and initial conditions.
c-----------------------------------------------------------------
c
      common/rdata/rho(1000+1),gen(1000+1),pow,time,
```

```
     * betat(1000+1),tfuel,
     * tcool1,tcool2,tin,rhon,genn,betatn,iter,
     * tfuel0,tcol10,tcol20,pow0,rhoti(1000+1),
     * betaeff(1000+1,100)
      common/flags/irhonum,ipknum
      common/betdat/beta(100),alamda(100)
      common/idata/inn,iout,ikin,itime0,ndg,itimen,igen,iflux,irho
      common/thdata/xmassf,cpf,fract,h,af,xmdot,xmassc,cpc,alphac,
     * alphaf,haf,hafmcc,hafmcf,powbtu,cc,dd,bb
      dimension c(100),c0(100)
c-----power is in mw, to convert to btu/sec mult by following
      data conp2b/948.05/
c
c-----initial c's
c
      if(time.gt.0.0)go to 200
      do 100 i=1,ndg
          c(i) = pow0*beta(i)/(alamda(i)*gen(1))
          c0(i) = c(i)
  100 continue
c
c
  200 continue
c
      return
      end
c
      subroutine dloadpk(y,neq,c,c0)
c---------------------------------------------------------------
c     dload places dependent variables in y array
c---------------------------------------------------------------
c
      common/rdata/rho(1000+1),gen(1000+1),pow,time,
     * betat(1000+1),tfuel,
     * tcool1,tcool2,tin,rhon,genn,betatn,iter,
     * tfuel0,tcol10,tcol20,pow0,rhoti(1000+1),
     * betaeff(1000+1,100)
      common/flags/irhonum,ipknum
      common/idata/inn,iout,ikin,itime0,ndg,itimen,igen,iflux,irho
      double precision y
      dimension y(101),c(100),c0(100)
c
c-----y(1)=power/initial power
c
      y(1) = pow/pow0
c
      do 100 i=1,ndg
          y(i+1) = c(i)/pow0
  100 continue
c
      return
      end
c
c
      subroutine f(neq,t,y,ydot)
c---------------------------------------------------------------
c     f is called by lsode to determine derivatives of the
c     dependent variables
c---------------------------------------------------------------
c
      common/betdat/beta(100),alamda(100)
      common/rdata/rho(1000+1),gen(1000+1),pow,time,
     * betat(1000+1),tfuel,
     * tcool1,tcool2,tin,rhon,genn,betatn,iter,
     * tfuel0,tcol10,tcol20,pow0,rhoti(1000+1),
     * betaeff(1000+1,100)
      common/flags/irhonum,ipknum
      common/idata/inn,iout,ikin,itime0,ndg,itimen,igen,iflux,irho
      common/thdata/xmassf,cpf,fract,h,af,xmdot,xmassc,cpc,alphac,
     * alphaf,haf,hafmcc,hafmcf,powbtu,cc,dd,bb
      double precision y,ydot
      dimension y(101),ydot(101)
c
```

```
      sum=0.0
      do 100 i=1,ndg
          sum = sum + alamda(i)*y(i+1)
  100 continue
      ydot(1) = (rhon-betatn)*y(1)/genn + sum
c
c-----equations 2-ndg+1: precursors
c
      do 200 i=1,ndg
          ydot(i+1) = betaeff(iter,i)*y(1)/genn - alamda(i)*y(i+1)
  200 continue
c
c      write(iout,*)('iter = ',iter,'Time = ',t)
c      write(iout,999)(rhon,genn,betatn,sum)
c      write(iout,998)(alamda(ig),ig=1,ndg)
c      write(iout,998)(betaeff(iter,ig),ig=1,ndg)
  998 format(6e17.4)
  999 format(4e17.4)
      return
      end
c
      subroutine unloadpk(y,neq,c)
c-----------------------------------------------------------------
c     unload takes dependent variables out of y array
c-----------------------------------------------------------------
c
      common/idata/inn,iout,ikin,itime0,ndg,itimen,igen,iflux,irho
      common/rdata/rho(1000+1),gen(1000+1),pow,time,
     * betat(1000+1),tfuel,
     * tcool1,tcool2,tin,rhon,genn,betatn,iter,
     * tfuel0,tcol10,tcol20,pow0,rhoti(1000+1),
     * betaeff(1000+1,100)
      common/flags/irhonum,ipknum
      double precision y
      dimension y(101),c(100)
c
c-----y(1)=power/initial power
c
      pow = pow0*y(1)
c
      do 100 i=1,ndg
          c(i) = y(i+1)*pow0
  100 continue
c
      return
      end
c
c
      subroutine savpowpk(c)
c-----------------------------------------------------------------
c     savpow writes dependent variables to the time file
c-----------------------------------------------------------------
c
      common/rdata/rho(1000+1),gen(1000+1),pow,time,
     * betat(1000+1),tfuel,
     * tcool1,tcool2,tin,rhon,genn,betatn,iter,
     * tfuel0,tcol10,tcol20,pow0,rhoti(1000+1),
     * betaeff(1000+1,100)
      common/flags/irhonum,ipknum
      common/idata/inn,iout,ikin,itime0,ndg,itimen,igen,iflux,irho
      dimension c(100)
      data iline/0/
c
c
c-----save variables
c
      write(itimen,900)time,pow,tfuel,tcool1,tcool2
      write(itimen,907)(c(i),i=1,ndg)
  900 format(7e11.4)
  907 format(11x,6e11.4)
c
c
      iline = iline + 2
```

```
      if(iline .gt. 20000)then
          write(iout,700)
  700     format(1x,'exceeded 20000 lines in ptkin')
          stop
      end if
c
      return
      end
c
      subroutine intrhopk(xnew,tk)
      common/rdata/rho(1000+1),gen(1000+1),pow,time,
     * betat(1000+1),tfuel,
     * tcool1,tcool2,tin,rhon,genn,betatn,iter,
     * tfuel0,tcol10,tcol20,pow0,rhoti(1000+1),
     * betaeff(1000+1,100)
      common/flags/irhonum,ipknum
      common/idata/inn,iout,ikin,itime0,ndg,itimen,igen,iflux,irho
      integer imethod
      real x0,x1,x2,xnew,t0,t1,t2,tk,dt1,dt2,dtk,dtk2
c
c imethod = 1 if linear interpolation
c imethod = 2 if quadratic interpolation
c x0,x1,x2 are last three values of parameter being interpolated
c t0,t1,t2 are the associated times
c xnew is the interpolated value
c tk is the time the parameter is being interpolated at
c
      if (iter.eq.1) then
        imethod = 1
        t0   = rhoti(iter)
        t1   = rhoti(iter+1)
        x0   = rho(iter)
        x1   = rho(iter+1)
      else
        imethod = 2
        t0   = rhoti(iter-1)
        t1   = rhoti(iter)
        t2   = rhoti(iter+1)
        x0   = rho(iter-1)
        x1   = rho(iter)
        x2   = rho(iter+1)
      end if
c
      dt1 = t1 - t0
      dt2 = t2 - t1
      dtk = tk - t0
      dtk2 = tk - t1
      if(imethod.eq.1) then
      xnew = x0 + ((x1 - x0)*dtk)/(dt1)
      end if
      if(imethod.eq.2) then
      xnew = x0 + ((x1 - x0)*(dtk))/(dt1) + ((dtk)*(dtk2))*
     * ((((x2 - x1)/dt2) - ((x1 - x0)/dt1))/(dt2 + dt1))
      end if
c     write(6,*) (' xnew = ',xnew,' x0 = ',x0,
c    * ' x1 = ',x1,' x2 = ',x2)
c     write(6,*) (' tk = ',tk,' t0 = ',t0,' t1 = ',t1,
c    * ' t2 = ',t2)
      return
      end
c
      subroutine intgenpk(xnew,tk)
      common/rdata/rho(1000+1),gen(1000+1),pow,time,
     * betat(1000+1),tfuel,
     * tcool1,tcool2,tin,rhon,genn,betatn,iter,
     * tfuel0,tcol10,tcol20,pow0,rhoti(1000+1),
     * betaeff(1000+1,100)
      common/flags/irhonum,ipknum
      common/idata/inn,iout,ikin,itime0,ndg,itimen,igen,iflux,irho
      integer imethod
      real x0,x1,x2,xnew,t0,t1,t2,tk,dt1,dt2,dtk,dtk2
c
c imethod = 1 if linear interpolation
```

```
c imethod = 2 if quadratic interpolation
c x0,x1,x2 are last three values of parameter being interpolated
c t0,t1,t2 are the associated times
c xnew is the interpolated value
c tk is the time the parameter is being interpolated at
c
      if (iter.eq.1) then
        imethod = 1
        t0   = rhoti(iter)
        t1   = rhoti(iter+1)
        x0   = gen(iter)
        x1   = gen(iter+1)
      else
        imethod = 2
        t0   = rhoti(iter-1)
        t1   = rhoti(iter)
        t2   = rhoti(iter+1)
        x0   = gen(iter-1)
        x1   = gen(iter)
        x2   = gen(iter+1)
      end if
c
      dt1 = t1 - t0
      dt2 = t2 - t1
      dtk = tk - t0
      dtk2 = tk - t1
      if(imethod.eq.1) then
      xnew = x0 + ((x1 - x0)*dtk)/(dt1)
      end if
      if(imethod.eq.2) then
      xnew = x0 + ((x1 - x0)*(dtk))/(dt1) + ((dtk)*(dtk2))*
     * ((((x2 - x1)/dt2) - ((x1 - x0)/dt1))/(dt2 + dt1))
      end if
c     write(6,*) (' xnew = ',xnew,' x0 = ',x0,
c    * ' x1 = ',x1,' x2 = ',x2)
c     write(6,*) (' tk = ',tk,' t0 = ',t0,' t1 = ',t1,
c    * ' t2 = ',t2)
      return
      end
*********************************************************************
*                                                                   *
*                      SUBROUTINE NORM                              *
*                                                                   *
*********************************************************************
      subroutine norm
c-------------------------------------------------------------------
c
c     merge & reformat flux output of keno into flux shape
c     interface file.
c
c-------------------------------------------------------------------
      dimension flux(300,300)
      dimension beta(20)
      data lim/20/
      data ifluxi/48/
      data ifluxo/55/
      data inew/57/
      data iout/56/
      data ikin/49/
c
c
c-----open files
c
      open(56,file='norm.out',status='unknown',form='formatted')
      open(49,file='kinetics.dat',status='old',form='formatted')
      open(48,file='ft48f001',status='old',form='unformatted')
      open(55,file='flux.dat',status='old',form='formatted')
      open(57,file='fluxnew.dat',status='new',form='formatted')
c
      rewind 56
      rewind 49
      rewind 48
      rewind 57
```

```
      rewind 55
c
c
c-----input no. of delayed neutron groups
      read(ikin,940)ndg
      if (ndg .gt. lim)then
          write(iout,980)ndg,lim
          stop
      end if
c
      read(ifluxi)kmax,ngp
      read(ifluxi)time2
c
      if(kmax.gt.300)then
          write(iout,900)kmax
          stop
      end if
c
      if(ngp.gt.300)then
          write(iout,910)ngp
          stop
      end if
c
      read(ifluxo,940)kmaxo,ngpo
c
      if(kmax.ne.kmaxo)then
          write(iout,950)kmax,kmaxo
          stop
      end if
c
      if(ngp.ne.ngpo)then
          write(iout,960)ngp,ngpo
          stop
      end if
c
c-----transfer data from old flux.dat to new flux.dat.
c     first const on file will be the constant value of the integral.
c
      write(inew,940)kmax,ngp
  40  continue
          read(ifluxo,970,end=50)time,rho,xgen,const
c
c---------replace time step (iteration), do not append time step
c
          iyes = 1
          if(time2 .eq. time)then
              iyes = 0
          end if
c
          if(iyes .eq. 1)then
              write(inew,970)time,rho,xgen,const
          end if
          read(ifluxo,930,end=50)((flux(ig,k),ig=1,ngp),k=1,kmax)
          if(iyes .eq. 1)then
              write(inew,930)((flux(ig,k),ig=1,ngp),k=1,kmax)
          end if
          read(ifluxo,930,end=50)(beta(j),j=1,ndg)
          if(iyes .eq. 1)then
              write(inew,930)(beta(j),j=1,ndg)
          end if
          go to 40
  50  continue
c
 100  continue
c
c-----input fluxes
c
          read(ifluxi)ig,ll,lu
          if(ig .eq. -1)then
              go to 200
          end if
          read(ifluxi)(flux(ig,k),k=ll,lu)
          go to 100
```

```
c
 200  continue
c
c-----output latest data
c
      read(ifluxi)rho,xgen,const
      write(inew,970)time2,rho,xgen,const
      write(inew,930)((flux(ig,k),ig=1,ngp),k=1,kmax)
      read(ifluxi)(beta(j),j=1,ndg)
      write(inew,930)(beta(j),j=1,ndg)
c
      write(iout,1000)time2
1000  format(1x,'flux.dat file updated by norm, time=',e11.4)
      write(iout,1100)rho,xgen
1100  format(1x,'reactivity=',e11.4,/,1x,'generation time=',e11.4)
      write(iout,1200)const
1200  format(1x,'constraint constant=',e11.4)
c
      close (56)
      close (49)
      close (48)
      close (57)
      close (55)
c
      return
 900  format(1x,'kmax exceeds 300, kmax=',i7)
 910  format(1x,'ngp exceeds 300, ngp=',i7)
 920  format(1x,'time=',e11.4)
 930  format(6e11.4)
 940  format(2i5)
 950  format(1x,'kmax not consistent in interface files in norm',2i5)
 960  format(1x,'ngp not consistent in interface files in norm',2i5)
 970  format(4e11.4)
 980  format(1x,'ndg exceeds lim, ndg,lim=',2i5)
      end
**********************************************************************
*                                                                    *
*                     SUBROUTINE CDELAY                              *
*                                                                    *
**********************************************************************
      subroutine cdelay
c---------------------------------------------------------------------
c
c     calculate delayed neutron shape
c
c     nntmef = no. of time steps on flux.dat
c     nntmes = no. of time steps on nusigf.dat
c     nnpow  = no. of time steps on timen.dat
c     ndglim = max no of delayed neutron groups
c     ntimef = max no of flux shape calcs
c     ntimes = max no of times on nusigf.dat file
c     npow   = max no of times on timen.dat file
c     ntegr  = max no of integrand values
c---------------------------------------------------------------------
c
      common/unitscd/iout,ikin,isigf,itime,iflux,idelay
      common/paramcd/kmax,ngp,ntimef,ntimes,npow,ndglim,ntegr,
     * nntmef,nntmes,nnpow,ndg,nmat
      dimension d(100000)
      dimension beta(20),alamda(20)
      data limmem/100000/
c
      ndglim = 20
      ntimef = 40
      ntimes = 20
      npow = 10000
      ntegr = 3000
c
      call ofilecd
c
      read(iflux,940)kmax,ngp
c
      read(isigf,940)kmax0,ngp0,nmat
```

```
      if(kmax0 .ne. kmax)then
          write(iout,950)kmax,kmax0
          stop
      end if
      if(ngp0 .ne. ngp)then
          write(iout,960)ngp,ngp0
          stop
      end if
c
c
c-----generate array pointers
      ngp1 = ngp + 1
      lflux = 1
      lfluxt = lflux + kmax*ngp*ntimef
      lsigf = lfluxt + ntimef
      lsigft = lsigf + nmat*ngp*ntimes
      lncor = lsigft + ntimes
      lpow  = lncor + kmax
      lpowt = lpow + npow
      lchi = lpowt + npow
      lntegr = lchi + ngp*ndglim
      ltime = lntegr + ntegr
      lqd = ltime + ntegr
      lqdk = lqd + kmax*ngp
      lqdkig = lqdk + kmax
      lvol = lqdkig + kmax*ngp
      lvel = lvol + kmax
      lf = lvel + ngp1
      lfk = lf + kmax*ngp
      lfkig = lfk + kmax
      iend = lfkig + kmax*ngp
      if(iend .gt.limmem)then
          write(iout,900)limmem,iend
          stop
      end if
c
      call master1(d(lflux),d(lfluxt),d(lsigf),d(lpow),d(lchi),
     * d(lntegr),d(ltime),d(lqd),d(lsigft),d(lpowt),beta,alamda,
     * d(lncor),d(lqdk),d(lqdkig),d(lvol),d(lvel),d(lf),
     * d(lfk),d(lfkig),ngp1)
c
      close (iout)
      close (ikin)
      close (isigf)
      close (itime)
      close (iflux)
      close (idelay)
c
      return
 900  format(1x,'insufficient memory',2i10)
 940  format(3i5)
 950  format(1x,'kmax not consistent in interface files in main',2i5)
 960  format(1x,'ngp not consistent in interface files in main',2i5)
      end
      subroutine master1(flux,fluxti,sigf,pow,chi,xntegr,time,qd,
     * sigfti,powti,beta,alamda,ncor,qdk,
     * qdkig,vol,vel,f,fk,fkig,ngp1)
c-----------------------------------------------------------------
c
c     master1 is the controlling subroutine
c
c-----------------------------------------------------------------
c
      common/unitscd/iout,ikin,isigf,itime,iflux,idelay
      common/paramcd/kmax,ngp,ntimef,ntimes,npow,ndglim,ntegr,
     * nntmef,nntmes,nnpow,ndg,nmat
      dimension flux(ngp,kmax,ntimef),sigf(ngp,nmat,ntimes)
      dimension pow(npow),xntegr(ntegr),time(ntegr),qd(ngp,kmax)
      dimension beta(ndglim),alamda(ndglim),chi(ngp,ndglim)
      dimension fluxti(ntimef),sigfti(ntimes),powti(npow),ncor(kmax)
      dimension qdk(kmax),qdkig(ngp,kmax),vol(kmax),vel(ngp1)
      dimension f(ngp,kmax),fk(kmax),fkig(ngp,kmax)
      logical istop
```

```
      data delmin/.001/
c
c-----input no. of delayed neutron groups
      read(ikin,940)ndg
      if (ndg .gt. ndglim)then
          write(iout,980)ndg,ndglim
          stop
      end if
      read(ikin,803)(alamda(i),i=1,ndg)
      read(ikin,803)(beta(i),i=1,ndg)
c
c-----input flux shapes
      call nflux(flux,fluxti)
c
c-----input nusigf & chi's & velocities & volumes
      call nsigf(sigf,chi,sigfti,ncor,vol,vel,ngp1)
c
c-----input powers
c     xtime = last time on timen.dat file and will be time integrated
c        to
c     power = last amplitude on timen.dat file
      call inpow(pow,xtime,powti,power)
c
c
      do 55 k=1,kmax
          qdk(k) = 0.0
          fk(k) = 0.0
          do 50 ig=1,ngp
              qd(ig,k) = 0.0
              qdkig(ig,k) = 0.0
              f(ig,k) = 0.0
              fkig(ig,k) = 0.0
  50      continue
  55  continue
c
      do 95 k=1,kmax
       m = ncor(k)
       do 91 j=1,ndg
c
c-----steady-state fissions (assumes first time step on timen.dat,
c       flux.dat, nusigf.dat files is steady-state.)
c
          ssfiss = 0.0
          do 80 ig=1,ngp
              ssfiss = ssfiss + sigf(ig,m,1)*pow(1)*flux(ig,k,1)
  80      continue
c
          do 90 ignew=1,ngp
c---------------following assumes all chi's are the same for
c                   all j's
              qd(ignew,k) = qd(ignew,k) + ssfiss*chi(ignew,1)*
     *              alamda(j)*beta(j)*exp(-alamda(j)*xtime)
  90      continue
c
  91   continue
  95  continue
c
c
c-----integrate
      do 500 j=1,ndg
          deltat = delmin
          t = xtime
          i = 1
          time(i) = t
          xntegr(i) = 1.0
          istop = .false.
c
c
 100      continue
              t = t - deltat
c
              if(t .le. 0.0)then
                  t = 0.0
```

```
                istop = .true.
            end if
c
            i = i + 1
c
            if(i .gt. ntegr)then
                write(iout,990)ntegr
                stop
            end if
c
            xntegr(i) = exp(-alamda(j)*(xtime-t))
c
            trial = abs(xntegr(i)-xntegr(i-1))
            if(trial .gt. .005)then
                deltat = deltat/2.0
                i = i - 1
            else if(trial .lt. .001)then
                if(.not. istop)then
                    deltat = deltat*2.0
                    i = i - 1
                endif
            else
                time(i) = t
            end if
c
            if(istop)then
                if(i .eq. 1)then
                    i = 2
                    time(i) = t
                end if
                go to 200
            else
                go to 100
            end if
 200      continue
c
c-----add times from timen.dat file
      ii = i + nnpow
c
      if(ii .gt. ntegr)then
          write(iout,990)ntegr
          stop
      end if
c
      call addmor(time,i,powti,xntegr,alamda(j),xtime)
c
c
c-----finished with time mesh for group j
c     now fold other terms of the integrand in
c
          do 450 k=1,kmax
            sum = 0.0
            do 400 ig=1,ngp
c
              do 300 ii=1,i
c
c-----------------interpolate values
c
                  call intflx(flux,time(ii),xflux,fluxti,ig,k)
c
                  call intsig(sigf,time(ii),xsig,sigfti,ig,k,ncor)
c
                  call intpow(pow,time(ii),xpow,powti)
c
                  fact = xflux*xsig*xpow*alamda(j)*beta(j)
                  xntegr(ii) = fact*exp(-alamda(j)*(xtime-time(ii)))
 300          continue
c
              do 310 ii=2,i
c
c-----------------integrate using trapezoid rule
c
c
```

```
                del = abs(time(ii)-time(ii-1))/2.0
                sum = sum + (xntegr(ii) + xntegr(ii-1))*del
 310          continue
 400        continue
c
            do 410 ignew=1,ngp
c---------------following assumes all chi's are the same for
c                all j's
               qd(ignew,k) = qd(ignew,k) + sum*chi(ignew,1)
 410        continue
 450      continue
 500  continue
c
c
c-----calculate (1/vdelt)*flux(t-delt) part of shape derivative.
c
      call fluxsh(flux,fluxti,f,vel,ngp1,xtime)
c
c
c-----d = ratio of delayed & flux shift source neutrons to total source
c-----fs = ratio of flux shift source neutrons to delayed + flux shift
c          source neutrons
c
      fs = 0.0
      d = 0.0
      do 520 k=1,kmax
          sumf = 0.0
          sum = 0.0
          do 510 ig=1,ngp
              sumf = sumf + f(ig,k)
              sum = sum + qd(ig,k)
 510      continue
          fs = fs + sumf*vol(k)
          d = d + sum*vol(k)
 520  continue
c
c
c-----calculate pdf's for spatial & energy distributions of fixed
c        source neutrons
c
      do 540 k=1,kmax
          sumf = 0.0
          sum = 0.0
          do 530 ig=1,ngp
              sumf = sumf + f(ig,k)
              sum = sum + qd(ig,k)
 530      continue
          do 535 ig=1,ngp
              if(sum .gt. 0.0)then
                  qdkig(ig,k) = qd(ig,k)/sum
              end if
              if(sumf .gt. 0.0)then
                  fkig(ig,k) = f(ig,k)/sumf
              end if
 535      continue
          if(d .gt. 0.0)then
              qdk(k) = sum*vol(k)/d
          end if
          if(fs .gt. 0.0)then
              fk(k) = sumf*vol(k)/fs
          end if
 540  continue
      d = d/pow(nnpow)
c
c
c-----Flux used to calculate prompt fission neutrons is that at the
c     latest time on flux.dat file.
c
      betat = 0.0
      do 550 j=1,ndg
          betat = betat + beta(j)
 550  continue
c
```

```
      xtime2 = fluxti(nntmef)
      p = 0.0
c
      do 700 k=1,kmax
          sum = 0.0
          do 600 ig=1,ngp
              call intsig(sigf,xtime2,xsig,sigfti,ig,k,ncor)
              sum = sum + chi(ig,1)*(1.0-betat)*xsig*flux(ig,k,nntmef)
 600      continue
          p = p + sum*vol(k)
 700  continue
c
      r1 = (d + fs)/(d + fs + p)
      r2 = fs/(d + fs)
      r3 = d/p
      write(idelay,940)kmax,ngp
      write(idelay,930)r1,r2,r3,fluxti(nntmef)
      write(idelay,930)(qdk(k),k=1,kmax)
      write(idelay,930)((qdkig(ig,k),ig=1,ngp),k=1,kmax)
      write(idelay,930)(fk(k),k=1,kmax)
      write(idelay,930)((fkig(ig,k),ig=1,ngp),k=1,kmax)
c
      write(iout,1000)xtime
      write(iout,1100)r1
      write(iout,1200)r2
      Write(iout,1300)r3
c
 803  format(7e11.4)
 930  format(6e11.4)
 940  format(2i5)
 980  format(1x,'ndg exceeds lim, ndg,lim=',2i5)
 990  format(1x,'number of intrgrand values exceeds limit',i6)
1000  format(1x,'cdelay calc. ended at time=',e11.4)
1100  format(1x,'ratio of fixed source to total source=',e11.4)
1200  format(1x,'ratio of flux shift to total fixed source=',e11.4)
1300  format(1x,'ratio of delayed n. source to prompt source=',e11.4)
      return
      end
      subroutine ofilecd
      common/unitscd/iout,ikin,isigf,itime,iflux,idelay
c
      iout = 59
      ikin = 49
      isigf = 54
      itime = 46
      iflux = 55
      idelay = 47
c
c
c-----open files
c
      open(59,file='cdelay.out',status='new',form='formatted')
      open(49,file='kinetics.dat',status='old',form='formatted')
      open(54,file='nusigf.dat',status='old',form='formatted')
      open(46,file='timen.dat',status='old',form='formatted')
      open(55,file='flux.dat',status='old',form='formatted')
      open(47,file='delay.dat',status='unknown',form='formatted')
      rewind 59
      rewind 49
      rewind 54
      rewind 46
      rewind 55
      rewind 47
c
      return
      end
c
      subroutine nflux(flux,fluxti)
c-----------------------------------------------------------------
c
c     input flux shapes
c
c-----------------------------------------------------------------
```

```
c
      common/unitscd/iout,ikin,isigf,itime,iflux,idelay
      common/paramcd/kmax,ngp,ntimef,ntimes,npow,ndglim,ntegr,
     * nntmef,nntmes,nnpow,ndg,nmat
      dimension flux(ngp,kmax,ntimef),fluxti(ntimef)
c
c
      i = 0
  40  continue
c
          ip1 = i + 1
          if(ip1 .gt. ntimef)then
              write(iout,910)ntimef
              stop
          end if
c
          read(iflux,970,end=50)fluxti(ip1),rho,gen,xdum
          read(iflux,930,end=50)((flux(ig,k,ip1),ig=1,ngp),k=1,kmax)
          read(iflux,930,end=50)(xdum,j=1,ndg)
          i = i + 1
          go to 40
  50  continue
c
      nntmef = i
      if(nntmef .le. 0) then
          write(iout,990)
          stop
      end if
c
c
      if(fluxti(1) .ne. 0.0) then
          write(iout,980)
          stop
      end if
c
c
      return
 910  format(1x,'more times on flux.dat than allowed',i11)
 930  format(6e11.4)
 970  format(4e11.4)
 980  format(1x,'first time on flux.dat file must be 0')
 990  format(1x,'no data on flux.dat')
      end
c
      subroutine nsigf(sigf,chi,sigfti,ncor,vol,vel,ngp1)
c-----------------------------------------------------------------
c
c     input nusigf, chi, volumes, and velocities
c
c-----------------------------------------------------------------
c
      common/unitscd/iout,ikin,isigf,itime,iflux,idelay
      common/paramcd/kmax,ngp,ntimef,ntimes,npow,ndglim,ntegr,
     * nntmef,nntmes,nnpow,ndg,nmat
      dimension sigf(ngp,nmat,ntimes),sigfti(ntimes),chi(ngp,ndglim)
      dimension ncor(kmax),vel(ngp1),vol(kmax)
c
c-----following assumes all chi's are the same for
c        all j's
      read(isigf,930)(vol(k),k=1,kmax)
      read(isigf,930)(chi(ig,1),ig=1,ngp)
      read(isigf,930)(vel(ig),ig=1,ngp1)
      read(isigf,950)(ncor(k),k=1,kmax)
c
      i = 0
  40  continue
c
          i = i + 1
          if(i .gt. ntimes)then
              write(iout,910)ntimes
              stop
          end if
c
```

```
          read(isigf,930)sigfti(i),
     *             ((sigf(ig,m,i),ig=1,ngp),m=1,nmat)
          if(sigfti(i) .lt. 0.0)then
              i = i - 1
              go to 50
          end if
          go to 40
   50  continue
c
       nntmes = i
       if(nntmes .le. 0) then
           write(iout,990)
           stop
       end if
c
c-----convert energies to velocities
c
       ig = 0
   80  continue
          ig = ig + 1
          eb = sqrt(vel(ig)*vel(ig+1))
          vel(ig) = 1.3859e+6*sqrt(eb)
          if(ig .lt. ngp)then
              go to 80
          end if
c
       return
  910  format(1x,'more times on nusigf.dat than allowed',i11)
  930  format(6e11.4)
  940  format(2i5)
  950  format(12i5)
  990  format(1x,'no data on nusigf.dat')
       end
c
       subroutine inpow(pow,xtime,powti,power)
       common/unitscd/iout,ikin,isigf,itime,iflux,idelay
       common/paramcd/kmax,ngp,ntimef,ntimes,npow,ndglim,ntegr,
      * nntmef,nntmes,nnpow,ndg,nmat
       dimension pow(npow),powti(npow)
c
       i = 0
   100 continue
c
           ip1 = i + 1
           if(ip1 .gt. npow)then
               write(iout,910)npow
               stop
           end if
c
           read(itime,900,end=200)powti(ip1),pow(ip1),tfuel,
     *      tcool1,tcool2
           read(itime,907,end=200)(xdum,j=1,ndg)
           i = ip1
       go to 100
c
   200 continue
c
       nnpow = i
       if(nnpow .le. 0) then
           write(iout,990)
           stop
       end if
c
       xtime = powti(nnpow)
       power = pow(nnpow)
c
c
  900  format(7e11.4)
  907  format(11x,6e11.4)
  910  format(1x,'more times on timen.dat than allowed',i11)
  990  format(1x,'no data on timen.dat')
       return
       end
```

```
      subroutine addmor(time,i,powti,xntegr,alamda,xtime)
c-----------------------------------------------------------------
c
c     add times from timen.dat file
c
c-----------------------------------------------------------------
c
      common/paramcd/kmax,ngp,ntimef,ntimes,npow,ndglim,ntegr,
     * nntmef,nntmes,nnpow,ndg,nmat
      dimension xntegr(ntegr),time(ntegr)
      dimension powti(npow)
      logical isort
c
      if(nnpow .le. 1)then
          i = i + 1
          time(i) = powti(1)
      else
          do 100 j=1,nnpow
              time(i+j) = powti(j)
 100      continue
          i = i + nnpow
      end if
c
c-----sort times
c
 150  continue
      isort = .false.
      do 200 j=2,i
          if(time(j) .lt. time(j-1))then
               a = time(j)
               time(j) = time(j-1)
               time(j-1) = a
               isort = .true.
          end if
 200  continue
c
      if(isort)then
          go to 150
      end if
c
c-----times have been sorted
c
      do 300 ii=1,i
          xntegr(ii) = exp(-alamda*(xtime-time(ii)))
 300  continue
c
      return
      end
      subroutine intflx(flux,t,x,fluxti,ig,k)
c-----------------------------------------------------------------
c
c     interpolate flux values
c
c-----------------------------------------------------------------
c
      common/paramcd/kmax,ngp,ntimef,ntimes,npow,ndglim,ntegr,
     * nntmef,nntmes,nnpow,ndg,nmat
      dimension flux(ngp,kmax,ntimef)
      dimension fluxti(ntimef)
c
      if(nntmef .le. 1)then
          x = flux(ig,k,1)
      else
          do 100 i=1,nntmef
              if(t .le. fluxti(i))then
                  go to 200
              end if
 100      continue
c
c---------time is > than any time on flux file. set equal to
c           flux at last flux file time step
c
          x = flux(ig,k,nntmef)
```

```
          go to 300
c
 200      continue
          if(i .le. 1)then
              x = flux(ig,k,1)
          else
              x = flux(ig,k,i-1) + (flux(ig,k,i) - flux(ig,k,i-1))*
     *            (t - fluxti(i-1))/(fluxti(i) - fluxti(i-1))
          end if
 300      continue
c
       end if
       return
       end
       subroutine intsig(sigf,t,x,sigfti,ig,k,ncor)
c-----------------------------------------------------------------
c
c      interpolate nusigf values
c
c-----------------------------------------------------------------
c
       common/paramcd/kmax,ngp,ntimef,ntimes,npow,ndglim,ntegr,
      * nntmef,nntmes,nnpow,ndg,nmat
       dimension sigf(ngp,nmat,ntimes)
       dimension sigfti(ntimes),ncor(kmax)
c
       m = ncor(k)
       if(nntmes .le. 1)then
           x = sigf(ig,m,1)
       else
           do 100 i=1,nntmes
               if(t .le. sigfti(i))then
                   go to 200
               end if
 100       continue
c
c---------time is > than any time on sigf file. set equal to
c          sigf at last sigf file time step
c
           x = sigf(ig,m,nntmes)
           go to 300
c
 200       continue
           if(i .le. 1)then
               x = sigf(ig,m,1)
           else
               x = sigf(ig,m,i-1) + (sigf(ig,m,i) - sigf(ig,m,i-1))*
     *             (t - sigfti(i-1))/(sigfti(i) - sigfti(i-1))
           end if
 300       continue
c
       end if
       return
       end
       subroutine intpow(pow,t,x,powti)
c-----------------------------------------------------------------
c
c      interpolate power values
c
c-----------------------------------------------------------------
c
       common/paramcd/kmax,ngp,ntimef,ntimes,npow,ndglim,ntegr,
      * nntmef,nntmes,nnpow,ndg,nmat
       dimension pow(npow)
       dimension powti(npow)
c
       if(nnpow .le. 1)then
           x = pow(1)
       else
           do 100 i=1,nnpow
               if(t .le. powti(i))then
                   go to 200
               end if
```

```
 100      continue
c
c---------time is > than any time on time file. set equal to
c           pow at last time file time step
c
          x = pow(nnpow)
          go to 300
c
 200      continue
          if(i .le. 1)then
              x = pow(1)
          else
              x = pow(i-1) + (pow(i) - pow(i-1))*
     *            (t - powti(i-1))/(powti(i) - powti(i-1))
          end if
 300      continue
c
      end if
      return
      end
      subroutine fluxsh(flux,fluxti,f,vel,ngp1,xtime)
c-----------------------------------------------------------------
c
c     calculate (1/(delt*vel)) * flux(t-delt)
c      t = xtime = last time on timen.dat file
c      t - delt = last time on flux.dat file
c
c-----------------------------------------------------------------
c
      common/paramcd/kmax,ngp,ntimef,ntimes,npow,ndglim,ntegr,
     * nntmef,nntmes,nnpow,ndg,nmat
      common/unitscd/iout,ikin,isigf,itime,iflux,idelay
      dimension flux(ngp,kmax,ntimef)
      dimension fluxti(ntimef)
      dimension f(ngp,kmax),vel(ngp1)
c
c
      delt = xtime - fluxti(nntmef)
c     write(6,*)('delt =',delt)
      if(delt .le. 0.0)then
          write(iout,950)
c         stop
      end if
c
      do 500 k=1,kmax
          do 400 ig=1,ngp
              f(ig,k) = 0.0
              f(ig,k) = (1.0/vel(ig))*flux(ig,k,nntmef)/delt
c
 400      continue
 500  continue
c
c
 950  format(1x,'delt le 0 in fluxsh')
      return
      end
*****************************************************************
*                                                               *
*                     SUBROUTINE POWER                          *
*                                                               *
*****************************************************************
      subroutine power
c-----------------------------------------------------------------
c
c     calculate power from amplitude and shape functions
c     also determine normalized flux distribution
c
c     nntmef = no. of time steps on flux.dat
c     nntmes = no. of time steps on nusigf.dat
c     nnamp  = no. of time steps on timen.dat
c     ntimef = max no of flux shape calcs
c     ntimes = max no of times on nusigf.dat file
c     namp   = max no of times on timen.dat file
```

```
c-----------------------------------------------------------------
c
      common/units2/inn,iout,isigf,itime,iflux,ikin,ifluxo
      common/param2/kmax,ngp,ntimef,ntimes,namp,
     * nntmef,nntmes,nnamp,nmat,ndg
      dimension d(50000)
      data limmem/50000/
c
      ntimef = 40
      ntimes = 20
      namp = 10000
c
      call ofiles2
c
      read(iflux,940)kmax,ngp
c
      read(isigf,940)kmax0,ngp0,nmat
      if(kmax0 .ne. kmax)then
          write(iout,950)kmax,kmax0
          stop
      end if
      if(ngp0 .ne. ngp)then
          write(iout,960)ngp,ngp0
          stop
      end if
c
c
c-----generate array pointers
      ngp1 = ngp + 1
      lflux = 1
      lfluxt = lflux + kmax*ngp*ntimef
      lsigf = lfluxt + ntimef
      lsigft = lsigf + nmat*ngp*ntimes
      lncor = lsigft + ntimes
      lamp  = lncor + kmax
      lampt = lamp + namp
      lvol = lampt + namp
      lcon = lvol + kmax
      iend = lcon + ntimef
      if(iend .gt.limmem)then
          write(iout,900)limmem,iend
          stop
      end if
c
      call master2(d(lflux),d(lfluxt),d(lsigf),d(lamp),
     * d(lsigft),d(lampt),d(lncor),d(lvol),d(lcon),ngp1)
c
      close (inn)
      close (iout)
      close (isigf)
      close (itime)
      close (iflux)
      close (ikin)
      close (ifluxo)
c
      return
 900  format(1x,'insufficient memory',2i10)
 940  format(3i5)
 950  format(1x,'kmax not consistent in interface files in main',2i5)
 960  format(1x,'ngp not consistent in interface files in main',2i5)
      end
      subroutine master2(flux,fluxti,sigf,amp,
     * sigfti,ampti,ncor,vol,con,ngp1)
c-----------------------------------------------------------------
c
c     master2 is the controlling subroutine
c
c-----------------------------------------------------------------
c
      common/units2/inn,iout,isigf,itime,iflux,ikin,ifluxo
      common/param2/kmax,ngp,ntimef,ntimes,namp,
     * nntmef,nntmes,nnamp,nmat,ndg
      dimension flux(ngp,kmax,ntimef),sigf(ngp,nmat,ntimes)
```

```
      dimension amp(namp)
      dimension fluxti(ntimef),sigfti(ntimes),ampti(namp),ncor(kmax)
      dimension vol(kmax),con(ntimef)
c
c-----input flux shapes
      call nflux2(flux,fluxti,con)
c
c-----input sigf & volumes
      call nsigf2(sigf,sigfti,ncor,vol,ngp1)
c
c-----input amplitudes
      call inamp(amp,ampti)
c
c-----determine power normalization factor
c      defaults to 1.0 if the file power.in does not exist.
c
      powini = 1.0
      read(inn,803,end=50)powini
  50  continue
c
      p = 0.0
      do 200 k=1,kmax
          sum = 0.0
          do 100 ig=1,ngp
              m = ncor(k)
              sum = sum + sigf(ig,m,1)*flux(ig,k,1)
 100      continue
          p = p + sum*vol(k)
 200  continue
      p = p*amp(1)
      xnormp = powini/p
c
c-----determine powers
c
      do 500 i=1,nnamp
          p = 0.0
          do 400 k=1,kmax
              sum = 0.0
              do 300 ig=1,ngp
c
c-----------------interpolate values
c
                  call intflx2(flux,ampti(i),xflux,fluxti,ig,k,con,xc)
                  call intsig2(sigf,ampti(i),xsig,sigfti,ig,k,ncor)
                  xnormc = con(1)/xc
c
                  sum = sum + xsig*xflux
 300          continue
              p = p + sum*vol(k)
 400      continue
          pow = p*amp(i)*xnormc*xnormp
          write(iout,803)ampti(i),pow,amp(i)
 500  continue
c
c-----determine normalized flux distribution
c
      do 800 i=1,nntmef
          call intamp(amp,fluxti(i),x,ampti)
          do 700 k=1,kmax
              do 600 ig=1,ngp
                  flux(ig,k,i) = flux(ig,k,i)*x*xnormp*con(1)/con(i)
 600          continue
 700      continue
 800  continue
c
c-----output flux distribution
      call oflux(flux,fluxti)
c
 803  format(3e11.4)
      return
      end
      subroutine ofiles2
      common/units2/inn,iout,isigf,itime,iflux,ikin,ifluxo
```

```
c
      inn = 63
      iout =  60
      isigf = 54
      itime = 46
      iflux = 55
      ikin  = 49
      ifluxo = 62
c
c-----open files
c
      open(63,file='power.in',status='unknown',form='formatted')
      open(60,file='power.out',status='new',form='formatted')
      open(54,file='nusigf.dat',status='old',form='formatted')
      open(46,file='timen.dat',status='old',form='formatted')
      open(55,file='flux.dat',status='old',form='formatted')
      open(49,file='kinetics.dat',status='old',form='formatted')
      open(62,file='flux.out',status='new',form='formatted')
c
      rewind 63
      rewind 60
      rewind 54
      rewind 46
      rewind 55
      rewind 49
      rewind 62
c
      return
      end
c
      subroutine nflux2(flux,fluxti,con)
c---------------------------------------------------------------
c
c     input flux shapes
c
c---------------------------------------------------------------
c
      common/units2/inn,iout,isigf,itime,iflux,ikin,ifluxo
      common/param2/kmax,ngp,ntimef,ntimes,namp,
     * nntmef,nntmes,nnamp,nmat,ndg
      dimension flux(ngp,kmax,ntimef),fluxti(ntimef)
      dimension con(ntimef)
c
c-----input no. of delayed neutron groups
      read(ikin,940)ndg
c
      i = 0
  40  continue
c
          ip1 = i + 1
          if(ip1 .gt. ntimef)then
              write(iout,910)ntimef
              stop
          end if
c
          read(iflux,970,end=50)fluxti(ip1),rho,gen,con(ip1)
          read(iflux,930,end=50)((flux(ig,k,ip1),ig=1,ngp),k=1,kmax)
          read(iflux,930,end=50)(xdum,j=1,ndg)
          i = i + 1
          go to 40
  50  continue
c
      nntmef = i
      if(nntmef .le. 0) then
          write(iout,990)
          stop
      end if
c
      return
 910  format(1x,'more times on flux.dat than allowed',i11)
 930  format(6e11.4)
 940  format(i5)
 970  format(4e11.4)
```

```
 990  format(1x,'no data on flux.dat')
      end
c
      subroutine nsigf2(sigf,sigfti,ncor,vol,ngp1)
c-----------------------------------------------------------------
c
c     input sigf,  volumes
c
c-----------------------------------------------------------------
c
      common/units2/inn,iout,isigf,itime,iflux,ikin,ifluxo
      common/param2/kmax,ngp,ntimef,ntimes,namp,
     * nntmef,nntmes,nnamp,nmat,ndg
      dimension sigf(ngp,nmat,ntimes),sigfti(ntimes)
      dimension ncor(kmax),vol(kmax)
c
      read(isigf,930)(vol(k),k=1,kmax)
      read(isigf,930)(xdum,ig=1,ngp)
      read(isigf,930)(xdum,ig=1,ngp1)
      read(isigf,950)(ncor(k),k=1,kmax)
c
      i2 = 0
  30  continue
c
      i = 0
  40  continue
c
          i = i + 1
          if(i .gt. ntimes)then
              write(iout,910)ntimes
              stop
          end if
c
          read(isigf,930)sigfti(i),
     *            ((sigf(ig,m,i),ig=1,ngp),m=1,nmat)
          if(sigfti(i) .lt. 0.0)then
              i = i - 1
              go to 50
          end if
          go to 40
  50  continue
c
c-----just read nusigf, go back for sigf
c
      if(i2 .eq. 0)then
          i2 = 1
          go to 30
      end if
c
      nntmes = i
      if(nntmes .le. 0) then
          write(iout,990)
          stop
      end if
c
      return
 910  format(1x,'more times on nusigf.dat than allowed',i11)
 930  format(6e11.4)
 940  format(2i5)
 950  format(12i5)
 990  format(1x,'no data on nusigf.dat')
      end
c
      subroutine inamp(amp,ampti)
c-----------------------------------------------------------------
c
c     input amplitude function
c
c-----------------------------------------------------------------
c
      common/units2/inn,iout,isigf,itime,iflux,ikin,ifluxo
      common/param2/kmax,ngp,ntimef,ntimes,namp,
     * nntmef,nntmes,nnamp,nmat,ndg
```

```
      dimension amp(namp),ampti(namp)
c
      i = 0
  100 continue
c
          ip1 = i + 1
          if(ip1 .gt. namp)then
              write(iout,910)namp
              stop
          end if
c
          read(itime,900,end=200)ampti(ip1),amp(ip1),tfuel,
     *      tcool1,tcool2
          read(itime,907,end=200)(xdum,j=1,ndg)
          i = ip1
      go to 100
c
  200 continue
c
      nnamp = i
      if(nnamp .le. 0) then
          write(iout,990)
          stop
      end if
c
c
 900  format(7e11.4)
 907  format(11x,6e11.4)
 910  format(1x,'more times on timen.dat than allowed',i11)
 990  format(1x,'no data on timen.dat')
      return
      end
      subroutine intflx2(flux,t,x,fluxti,ig,k,con,xc)
c-----------------------------------------------------------------
c
c     interpolate flux values
c
c-----------------------------------------------------------------
c
      common/param2/kmax,ngp,ntimef,ntimes,namp,
     * nntmef,nntmes,nnamp,nmat,ndg
      dimension flux(ngp,kmax,ntimef)
      dimension fluxti(ntimef),con(ntimef)
c
      if(nntmef .le. 1)then
          x = flux(ig,k,1)
          xc = con(1)
      else
          do 100 i=1,nntmef
              if(t .le. fluxti(i))then
                  go to 200
              end if
 100      continue
c
c---------time is > than any time on flux file. set equal to
c          flux at last flux file time step
c
          x = flux(ig,k,nntmef)
          xc = con(nntmef)
          go to 300
c
 200      continue
          if(i .le. 1)then
              x = flux(ig,k,1)
              xc = con(1)
          else
              x = flux(ig,k,i-1) + (flux(ig,k,i) - flux(ig,k,i-1))*
     *            (t - fluxti(i-1))/(fluxti(i) - fluxti(i-1))
              xc = con(i-1) + (con(i) - con(i-1))*
     *            (t - fluxti(i-1))/(fluxti(i) - fluxti(i-1))
          end if
 300      continue
c
```

```
      end if
      return
      end
      subroutine intsig2(sigf,t,x,sigfti,ig,k,ncor)
c-----------------------------------------------------------------
c
c     interpolate nusigf values
c
c-----------------------------------------------------------------
c
      common/param2/kmax,ngp,ntimef,ntimes,namp,
     * nntmef,nntmes,nnamp,nmat,ndg
      dimension sigf(ngp,nmat,ntimes)
      dimension sigfti(ntimes),ncor(kmax)
c
      m = ncor(k)
      if(nntmes .le. 1)then
          x = sigf(ig,m,1)
      else
          do 100 i=1,nntmes
              if(t .le. sigfti(i))then
                  go to 200
              end if
 100      continue
c
c---------time is > than any time on sigf file. set equal to
c           sigf at last sigf file time step
c
          x = sigf(ig,m,nntmes)
          go to 300
c
 200      continue
          if(i .le. 1)then
              x = sigf(ig,m,1)
          else
              x = sigf(ig,m,i-1) + (sigf(ig,m,i) - sigf(ig,m,i-1))*
     *            (t - sigfti(i-1))/(sigfti(i) - sigfti(i-1))
          end if
 300      continue
c
      end if
      return
      end
      subroutine oflux(flux,fluxti)
c-----------------------------------------------------------------
c
c     output flux distribution
c
c-----------------------------------------------------------------
c
      common/units2/inn,iout,isigf,itime,iflux,ikin,ifluxo
      common/param2/kmax,ngp,ntimef,ntimes,namp,
     * nntmef,nntmes,nnamp,nmat,ndg
      dimension flux(ngp,kmax,ntimef),fluxti(ntimef)
c
c
      do 200 i=1,nntmef
          do 100 ig=1,ngp
              write(ifluxo,910)fluxti(i),ig
              write(ifluxo,930)(flux(ig,k,i),k=1,kmax)
 100      continue
 200  continue
c
      return
 910  format(1x,'flux at time=',1pe10.4,4x,'group=',i3)
 930  format(6e11.4)
      end
      subroutine intamp(amp,t,x,ampti)
c-----------------------------------------------------------------
c
c     interpolate amplitude values
c
c-----------------------------------------------------------------
```

```
c
      common/param2/kmax,ngp,ntimef,ntimes,namp,
     * nntmef,nntmes,nnamp,nmat,ndg
      dimension amp(namp),ampti(namp)
c
      if(nnamp .le. 1)then
          x = amp(1)
      else
          do 100 i=1,nnamp
              if(t .le. ampti(i))then
                  go to 200
              end if
 100      continue
c
c---------time is > than any time on time file. set equal to
c           amp at last time file time step
c
          x = amp(nnamp)
          go to 300
c
 200      continue
          if(i .le. 1)then
              x = amp(1)
          else
              x = amp(i-1) + (amp(i) - amp(i-1))*
     *            (t - ampti(i-1))/(ampti(i) - ampti(i-1))
          end if
 300      continue
c
      end if
      return
      end
*****************************************************************
*                                                               *
*                     SUBROUTINE KENOVA                         *
*                                                               *
*****************************************************************
      subroutine kenova
c
c     book 1
c
c###############################################################################
c
c                audit trail information
c
c     date the module was last permanently updated:      none
c     time the module was last permanently updated:      none
c     programmer name:                                   n.f.landers
c     module name:                                       k5kenova
c     current archiving level number:                    00000
c     current number of permanent updates:               00000
c     date of last access by librarian:                  91/10/18
c     dateset name:  zaz.ncss.master
c
c###############################################################################
c.....................................................................
      common /unit/ inpt, outpt, icexs, albdo, wts, skrt, rstrt, wstrt,
     1 ampxs, direct(3), i0, i1, i2, i3, i4, nspare(4)
      integer outpt,albdo,skrt,rstrt,wstrt,direct,wts,icexs,ampxs
c.....................................................................
c.....................................................................
      common/runtyp/bfalse,btrue,bin
      integer bin
      logical btrue,bfalse
c.....bfalse=true for standard keno input
c.....btrue=true for binary input
c.....bin=binary input unit (unit=95)
c.....................................................................
      external master

cmfe
c     call link('print6//')
cmfe
```

```
      call opnfil(outpt, 'new', 'formatted', '$out')
      call opnfil(inpt ,'old','formatted','input')
      rewind(inpt)
      bfalse = .true.
      btrue  = .false.
      call inital(lng)
      call alocat(master,lng,*100)
      call closda(direct(1))
      call closda(direct(2))
      call closda(direct(3))
      close(inpt)
      return
  100 continue
      call stop ('keno message number  k5-213'//
     *           '   *** error ***  insufficient space allocation.'//
     *           '  master was not called.$',162,0,1)
      return
      end
c
c
c
      subroutine cleanup

      character*60 link1,link2,link3,link4,link5,link6,link7
      character*60 link8,link9,link10,link11,link12,link13
      character*60 link14,link15,link16,link17,link18
      character*60 link19,link20,link21
c
      link1='cat _out* > keno.out               '
      link2='cat infile >> outfile              '
      link3='cat input input.adj >> outfile     '
      link4='rm ft08f001 ft09f001 ft14f001      '
      link19='rm ft46f001 ft47f001 ft49f001     '
      link5='rm _out* _P* tdkeno tdkenoadj      '
      link20='cat tmeout >> outfile             '
      link21='rm tmeout                         '
      link7='mv rho.dat output                  '
      link8='mv cdelay.dat output               '
      link9='cp flux.dat output                 '
      link10='mv power.out output               '
      link11='mv timen.dat output               '
      link12='rm kinetics.dat delay.dat         '
      link13='mv flux.out output                '
      link14='rm general.dat                    '
      link15='mv power.in output                '
      link16='rm keno.meth rho.out              '
      link17='rm time0.dat nusigf.dat           '
      link18='mv keno.out output                '
      link6='mv outfile output                  '
c
      status=system(link1)
      status=system(link2)
      status=system(link3)
      status=system(link4)
      status=system(link19)
      status=system(link5)
      status=system(link20)
      status=system(link21)
      status=system(link7)
      status=system(link8)
      status=system(link9)
      status=system(link10)
      status=system(link11)
      status=system(link12)
      status=system(link13)
      status=system(link14)
      status=system(link15)
      status=system(link16)
      status=system(link17)
      status=system(link18)
      status=system(link6)
      return
      end
```

Appendix B:

TDKENO-M User's Manual

TDKENO-M User's Manual

by

Charles L. Bentley

The University of Tennessee, Knoxville

Department of Nuclear Engineering

December 1996

Table of Contents

1. Introduction

TDKENO-M is a Fortran code which solves the time-dependent, three-dimensional Boltzmann transport equation with explicit representation of delayed neutrons as given by equations 1 and 2. The code consists of several large subroutines which perform specific functions as described below:

1. KENO V.a — Modified to perform a fixed source calculation and to explicitly track delayed neutrons based on the distibution function given by equation 18. KENO V.a was also modified to include the flux shape derivative with the fixed source terms using a backwards difference approximation of first order (equation 19).

2. RHO — Determines the point kinetics parameters (reactivity, generation time and effective delayed neutron fraction) from their inner product definitions as given by equations 9 through 15.

3. PTKIN — Solves the point kinetics equations using the Livermore Solver of Ordinary Differential Equations (LSODE).

4. CDELAY — Calculates the delayed neutron distribution (equation 18) as well as the cumulative distribution functions used to incorporate the flux shape derivative.

5. POWER — Determines the power history using equation 22.

The calculational flow as well as the subroutines mentioned above are described in greater detail in the subsequent chapters of the user's manual.

2. Calculational Flow

The calculational flow for a typical TDKENO-M calculation is given in Figure 1. Initially, a steady-state adjoint calculation is performed using the modified version of KENO V.a. This steady state adjoint flux is used for all subsequent calculations of the point kinetics parameters as well as for the calculation of the system power. Next, a steady-state forward KENO V.a calculation is performed to determine the initial flux shape as well as the effective multiplication factor and the constraint integral (equation 7). Next, subroutines RHO and PTKIN are called to determine the initial values of the point kinetics parameters and the flux amplitude, respectively. In the case of thermalhydraulic feedback (i.e., Benchmark 14-A2), subroutine FEEDBACK is called to calculate the system temperature using the adiabatic heatup model described in Appendix C. The subroutine CDELAY is then called by TDKENO-M to determine the delayed neutron source distribution as well as the cumulative distribution functions needed for the implementation of the flux shape derivative for the transient calculations.

TDKENO-M next calls KENO V.a to perform the first flux shape calculation of the perturbed system. Subsequently, subroutines RHO, PTKIN and CDELAY are called again to update the system parameters and the amplitude as described above. This sequence is continued until the desired cutoff time is reached for the transient calculation. Upon the completion of the transient calculation, subroutine POWER is used to determine the power trace for the system.

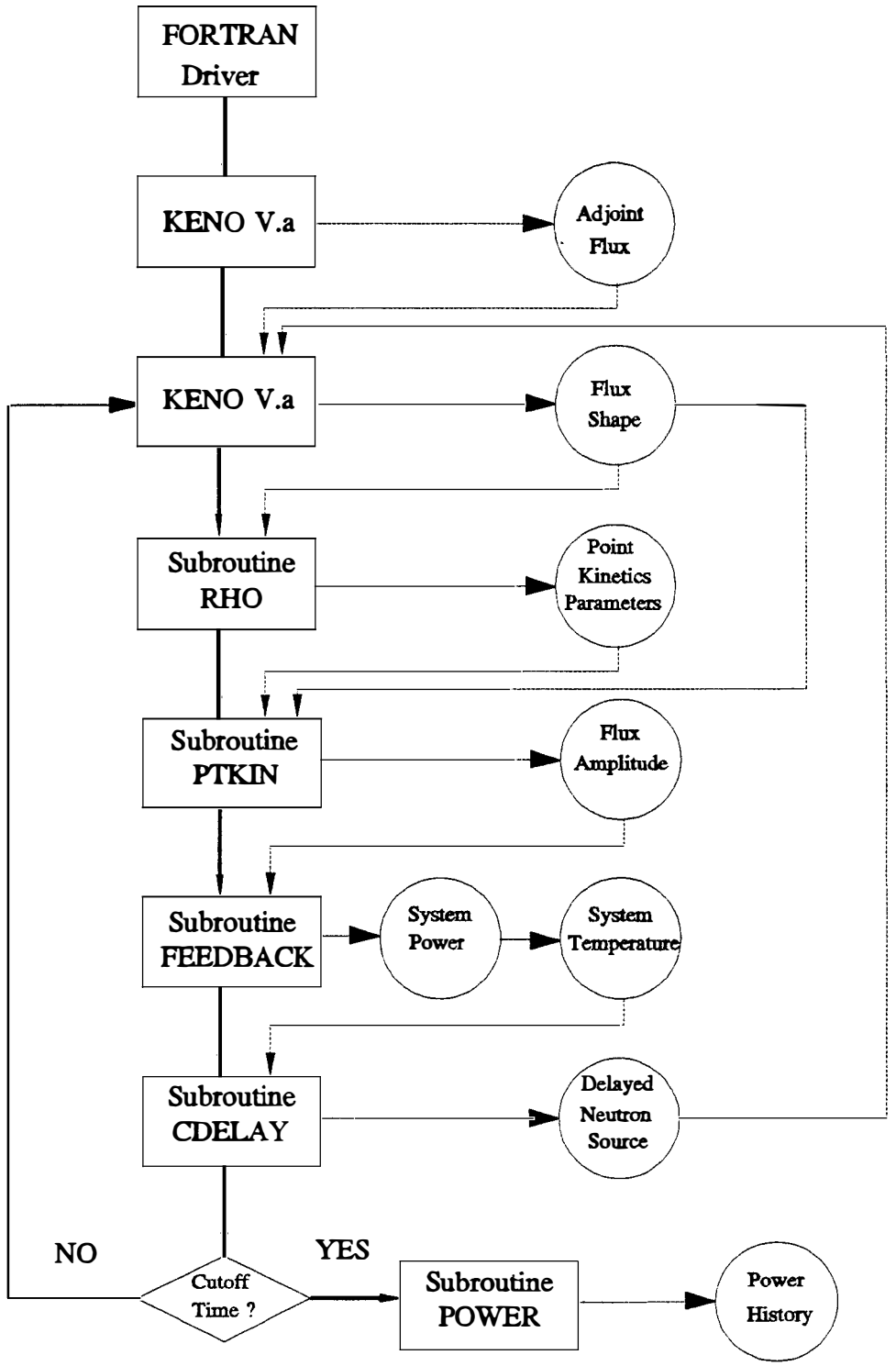

Figure 1: Computational Flow for TDKENO-M

3. Subroutines

3.1 KENO V.a

The three-dimensional transport code KENO V.a was modified to solve the inhomogeneous shape equation (equation 18) which includes the delayed neutron source, the amplitude derivative as well as the shape derivative.[42,43,44] Several KENO V.a subroutines were modified so that the fission source for a generation would explicitly represent the delayed neutrons as well as those neutrons contributing to the flux shape derivative (i.e., the right hand side of the backwards difference approximation). KENO V.a selects the energy, angle and direction of the delayed neutrons based on the delayed neutron source distribution given by equation 18. The energy, angle and direction of the neutrons contributing to the flux shape derivative term are selected based on cumulative distribution functions calculated by subroutine CDELAY as described below.

3.2 RHO

Subroutine RHO calculates the point kinetics parameters including reactivity, generation time and effective delayed neutron fraction using an interpolated flux shape. The point kinetics parameters are calculated based on their inner product definitions. Subroutine RHO uses the steady-state adjoint fluxes and the flux shapes produced by KENO V.a to numerically integrate the appropriate equations. The cross sections, fission spectra and delayed neutron data are read from interface files as described in the following chapter of this manual.

3.3 PTKIN

Subroutine PTKIN solves the point kinetics equations using the Livermore Solver of Ordinary Differential Equations (LSODE). LSODE solves initial value problems for either stiff or non-stiff systems of first order differential equations. Integration time steps are selected automatically by LSODE based on user-specified tolerance parameters.

PTKIN uses quadratically interpolated (second order) point kinetics parameters with linearly interpolated cross sections. Subroutine PTKIN outputs the flux amplitude as well as the delayed precursor concentrations and the flux amplitude derivative.

3.4 CDELAY

Subroutine CDELAY solves the delayed neutron spatial distribution using a numerical integration technique with linearly interpolated flux shapes and cross sections. CDELAY selects the integration time step automatically using an algorithm to minimize integration errors.

CDELAY also determines cumulative distribution density functions used by KENO V.a to incorporate the flux shape derivative. Specifically, the right hand side of the backwards difference approximation is used to modify the source distribution in KENO V.a for each generation of neutrons. A random selection is made based on the cumulative distribution functions to determine whether each neutron is prompt, delayed, or contributes to the flux shape derivative. If a neutron is selected to be delayed, the

starting energy, angle, and position are determined from the delayed source distribution. If the neutron is determined to contribute to the flux shape derivative, the starting parameters are selected based on the source distribution calculated from the numerical integration of the backwards difference approximation term. Note that prompt neutrons are tracked by KENO V.a without any modifications.

3.5 POWER

Subroutine POWER determines the power history for a calculation by numerically integrating the product of the shape, amplitude, and fission cross sections divided by the particle speed. POWER uses linearly interpolated flux shapes and cross sections. The time, flux amplitude, and power are all written to an output file as describe later in the manual.

4. Description of Input Files

TDKENO-M uses several input files for a calculation including cross section files, standard KENO V.a input files, and a single input file which includes all other necessary data to perform a TDKENO-M calculation. Note that the standard KENO V.a input (except angular input) and the KENO V.a cross section files are not described in this manual since a complete description exists in the SCALE manuals. Described below is the TDKENO-M input file format followed by a sample input file for Benchmark 16-A1:

4.1 Global Parameters

Record	Format	Input Variable	
1	6I5	iadj	Flags TDKENO-M to perform an adjoint calculation if set equal to 1. Otherwise, TDKENO-M does not perform an adjoint calculation and uses a user-specified adjoint file.
		imeth	Determines the method used to calculate the flux shape. If set equal to 1, TDKENO-M performs a point kinetics calculation. For imeth=2, TDKENO-M performs an adiabatic calculation. For imeth=3, TDKENO-M does a quasistatic calculation. Finally, for imeth=4, TDKENO-M performs an improved quasistatic calculation.
		iss	Flags TDKENO-M to perform a steady-state forward calculation. If iss=1, TDKENO-M does a steady-state calculation. Otherwise, TDKENO-M uses a user specified steady-state flux file.
		itfs	The number of transient flux shape calculations to be performed.
		irho	The number of reactivity calculations to be performed between flux shape calculations.

Record	Format	Input Variable	
		ipk	The number of point kinetics calculations to be performed between each reactivity calculation.
2	NI5	icor	This record contains N entries where N is the number of transient shape calculations. The parameter icor corresponds to the standard KENO V.a input file to be used for each flux shape calculation.
3	2I5	kmax	The number of geometry regions.
		ngp	The number of energy groups.
4	4I5	dpf	The initial ratio of delayed neutrons to total neutrons.
		f2d	The initial ratio of neutrons contibuting to the flux shape derivative to delayed plus flux shape contibuting neutrons.
		d2p	The initial ratio of delayed neutrons to prompt neutrons.
		xdelt	The initial time of a delayed neutron source distribution calculation.
5	1I5	ifr	A flag used for the initial point kinetics calculation. This parameter should be set equal to -2 if the first calculation is a steady-state forward calculation.
6	1I5	ndg	The number of delayed neutron groups.
7	NI5	beta	The initial delayed neutron fraction for each delayed neutron group where N is the total number of groups.
8	NI5	alam	The initial delayed neutron decay constants where N is the number of delayed neutron groups.
9	6E7.4	time	The initial time of the calculation.
		pow	The initial power of the system.
		tfuel	The initial fuel temperature of the system.
		tcool1	The initial temperature of coolant 1.

Record	Format	Input Variable	
		tcool2	The initial temperature of coolant 2.
		dpdt	The initial amplitude derivative.
10	NE11.4	c(i)	The initial delayed neutron precursor concentrations, where N is the number of delayed neutron groups.
11	3I5	kmax	The number of geometrical regions.
		ngp	The number of energy groups.
		nmat	The number of materials.
12	NE11.4	vol(i)	The volumes for each region, where N is the number of geometrical regions.
13	NE11.4	chi(i)	The fission spectrum for each energy group, where N is the number of energy groups.
14	NE11.4	eb(i)	The energy boundaries for each energy group, where N is the number of energy groups plus one.
15	NE11.4	nc(i)	The material identification number corresponding to each geometrical region, where N is the number of geometrical regions.
16	6E11.4	time	The time of the first fission cross-section entry.
		sf(i,j)	The fission cross section at the specified time where i denotes the energy group and j denotes the geometrical region.
17	E11.4	xdum	This variable is set equal to -1.0 to flag a new sequence of fission cross section data.
18	6E11.4	sf(i,j)	These entries are the same as record 16 and are repeated for each time where fission cross section data is entered.

4.2 General Parameters

These parameters are specific to each problem and allow the user to input specific data used during a calculation. Each section starts with a record giving the section number and the number of entries in the section. The format of this series of data is:

Record	Format		Input Variable
1	2I5	isec	The section number.
		nrec	The number of entries for this section.
2-6	80A1	title	These five lines can be used to give a descriptive title for the problem being modeled.
7-28			These 21 lines are not used in the current version of TDKENO-M.
29	2I5	isec	The section number for the time step data.
		istep	The number of time steps given in the time step data.
30	E11.4	tstep	The time for each flux shape calculation.

For the current version of TDKENO-M the first section is used to enter title data in a series of five lines. The second and third sections are used for thermalhydraulic data (Benchmark Problem 14-A2 only). Finally, the fourth section is used to enter time interval data.

Refer to the following section of this document for a sample TDKENO-M input file. Note that all of the sample files presented in this manual are for the modeling of ANL Benchmark Problem 16-A1.[45]

4.3 Sample TDKENO-M Input File

```
  0    4    1    3   10   10
  1    1    1
 21    2
.0000E+00  .0000E+00  .0000E+00  .0000E+00
 -2
  6
.1290E-01  .3110E-01  .1340E+00  .3310E+00  .1260E+01  .3210E+01
.8100E-04  .6870E-03  .6120E-03  .1138E-02  .5120E-03  .1700E-03
.0000E+00  .1000E+01  .0000E+00  .0000E+00  .0000E+00
.1680E+05  .5912E+05  .1223E+05  .9215E+04  .1094E+04  .1439E+03
 21    2    5
.4000E+02  .3000E+01  .7000E+01  .9100E+01  .9100E+01  .9174E+01
.7000E+01  .3000E+01  .9000E+01  .5000E+01  .2400E+02  .5000E+01
.9000E+01  .3000E+01  .7000E+01  .9126E+01  .9100E+01  .9148E+01
.7000E+01  .3000E+01  .4000E+02
.1000E+01  .0000E+00
.1194E+07  .1934E+05  .1000E+04
  1    4    4    4    4    4    4    4    3    2    2    2
  3    5    5    5    5    5    5    5    1
.0000E+00  .8344E-03  .3278E-03  .7452E-02  .1106E-01  .0000E+00
.0000E+00  .7452E-02  .1106E-01  .7452E-02  .1106E-01
.1000E-08  .8344E-03  .3278E-03  .7452E-02  .1106E-01  .0000E+00
.0000E+00  .7824E-02  .1161E-01  .7079E-02  .1051E-01
.1500E+02  .8344E-03  .3278E-03  .7452E-02  .1106E-01  .0000E+00
.0000E+00  .7824E-02  .1161E-01  .7079E-02  .1051E-01
-.1000E+01  .0000E+00  .0000E+00  .0000E+00  .0000E+00  .0000E+00
.0000E+00  .0000E+00  .0000E+00  .0000E+00  .0000E+00
.0000E+00  .8344E-03  .3278E-03  .7452E-02  .1106E-01  .0000E+00
.0000E+00  .7452E-02  .1106E-01  .7452E-02  .1106E-01
.1000E-08  .8344E-03  .3278E-03  .7452E-02  .1106E-01  .0000E+00
.0000E+00  .7824E-02  .1161E-01  .7079E-02  .1051E-01
.1500E+02  .8344E-03  .3278E-03  .7452E-02  .1106E-01  .0000E+00
.0000E+00  .7824E-02  .1161E-01  .7079E-02  .1051E-01
-.1000E+01  .0000E+00  .0000E+00  .0000E+00  .0000E+00  .0000E+00
.0000E+00  .0000E+00  .0000E+00  .0000E+00  .0000E+00
  1    5
  BENCHMARK PROBLEM 16-A1
  P. 26 OF ORNL/TM-7086


  2   14
  4    7    1    1    0
  0    0    0    0    0    0
 10   10  .1000E-02  .1000E-02
 20   24    5   16    5   24   20
  1
  1
.0000E+00  .4000E+02  .8737E+02  .9637E+02  .1304E+03  .1394E+03  .1867E+03
.2267E+03
.1000E+01  .1000E+01
.1000E+01  .1000E+01
.2500E+00  .2500E+00  .2500E+00  .2500E+00
-.7887E+00 -.2113E+00  .2113E+00  .7887E+00
.1000E+01  .1000E+01  .1000E+01  .1000E+01
.1000E+01  .1000E+01  .1000E+01  .1000E+01
  3    6
  3    3    2
  1    2    3    2    3    2    1
.0000E+00  .0000E+00  .0000E+00
  1    1  .1000E+01
  2    2  .1000E+01
  3    3  .1000E+01
  4    5
.0000E+00
.1000E-05
.1000E-04
.1000E-01
.1000E+02
```

4.4 KENO V.a Angular Input

The KENO V.a input is identical to a normal KENO V.a input file with the exception that angular data is appended to the end of the standard input file. The angular data is entered in the KENO V.a extra data form (i.e., READ EXTRA data END EXTRA) where data is in the form:

nphi	The number of phi angle bins.
ntheta	The number of theta (azimuthal) angle bins.
phi(i)	The boundaries for the phi angles. The first angle entered must be 0.0 and the last angle must be twice the variable pi.
theta(i)	The boundaries for the theta (azimuthal) angles. The first angle entered must be 0.0 and the last angle must be pi.

Refer to the following section of this manual for a sample KENO V.a input file with angular data.

4.5 Sample KENO V.a Input File

```
  benchmark problem 16-a1     steady state adjoint   k16a1a
  read parm  tme=450  tba=10.0 gen=706  npg=2500  nsk=6  flx=yes
    rnd=2bc38cfe126d amx=yes  xap=yes  plt=no  adj=yes
    lib=2   end parm
  read geom
cuboid  1   1   40.000  0.0  1000.0 -1000.0  1000.0 -1000.0
cuboid  4   1   43.000  0.0  1000.0 -1000.0  1000.0 -1000.0
cuboid  4   1   50.000  0.0  1000.0 -1000.0  1000.0 -1000.0
cuboid  4   1   59.100  0.0  1000.0 -1000.0  1000.0 -1000.0
cuboid  4   1   68.200  0.0  1000.0 -1000.0  1000.0 -1000.0
cuboid  4   1   77.374  0.0  1000.0 -1000.0  1000.0 -1000.0
cuboid  4   1   84.374  0.0  1000.0 -1000.0  1000.0 -1000.0
cuboid  4   1   87.374  0.0  1000.0 -1000.0  1000.0 -1000.0
cuboid  3   1   96.374  0.0  1000.0 -1000.0  1000.0 -1000.0
cuboid  2   1  101.374  0.0  1000.0 -1000.0  1000.0 -1000.0
cuboid  2   1  125.374  0.0  1000.0 -1000.0  1000.0 -1000.0
cuboid  2   1  130.374  0.0  1000.0 -1000.0  1000.0 -1000.0
cuboid  3   1  139.374  0.0  1000.0 -1000.0  1000.0 -1000.0
cuboid  5   1  142.374  0.0  1000.0 -1000.0  1000.0 -1000.0
cuboid  5   1  149.374  0.0  1000.0 -1000.0  1000.0 -1000.0
cuboid  5   1  158.500  0.0  1000.0 -1000.0  1000.0 -1000.0
cuboid  5   1  167.600  0.0  1000.0 -1000.0  1000.0 -1000.0
```

```
cuboid  5   1  176.748  0.0  1000.0 -1000.0  1000.0 -1000.0
cuboid  5   1  183.748  0.0  1000.0 -1000.0  1000.0 -1000.0
cuboid  5   1  186.748  0.0  1000.0 -1000.0  1000.0 -1000.0
cuboid  1   1  226.748  0.0  1000.0 -1000.0  1000.0 -1000.0
  end geom
  read boun  yzf=specular  end boun
  read mixt  sct=0   mix=1  1  1.0
                     mix=2  2  1.0
                     mix=3  3  1.0
                     mix=4  2  1.0
                     mix=5  2  1.0
  end mixt
  read extr
  20  1
   0.0    0.314 0.628 0.942 1.257 1.571 1.885 2.199 2.513 2.827
   3.142  3.456 3.770 4.084 4.398 4.712 5.027 5.341 5.655 5.969 6.283
   0.0  3.14159
  end extr
  end data
```

5. Description of Output Files

The output from TDKENO-M consists of six files. Each of these files is described below and excerpts from sample output files are given.

5.1 Output file power.out

This output file consists of three columns and contains the power history as well as the flux amplitudes. The first column is the time each flux amplitude was calculated. The second and third columns are the power and flux amplitude, respectively.

5.1.1 Sample power.out file

```
0.000000E+00 1.000000E+00 1.000000E+00
5.000000E-08 1.000793E+00 1.000000E+00
1.000000E-07 1.001144E+00 1.000010E+00
1.500000E-07 1.001485E+00 1.000010E+00
2.000000E-07 1.001827E+00 1.000010E+00
2.500000E-07 1.002179E+00 1.000020E+00
3.000000E-07 1.002521E+00 1.000020E+00
3.500000E-07 1.002864E+00 1.000020E+00
4.000000E-07 1.003216E+00 1.000030E+00
4.500000E-07 1.003559E+00 1.000030E+00
5.000000E-07 1.003902E+00 1.000030E+00
5.500000E-07 1.004255E+00 1.000040E+00
6.000000E-07 1.004599E+00 1.000040E+00
6.500000E-07 1.004942E+00 1.000040E+00
7.000000E-07 1.005296E+00 1.000050E+00
7.500000E-07 1.005640E+00 1.000050E+00
8.000000E-07 1.005984E+00 1.000050E+00
8.500000E-07 1.006339E+00 1.000060E+00
9.000000E-07 1.006684E+00 1.000060E+00
9.500000E-07 1.007038E+00 1.000070E+00
1.000000E-06 1.007384E+00 1.000070E+00
2.080000E-06 1.007588E+00 1.002140E+00
3.070000E-06 1.007756E+00 1.004030E+00
4.150000E-06 1.007904E+00 1.006070E+00
5.230000E-06 1.008028E+00 1.008100E+00
6.310000E-06 1.008129E+00 1.010120E+00
7.030000E-06 1.008171E+00 1.011450E+00
8.020000E-06 1.008221E+00 1.013280E+00
9.010000E-06 1.008240E+00 1.015090E+00
1.000000E-05 1.008247E+00 1.016900E+00
2.098000E-04 1.198779E+00 1.208920E+00
3.097000E-04 1.231955E+00 1.242200E+00
4.096000E-04 1.249170E+00 1.259380E+00
5.095000E-04 1.258157E+00 1.268260E+00
```

5.2 Output file outfile

This output file contains the status from each subroutine upon its completion. Information in this file includes the time of completion of each subroutine, the status of the completion, and the final values from the calculation. The final five lines of this output file contains information describing the computational method used for the calculation (i.e., point kinetics, adiabatic, quasistatic, or improved quasistatic) as well as the CPU time for the problem.

5.2.1 Sample outfile file

```
****************************************
Time,tstop,delt,tout,tk =   .0000E+00  .1000E-05  .1000E-07  .1000E-07  .1000E-06
iter =  1  istate =  1
Reactivity =  2.58012E-05
Gen =  3.74433E-07
Beta  =  8.10000E-05
Beta  =  6.86999E-04
Beta  =  6.11999E-04
Beta  =  1.13800E-03
Beta  =  5.11999E-04
Beta  =  1.69999E-04
Betatn =  3.19999E-03
Lamda =  1.29000E-02
Lamda =  3.10999E-02
Lamda =  .134
Lamda =  .331
Lamda =  1.26
Lamda =  3.21
****************************************
Time,tstop,delt,tout,tk =   .1000E-06  .1000E-05  .1000E-07  .1100E-06  .2000E-06
iter =  2  istate =  2
Reactivity =  2.58012E-05
Gen =  3.74433E-07
Beta  =  8.10000E-05
Beta  =  6.86999E-04
Beta  =  6.11999E-04
Beta  =  1.13800E-03
Beta  =  5.11999E-04
Beta  =  1.69999E-04
Betatn =  3.19999E-03
Lamda =  1.29000E-02
Lamda =  3.10999E-02
Lamda =  .134
Lamda =  .331
Lamda =  1.26
Lamda =  3.21
****************************************
Time,tstop,delt,tout,tk =   .9900E-06  .1000E-05  .1000E-07  .1000E-05  .1000E-05
iter =  10  istate =  2
Reactivity =  2.58012E-05
Gen =  3.74433E-07
```

```
Beta  =  8.10000E-05
Beta  =  6.86999E-04
Beta  =  6.11999E-04
Beta  =  1.13800E-03
Beta  =  5.11999E-04
Beta  =  1.69999E-04
Betatn =  3.19999E-03
Lamda =  1.29000E-02
Lamda =  3.10999E-02
Lamda =  .134
Lamda =  .331
Lamda =  1.26
Lamda =  3.21
****************************************
ptkin finished at time=  .1000E-05
    last power=  .1000E+01
cdelay calc. ended at time=  .1000E-05
ratio of fixed source to total source=  .3801E+00
ratio of flux shift to total fixed source=  .9966E+00
ratio of delayed n. source to prompt source=  .2055E-02
Flux.dat file updated by norm, time =  .1000E-05
****************************************
Time,tstop,delt,tout,tk =   .1000E-05  .1000E-04  .9000E-07  .1090E-05  .1900E-05
iter =  1  istate =  1
Reactivity =  7.30358E-04
Gen =  3.72856E-07
Beta  =  8.10000E-05
Beta  =  6.87000E-04
Beta  =  6.11999E-04
Beta  =  1.13800E-03
Beta  =  5.11999E-04
Beta  =  1.69999E-04
Betatn =  3.19999E-03
Lamda =  1.29000E-02
Lamda =  3.10999E-02
Lamda =  .134
Lamda =  .331
Lamda =  1.26
Lamda =  3.21
****************************************
Time,tstop,delt,tout,tk =   .1000E-04  .1000E-04  .9000E-07  .1009E-04  .1000E-04
iter =  10  istate =  2
Reactivity =  7.30358E-04
Gen =  3.72856E-07
Beta  =  8.10000E-05
Beta  =  6.87000E-04
Beta  =  6.11999E-04
Beta  =  1.13800E-03
Beta  =  5.11999E-04
Beta  =  1.69999E-04
Betatn =  3.19999E-03
Lamda =  1.29000E-02
Lamda =  3.10999E-02
Lamda =  .134
Lamda =  .331
Lamda =  1.26
Lamda =  3.21
****************************************
ptkin finished at time=  .1000E-04
    last power=  .1017E+01
cdelay calc. ended at time=  .1009E-04
ratio of fixed source to total source=  .6446E-01
ratio of flux shift to total fixed source=  .9707E+00
ratio of delayed n. source to prompt source=  .2017E-02
Flux.dat file updated by norm, time =  .1009E-04
****************************************
TDKENO-M performed an improved quasistatic calculation.
TDKENO-M performed a steady-state adjoint calculation.
TDKENO-M performed a steady-state forward calculation.
TDKENO-M performed  3  transient calculations.
The calculation was completed in    0.4795734E+01 min.
****************************************
```

5.3 Output file rho.dat

The values of reactivity calculated by subroutine RHO are contained in this output file along with the generation time and effective delayed neutron fractions. This file contains a series of outputs from different reactivity calculations. Each output begins with either a -1 or a -2 which indicates whether the calculation used interpolated or extrapolated fluxes. A -1 flag denotes that the calculation used extrapolated fluxes and a -2 denotes that the calculation used interpolated fluxes.

5.3.1 Sample rho.dat file

```
-1
 .0000000E+00  .3744335E-06  .3055803E-04  .3907257E-13
 .1676800E+05  .5899200E+05  .1219700E+05  .9181500E+04  .1085200E+04  .1414300E+03
 .8100001E-04  .6869999E-03  .6119999E-03  .1138000E-02  .5120000E-03  .1700000E-03
 .1000000E-06  .3744335E-06  .3055803E-04  .3907257E-13
 .1676800E+05  .5899200E+05  .1219700E+05  .9181500E+04  .1085200E+04  .1414300E+03
 .8100001E-04  .6869999E-03  .6119999E-03  .1138000E-02  .5120000E-03  .1700000E-03
 .2000000E-06  .3744335E-06  .3055803E-04  .3907257E-13
 .1676800E+05  .5899200E+05  .1219700E+05  .9181500E+04  .1085200E+04  .1414300E+03
 .8100001E-04  .6869999E-03  .6119999E-03  .1138000E-02  .5120000E-03  .1700000E-03
 .3000000E-06  .3744335E-06  .3055803E-04  .3907257E-13
 .1676800E+05  .5899200E+05  .1219700E+05  .9181500E+04  .1085200E+04  .1414300E+03
 .8100001E-04  .6869999E-03  .6119999E-03  .1138000E-02  .5120000E-03  .1700000E-03
 .4000000E-06  .3744335E-06  .3055803E-04  .3907257E-13
 .1676800E+05  .5899200E+05  .1219700E+05  .9181500E+04  .1085200E+04  .1414300E+03
 .8100001E-04  .6869999E-03  .6119999E-03  .1138000E-02  .5120000E-03  .1700000E-03
 .5000000E-06  .3744335E-06  .3055803E-04  .3907257E-13
 .1676800E+05  .5899200E+05  .1219700E+05  .9181500E+04  .1085200E+04  .1414300E+03
 .8100001E-04  .6869999E-03  .6119999E-03  .1138000E-02  .5120000E-03  .1700000E-03
 .6000000E-06  .3744335E-06  .3055803E-04  .3907257E-13
 .1676800E+05  .5899200E+05  .1219700E+05  .9181500E+04  .1085200E+04  .1414300E+03
 .8100001E-04  .6869999E-03  .6119999E-03  .1138000E-02  .5120000E-03  .1700000E-03
 .7000000E-06  .3744335E-06  .3055803E-04  .3907257E-13
 .1676800E+05  .5899200E+05  .1219700E+05  .9181500E+04  .1085200E+04  .1414300E+03
 .8100001E-04  .6869999E-03  .6119999E-03  .1138000E-02  .5120000E-03  .1700000E-03
 .8000000E-06  .3744335E-06  .3055803E-04  .3907257E-13
 .1676800E+05  .5899200E+05  .1219700E+05  .9181500E+04  .1085200E+04  .1414300E+03
 .8100001E-04  .6869999E-03  .6119999E-03  .1138000E-02  .5120000E-03  .1700000E-03
 .9000000E-06  .3744335E-06  .3055803E-04  .3907257E-13
 .1676800E+05  .5899200E+05  .1219700E+05  .9181500E+04  .1085200E+04  .1414300E+03
 .8100001E-04  .6869999E-03  .6119999E-03  .1138000E-02  .5120000E-03  .1700000E-03
 .1000000E-05  .3744335E-06  .3055803E-04  .3907257E-13
 .1676800E+05  .5899200E+05  .1219700E+05  .9181500E+04  .1085200E+04  .1414300E+03
 .8100001E-04  .6869999E-03  .6119999E-03  .1138000E-02  .5120000E-03  .1700000E-03
-2
 .0000000E+00  .3744335E-06  .3055803E-04  .3914710E-13
 .1676800E+05  .5899200E+05  .1219700E+05  .9181500E+04  .1085200E+04  .1414300E+03
 .8100001E-04  .6869999E-03  .6119999E-03  .1138000E-02  .5120000E-03  .1700000E-03
 .1000000E-06  .3742900E-06  .1323545E-03  .3914710E-13
 .1676800E+05  .5899200E+05  .1219700E+05  .9181500E+04  .1085200E+04  .1414300E+03
 .8100001E-04  .6870000E-03  .6120000E-03  .1138000E-02  .5120000E-03  .1700000E-03
```

```
.2000000E-06  .3741465E-06  .2341905E-03  .3914710E-13
.1676800E+05  .5899200E+05  .1219700E+05  .9181500E+04  .1085200E+04  .1414300E+03
.8099998E-04  .6869999E-03  .6119999E-03  .1138000E-02  .5120000E-03  .1700000E-03
.3000000E-06  .3740031E-06  .3360515E-03  .3914710E-13
.1676800E+05  .5899200E+05  .1219700E+05  .9181500E+04  .1085200E+04  .1414300E+03
.8099999E-04  .6869999E-03  .6120000E-03  .1138000E-02  .5120000E-03  .1700000E-03
.4000000E-06  .3738596E-06  .4379657E-03  .3914710E-13
.1676800E+05  .5899200E+05  .1219700E+05  .9181500E+04  .1085200E+04  .1414300E+03
.8099999E-04  .6869999E-03  .6119999E-03  .1138000E-02  .5119999E-03  .1700000E-03
.5000000E-06  .3737161E-06  .5398598E-03  .3914710E-13
.1676800E+05  .5899200E+05  .1219700E+05  .9181500E+04  .1085200E+04  .1414300E+03
.8099999E-04  .6869999E-03  .6119999E-03  .1138000E-02  .5120000E-03  .1700000E-03
.6000000E-06  .3735724E-06  .6417960E-03  .3914710E-13
.1676800E+05  .5899200E+05  .1219700E+05  .9181500E+04  .1085200E+04  .1414300E+03
.8100001E-04  .6870001E-03  .6120000E-03  .1138000E-02  .5120000E-03  .1700000E-03
.7000000E-06  .3734287E-06  .7437427E-03  .3914710E-13
.1676800E+05  .5899200E+05  .1219700E+05  .9181500E+04  .1085200E+04  .1414300E+03
.8100000E-04  .6870000E-03  .6119999E-03  .1138000E-02  .5120000E-03  .1700000E-03
.8000000E-06  .3732851E-06  .8457223E-03  .3914710E-13
.1676800E+05  .5899200E+05  .1219700E+05  .9181500E+04  .1085200E+04  .1414300E+03
.8100000E-04  .6870001E-03  .6120000E-03  .1138000E-02  .5120000E-03  .1700000E-03
.9000000E-06  .3731414E-06  .9477155E-03  .3914710E-13
.1676800E+05  .5899200E+05  .1219700E+05  .9181500E+04  .1085200E+04  .1414300E+03
.8099999E-04  .6869999E-03  .6119998E-03  .1138000E-02  .5119999E-03  .1700000E-03
.1000000E-05  .3729977E-06  .1049727E-02  .3914710E-13
.1676800E+05  .5899200E+05  .1219700E+05  .9181500E+04  .1085200E+04  .1414300E+03
.8099998E-04  .6870000E-03  .6119999E-03  .1138000E-02  .5120000E-03  .1700000E-03
```

5.4 Output file flux.out

The normalized flux shape distributions are contained in this output file for each of the energy groups as well as each of the spatial regions. Also contained in this file is the time of each of the flux shape calculations.

5.4.1 Sample flux.out file

```
flux at time= .0000E+00    group=  1
1.670472E-01 5.563998E-01 6.784357E-01 8.293718E-01 9.266717E-01 9.418316E-01
8.852198E-01 8.093197E-01 7.454078E-01 8.177118E-01 9.015638E-01 8.112158E-01
7.369277E-01 7.956918E-01 8.664557E-01 9.198877E-01 9.018717E-01 8.018478E-01
6.573238E-01 5.392158E-01 1.623264E-01
flux at time= .0000E+00    group=  2
5.467278E-02 1.092384E-01 1.162852E-01 1.287152E-01 1.382828E-01 1.387520E-01
1.293760E-01 1.191160E-01 1.121264E-01 1.207244E-01 1.323544E-01 1.188724E-01
1.097808E-01 1.180464E-01 1.290672E-01 1.366644E-01 1.360867E-01 1.256812E-01
1.125708E-01 1.070184E-01 5.365318E-02
flux at time=1.0000E-06    group=  1
1.823611E-01 6.087489E-01 7.512240E-01 9.188441E-01 1.025996E+00 1.029687E+00
9.530403E-01 8.589637E-01 7.795646E-01 8.435740E-01 8.968463E-01 7.847601E-01
6.999922E-01 7.462457E-01 8.035786E-01 8.402659E-01 8.200388E-01 7.297654E-01
6.012915E-01 4.978580E-01 1.489566E-01
flux at time=1.0000E-06    group=  2
6.033601E-02 1.198631E-01 1.272698E-01 1.409502E-01 1.526740E-01 1.518204E-01
1.399260E-01 1.252825E-01 1.156989E-01 1.234606E-01 1.317317E-01 1.146232E-01
1.044292E-01 1.103874E-01 1.177961E-01 1.243005E-01 1.241395E-01 1.137117E-01
1.031216E-01 9.855019E-02 4.883561E-02
flux at time=1.0090E-05    group=  1
```

```
1.802600E-01 5.942785E-01 7.350117E-01 9.004663E-01 9.982772E-01 9.987129E-01
9.283276E-01 8.413092E-01 7.661208E-01 8.315411E-01 8.935103E-01 7.793484E-01
6.961046E-01 7.451782E-01 7.994739E-01 8.329597E-01 8.114197E-01 7.226345E-01
5.917856E-01 4.889515E-01 1.484267E-01
flux at time=1.0090E-05    group=  2
6.501836E-02 1.268395E-01 1.340680E-01 1.509453E-01 1.623615E-01 1.612706E-01
1.481227E-01 1.331265E-01 1.223619E-01 1.312151E-01 1.423173E-01 1.233794E-01
1.113750E-01 1.170124E-01 1.265836E-01 1.342684E-01 1.328469E-01 1.229227E-01
1.115646E-01 1.062308E-01 5.415092E-02
flux at time=1.0000E-02    group=  1
2.295858E-01 7.704574E-01 9.482068E-01 1.164879E+00 1.294740E+00 1.297853E+00
1.197407E+00 1.083756E+00 9.867509E-01 1.076199E+00 1.149749E+00 1.002511E+00
8.979581E-01 9.656869E-01 1.035059E+00 1.078718E+00 1.042191E+00 9.218639E-01
7.539604E-01 6.220564E-01 1.864705E-01
flux at time=1.0000E-02    group=  2
7.545748E-02 1.518985E-01 1.601864E-01 1.783776E-01 1.929991E-01 1.931676E-01
1.768196E-01 1.583309E-01 1.469479E-01 1.565476E-01 1.690063E-01 1.468470E-01
1.328318E-01 1.397690E-01 1.515151E-01 1.608142E-01 1.570381E-01 1.448328E-01
1.293624E-01 1.233330E-01 6.152108E-02
```

5.5 Output file cdelay.dat

This output file contains the delayed neutron precursor spatial distribution data as well as the data used for incorporating the flux shape derivative. This output file consists of a header line which contains the time as well as the delayed neutron fraction and the flux shape derivative contribution factor.

Following the header line is a sequence of cumulative distribution functions. The first sequence of data is the cumulative density functions for the energy, angular, and spatial distributions for the delayed neutrons. The second sequence of data is similar to the first sequence with the exception that it is for the flux shape derivative probabilities.

5.5.1 Sample cdelay.dat file

```
  21    2
.380143E+00 .996650E+00 .205475E-02 .000000E+00
.629260E-02 .160645E-01 .443955E-01 .692012E-01 .767628E-01 .784710E-01
.561962E-01 .220467E-01 .000000E+00 .371462E-01 .196387E+00 .368017E-01
.000000E+00 .217066E-01 .551935E-01 .763573E-01 .748599E-01 .673828E-01
.430066E-01 .156066E-01 .612166E-02
.100000E+01 .000000E+00 .100000E+01 .000000E+00 .100000E+01 .000000E+00
.100000E+01 .000000E+00 .100000E+01 .000000E+00 .100000E+01 .000000E+00
```

```
 .100000E+01 .000000E+00 .100000E+01 .000000E+00 .000000E+00 .000000E+00
 .100000E+01 .000000E+00 .100000E+01 .000000E+00 .100000E+01 .000000E+00
 .000000E+00 .000000E+00 .100000E+01 .000000E+00 .100000E+01 .000000E+00
 .100000E+01 .000000E+00 .100000E+01 .000000E+00 .100000E+01 .000000E+00
 .100000E+01 .000000E+00 .100000E+01 .000000E+00 .100000E+01 .000000E+00
 .727156E-01 .133826E-01 .354849E-01 .537173E-01 .589178E-01 .600087E-01
 .428773E-01 .168551E-01 .470480E-01 .284239E-01 .150032E+00 .281004E-01
 .463021E-01 .166327E-01 .423411E-01 .585325E-01 .576406E-01 .524563E-01
 .343660E-01 .130450E-01 .711204E-01
 .342024E+00 .657976E+00 .464250E+00 .535750E+00 .498136E+00 .501864E+00
 .522951E+00 .477049E+00 .532728E+00 .467272E+00 .535923E+00 .464077E+00
 .537906E+00 .462094E+00 .536161E+00 .463839E+00 .530738E+00 .469262E+00
 .535391E+00 .464609E+00 .536796E+00 .463204E+00 .537252E+00 .462748E+00
 .533153E+00 .466847E+00 .534180E+00 .465820E+00 .533171E+00 .466829E+00
 .533829E+00 .466171E+00 .529959E+00 .470041E+00 .520481E+00 .479519E+00
 .498349E+00 .501651E+00 .461555E+00 .538445E+00 .339812E+00 .660188E+00
```

5.6 Output file keno.out

This output file is simply the standard KENO V.a output files for each of the flux shape calculations. Note that the output has been modified to include the amplitude derivative and shape derivative information as well as the delayed neutron source distribution information.

5.6.1 Sample keno.out file

```
*******************************************************************************
*****                  program verification information                   *****
*****                                                                     *****
*****                 code system:  ncsshp   version:  1.0                *****
*****                                                                     *****
*******************************************************************************

*******************************************************************************
*****                                                                     *****
*****                                                                     *****
*****              program:  tdkenom                                      *****
*****                                                                     *****
*****        creation date:  09/11/96                                     *****
*****                                                                     *****
*****              library:                                               *****
*****                                                                     *****
*****                                                                     *****
*****    this is not a ncsshp   configuration controlled code            *****
*****                                                                     *****
*****              jobname:  ut9                                          *****
*****                                                                     *****
*****    date of execution:  09/11/96                                     *****
*****                                                                     *****
*****    time of execution:  08:28:05                                     *****
*****                                                                     *****
*****                                                                     *****
*******************************************************************************
*******************************************************************************
*******************************************************************************
***               benchmark problem 16-a1     perturbed state    k16a1p1    ***
```

```
***                                                                          ***
******************************************************************************
***                      numeric parameters                                  ***
***                                                                          ***
***                                                                          ***
***                     tme         maximum problem time (min)     430.00    ***
***                                                                          ***
***                     tba         time per generation (min)       20.00    ***
***                                                                          ***
***                     gen         number of generations           353      ***
***                                                                          ***
***                     npg         number per generation           2100     ***
***                                                                          ***
***                     nsk         number of generations to be skipped   6  ***
***                                                                          ***
***                     beg         beginning generation number           1  ***
***                                                                          ***
***                     res         generations between checkpoints       0  ***
***                                                                          ***
***                     x1d         number of extra 1-d cross sections    0  ***
***                                                                          ***
***                     nbk         neutron bank size                  125   ***
***                                                                          ***
***                     xnb         extra positions in neutron bank       0  ***
***                                                                          ***
***                     nfb         fission bank size                  2100  ***
***                                                                          ***
***                     xfb         extra positions in fission bank       0  ***
***                                                                          ***
***                     wta         default value of weight average  .5000   ***
***                                                                          ***
***                     wth         weight high for splitting       3.0000   ***
***                                                                          ***
***                     wtl         weight low for russian roulette  .3333   ***
***                                                                          ***
***                     rnd         starting random number   2BC5 8CFE126D   ***
***                                                                          ***
***                     nb8         number of d.a. blocks on unit  8   200   ***
***                                                                          ***
***                     nl8         length of d.a. blocks on unit  8   512   ***
***                                                                          ***
***                     adj         mode of calculation       forward        ***
***                                                                          ***
***                                 input data written on restart unit   no   ***
***                                                                          ***
***                                 binary data interface                no   ***
***                                                                          ***
***                                                                          ***
******************************************************************************
******************************************************************************
***                                                                          ***
***         benchmark problem 16-a1     perturbed state    k16a1p1           ***
***                                                                          ***
******************************************************************************
***                        logical parameters                                ***
***                                                                          ***
***                                                                          ***
***  run  execute problem after checking data    yes                         ***
***                                                                          ***
***  plt  plot picture map(s)                     no                         ***
***                                                                          ***
***  flx  compute flux                           yes                         ***
***                                                                          ***
***  fdn  compute fission densities               no                         ***
***                                                                          ***
***  smu  compute avg unit self-multiplication    no                         ***
***                                                                          ***
***  nub  compute nu-bar & avg fission group      no                         ***
***                                                                          ***
***  mku  compute matrix k-eff by unit number     no                         ***
***                                                                          ***
***  mkp  compute matrix k-eff by unit location   no                         ***
***                                                                          ***
```

```
***  cku  compute cofactor k-eff by unit number   no                        ***
***                                                                          ***
***  ckp  compute cofactor k-eff by unit location no                        ***
***                                                                          ***
***  fmu  print fiss prod matrix by unit number   no                        ***
***                                                                          ***
***  fmp  print fiss prod matrix by unit location no                        ***
***                                                                          ***
*** mkh  compute matrix k-eff by hole number      no                        ***
***                                                                          ***
*** mka  compute matrix k-eff by array number     no                        ***
***                                                                          ***
***  ckh  compute cofactor k-eff by hole number   no                        ***
***                                                                          ***
***  cka  compute cofactor k-eff by array number  no                        ***
***                                                                          ***
***  fmh  print fiss prod matrix by hole number   no                        ***
***                                                                          ***
***  fma  print fiss prod matrix by array number  no                        ***
***                                                                          ***
***  hhl  collect matrix by highest hole level    no                        ***
***                                                                          ***
***  hal  collect matrix by highest array level   no                        ***
***                                                                          ***
***  amx  print all mixed cross sections         yes                        ***
***                                                                          ***
***  far  print fis. and abs. by region           no                        ***
***                                                                          ***
***  xs1  print 1-d mixture x-sections           yes                        ***
***                                                                          ***
***  pax  print xsec-albedo correlation tables    no                        ***
***                                                                          ***
***  xs2  print 2-d mixture x-sections           yes                        ***
***                                                                          ***
***  pwt  print weight average array              no                        ***
***                                                                          ***
***  xap  print mixture angles & probabilities   yes                        ***
***                                                                          ***
***  pgm  print input geometry                    no                        ***
***                                                                          ***
***  pki  print fission spectrum                 yes                        ***
***                                                                          ***
*** bug  print debug information                  no                        ***
***                                                                          ***
***  p1d  print extra 1-d cross sections         yes                        ***
***                                                                          ***
***  trk  print tracking information              no                        ***
***                                                                          ***
***                                                                          ***
******************************************************************************
******************************************************************************
***                                                                          ***
***                        parameter input completed                         ***
***                                                                          ***
***   ........    0 io's were used reading the parameter data      ........  ***
***                                                                          ***
******************************************************************************
******************************************************************************
 This code used method  4 to calculate shape
 dpf2t,f2dpf,d2p,time =  .380143E+00 .996650E+00 .205475E-02 .000000E+00
 dpdt,time,pow,xdelt  =  .715206E+02 .100000E-05 .100007E+01 .000000E+00

          *************** data reading completed ***************
******************************************************************************
***                                                                          ***
***           benchmark problem 16-a1     perturbed state    k16a1p1         ***
***                                                                          ***
******************************************************************************
******************************************************************************
***                                                                          ***
1 =id          material=    1

group     sgt          nap          abp          nfp          chi             mwa1         mwa2         mwa3
```

```
     1  2.41100E-01 9.83998E-01 1.60017E-02 3.46085E-03 1.00000E+00        1           2           1
     2  4.17200E-01 9.75561E-01 2.44391E-02 7.85618E-04 1.00000E+00        3           3           2

  scattering transfer array for material     1
      from   grp   1     grp   2
      to grp
    +  0    9.84834E-01 1.00000E+00
    +  1    1.51659E-02

2 =id         material=    2

group     sgt         nap         abp         nfp         chi             mwa1        mwa2        mwa3
  1  1.84900E-01 9.72396E-01 2.76041E-02 4.03018E-02 1.00000E+00          1           2           1
  2  3.66800E-01 9.64343E-01 3.56570E-02 3.01559E-02 1.00000E+00          3           3           2

  scattering transfer array for material     2
      from   grp   1     grp   2
      to grp
    +  0    9.88403E-01 1.00000E+00
    +  1    1.15965E-02

3 =id         material=    3

group     sgt         nap         abp         nfp         chi             mwa1        mwa2        mwa3
  1  9.43200E-02 9.26917E-01 7.30831E-02  .00000E+00  .00000E+00          1           2           1
  2  1.87620E-01 9.13069E-01 8.69310E-02  .00000E+00  .00000E+00          3           3           2

  scattering transfer array for material     3
     from   grp   1     grp   2
     to grp
    +  0    9.80363E-01 1.00000E+00
    +  1    1.96370E-02

4 =id         material=    4

group     sgt         nap         abp         nfp         chi             mwa1        mwa2        mwa3
  1  1.94145E-01 9.72396E-01 2.76041E-02 4.03018E-02 1.00000E+00          1           2           1
  2  3.85140E-01 9.64343E-01 3.56570E-02 3.01559E-02 1.00000E+00          3           3           2

  scattering transfer array for material     4
      from   grp   1     grp   2
      to grp
    +  0    9.88404E-01 1.00000E+00
    +  1    1.15965E-02

5 =id         material=    5

group     sgt         nap         abp         nfp         chi             mwa1        mwa2        mwa3
  1  1.75655E-01 9.72396E-01 2.76041E-02 4.03018E-02 1.00000E+00          1           2           1
  2  3.48460E-01 9.64343E-01 3.56570E-02 3.01559E-02 1.00000E+00          3           3           2

  scattering transfer array for material     5
      from   grp   1     grp   2
      to grp
    +  0    9.88403E-01 1.00000E+00
    +  1    1.15965E-02
********************************************************************************
***                                                                          ***
***            benchmark problem 16-a1     perturbed state    k16a1p1        ***
***                                                                          ***
********************************************************************************
********************************************************************************
***                                                                          ***
***                               ****** additional information ******       ***
***                                                                          ***
***  number of energy groups               2       use lattice geometry  no  ***
***                                                                          ***
***  no. of fission spectrum source group  1       global array number    0  ***
***                                                                          ***
***  no. of scattering angles in xsecs     0                                 ***
***                                                                          ***
***  number of units in the global x dir.      0                             ***
***                                                                          ***
```

```
***  entries/neutron in the neutron bank  16                                  ***
***                                                                           ***
***  number of units in the global y dir.      0                              ***
***                                                                           ***
***  entries/neutron in the fission bank   9                                  ***
***                                                                           ***
***  number of units in the global z dir.      0                              ***
***                                                                           ***
***  number of mixtures used   5       use a global reflector           yes  ***
***                                                                           ***
***  number of bias id's used  1       use nested holes                  no  ***
***                                                                           ***
***  number of differential albedos used   0       number of holes        0  ***
***                                                                           ***
***  total input geometry regions  21       maximum hole nesting level    0  ***
***                                                                           ***
***  number of geometry regions used      21       use nested arrays     no  ***
***                                                                           ***
***  largest geometry unit number      1       number of arrays used      0  ***
***                                                                           ***
***  largest array number     1       maximum array nesting level         0  ***
***                                                                           ***
***                                                                           ***
***  +x boundary condition            vacuum -x boundary condition   vacuum  ***
***                                                                           ***
***  +y boundary condition          specular -y boundary condition specular  ***
***                                                                           ***
***  +z boundary condition          specular -z boundary condition specular  ***
***                                                                           ***
*******************************************************************************
*******************************************************************************
***                                                                           ***
***            benchmark problem 16-a1     perturbed state    k16a1p1         ***
***                                                                           ***
*******************************************************************************
*******************************************************************************

0                media bias      geometry description for those units utilized in this problem
 region           num   id
                   ******************   global   ******************
                                                -----   unit     1   -----
0  1 cuboid          1  1      +x =  40.000     -x =  .00000E+00 +y =  1000.0     -y = -1000.0     +z
=  1000.0     -z = -1000.0
0  2 cuboid          4  1      +x =  43.000     -x =  .00000E+00 +y =  1000.0     -y = -1000.0     +z
=  1000.0     -z = -1000.0
0  3 cuboid          4  1      +x =  50.000     -x =  .00000E+00 +y =  1000.0     -y = -1000.0     +z
=  1000.0     -z = -1000.0
0  4 cuboid          4  1      +x =  59.100     -x =  .00000E+00 +y =  1000.0     -y = -1000.0     +z
=  1000.0     -z = -1000.0
0  5 cuboid          4  1      +x =  68.200     -x =  .00000E+00 +y =  1000.0     -y = -1000.0     +z
=  1000.0     -z = -1000.0
0  6 cuboid          4  1      +x =  77.374     -x =  .00000E+00 +y =  1000.0     -y = -1000.0     +z
=  1000.0     -z = -1000.0
0  7 cuboid          4  1      +x =  84.374     -x =  .00000E+00 +y =  1000.0     -y = -1000.0     +z
=  1000.0     -z = -1000.0
0  8 cuboid          4  1      +x =  87.374     -x =  .00000E+00 +y =  1000.0     -y = -1000.0     +z
=  1000.0     -z = -1000.0
0  9 cuboid          3  1      +x =  96.374     -x =  .00000E+00 +y =  1000.0     -y = -1000.0     +z
=  1000.0     -z = -1000.0
0 10 cuboid          2  1      +x =  101.37     -x =  .00000E+00 +y =  1000.0     -y = -1000.0     +z
=  1000.0     -z = -1000.0
0 11 cuboid          2  1      +x =  125.37     -x =  .00000E+00 +y =  1000.0     -y = -1000.0     +z
=  1000.0     -z = -1000.0
0 12 cuboid          2  1      +x =  130.37     -x =  .00000E+00 +y =  1000.0     -y = -1000.0     +z
=  1000.0     -z = -1000.0
0 13 cuboid          3  1      +x =  139.37     -x =  .00000E+00 +y =  1000.0     -y = -1000.0     +z
=  1000.0     -z = -1000.0
0 14 cuboid          5  1      +x =  142.37     -x =  .00000E+00 +y =  1000.0     -y = -1000.0     +z
=  1000.0     -z = -1000.0
0 15 cuboid          5  1      +x =  149.37     -x =  .00000E+00 +y =  1000.0     -y = -1000.0     +z
=  1000.0     -z = -1000.0
0 16 cuboid          5  1      +x =  158.50     -x =  .00000E+00 +y =  1000.0     -y = -1000.0     +z
=  1000.0     -z = -1000.0
```

```
0 17 cuboid          5  1      +x =  167.60     -x =  .00000E+00 +y =  1000.0     -y = -1000.0     +z
=  1000.0     -z = -1000.0
0 18 cuboid          5  1      +x =  176.75     -x =  .00000E+00 +y =  1000.0     -y = -1000.0     +z
=  1000.0     -z = -1000.0
0 19 cuboid          5  1      +x =  183.75     -x =  .00000E+00 +y =  1000.0     -y = -1000.0     +z
=  1000.0     -z = -1000.0
0 20 cuboid          5  1      +x =  186.75     -x =  .00000E+00 +y =  1000.0     -y = -1000.0     +z
=  1000.0     -z = -1000.0
0 21 cuboid          1  1      +x =  226.75     -x =  .00000E+00 +y =  1000.0     -y = -1000.0     +z
=  1000.0     -z = -1000.0
                    benchmark problem 16-a1     perturbed state    k16a1p1
```

volumes for those units utilized in this problem

unit	region	geometry region	volume	cumulative volume
1	1	1	1.60000E+08 cm**3	1.60000E+08 cm**3
	2	2	1.20000E+07 cm**3	1.72000E+08 cm**3
	3	3	2.80000E+07 cm**3	2.00000E+08 cm**3
	4	4	3.64000E+07 cm**3	2.36400E+08 cm**3
	5	5	3.64000E+07 cm**3	2.72800E+08 cm**3
	6	6	3.66960E+07 cm**3	3.09496E+08 cm**3
	7	7	2.80000E+07 cm**3	3.37496E+08 cm**3
	8	8	1.20000E+07 cm**3	3.49496E+08 cm**3
	9	9	3.60000E+07 cm**3	3.85496E+08 cm**3
	10	10	2.00000E+07 cm**3	4.05496E+08 cm**3
	11	11	9.60000E+07 cm**3	5.01496E+08 cm**3
	12	12	2.00000E+07 cm**3	5.21496E+08 cm**3
	13	13	3.60000E+07 cm**3	5.57496E+08 cm**3
	14	14	1.20000E+07 cm**3	5.69496E+08 cm**3
	15	15	2.80000E+07 cm**3	5.97496E+08 cm**3
	16	16	3.65040E+07 cm**3	6.34000E+08 cm**3
	17	17	3.64000E+07 cm**3	6.70400E+08 cm**3
	18	18	3.65920E+07 cm**3	7.06992E+08 cm**3
	19	19	2.80000E+07 cm**3	7.34992E+08 cm**3
	20	20	1.20000E+07 cm**3	7.46992E+08 cm**3
	21	21	1.60000E+08 cm**3	9.06992E+08 cm**3

unit	uses	region	mixture	total volume
1	1	1	1	1.60000E+08 cm**3
		2	4	1.20000E+07 cm**3
		3	4	2.80000E+07 cm**3
		4	4	3.64000E+07 cm**3
		5	4	3.64000E+07 cm**3
		6	4	3.66960E+07 cm**3
		7	4	2.80000E+07 cm**3
		8	4	1.20000E+07 cm**3
		9	3	3.60000E+07 cm**3
		10	2	2.00000E+07 cm**3
		11	2	9.60000E+07 cm**3
		12	2	2.00000E+07 cm**3
		13	3	3.60000E+07 cm**3

```
                                    14          5          1.20000E+07 cm**3
                                    15          5          2.80000E+07 cm**3
                                    16          5          3.65040E+07 cm**3
                                    17          5          3.64000E+07 cm**3
                                    18          5          3.65920E+07 cm**3
                                    19          5          2.80000E+07 cm**3
                                    20          5          1.20000E+07 cm**3
                                    21          1          1.60000E+08 cm**3


                                          total mixture volumes
                                       mixture        total volume
                                          1         3.20000E+08 cm**3
                                          2         1.36000E+08 cm**3
                                          3         7.20000E+07 cm**3
                                          4         1.89496E+08 cm**3
                                          5         1.89496E+08 cm**3

keno message number k5-123          execution terminated due to completion of the specified number of
generations.
                        benchmark problem 16-a1     perturbed state    k16a1p1
lifetime =  6.92827E-07 + or -  9.16060E-10         generation time =  3.79794E-07 + or -  4.53918E-10
no. of initial
generations      average                            67 per cent        95 per cent      99 per cent
 skipped      k-effective      deviation    confidence interval   confidence interval   confidence
   6       1.00101   + or -  .00044     1.00058 to 1.00145    1.00014 to 1.00188     .99970 to 1.00232
   7       1.00098   + or -  .00044     1.00055 to 1.00142    1.00011 to 1.00186     .99968 to 1.00229
   8       1.00101   + or -  .00044     1.00057 to 1.00144    1.00013 to 1.00188     .99969 to 1.00232
   9       1.00098   + or -  .00044     1.00054 to 1.00142    1.00011 to 1.00186     .99967 to 1.00230
  10       1.00098   + or -  .00044     1.00054 to 1.00142    1.00010 to 1.00186     .99966 to 1.00230
  11       1.00100   + or -  .00044     1.00056 to 1.00144    1.00012 to 1.00188     .99968 to 1.00232
  12       1.00100   + or -  .00044     1.00056 to 1.00144    1.00012 to 1.00189     .99968 to 1.00233
  13       1.00105   + or -  .00044     1.00061 to 1.00149    1.00017 to 1.00193     .99973 to 1.00237
  14       1.00105   + or -  .00044     1.00061 to 1.00149    1.00017 to 1.00193     .99973 to 1.00237
  15       1.00100   + or -  .00044     1.00056 to 1.00144    1.00012 to 1.00188     .99968 to 1.00232
  20       1.00096   + or -  .00044     1.00051 to 1.00140    1.00007 to 1.00185     .99962 to 1.00229
  25       1.00104   + or -  .00045     1.00059 to 1.00149    1.00014 to 1.00194     .99969 to 1.00239

                    benchmark problem 16-a1     perturbed state    k16a1p1
fluxes for this problem
region   1       region   2       region   3       region   4       region   5          region  6
group   flux  percent   flux  percent   flux  percent   flux  percent   flux  percent   flux percent
              deviation       deviation       deviation       deviation       deviation    deviation
1 4.531E-08   .46  1.513E-07  .43 1.867E-07  .39 2.283E-07  .35 2.549E-07  .34    2.559E-07     .30
2 1.499E-08   .55  2.978E-08  .71 3.162E-08  .62 3.502E-08  .55 3.794E-08  .56    3.772E-08     .53

                    benchmark problem 16-a1     perturbed state    k16a1p1
fluxes for this problem
region   7       region   8       region   9       region   10      region   11        region  12
group   flux  percent   flux  percent   flux  percent   flux  percent   flux  percent   flux percent
              deviation       deviation       deviation       deviation       deviation    deviation
1  2.368E-07  .29 2.134E-07  .30 1.937E-07  .28 2.096E-07  .30 2.228E-07     .25    1.950E-07     .32
2  3.477E-08  .54 3.113E-08  .57 2.875E-08  .55 3.068E-08  .58 3.273E-08     .44    2.848E-08     .59

                    benchmark problem 16-a1     perturbed state    k16a1p1
fluxes for this problem
region  13       region  14       region  15       region   16      region   17        region  18
group   flux  percent   flux  percent   flux  percent   flux  percent   flux  percent   flux percent
              deviation       deviation       deviation       deviation       deviation    deviation
1 1.739E-07  .32 1.854E-07  .35 1.997E-07  .36 2.088E-07  .38   2.038E-07     .40    1.813E-07     .41
2 2.595E-08  .59 2.743E-08  .69 2.927E-08  .62 3.089E-08  .55   3.085E-08     .61    2.825E-08     .61

                    benchmark problem 16-a1     perturbed state    k16a1p1
fluxes for this problem
           region  19           region  20           region  21
group   flux     percent     flux     percent     flux     percent
              deviation            deviation            deviation
1    1.494E-07     .45    1.237E-07     .47    3.701E-08     .50
2    2.562E-08     .67    2.449E-08     .80    1.213E-08     .65
*************************************************************************************************
congratulations!  you have  successfully traversed the perilous path through keno v in 116.84550
                  minutes
*************************************************************************************************
```

Appendix C: Description of Benchmark Problems

Problem 16-A1

Identification:	16-A1
By:	H.L. Dodds, Jr. (University of Tennessee, Knoxville)
Date Submitted:	November, 1977
Descriptive Title:	Delayed Supercritical Transient; One-Dimensional, Two-Group Neutron Transport Problem in a Fast Reactor
Suggested Function:	Test transient one-dimensional neutron transport methods
Reactor Configuration:	Figure C.1
Data:	Initial two-group constants are shown in Table C.1. Delayed neutron parameters, prompt and delayed spectra, and velocities are shown in Table C.2.

Reduction of Source Situation:

1. One-dimensional (slab), two-group neutron transport theory
2. Isotropic scattering
3. Zero return current boundary conditions on external surfaces
4. Steady-state initial conditions
5. Six delayed neutron precursor groups

Initiating Perturbation:	At time = 0.0 sec, the density of the material in Zone 2 is increased be 5% and the density in Zone 6 is decreased by 5% resulting in a step insertion of reactivity at t = 0.0 sec.

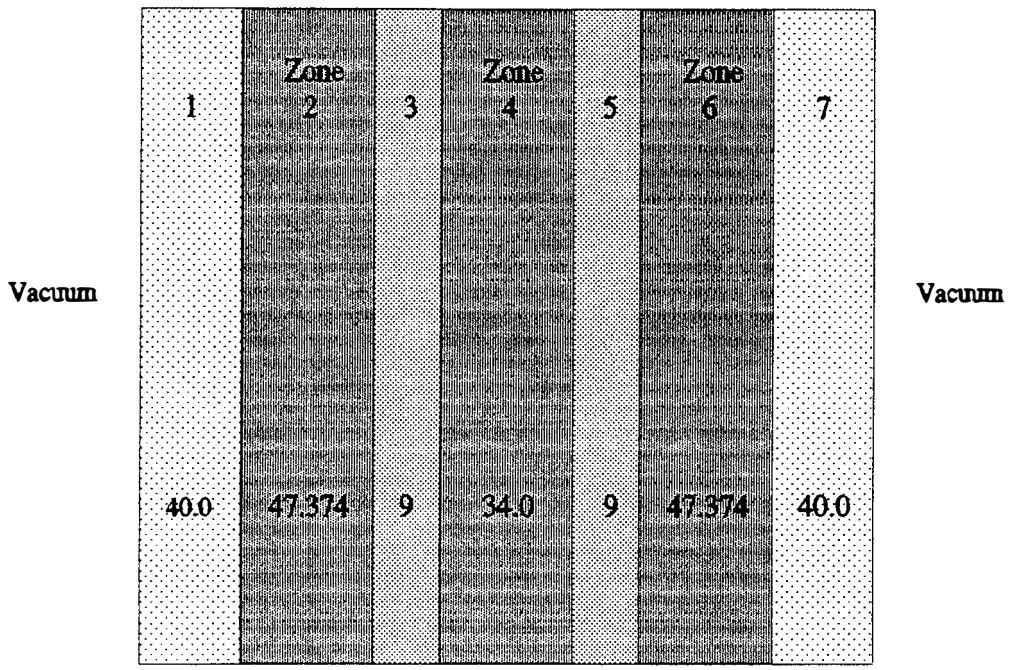

Figure C.1: Schematic Diagram of Benchmark Problem 16-A1

Table C.1: Benchmark Problem 16-A1 Initial Two-Group Data

Zone	Group	$\nu\Sigma_f^i$	Σ_t^i	$\Sigma_s^{i\to i}$	$\Sigma_s^{i\to j}$
1,7	1	8.3441×10^{-4}	2.411×10^{-1}	2.336×10^{-1}	3.598×10^{-3}
	2	3.2776×10^{-4}	4.172×10^{-1}	4.070×10^{-1}	0.0
2,4,6	1	7.4518×10^{-3}	1.849×10^{-1}	1.777×10^{-1}	2.085×10^{-3}
	2	1.1061×10^{-2}	3.668×10^{-1}	3.537×10^{-1}	0.0
3,5	1	0.0	9.432×10^{-2}	8.571×10^{-2}	1.717×10^{-3}
	2	0.0	1.876×10^{-1}	1.713×10^{-1}	0.0

Table C.2: Benchmark Problem 16-A1 Delayed Neutron Parameters[a]

Delayed Neutron Group	Delayed Neutron Fraction, β	Decay Constant, λ
1	8.10×10^{-5}	1.29×10^{-2}
2	6.87×10^{-4}	3.11×10^{-2}
3	6.12×10^{-4}	1.34×10^{-1}
4	1.14×10^{-3}	3.31×10^{-1}
5	5.12×10^{-4}	1.26
6	1.70×10^{-4}	3.21

[a]Prompt and delayed neutron spectra are identical with $\chi_1 = 1.0$ and $\chi_2 = 0.0$. Also, $1/v_1 = 1.851 \times 10^{-9}$ sec/cm and $1/v_2 = 1.088 \times 10^{-8}$ sec/cm.

Expected Primary Results:

1. Initial k-eff and initial scalar flux distribution for each group.
2. Convergence requirements on flux (and eigenvalue) if an iterative solution is used.
3. Total reactor power versus time (normalized so that total power = 1.0 MW at t = 0.0).
4. Time-dependent group flux distributions.
5. Sensitivity of results to time step size.
6. CPU time, I/O time, and core storage requirements.

Possible Additional Results:

7. Zone-averaged power fractions versus time.
8. Sensitivity of results to spatial mesh size.
9. Sensitivity of results to angular quadrature (if discrete ordinates method).

The initial configuration is made critical by dividing the production cross sections by k-eff, and the initial precursor concentrations are in equilibrium with the initial critical flux distribution.

Based on preliminary static k-eff calculations of the initial critical configuration using ANISN[32], an S_4 quadrature with 114 spatial intervals (as defined in Tables C.3 and C.4) is a sufficiently accurate representation for the angular and spatial discretization. Specifically, by using a finer spatial mesh (i.e., 228 intervals), k-eff changed by 0.0001 and using an S_8 ans S_{16} quadrature, k-eff changed by 0.0003 and 0.0003, respectively. Therefore, it is suggested that the "Expected Primary Results" given above be obtained initially using the discretization indicated in Tables C.3 and C.4 if the conventional discrete ordinates method is used.

Table C.3: S4 Angular Quadrature

Cosine (μ)	Weight
-1.0	0.00
-0.788675	0.25
-0.211325	0.25
+0.211325	0.25
+0.788675	0.25

Table C.4: Spatial Mesh

Zone	Number of Intervals
1	20
2	24
3	5
4	16
5	5
6	24
7	20

Problem 16-A2

Identification:	16-A2
By:	H.L. Dodds, Jr. (University of Tennessee, Knoxville)
Date Submitted:	November, 1977
Descriptive Title:	Prompt Supercritical Transient; One-Dimensional, Two-Group Neutron Transport Problem in a Fast Reactor
Suggested Function:	Test transient one-dimensional neutron transport methods
Reactor Configuration:	Same as Problem 16-A1
Data:	Same as Problem 16-A1

Reduction of Source Situation:

1. One-dimensional (slab), two-group neutron transport theory
2. Isotropic scattering
3. Zero return current boundary conditions on external surfaces
4. Steady-state initial conditions
5. Six delayed neutron precursor groups

Initiating Perturbation:	At time = 0.0 sec, the density of the material in Zone 2 is increased be 10% and the density in Zone 6 is decreased by 10% resulting in a step insertion of reactivity at t = 0.0 sec.

Problem 16-A3

Identification:	16-A3
By:	H.L. Dodds, Jr. (University of Tennessee, Knoxville)
Date Submitted:	November, 1977
Descriptive Title:	Transient Initially Prompt Supercritical, then Subcritical; One-Dimensional, Two-Group Neutron Transport Problem in a Fast Reactor
Suggested Function:	Same as Problem 16-A1
Reactor Configuration:	Same as Problem 16-A1
Data:	Same as Problem 16-A1

Reduction of Source Situation:

1. One-dimensional (slab), two-group neutron transport theory
2. Isotropic scattering
3. Zero return current boundary conditions on external surfaces
4. Steady-state initial conditions
5. Six delayed neutron precursor groups

Initiating Perturbation: At time = 0.0 sec, the material in Zone 5 is changed from a mixture of sodium and control rod materials to 100% sodium (i.e., control rod bank ejection). Then, at t=0.0001 sec, the material in Zone 3 is changed from a mixture of sodium and control rod materials to 100% control rod material (i.e., full insertion of the control rod bank). The cross sections for sodium and for the control rod material are shown in Table C.5.

Table C.5: Two-Group Constants for Sodium and Control Rod Materials

Material	Energy Group	Σ_t^i	$\Sigma_s^{i \to i}$	$\Sigma_s^{i \to j}$
Sodium	1	6.830×10^{-2}	6.3293×10^{-2}	1.294×10^{-3}
	2	1.257×10^{-1}	1.21099×10^{-1}	0.0
Control Rod	1	1.795×10^{-1}	1.59078×10^{-1}	3.101×10^{-3}
	2	3.903×10^{-1}	3.35661×10^{-1}	0.0

Problem 16-A6

Identification:	16-A6
By:	H.L. Dodds, Jr. (University of Tennessee, Knoxville)
Date Submitted:	May, 1978
Descriptive Title:	Material Motion Transient Initially Prompt Supercritical (Step Perturbation), then Subcritical (Ramp Perturbation); One-Dimensional, Two-Group Neutron Transport Problem in a Fast Reactor
Suggested Function:	Same as Problem 16-A1
Reactor Configuration:	Same as Problem 16-A1
Data:	Same as Problem 16-A1

Reduction of Source Situation:

1. One-dimensional (slab), two-group neutron transport theory
2. Isotropic scattering
3. Zero return current boundary conditions on external surfaces
4. Steady-state initial conditions
5. Six delayed neutron precursor groups

Initiating Perturbation: At time = 0.0 sec, the mixture of sodium and control rod material in Zone 5 is changed (a decrease in rod material and an increase in sodium) resulting in the cross sections for Zone 5 at t=0.0 sec given in Table C.6. In addition, for the time interval between t = 0.0 sec and t = 0.5 sec, the densities of the materials in Zones 1, 2, 4, 6, and 7 are changed linearly (i.e., materials motion) resulting in final cross sections for each zone at t = 0.5 sec as shown in Table C.7.

Table C.6: Two-Group Constants for Step Perturbation in Zone 5

Energy Group	Σ_t^i	$\Sigma_s^{i\to i}$	$\Sigma_s^{i\to j}$
1	9.05399×10^{-2}	8.24499×10^{-2}	1.6550×10^{-3}
2	1.7860×10^{-1}	1.640×10^{-1}	0.0

Table C.7: Two-Group Constants for Ramp Perturbations

Zone	Group	$\nu\Sigma_f^i$	Σ_t^i	$\Sigma_s^{i\to i}$	$\Sigma_s^{i\to j}$
1	1	1.71696×10^{-3}	2.62999×10^{-1}	2.54691×10^{-1}	3.8449×10^{-3}
	2	1.63779×10^{-3}	4.60642×10^{-1}	4.48897×10^{-1}	0.0
2	1	6.70661×10^{-3}	1.6641×10^{-1}	1.5994×10^{-1}	1.8765×10^{-3}
	2	9.95507×10^{-3}	3.3012×10^{-1}	3.18349×10^{-1}	0.0
4,6	1	2.98072×10^{-3}	7.39599×10^{-2}	7.10843×10^{-2}	8.3399×10^{-3}
	2	4.42448×10^{-3}	1.4672×10^{-1}	1.41488×10^{-1}	0.0
7	1	9.93014×10^{-3}	4.66791×10^{-1}	4.50559×10^{-1}	6.1429×10^{-3}
	2	1.38292×10^{-2}	8.64919×10^{-1}	8.38759×10^{-1}	0.0

Problem 16-A7

Identification:	16-A7
By:	H.L. Dodds, Jr. (University of Tennessee, Knoxville)
Date Submitted:	May, 1978
Descriptive Title:	Material Motion Transient Initially Prompt Supercritical (Step Perturbation), then Subcritical (Step Perturbation); One-Dimensional Two-Group Neutron Transport Problem in a Fast Reactor
Suggested Function:	Same as Problem 16-A1
Reactor Configuration:	Same as Problem 16-A1
Data:	Same as Problem 16-A1

Reduction of Source Situation:

1. One-dimensional (slab), two-group neutron transport theory
2. Isotropic scattering
3. Zero return current boundary conditions on external surfaces
4. Steady-state initial conditions
5. Six delayed neutron precursor groups

Initiating Perturbation:	This perturbation is identical to the perturbation for Problem 16-A6 except for the ramp representing material motion. Specifically, the ramp between 0.0 sec and 0.5 sec is replaced by a step which occurs at t = 0.01 sec. The magnitude of this step change at t = 0.01 sec is determined by the cross sections given in Table C.7.

Problem 14-A2

Identification:	14-A2
By:	S. Langenbuch (GRS-Munich) W. Werner (GRS-Munich)
Date Submitted:	June, 1976
Descriptive Title:	Super Prompt-critical Transient; Three-dimensional, Two-group Neutron Diffusion Problem, with Adiabatic Heatup and Doppler Feedback in a Thermal Reactor
Suggested Function:	Test 3-D Neutron Kinetics Solution, Especially for Coarse Mesh Methods
Reactor Configuration:	Figure C.2 and C.3
Data:	Initial two-group constants are shown in Table C.8. Delayed neutron parameters, prompt and delayed spectra, and velocities are shown in Table C.9.

Reduction of Source Situation:

1. Three-dimensional (x-y-z), two-group diffusion theory

2. Two delayed neutron precursor groups

3. Adiabatic heatup:

$$\alpha[\Sigma_{f_1}(\bar{x},t)\phi_1(\bar{x},t) + \Sigma_{f_2}(\bar{x},t)\phi_2(\bar{x},t)] = \frac{\partial}{\partial t}T(\bar{x},t)$$

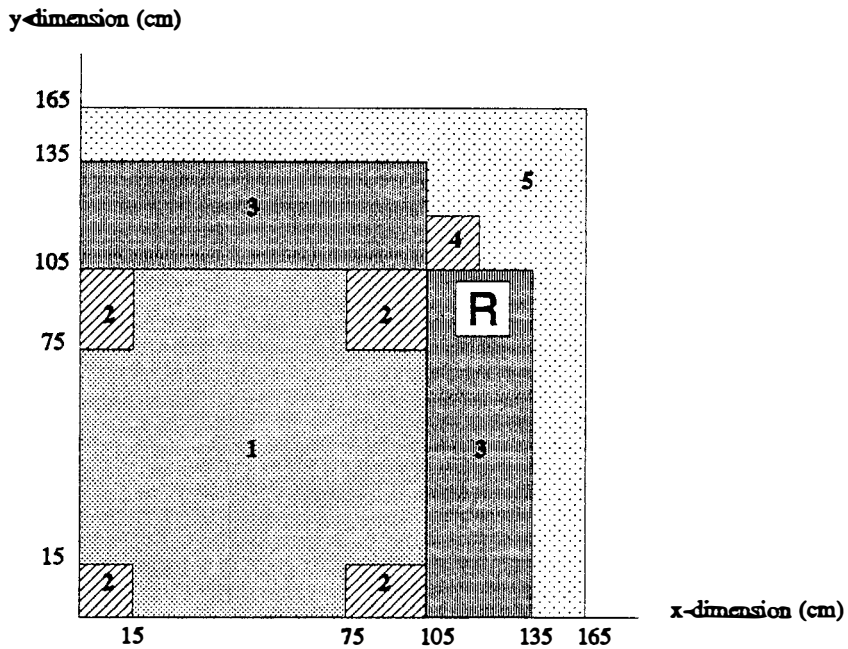

Figure C.2: Quadrant of Reactor Horizontal Cross Section for Benchmark 14-A2

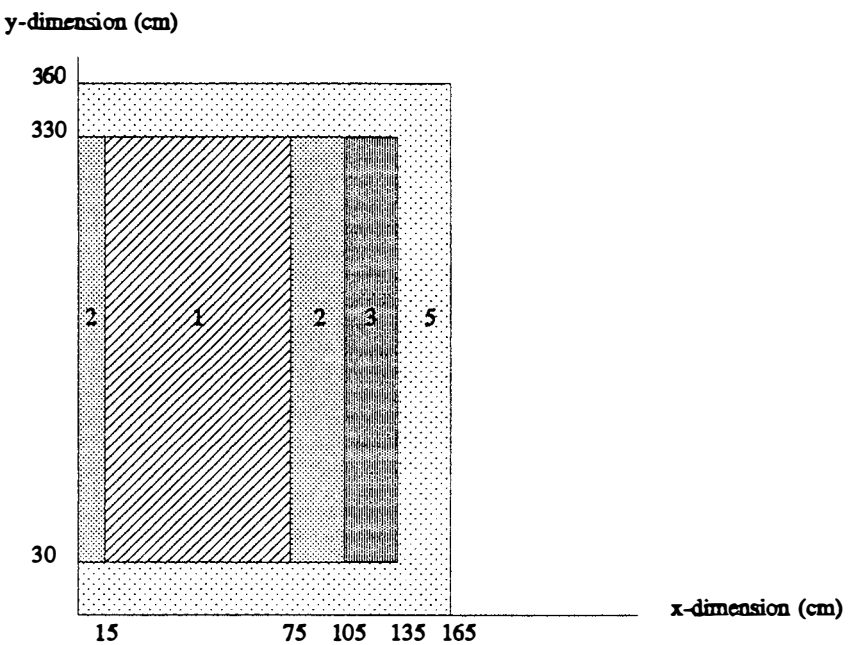

Figure C.3: Reactor Vertical Cross Section for Benchmark 14-A2

Table C.8: Benchmark Problem 14-A2 Delayed Neutron Parameters[a]

Delayed Neutron Group	Delayed Neutron Fraction, β	Decay Constant, λ
1	5.40×10^{-3}	6.54×10^{-3}
2	1.087×10^{-3}	1.35

[a]Prompt and delayed neutron spectra are identical with $\chi_1 = 1.0$ and $\chi_2 = 0.0$. Also, $1/v_1 = 3.333 \times 10^{-6}$ sec/cm and $1/v_2 = 3.333 \times 10^{-4}$ sec/cm.

Table C.9: Benchmark Problem 14-A2 Initial Two-Group Data

Zone	Material	Group	D_i	Σ_a^i	$\nu\Sigma_f^i$	$\Sigma_s^{1\to2}$
1	Fuel 1 with rod	1 2	1.255 0.211	8.252×10^{-3} 1.003×10^{-1}	4.602×10^{-3} 1.091×10^{-1}	2.533×10^{-2}
2	Fuel 2 without rod	1 2	1.268 0.1902	7.181×10^{-3} 7.047×10^{-2}	4.609×10^{-3} 8.675×10^{-2}	2.767×10^{-2}
3	Fuel 2 with rod	1 2	1.259 0.2091	8.002×10^{-3} 8.344×10^{-2}	4.663×10^{-3} 1.021×10^{-1}	2.617×10^{-2}
4	Fuel 2 without rod	1 2	1.259 0.2091	8.002×10^{-3} 7.332×10^{-2}	4.663×10^{-3} 1.021×10^{-1}	2.617×10^{-2}
5	Reflector	1 2	1.257 0.1592	6.034×10^{-4} 1.911×10^{-2}	0.0 0.0	4.754×10^{-2}

4. Doppler feedback:

$$\Sigma_{a_1}(\bar{x},t) = \Sigma_{a_1}(\bar{x},t=0)\ [1 + \gamma(\sqrt{T(\bar{x},t)} - \sqrt{T_0})]$$

5. Power:

$$P(\bar{x},t) = \varepsilon[\Sigma_{f_1}(\bar{x},t)\phi_1(\bar{x},t) + \Sigma_{f_2}(\bar{x},t)\phi_2(\bar{x},t)]$$

Data for feedback model:

$\alpha = 3.83 \times 10^{-11}$ K cm^3	conversion factor
$\gamma = 3.034 \times 10^{-3}$ K$^{1/2}$	feedback constant
$\epsilon = 3.204 \times 10^{-11}$ W sec / fission	energy conversion factor

Initiating Perturbation:

The initial configuration is made critical by dividing the production cross sections by k_{eff}. The initial flux distribution is normalized such that the average power density is:

$$\bar{P} = \frac{\varepsilon}{V_{core}}\int(\Sigma_{f_1}\phi_1 + \Sigma_{f_2}\phi_2)dV = 1.0x10^{-6}\ Wcm^{-3}$$

Further, the initial precursor concentrations are in equilibrium with the initial critical flux distribution. Finally, the initial temperature is 300K.

Cross Section Perturbation:

$$\frac{\Sigma_{a_2}(t)}{\Sigma_{a_2}(t=0)} = 1 - 0.0606184t \qquad t \leq 2$$

$$0.8787631 \qquad t \geq 2$$

where, t = time (sec).

Appendix D: Description of Mesh Intervals

The number of mesh intervals for Benchmark Problems 16-A1, 16-A2, and 16-A3 are 21, 32, and 29, respectively. The mesh intervals are defined in detail in Tables D.1, D.2, and D.3 as x-dimensions since these problems are one-dimensional. Note that the x-dimensions are given as ranges so that the top and bottom numbers correspond to starting and ending points for each mesh interval, respectively. Similar information for Benchmark Problems 16-A6 and 16-A7 (36 intervals each) is presented in Table D.4. Note that the y and z dimensions are ± 1000.0 cm for each of the Benchmark 16 problems. Also given in the tables is the reactor zone (refer to Figure 3 on page 25) corresponding to each mesh interval.

The 217 mesh intervals for Benchmark Problem 14-A2 are given in Table D.5 as x-dimensions, y-dimensions, and z-dimensions since the problem is three-dimensional. Note that the dimensions are given as ranges so that the top and bottom numbers correspond to starting and ending points for each mesh interval, respectively. Also given in Table D.5 is the reactor zone (refer to Figures 11 and 12 on pages 39 and 40) corresponding to each mesh interval.

Table D.1: Mesh Intervals for Benchmark Problem 16-A1

Mesh Interval	x-dimension (cm)	Zone
1	0.0 40.0	1
2	40.0 43.0	2
3	43.0 50.0	2
4	50.0 59.1	2
5	59.1 68.2	2
6	68.2 77.374	2
7	77.374 84.374	2
8	84.374 87.374	2
9	87.374 96.374	3
10	96.374 101.374	4
11	101.374 125.374	4
12	125.374 130.374	4
13	130.374 139.374	5
14	139.374 142.374	6
15	142.374 149.374	6

16	149.374 158.5	6
17	158.5 167.6	6
18	167.6 176.748	6
19	176.748 183.748	6
20	183.748 186.748	6
21	186.748 226.748	7

Table D.2: Mesh Intervals for Benchmark Problem 16-A2

Mesh Interval	x-dimension (cm)	Zone
1	0.0 5.0	1
2	5.0 20.0	1
3	20.0 35.0	1
4	35.0 40.0	1
5	40.0 43.0	2
6	43.0 50.0	2
7	50.0 59.1	2
8	59.1 68.2	2

9	68.2 77.374	2
10	77.374 84.374	2
11	84.374 87.374	2
12	87.374 89.374	3
13	89.374 94.374	3
14	94.374 96.374	3
15	96.374 101.374	4
16	101.374 113.374	4
17	113.374 125.374	4
18	125.374 130.374	4
19	130.374 132.374	5
20	132.374 137.374	5
21	137.374 139.374	5
22	139.374 142.374	6
23	142.374 149.374	6
24	149.374 158.5	6
25	158.5 167.6	6

26	167.6 176.748	6
27	176.748 183.748	6
28	183.748 186.748	6
29	186.748 191.748	7
30	191.748 206.748	7
31	206.748 221.748	7
32	221.748 226.748	7

Table D.3: Mesh Intervals for Benchmark Problem 16-A3

Mesh Interval	x-dimension (cm)	Zone
1	0.0 10.0	1
2	10.0 30.0	1
3	30.0 40.0	1
4	40.0 45.0	2
5	45.0 55.0	2
6	55.0 72.374	2
7	72.374 82.374	2

8	82.374 87.374	2
9	87.374 88.374	3
10	88.374 90.374	3
11	90.374 93.374	3
12	93.374 95.374	3
13	95.374 96.374	3
14	96.374 101.374	4
15	101.374 125.374	4
16	125.374 130.374	4
17	130.374 131.374	5
18	131.374 133.374	5
19	133.374 136.374	5
20	136.374 138.374	5
21	138.374 139.374	5
22	139.374 144.374	6
23	144.374 154.374	6
24	154.374 171.748	6

25	171.748 181.748	6
26	181.748 186.748	6
27	186.748 196.748	7
28	196.748 216.748	7
29	216.748 226.748	7

Table D.4: Mesh Intervals for Benchmark Problems 16-A6 and 16-A7

Mesh Interval	x-dimension (cm)	Zone
1	0.0 5.0	1
2	5.0 10.0	1
3	10.0 20.0	1
4	20.0 30.0	1
5	30.0 35.0	1
6	35.0 40.0	1
7	40.0 43.0	2
8	43.0 50.0	2
9	50.0 59.1	2

10	59.1 68.2	2
11	68.2 77.374	2
12	77.374 84.374	2
13	84.374 87.374	2
14	87.374 96.374	3
15	96.374 101.374	4
16	101.374 106.374	4
17	106.374 115.374	4
18	115.374 125.374	4
19	125.374 130.374	4
20	130.374 131.374	5
21	131.374 134.374	5
22	134.374 138.374	5
23	138.374 139.374	5
24	139.374 142.374	6
25	142.374 149.374	6
26	149.374 158.5	6

27	158.5 167.6	6
28	167.6 176.748	6
29	176.748 183.748	6
30	183.748 186.748	6
31	186.748 191.748	7
32	191.748 196.748	7
33	196.748 206.748	7
34	206.748 216.748	7
35	216.748 221.748	7
36	221.748 226.748	7

Table D.5: Mesh Intervals for Benchmark Problem 14-A2

Mesh Interval	x-dimensions (cm)	y-dimensions (cm)	z-dimensions (cm)	Zone
1	0.0 15.0	0.0 15.0	30.0 80.0	2
2	15.0 75.0	0.0 15.0	30.0 80.0	1
3	75.0 105.0	0.0 15.0	30.0 80.0	2
4	105.0 135.0	0.0 15.0	30.0 80.0	3

5	0.0 75.0	15.0 45.0	30.0 80.0	1
6	75.0 105.0	15.0 45.0	30.0 80.0	1
7	105.0 135.0	15.0 45.0	30.0 80.0	3
8	0.0 75.0	45.0 65.0	30.0 80.0	1
9	75.0 105.0	45.0 65.0	30.0 80.0	1
10	105.0 135.0	45.0 65.0	30.0 80.0	3
11	0.0 60.0	60.0 75.0	30.0 80.0	1
12	60.0 90.0	60.0 75.0	30.0 80.0	1
13	90.0 105.0	60.0 75.0	30.0 80.0	1
14	105.0 135.0	60.0 75.0	30.0 80.0	3
15	0.0 15.0	75.0 90.0	30.0 80.0	2
16	15.0 45.0	75.0 90.0	30.0 80.0	1
17	45.0 75.0	75.0 90.0	30.0 80.0	1
18	75.0 90.0	75.0 90.0	30.0 80.0	2
19	90.0 105.0	75.0 90.0	30.0 80.0	2
20	105.0 120.0	75.0 90.0	30.0 80.0	3
21	120.0 135.0	75.0 90.0	30.0 80.0	3

22	0.0 15.0	90.0 105.0	30.0 80.0	2
23	15.0 45.0	90.0 105.0	30.0 80.0	1
24	45.0 75.0	90.0 105.0	30.0 80.0	1
25	75.0 90.0	90.0 105.0	30.0 80.0	2
26	90.0 105.0	90.0 105.0	30.0 80.0	2
27	105.0 120.0	90.0 105.0	30.0 80.0	3
28	120.0 135.0	90.0 105.0	30.0 80.0	3
29	0.0 60.0	105.0 120.0	30.0 80.0	3
30	60.0 90.0	105.0 120.0	30.0 80.0	3
31	90.0 105.0	105.0 120.0	30.0 80.0	3
32	105.0 120.0	105.0 120.0	30.0 80.0	4
33	120.0 135.0	105.0 120.0	30.0 80.0	5
34	0.0 75.0	120.0 135.0	30.0 80.0	3
35	75.0 105.0	120.0 135.0	30.0 80.0	3
36	105.0 135.0	120.0 135.0	30.0 80.0	5
37	0.0 15.0	0.0 15.0	80.0 130.0	2
38	15.0 75.0	0.0 15.0	80.0 130.0	1

39	75.0 105.0	0.0 15.0	80.0 130.0	2
40	105.0 135.0	0.0 15.0	80.0 130.0	3
41	0.0 75.0	15.0 45.0	80.0 130.0	1
42	75.0 105.0	15.0 45.0	80.0 130.0	1
43	105.0 135.0	15.0 45.0	80.0 130.0	3
44	0.0 75.0	45.0 60.0	80.0 130.0	1
45	75.0 105.0	45.0 60.0	80.0 130.0	1
46	105.0 135.0	45.0 60.0	80.0 130.0	3
47	0.0 60.0	60.0 75.0	80.0 130.0	1
48	60.0 90.0	60.0 75.0	80.0 130.0	1
49	90.0 105.0	60.0 75.0	80.0 130.0	1
50	105.0 135.0	60.0 75.0	80.0 130.0	3
51	0.0 15.0	75.0 90.0	80.0 130.0	2
52	15.0 45.0	75.0 90.0	80.0 130.0	1
53	45.0 75.0	75.0 90.0	80.0 130.0	1
54	75.0 90.0	75.0 90.0	80.0 130.0	2
55	90.0 105.0	75.0 90.0	80.0 130.0	2

56	105.0 120.0	75.0 90.0	80.0 130.0	3
57	120.0 135.0	75.0 90.0	80.0 130.0	3
58	0.0 15.0	90.0 105.0	80.0 130.0	2
59	15.0 45.0	90.0 105.0	80.0 130.0	1
60	45.0 75.0	90.0 105.0	80.0 130.0	1
61	75.0 90.0	90.0 105.0	80.0 130.0	2
62	90.0 105.0	90.0 105.0	80.0 130.0	2
63	105.0 120.0	90.0 105.0	80.0 130.0	3
64	120.0 135.0	90.0 105.0	80.0 130.0	3
65	0.0 60.0	105.0 120.0	80.0 130.0	3
66	60.0 90.0	105.0 120.0	80.0 130.0	3
67	90.0 105.0	105.0 120.0	80.0 130.0	3
68	105.0 120.0	105.0 120.0	80.0 130.0	4
69	120.0 135.0	105.0 120.0	80.0 130.0	5
70	0.0 75.0	120.0 135.0	80.0 130.0	3
71	75.0 105.0	120.0 135.0	80.0 130.0	3
72	105.0 135.0	120.0 135.0	80.0 130.0	5

73	0.0 15.0	0.0 15.0	130.0 180.0	2
74	15.0 75.0	0.0 15.0	130.0 180.0	1
75	75.0 105.0	0.0 15.0	130.0 180.0	2
76	105.0 135.0	0.0 15.0	130.0 180.0	3
77	0.0 75.0	15.0 45.0	130.0 180.0	1
78	75.0 105.0	15.0 45.0	130.0 180.0	1
79	105.0 135.0	15.0 45.0	130.0 180.0	3
80	0.0 75.0	45.0 60.0	130.0 180.0	1
81	75.0 105.0	45.0 60.0	130.0 180.0	1
82	105.0 135.0	45.0 60.0	130.0 180.0	3
83	0.0 60.0	60.0 75.0	130.0 180.0	1
84	60.0 90.0	60.0 75.0	130.0 180.0	1
85	90.0 105.0	60.0 75.0	130.0 180.0	1
86	105.0 135.0	60.0 75.0	130.0 180.0	3
87	0.0 15.0	75.0 90.0	130.0 180.0	2
88	15.0 45.0	75.0 90.0	130.0 180.0	1
89	45.0 75.0	75.0 90.0	130.0 180.0	1

90	75.0 90.0	75.0 90.0	130.0 180.0	2
91	90.0 105.0	75.0 90.0	130.0 180.0	2
92	105.0 120.0	75.0 90.0	130.0 180.0	3
93	120.0 135.0	75.0 90.0	130.0 180.0	3
94	0.0 15.0	90.0 105.0	130.0 180.0	2
95	15.0 45.0	90.0 105.0	130.0 180.0	1
96	45.0 75.0	90.0 105.0	130.0 180.0	1
97	75.0 90.0	90.0 105.0	130.0 180.0	2
98	90.0 105.0	90.0 105.0	130.0 180.0	2
99	105.0 120.0	90.0 105.0	130.0 180.0	3
100	120.0 135.0	90.0 105.0	130.0 180.0	3
101	0.0 60.0	105.0 120.0	130.0 180.0	3
102	60.0 90.0	105.0 120.0	130.0 180.0	3
103	90.0 105.0	105.0 120.0	130.0 180.0	3
104	105.0 120.0	105.0 120.0	130.0 180.0	4
105	120.0 135.0	105.0 120.0	130.0 180.0	5
106	0.0 75.0	120.0 135.0	130.0 180.0	3

107	75.0 105.0	120.0 135.0	130.0 180.0	3
108	105.0 135.0	120.0 135.0	130.0 180.0	5
109	0.0 15.0	0.0 15.0	180.0 230.0	2
110	15.0 75.0	0.0 15.0	180.0 230.0	1
111	75.0 105.0	0.0 15.0	180.0 230.0	2
112	105.0 135.0	0.0 15.0	180.0 230.0	3
113	0.0 75.0	15.0 45.0	180.0 230.0	1
114	75.0 105.0	15.0 45.0	180.0 230.0	1
115	105.0 135.0	15.0 45.0	180.0 230.0	3
116	0.0 75.0	45.0 60.0	180.0 230.0	1
117	75.0 105.0	45.0 60.0	180.0 230.0	1
118	105.0 135.0	45.0 60.0	180.0 230.0	3
119	0.0 60.0	60.0 75.0	180.0 230.0	1
120	60.0 90.0	60.0 75.0	180.0 230.0	1
121	90.0 105.0	60.0 75.0	180.0 230.0	1
122	105.0 135.0	60.0 75.0	180.0 230.0	3
123	0.0 15.0	75.0 90.0	180.0 230.0	2

124	15.0 45.0	75.0 90.0	180.0 230.0	1
125	45.0 75.0	75.0 90.0	180.0 230.0	1
126	75.0 90.0	75.0 90.0	180.0 230.0	2
127	90.0 105.0	75.0 90.0	180.0 230.0	2
128	105.0 120.0	75.0 90.0	180.0 230.0	3
129	120.0 135.0	75.0 90.0	180.0 230.0	3
130	0.0 15.0	90.0 105.0	180.0 230.0	2
131	15.0 45.0	90.0 105.0	180.0 230.0	1
132	45.0 75.0	90.0 105.0	180.0 230.0	1
133	75.0 90.0	90.0 105.0	180.0 230.0	2
134	90.0 105.0	90.0 105.0	180.0 230.0	2
135	105.0 120.0	90.0 105.0	180.0 230.0	3
136	120.0 135.0	90.0 105.0	180.0 230.0	3
137	0.0 60.0	105.0 120.0	180.0 230.0	3
138	60.0 90.0	105.0 120.0	180.0 230.0	3
139	90.0 105.0	105.0 120.0	180.0 230.0	3
140	105.0 120.0	105.0 120.0	180.0 230.0	4

141	120.0 135.0	105.0 120.0	180.0 230.0	5
142	0.0 75.0	120.0 135.0	180.0 230.0	3
143	75.0 105.0	120.0 135.0	180.0 230.0	3
144	105.0 135.0	120.0 135.0	180.0 230.0	5
145	0.0 15.0	0.0 15.0	230.0 280.0	2
146	15.0 75.0	0.0 15.0	230.0 280.0	1
147	75.0 105.0	0.0 15.0	230.0 280.0	2
148	105.0 135.0	0.0 15.0	230.0 280.0	3
149	0.0 75.0	15.0 45.0	230.0 280.0	1
150	75.0 105.0	15.0 45.0	230.0 280.0	1
151	105.0 135.0	15.0 45.0	230.0 280.0	3
152	0.0 75.0	45.0 60.0	230.0 280.0	1
153	75.0 105.0	45.0 60.0	230.0 280.0	1
154	105.0 135.0	45.0 60.0	230.0 280.0	3
155	0.0 60.0	60.0 75.0	230.0 280.0	1
156	60.0 90.0	60.0 75.0	230.0 280.0	1
157	90.0 105.0	60.0 75.0	230.0 280.0	1

158	105.0 135.0	60.0 75.0	230.0 280.0	3
159	0.0 15.0	75.0 90.0	230.0 280.0	2
160	15.0 45.0	75.0 90.0	230.0 280.0	1
161	45.0 75.0	75.0 90.0	230.0 280.0	1
162	75.0 90.0	75.0 90.0	230.0 280.0	2
163	90.0 105.0	75.0 90.0	230.0 280.0	2
164	105.0 120.0	75.0 90.0	230.0 280.0	3
165	120.0 135.0	75.0 90.0	230.0 280.0	3
166	0.0 15.0	90.0 105.0	230.0 280.0	2
167	15.0 45.0	90.0 105.0	230.0 280.0	1
168	45.0 75.0	90.0 105.0	230.0 280.0	1
169	75.0 90.0	90.0 105.0	230.0 280.0	2
170	90.0 105.0	90.0 105.0	230.0 280.0	2
171	105.0 120.0	90.0 105.0	230.0 280.0	3
172	120.0 135.0	90.0 105.0	230.0 280.0	3
173	0.0 60.0	105.0 120.0	230.0 280.0	3
174	60.0 90.0	105.0 120.0	230.0 280.0	3

175	90.0 105.0	105.0 120.0	230.0 280.0	3
176	105.0 120.0	105.0 120.0	230.0 280.0	4
177	120.0 135.0	105.0 120.0	230.0 280.0	5
178	0.0 75.0	120.0 135.0	230.0 280.0	3
179	75.0 105.0	120.0 135.0	230.0 280.0	3
180	105.0 135.0	120.0 135.0	230.0 280.0	5
181	0.0 15.0	0.0 15.0	280.0 330.0	2
182	15.0 75.0	0.0 15.0	280.0 330.0	1
183	75.0 105.0	0.0 15.0	280.0 330.0	2
184	105.0 135.0	0.0 15.0	280.0 330.0	3
185	0.0 75.0	15.0 45.0	280.0 330.0	1
186	75.0 105.0	15.0 45.0	280.0 330.0	1
187	105.0 135.0	15.0 45.0	280.0 330.0	3
188	0.0 75.0	45.0 60.0	280.0 330.0	1
189	75.0 105.0	45.0 60.0	280.0 330.0	1
190	105.0 135.0	45.0 60.0	280.0 330.0	3
191	0.0 60.0	60.0 75.0	280.0 330.0	1

192	60.0 90.0	60.0 75.0	280.0 330.0	1
193	90.0 105.0	60.0 75.0	280.0 330.0	1
194	105.0 135.0	60.0 75.0	280.0 330.0	3
195	0.0 15.0	75.0 90.0	280.0 330.0	2
196	15.0 45.0	75.0 90.0	280.0 330.0	1
197	45.0 75.0	75.0 90.0	280.0 330.0	1
198	75.0 90.0	75.0 90.0	280.0 330.0	2
199	90.0 105.0	75.0 90.0	280.0 330.0	2
200	105.0 120.0	75.0 90.0	280.0 330.0	3
201	120.0 135.0	75.0 90.0	280.0 330.0	3
202	0.0 15.0	90.0 105.0	280.0 330.0	2
203	15.0 45.0	90.0 105.0	280.0 330.0	1
204	45.0 75.0	90.0 105.0	280.0 330.0	1
205	75.0 90.0	90.0 105.0	280.0 330.0	2
206	90.0 105.0	90.0 105.0	280.0 330.0	2
207	105.0 120.0	90.0 105.0	280.0 330.0	3
208	120.0 135.0	90.0 105.0	280.0 330.0	3

209	0.0 60.0	105.0 120.0	280.0 330.0	3
210	60.0 90.0	105.0 120.0	280.0 330.0	3
211	90.0 105.0	105.0 120.0	280.0 330.0	3
212	105.0 120.0	105.0 120.0	280.0 330.0	4
213	120.0 135.0	105.0 120.0	280.0 330.0	5
214	0.0 75.0	120.0 135.0	280.0 330.0	3
215	75.0 105.0	120.0 135.0	280.0 330.0	3
216	105.0 135.0	120.0 135.0	280.0 330.0	5
217	135.0 165.0	135.0 165.0	0.0 360.0	5

Vita

Charles Lewis Bentley was born in Elizabethtown, Kentucky on December 22, 1968. He graduated from East Hardin High School in May 1987. The following August he entered Western Kentucky University where he received a Bachelor of Science degree in Physics in May 1990.

In August 1990 he accepted a Pasqua Nuclear Engineering Scholarship at the University of Tennessee, Knoxville. In May 1992 he received the degree of Bachelor of Science in Nuclear Engineering.

In June 1992 he accepted a Research Fellowship from the Nuclear Engineering/ Health Physics Fellowship Program sponsored by the U.S. Department of Energy. In August 1992 he began graduate study at the University of Tennessee toward a Master of Science degree in Nuclear Engineering. This degree was awarded in August 1994.

In August 1994 he began pursuing a Doctor of Philosophy degree in Nuclear Engineering which was awarded in December 1996.