To the Graduate Council:

I am submitting herewith a dissertation written by Sedat Goluoglu entitled "A 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

# A Deterministic Method for Transient, Three-Dimensional Neutron Transport

A Dissertation
Presented for the
Doctor of Philosophy
Degree
The University of Tennessee, Knoxville

Sedat Goluoglu

August 1997

## Dedication

This dissertation is dedicated to my parents, Aysel Goluoğlu and the late Hakkı Goluoğlu.

## 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. My committee members, Dr. M.W. Guidry, Dr. L.F. Miller, and Dr. B.R. Upadhyaya also provided invaluable support. In addition, I wish to thank the members, past and present, of the Nuclear Criticality Safety Group at the University of Tennessee, Knoxville; particularly, Dr. C.L. Bentley, R.V. DeMeglio, Dr. M.E. Dunn, K.L. Norton, and Dr. R.E. Pevey. I would also like to thank Dr. Y. Azmy, Dr. B. Childs, P. Fox, Dr. J. Gehin, N. Hatmaker, Dr. R.A. Lillie, Dr. L.M. Petrie, D.B. Simpson, and Dr. R.M. Westfall of Oak Ridge National Laboratory for their support. Financial support for most of this work was provided by Lockheed Martin Energy Systems via a research contract with the University of Tennessee Nuclear Engineering Department. Finally, I would like to express my deepest appreciation to the members of my family who are always there for me.

# Abstract

A deterministic method for solving the time-dependent, three-dimensional Boltzmann transport equation with explicit representation of delayed neutrons has been developed and evaluated. The methodology used in this study for the time variable is the improved quasi-static (IQS) method. The position, energy, and angle variables of the neutron flux are computed using the three-dimensional (3-D) discrete ordinates code TORT. The resulting time-dependent, 3-D code is called TDTORT. The flux shape calculated by TORT is used to compute the point kinetics parameters (e.g., reactivity, generation time, etc.). The amplitude function is calculated by solving the point kinetics equations using LSODE (Livermore Solver of Ordinary differential Equations). Several transient 1-D, 2-D, and 3-D benchmark problems are used to verify TDTORT. The results show that methodology and code developed in this work have sufficient accuracy and speed to serve as a benchmarking tool for other less accurate models and codes. More importantly, a new computational tool based on transport theory now exists for analyzing the dynamic behavior of complex neutronic systems.

# Table of Contents


| Chapter | Page |
|---|---|
| 1. Introduction | 1 |
| 1.1 Neutronics Models | 1 |
| 1.2 Previous Work | 4 |
| 1.3 Objectives | 6 |
| 1.4 Justification and Originality of Work | 7 |
| 1.5 Thesis Organization | 8 |
| 2. Theory | 9 |
| 2.1 Improved Quasistatic Method | 9 |
| 2.2 Numerical Approximations | 13 |
| 3. Time-Dependent TORT Code (TDTORT) | 18 |
| 3.1 Introduction | 18 |
| 3.2 Perturbations and I/O Units | 20 |
| 4. Results | 22 |
| 4.1 General Remarks | 22 |
| 4.2 Analytical 1-D Test Problem | 23 |
| 4.3 Benchmark Problem 16-A1 | 24 |
| 4.4 Benchmark Problem 16-A2 | 31 |
| 4.5 Benchmark Problem 16-A3 | 33 |
| 4.6 Benchmark Problem 16-A6 | 36 |
| 4.7 Benchmark Problem 16-A7 | 38 |
| 4.8 Benchmark Problem 8-A1 | 41 |
| 4.9 2-D TWIGL Test Problem | 48 |
| 4.10 Benchmark Problem 14-A1 | 53 |
| 4.11 Benchmark Problem 14-A2 | 57 |
| 5. Summary, Conclusions, and Suggestions for Future Work | 61 |
| 5.1 Summary | 61 |
| 5.2 Conclusions | 62 |
| 5.3 Suggestions for Future Work | 62 |
| References | 64 |
| Appendices | 70 |
| Appendix A: Description of Benchmark Problems | 71 |

A.1 Problem 16-A1 . . . . . 71
A.2 Problem 16-A2 . . . . . 75
A.3 Problem 16-A3 . . . . . 76
A.4 Problem 16-A6 . . . . . 78
A.5 Problem 16-A7 . . . . . 80
A.6 Problem 8-A1 . . . . . 82
A.7 TWIGL Test Problem . . . . . 86
A.8 Problems 14-A1 and 14-A2 . . . . . 88
Appendix B: TDTORT Users Manual . . . . . 94
B.1 Input File Description . . . . . 94
B.2 Transient Input . . . . . 98
B.3 Cross Section Input Files . . . . . 103
B.4 Output Files . . . . . 104
B.5 TDTORT Code Listing . . . . . 113

Vita . . . . . 239

# List of Figures


| Figure | | Page |
|---|---|---|
| Figure 1 | Time-step Hierarchy of the Improved Quasistatic Method. | 16 |
| Figure 2 | Simplified Flow Chart for TDTORT | 19 |
| Figure 3 | Relative Power vs. Time Computed by TDTORT for the Analytical one-group Problem. | 25 |
| Figure 4 | Schematic Diagram of Benchmark Problem 16. | 27 |
| Figure 5 | Relative Power Comparisons of TDTORT and TIMEX(Benchmark) for Problem 16-A1. | 28 |
| Figure 6 | Normalized Flux vs. Radial Position for Benchmark 16-A1 (Group 1). | 29 |
| Figure 7 | Normalized Flux vs. Radial Position for Benchmark 16-A1 (Group 2). | 30 |
| Figure 8 | Relative Power Comparisons of TDTORT and TIMEX(Benchmark) for Problem 16-A2. | 32 |
| Figure 9 | Relative Power Comparisons of TDTORT and TIMEX(Benchmark) for Problem 16-A3. | 34 |
| Figure 10 | Relative Power Comparisons of TDTORT and TDA(Benchmark) for Problem 16-A6. | 37 |
| Figure 11 | Relative Power Comparisons of TDTORT and TDA(Benchmark) for Problem 16-A7. | 40 |
| Figure 12 | TWODQD Normalized Flux for Group 1 at 0.0 Seconds. | 42 |
| Figure 13 | TWODQD Normalized Flux for Group 2 at 0.0 Seconds. | 42 |
| Figure 14 | TWODQD Normalized Flux for Group 1 at 4.0 Seconds. | 43 |
| Figure 15 | TWODQD Normalized Flux for Group 2 at 4.0 Seconds. | 43 |
| Figure 16 | TDTORT Normalized Flux for Group 1 at 0.0 Seconds. | 44 |
| Figure 17 | TDTORT Normalized Flux for Group 2 at 0.0 Seconds. | 44 |

Figure 18 TDTORT Normalized Flux for Group 1 at 4.0 Seconds. . . . . 45
Figure 19 TDTORT Normalized Flux for Group 2 at 4.0 Seconds. . . . . 45
Figure 20 Relative Power Comparisons of TDTORT and TW0DTA(Benchmark) for Problem 8-A1. . . . . 46
Figure 21 Relative Power Comparisons of TDTORT and CONQUEST for the TWIGL Test Problem (step perturbation). . . . . 51
Figure 22 Relative Power Comparisons of TDTORT and CONQUEST for the TWIGL Test Problem (ramp perturbation). . . . . 52
Figure 23 Power vs. Time Comparisons of TDTORT with an $S_2$ Quadrature and Benchmark for Problem 14-A1. . . . . 54
Figure 24 Power vs. Time Comparisons of TDTORT with an $S_4$ Quadrature and Benchmark for Problem 14-A1. . . . . 55
Figure 25 Power vs. Time Comparisons of TDTORT and Benchmark for Problem 14-A1. . . . . 56
Figure 26 Average Power Density vs. Time Comparisons of Benchmarks for Problem 14-A2. . . . . 58
Figure 27 Average Power Density vs. Time Comparisons of TDTORT and Benchmark for Problem 14-A2. . . . . 59
Figure 28 Schematic Diagram of Benchmark Problem 8-A1. . . . . 83
Figure 29 Schematic Diagram of 2-D TWIGL Test Problem. . . . . 86
Figure 30 Quadrant of Reactor Horizontal Cross Section for Benchmark 14-A2. . . . . 89
Figure 31 Reactor Vertical Cross Section for Benchmark 14-A2. . . . . 90

# List of Tables


| Table | | Page |
|---|---|---|
| Table I | One Group Cross Sections for 1-D Analytical Test Problem | 24 |
| Table II | Point Kinetics Parameters for 1-D Analytical Test Problem | 24 |
| Table III | Point Kinetics Parameters for Benchmark Problem 16-A1 | 26 |
| Table IV | Point Kinetics Parameters for Benchmark Problem 16-A2 | 31 |
| Table V | Point Kinetics Parameters for Benchmark Problem 16-A3 | 35 |
| Table VI | Point Kinetics Parameters for Benchmark Problem 16-A6 | 38 |
| Table VII | Point Kinetics Parameters for Benchmark Problem 16-A7 | 39 |
| Table VIII | Point Kinetics Parameters for Benchmark Problem 8-A1 | 47 |
| Table IX | Point Kinetics Parameters for TWIGL Test Problem (step perturbation) | 49 |
| Table X | Point Kinetics Parameters for TWIGL Test Problem (ramp perturbation) | 50 |
| Table XI | Shape Update Time Intervals for 14-A1. | 57 |
| Table XII | Shape Update Time Intervals for 14-A2. | 60 |
| Table A.1 | Initial Cross Sections for Benchmark Problem 16-A1 | 73 |
| Table A.2 | $S_4$ Angular Quadrature | 74 |
| Table A.3 | Spatial Mesh | 74 |
| Table A.4 | Two-Group Constants for Sodium and Control Rod Materials | 77 |
| Table A.5 | Two-Group Constants for Step Perturbation in Zone 5 | 79 |
| Table A.6 | Two-Group Constants for Ramp Perturbations | 79 |
| Table A.7 | Initial Cross Sections for Benchmark Problem 8-A1 | 84 |
| Table A.8 | Delayed Neutron Parameters for Benchmark Problem 8-A1. | 85 |
| Table A.9 | Initial Cross Sections for 2-D TWIGL Test Problem | 87 |
| Table A.10 | Delayed Neutron Parameters for TWIGL Test Problem. | 87 |
| Table A.11 | Benchmark Problem 14-A2 Delayed Neutron Parameters | 90 |

Table A.12 Benchmark Problem 14-A2 Initial Two-Group Data . . . . . 91

# Chapter One

# Introduction

Multigroup, multi-dimensional, nodal diffusion theory has been the primary computational model in space-dependent neutron kinetics for applications in transient reactor analysis and reactor safety.[1,2] However, the validity of diffusion theory based methods is questionable for small high leakage reactors such as space reactors, coolant voiding situations in power reactors, and transients requiring detailed modeling of control rod motion. Homogenization of absorber regions with nearby moderator materials must be done for diffusion theory to be valid. Alternatively, transport theory based kinetics methods[3,4,5,6] requiring less homogenization could be used to provide better accuracy, and to serve as a benchmarking tool for verification of more approximate methods[7-13] based on diffusion theory. Due to long computing times required to obtain accurate numerical solutions, kinetics methods based on three-dimensional (3-D) transport theory with explicit representation of delayed neutrons have not been practical in the past. With the advent of high performance workstations, it is now possible to perform neutron kinetics calculations based on three-dimensional transport theory.

## 1.1 Neutronics Models

The neutron dynamics of a nuclear system can be modeled by the time-dependent Boltzmann transport equation with explicit representation of delayed neutrons as shown below:[7]

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

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

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

$$+ \sum_i \lambda_i C_i(\bar{r},t)\,\chi_i + S(\bar{r},\bar{\Omega},E,t)$$

The corresponding time-dependent precursor equations are:

$$\frac{\partial C_i(\bar{r},t)}{\partial t} + \lambda_i C_i(\bar{r},t) = \iint\beta_i\nu\Sigma_f(\bar{r},E,t)\,\phi(\bar{r},\bar{\Omega},E,t)\,d\Omega dE \quad i=1,NDG \qquad (2)$$

where, v= neutron speed,

$\phi$= time-dependent angular flux,

$\bar{r}$= spatial variable,

E= energy,

$\bar{\Omega}$= unit vector describing particle direction,

t= time,

$\Sigma_t$= total cross section,

$\Sigma_s$= differential scattering cross section,

S= external source,

$\chi_p$= normalized energy spectrum of prompt neutrons,

$\chi_i$= normalized energy spectrum for precursor group i,

$\beta_i$= delayed neutron fraction for precursor group i,

$\nu$= total number of neutrons emitted per fission,

$\Sigma_f$= fission cross section,

$\lambda_i$= decay constant for delayed neutron precursor group i,

$C_i$= density of delayed neutron precursors for group i,

NDG= number of precursor groups.

An approximation to the time-dependent transport equation is the time-dependent diffusion equation:[16]

$$\begin{aligned}\frac{1}{v}\frac{\partial}{\partial t}\phi(\bar{r},E,t) + \Sigma_t\phi(\bar{r},E,t) &= \bar{\nabla}\cdot D(\bar{r},E,t)\bar{\nabla}\phi(\bar{r},E,t) \\ &+ \int\Sigma_s(\bar{r};E'\rightarrow E;t)\phi(\bar{r},E',t)\,dE' \\ &+ \int\chi_p(E)\,(1-\beta)\,\nu\Sigma_f(\bar{r},E',t)\phi(\bar{r},E',t)\,dE' \\ &+ \sum_i \lambda_i C_i(\bar{r},t)\chi_i + S(\bar{r},E,t)\end{aligned} \tag{3}$$

and the corresponding time-dependent precursor equations:

$$\frac{\partial C_i(\bar{r},t)}{\partial t} + \lambda_i C_i(\bar{r},t) = \int\beta_i\nu\Sigma_f(\bar{r},E,t)\phi(\bar{r},E,t)\,dE \qquad i=1,NDG \tag{4}$$

These latter equations are simpler and require less computer time to solve making diffusion theory more widely used in reactor analyses.[17] For any real system, approximation techniques such as modal, quasistatic and direct integration are employed to solve the equations. However, since diffusion theory is inappropriate for many systems (e.g., systems with voided regions where the assumption of a linearly anisotropic flux fails within or near the void), it is desirable to apply these techniques instead to the

more exact transport equations.[18]

## 1.2 Previous Work

The modal expansion method involves expansion of the solution in specified functions of one or more of the independent variables. This is accomplished through use of combining coefficients which are determined by requiring that the modal equations be satisfied in a weighted integral sense.[19,20] Codes developed to use this method to solve the kinetics equations based on the diffusion model include 1-D RAUMZEIT[21], 2-D RADYVAR[22], and 3-D SMOKIN.[23]

In the nodal method, the reactor volume is divided into subregions or nodes.[18,24-27] The flux in each node is then approximated as a polynomial and an average flux for each node is determined. Diffusion codes that use this technique are a 1-D code by Alcouffe and Albrecht[28], and 3-D codes NEM[29], CUBBOX[30], IQSBOX[31], QUANDRY[32] and CONQUEST.[27]

Solutions can also be found by direct numerical integration of the transport or diffusion equations.[33] However, direct integration techniques such as finite difference or finite elements become time consuming when used for multigroup 3-D problems. Codes that use the finite difference method to solve the time-dependent diffusion equation and precursor equations are 1-D WIGLE[34], 2-D TWIGL[35], and 3-D 3DKIN.[36] The 3-D

code CRONOS[37] uses finite elements. Other direct numerical integration codes that solve the time-dependent transport equation[38] are the 1-D codes TDA[39] and TIMEX[40], and the 2-D code TRANZIT.[41] TDA is a version of the 1-D, multigroup, discrete ordinates transport code ANISN with the time dependence handled by direct integration.[42] TIMEX is a 1-D, multigroup transport code which uses a semi-implicit scheme for the time differencing. It uses a linearly discontinuous finite element approach to treat the spatial variable and conventional discrete ordinates to treat the angular variable. TRANZIT is a multigroup, time-dependent, discrete ordinates code in cylindrical ($\rho$,z) geometry. The space-time variables are approximated by using a weighted diamond difference scheme. Time differencing can vary between Crank-Nicholson and completely implicit schemes. There are also a few time-dependent 3-D transport codes without delayed neutrons which have never been published.[46]

The improved quasi-static method refers to a flux synthesis method[7-11] in which the time-dependent flux is expressed as product of a space, energy, and angle dependent shape function which is slowly varying in time, and a purely time-dependent amplitude function. This factorization is made unique by imposing a normalization condition. After some manipulation of the original equations, equations for the shape function and the amplitude function are obtained. The shape equation is solved only when a new shape is needed. The much simpler amplitude function, however, is solved over very small time intervals yielding a very accurate temporal solution and requires much less computer time.[12,13,43,44] The codes that use the quasistatic approach to solve the diffusion theory

based kinetics equations are the 1-D code QX1[10], 2-D codes FX2[45] and TWODQD[46], and the 3-D CERBERUS.[47] Also, the diffusion codes QUANDRY and CONQUEST and the Monte Carlo code TDKENO[6] use a quasistatic approach to treat the time dependence.[48] A code which uses the unstructured tetrahedral-mesh $SP_n$ method also exists for time-dependent, 3-D neutron transport problems but without delayed neutrons.[49] Finally, a transient 3-D transport method with delayed neutrons is currently being developed in France.[50] This method uses even-parity approach and a Fick's Law type of approximation and, therefore, is not rigorous transport theory.

## 1.3 Objectives

The purpose of this work is to develop the methodology and corresponding code to solve the time-dependent, 3-D neutron transport equation with explicit representation of delayed neutrons by using a discrete ordinates approach for the shape, energy, and angular variables and the improved quasi-static approach for the time variable.

The new code system, named TDTORT, is verified by solving the transient Benchmark Problems 16-A1, -A2, -A3, -A6, -A7, 8-A1, 14-A1, 14-A2 and a TWIGL test problem. 16-A1, -A2, -A3, -A6 and -A7 are 1-D transport problems for which solutions exist from the 1-D transport codes TIMEX and TDA. Problem 8-A1 is a 2-D (r-z), two group diffusion theory problem. The TWIGL test problem is a 2-D (x-y) transient, diffusion theory problem. 14-A1 and 14-A2 are transient, 2-D and 3-D

diffusion benchmark problems with an adiabatic feedback model, respectively. All diffusion theory problems are converted to transport problems (described later in this document) for use by TDTORT. TDTORT is also compared with the analytical solution of a 1-D homogeneous slab reactor modeled by transient diffusion theory that has been converted to a transport problem.

## 1.4 Justification and Originality of Work

Although most space-time codes utilize diffusion theory, the diffusion model is not valid in or near voided or strongly absorbing regions. To treat these regions accurately, transport theory is needed. It is also necessary to use transport theory for small, high leakage systems such as space reactors. In addition, many transients caused by asymmetric rod movements should be modeled with 3-D transport theory for accuracy. Currently, there is no deterministic code that solves the time-dependent 3-D transport model with explicit representation of delayed neutrons for orthogonal geometries. Hence, the purpose of this work is to develop the methodology and corresponding code utilizing this model. Finally, the code developed in this work could also be used as a benchmark tool for other computer codes which are more approximate and less CPU intensive such as the transient, multidimensional diffusion codes mentioned earlier.

## 1.5 Thesis Organization

In Chapter 2 the theory of the improved quasistatic method using transport theory is presented. A description of the code developed in this work, TDTORT, is presented in Chapter 3. A user's manual for TDTORT is presented in Appendix B. Chapter 4 presents TDTORT results for the benchmark problems considered in this work. Finally, Chapter 5 presents a summary and conclusions of this research along with suggestions for future work.

# Chapter Two

# Theory

## 2.1 Improved Quasistatic Method

The derivation of the improved quasistatic method starts with the time-dependent, 3-D neutron transport equation with explicit representation of delayed neutrons and the corresponding precursor equations presented earlier. The time-dependent spatial flux is expressed as product of a purely time-dependent amplitude function and a space, energy, angle, shape function which is also weakly time-dependent as follows:

$$\phi(\bar{r},\bar{\Omega},E,t) = T(t)\,\psi(\bar{r},\bar{\Omega},E,t) \tag{5}$$

This representation is substituted into time-dependent transport equation and every term is then divided by T(t) to obtain an equation for the flux shape as follows:

$$\begin{aligned}
&\frac{1}{v}\cdot\frac{1}{T(t)}\cdot\frac{dT(t)}{dt}\cdot\psi(\bar{r},\bar{\Omega},E,t) + \frac{1}{v}\frac{\partial\psi(\bar{r},\bar{\Omega},E,t)}{\partial t} + \bar{\Omega}\cdot\bar{\nabla}\psi(\bar{r},\bar{\Omega},E,t) \\
&+ \Sigma_t(\bar{r},E,t)\,\psi(\bar{r},\bar{\Omega},E,t) = \iint\Sigma_s(\bar{r};\bar{\Omega}',E'\to\bar{\Omega},E;t)\,\psi(\bar{r},\bar{\Omega}',E',t)\,d\Omega'dE' \\
&+ \iint\chi_p(E)\,(1-\beta)\,\nu\Sigma_f(\bar{r},E',t)\,\psi(\bar{r},\bar{\Omega}',E',t)\,d\Omega'dE' \\
&+ \frac{S_d(\bar{r},E,t)}{T(t)} + \frac{S(\bar{r},E,t)}{T(t)}
\end{aligned} \tag{6}$$

where, $S_d(\bar{r},E,t)$ represents the space and energy dependent delayed neutron source at time t and is given by:

$$S_d(\bar{r}, E, t) = \int_{-\infty}^{t} \int_{E'} \int_{\bar{\Omega}'} \nu\Sigma_f(\bar{r}, E', t')\, T(t')\, \psi(\bar{r}, \bar{\Omega}', E', t') \times \sum_i \beta_i \chi_i(E)\, \lambda_i e^{-\lambda_i (t-t')}\, d\Omega' dE' dt' \tag{7}$$

Since Equation (5) does not determine the amount of time dependance in the shape function, this factorization is made unique[7] by applying the normalization condition:

$$\iiint \frac{1}{V} \psi(\bar{r}, \bar{\Omega}, E, t)\, \phi^*(\bar{r}, \bar{\Omega}, E)\, d\Omega\, dE dV = \gamma \tag{8}$$

where $\gamma$ must remain constant throughout the transient.

The adjoint transport equation is given by:

$$-\bar{\Omega}\cdot\bar{\nabla}\phi^* + \Sigma_t\phi^* = \iint \Sigma_s(\bar{r}; \bar{\Omega}, E \to \bar{\Omega}', E')\, \phi^*(\bar{r}, \bar{\Omega}', E')\, d\Omega' dE' + \iint \chi(E')\, \nu\Sigma_f(\bar{r}, E)\, \phi^*(\bar{r}, \bar{\Omega}', E')\, d\Omega' dE' \tag{9}$$

The forward equation is then multiplied by the adjoint flux and integrated over $(\bar{r}, E, \bar{\Omega})$ phase space. Likewise, the adjoint transport equation is multiplied by the forward flux and integrated over the same phase space. The difference between the two is taken, and after some manipulation and use of the normalization condition, the resulting expression for the amplitude is obtained:

$$\frac{dT(t)}{dt} = \frac{\rho(t) - \overline{\beta}(t)}{\Lambda(t)} T(t) + \sum_i \lambda_i C_i(t) + \overline{S}(t) \tag{10}$$

where

$$\begin{aligned}\rho(t) = & -\frac{1}{F(t)} \iiint (\Sigma_t(\overline{r}, E, t) - \Sigma_t(\overline{r}, E)) \psi(\overline{r}, \overline{\Omega}, E, t) \phi^*(\overline{r}, \overline{\Omega}, E)\, d\Omega\, dE dV \\ & + \frac{1}{F(t)} \iiint\!\!\iint \phi^*(\overline{r}, \overline{\Omega}, E) \left[ \left( \Sigma_s(\overline{r}; \overline{\Omega}', E' \to \overline{\Omega}, E; t) - \Sigma_s(\overline{r}; \overline{\Omega}', E' \to \overline{\Omega}, E) \right) \right. \\ & \left. + \chi_p(E) \left( \nu\Sigma_f(\overline{r}, E', t) - \nu\Sigma_f(\overline{r}, E') \right) \right] \psi(\overline{r}, \overline{\Omega}', E', t)\, d\Omega' dE' d\Omega\, dE dV\end{aligned} \tag{11}$$

$$\begin{aligned}\overline{\beta}_i(t) = & \frac{1}{F(t)} \iiint\!\!\iint \chi(E)\, \beta_i \nu\Sigma_f(\overline{r}, E', t)\, \psi(\overline{r}, \overline{\Omega}', E', t) \\ & \times \phi_0^*(\overline{r}, \overline{\Omega}, E)\, d\Omega' dE' d\Omega\, dE dV\end{aligned} \tag{12}$$

$$\overline{\beta}(t) = \sum_i \overline{\beta}_i(t) \tag{13}$$

$$\Lambda(t) = \frac{1}{F(t)} \iiint \frac{1}{V} \psi(\overline{r}, \overline{\Omega}, E, t) \phi^*(\overline{r}, \overline{\Omega}, E)\, d\Omega\, dE dV \tag{14}$$

$$C_i(t) = \frac{(1/F(t))}{\Lambda} \iiint \phi^*(\bar{r}, \bar{\Omega}, E)\, C_i(\bar{r}, t)\, \chi_i(E)\, d\Omega\, dE dV \qquad (15)$$

$$\bar{S}(t) = \frac{(1/F(t))}{\Lambda} \iiint \phi^*(\bar{r}, \bar{\Omega}, E)\, Q(\bar{r}, \bar{\Omega}, E)\, d\Omega\, dE dV \qquad (16)$$

and

$$F(t) = \iiiint \chi(E)\, \nu\Sigma_f(\bar{r}, E', t)\, \psi(\bar{r}, \bar{\Omega}', E', t)\, \phi_0^*(\bar{r}, \bar{\Omega}, E)\, d\Omega' dE' d\Omega\, dE dV \qquad (17)$$

where,

$\Sigma_t(\bar{r},E)$ = macroscopic total cross section at time zero.

$\Sigma_f(\bar{r},E)$ = macroscopic fission cross section at time zero.

$\Sigma_s(\bar{r},\bar{\Omega},E' \rightarrow \bar{\Omega},E)$ = macroscopic scattering cross section at time zero.

Usually the shape function varies more slowly in time than the amplitude function allowing larger time steps to be used for the shape calculations than for the amplitude calculations. The use of different time steps can reduce the time required to model a transient considerably.

The shape function is used to calculate the point kinetics parameters (i.e., reactivity, effective delayed neutron fractions, and generation time) using their inner

product definitions. These parameters are then used in the amplitude and precursor equations. The shape equation is solved only when necessary in order to maintain acceptable accuracy. This method is known as the improved quasistatic method (IQS). Another IQS approach is to solve the transport equation directly for the flux and use the normalization to obtain the shape. In the latter approach the amplitude function need not be known before the shape is calculated which is more efficient than the first approach in which the amplitude must be guessed.

If the shape is not changing or changing very slowly, the time derivative of the shape in Equation (6) can be neglected (will be nearly equal to zero). This approach is called quasistatic method (QS). If the amplitude is not changing rapidly as well, the term involving the amplitude time derivative in Equation (6) will also be approximately zero. The resulting approach is called adiabatic method. Further, if $S_d$ is ignored, the resulting approach is called the adiabatic method with precomputed shape functions. Note that if the external source, S, is zero, this latter approach is actually an eigenvalue calculation rather than a fixed source calculation. Finally, if the time behavior of the system is calculated by using only the initial flux shape, the method is called the point kinetics method. This is the least accurate but fastest of all four approaches.

## 2.2 Numerical Approximations

TDTORT obtains the solution of Equation (6) for the shape at a specific point in

time using TORT[14,51], a static, 3-D, multigroup discrete ordinates code. The shape time derivative term in Equation (6) is approximated by a backward difference which converts the time-dependent equation into a static equation as shown below:

$$\begin{aligned}
&\frac{1}{v}\cdot\frac{1}{T(t_{n+1})}\cdot\frac{dT(t_{n+1})}{dt_{n+1}}\cdot\psi(\bar{r},\bar{\Omega},E,t_{n+1}) + \Sigma_t(\bar{r},E,t_{n+1})\,\psi(\bar{r},\bar{\Omega},E,t_{n+1}) \\
&+ \frac{1}{v(t_{n+1}-t_n)}\left(\psi(\bar{r},\bar{\Omega},E,t_{n+1}) - \psi(\bar{r},\bar{\Omega},E,t_n)\right) + \bar{\Omega}\cdot\bar{\nabla}\psi(\bar{r},\bar{\Omega},E,t_{n+1}) \\
&= \iint \Sigma_s(\bar{r};\bar{\Omega}',E'\rightarrow\bar{\Omega},E;t_{n+1})\,\psi(\bar{r},\bar{\Omega}',E',t_{n+1})\,d\Omega'dE' \\
&+ \iint \chi_p(E)\,(1-\beta)\,\nu\Sigma_f(\bar{r},E',t_{n+1})\,\psi(\bar{r},\bar{\Omega}',E',t_{n+1})\,d\Omega'dE' \\
&+ \frac{S_d(\bar{r},E,t_{n+1})}{T(t_{n+1})} + \frac{S(\bar{r},E,t_{n+1})}{T(t_{n+1})}
\end{aligned} \tag{18}$$

where,

$$\begin{aligned}
S_d(\bar{r},E,t_{n+1}) = &\int_{-\infty}^{t_{n+1}} \int_{E'} \int_{\bar{\Omega}'} \nu\Sigma_f(\bar{r},E',t')\,T(t')\,\psi(\bar{r},\bar{\Omega}',E',t') \\
&\times \sum_i \beta_i \chi_i(E)\,\lambda_i e^{-\lambda_i(t-t')}\,d\Omega'dE'dt'
\end{aligned} \tag{19}$$

is the delayed neutron source. After rearrangement, Equation (18) becomes:

$$\left(\frac{1}{v}\cdot\frac{1}{T(t_{n+1})}\cdot\frac{dT(t_{n+1})}{dt_{n+1}} + \Sigma_t(\bar{r},E,t_{n+1}) + \frac{1}{v(t_{n+1}-t_n)}\right)\psi(\bar{r},\bar{\Omega},E,t_{n+1})$$

$$+ \bar{\Omega}\cdot\bar{\nabla}\psi(\bar{r},\bar{\Omega},E,t_{n+1})$$

$$= \iint\Sigma_s(\bar{r};\bar{\Omega}',E'\rightarrow\bar{\Omega},E;t_{n+1})\,\psi(\bar{r},\bar{\Omega}',E',t_{n+1})\,d\Omega'dE' \qquad (20)$$

$$+ \iint\chi_p(E)\,(1-\beta)\,\nu\Sigma_f(\bar{r},E',t_{n+1})\,\psi(\bar{r},\bar{\Omega}',E',t_{n+1})\,d\Omega'dE'$$

$$+ \frac{S_d(\bar{r},E,t_{n+1})}{T(t_{n+1})} + \frac{1}{v(t_{n+1}-t_n)}\psi(\bar{r},\bar{\Omega},E,t_n) + \frac{S(\bar{r},E,t_{n+1})}{T(t_{n+1})}$$

This is the steady-state transport equation with the total cross section adjusted by the term:

$$\frac{1}{v}\cdot\frac{1}{T(t_{n+1})}\cdot\frac{\partial T(t_{n+1})}{\partial t_{n+1}} + \frac{1}{v(t_{n+1}-t_n)} \qquad (21)$$

where the subscript n+1 represents the time at which the shape calculation is performed. The fixed source is

$$\frac{S_d(\bar{r},E,t_{n+1})}{T(t_{n+1})} + \frac{1}{v(t_{n+1}-t_n)}\psi(\bar{r},\bar{\Omega},E,t_n) + \frac{S(\bar{r},E,t_{n+1})}{T(t_{n+1})} \qquad (22)$$

The time-step hierarchy of the IQS method is shown in Figure 1. At first the flux shape is extrapolated out to the end of the shape calculation time step $\Delta t_n$. The shape function is uniquely determined by applying the normalization condition to the shape calculated above. The point kinetics parameters are calculated using $\Delta t_j$ time intervals with linearly

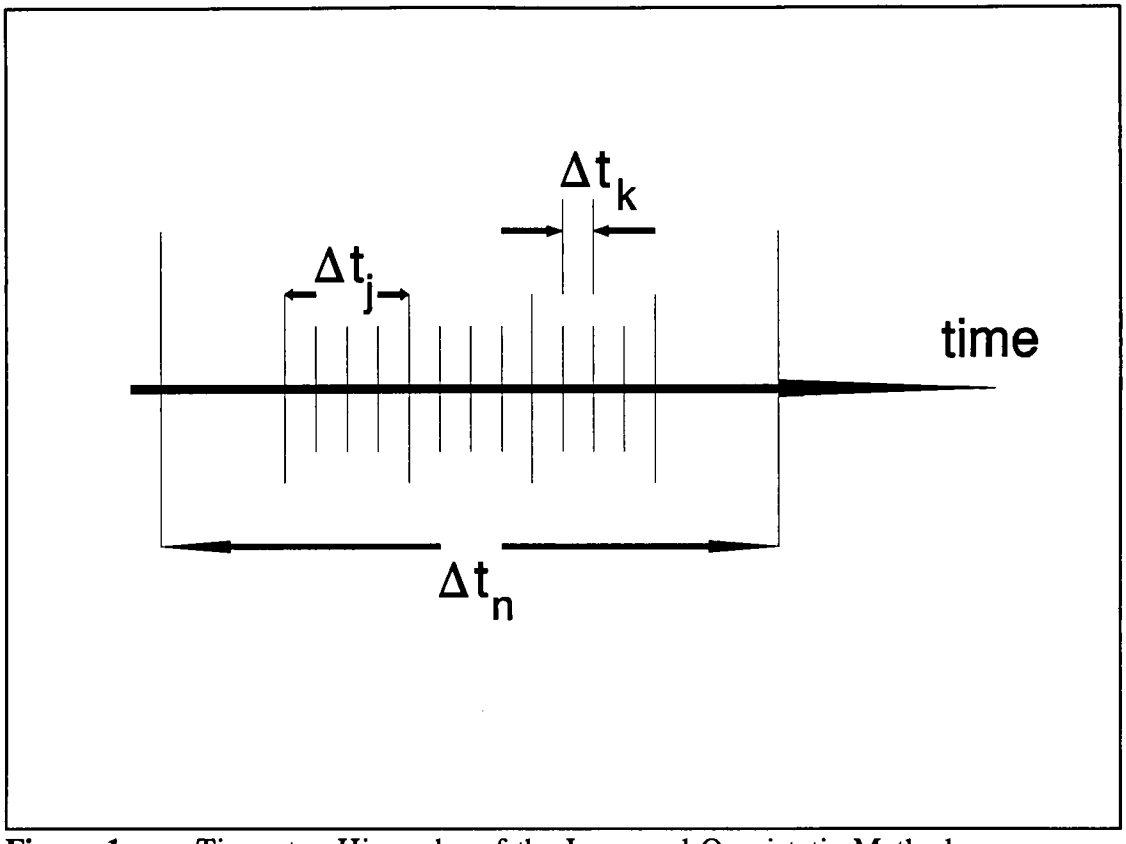



**Figure 1** Time-step Hierarchy of the Improved Quasistatic Method.

interpolated flux shapes. The point kinetics equations are solved using the differential equation solver, LSODE[52], over the smallest time interval $\Delta t_k$ using the quadratically interpolated point kinetics parameters. The delayed neutron source in Equation (19) is calculated by using the trapezoidal rule over the point kinetics time points.[53] As can be seen from Equation (19), the delayed neutron source is a function of the shape and amplitude values up to the end of current shape calculation time step. The linearly interpolated shape function values are used with newly computed amplitude values to calculate an approximate delayed neutron source, $S_d(r,E,t_{n+1})$. With the new delayed neutron source and newly calculated amplitude values, a new shape is calculated from Equation (20) using TORT. The new shape is then normalized by using Equation (7).

Using this new normalized shape, the above process is repeated over the same shape calculation time interval to update the amplitude and delayed neutron source. The entire process is repeated for all shape time intervals until the problem cut-off time is reached.

# Chapter Three

# Time-Dependent TORT Code (TDTORT)

## 3.1 Introduction

TDTORT is a new code developed at the University of Tennessee as part of this work. It utilizes the 3-D Discrete Ordinates code TORT to determine the flux shape throughout a transient. This shape is used to calculate the amplitude function using smaller time steps after which the amplitude and shape are recombined to yield the time-dependent spatial flux throughout the transient.

TDTORT is written in FORTRAN 77 and consists of 107 subroutines in addition to the TORT subroutines. A simple flow chart of TDTORT is presented in Figure 2. The DRIVER routine reads input and initializes all variables by calling the subroutines READES and INICAL. It also calls the subroutine STEADY to calculate the steady-state adjoint and forward flux shapes. It then calls the appropriate subroutine to perform the transient analysis according to the type of perturbation specified by user. The name of this family of subroutines is JOBWx where x is a one digit number representing the perturbation type as described later in the users manual. The JOBWx subroutine calls PKVAR to calculate point kinetics parameters, LSODE to calculate the amplitude, POWCAL to calculate power, DELSOR to calculate the delayed neutron source, and RXTEMP to calculate the temperature distribution if required. These calculated quantities are then fed back to JOBWx as required by the IQS methodology described in

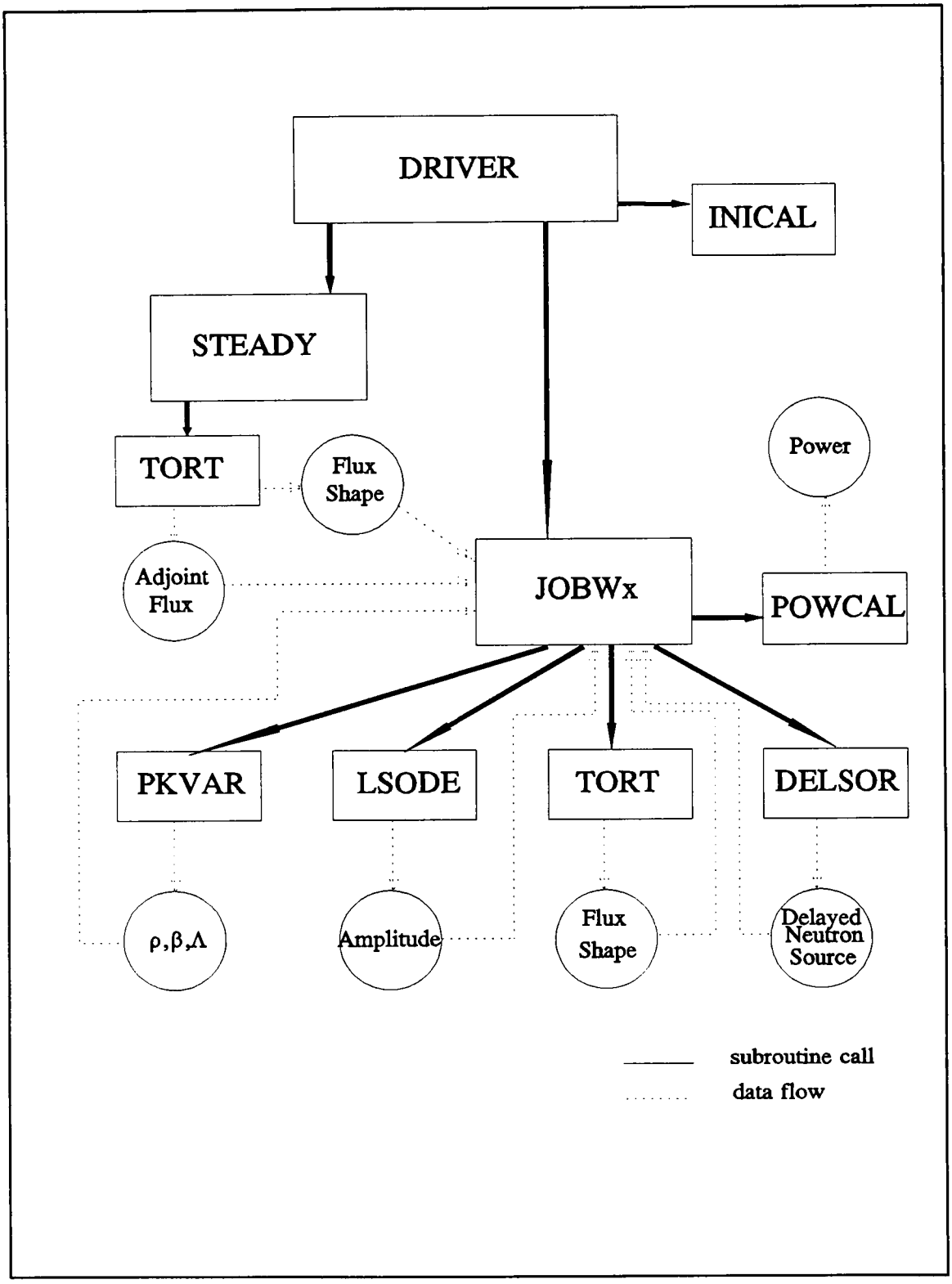



**Figure 2** Simplified Flow Chart for TDTORT

the previous chapter.

A users manual for TDTORT is provided in Appendix B which also includes a sample problem input and corresponding output.

## 3.2 Perturbations and I/O Units

TDTORT can be used to compute the transient behavior of neutronic systems which can be modeled by the discrete ordinates code TORT. Step, ramp, and certain combinations of the two perturbations are currently allowed. The transient can be initiated by a step perturbation at time zero or a ramp perturbation starting at time zero. The currently allowed perturbation types are:

- A single step perturbation at time zero
- A step perturbation at time zero followed by another step at a later time
- A step perturbation at time zero followed by a ramp starting at time zero
- A ramp perturbation starting at time zero
- A special case with thermal feedback.

While performing all these calculations, TDTORT uses several units to perform data I/O. It also uses some default units to interact with TORT. The unit numbers 9, 10, 11, 18, 25, 26, 27, 28, 31, and 35 should not be used in TORT input to specify a scratch file unit, source, flux guess input, or anything else because these units are used

by TDTORT.

# Chapter Four

# Results

## 4.1 General Remarks

All TDTORT calculations presented herein were performed on a SUN Ultra 1 workstation with a 167 MHz CPU and 192 MBytes of RAM. Also, the fully symmetric quadrature set supplied by TORT is used for all problems. All of the problems considered thus far have isotropic scattering. Problems 16-A1 thru -A7 are for a 1-D transport model of a liquid metal fast breeder reactor. It should be noted that the TDA results for problems 16-A6 and 16-A7 were not published in the Benchmark book.[15] Problem 8-A1 is a 2-D (r-z) diffusion model of a heavy water reactor. The TWIGL test problem is a 2-D (x-y) diffusion model of a pressurized water reactor. Problems 14-A1 and 14-A2 are diffusion problems for a boiling water reactor. All diffusion theory based benchmark problems are converted to transport theory problems using the following relationship between the diffusion coefficient and the total scattering cross section;

$$D=\frac{1}{3\Sigma_s(1-\overline{\cos\psi})} \qquad (23)$$

where,

D = diffusion coefficient,

$\Sigma_s$ = scattering cross section,

$\psi$ = scattering angle in the laboratory system.

If scattering is isotropic the above equation reduces to:

$$D=\frac{1}{3\Sigma_s} \tag{24}$$

which is the case for the diffusion problems considered in this work. Using this scattering cross section along with the specified values of the downscattering and absorption cross sections, the within group scattering cross section and the total cross section are calculated. For all the transients presented herein, the $\gamma$ normalization in Equation (8) is constant to within 1% throughout the transients except for problems 14-A1 and 14-A2 which are constant within 5%. Furthermore, the shape calculation time intervals are selected such that two iterations over the shape time interval are sufficient, and increasing the number of iterations beyond two does not result in significant improvement in accuracy. In addition, since there is no pre-defined relationship between the time-step size and a particular transient problem, a time-step size selection algorithm has not been developed in this work. It is recommended that a new TDTORT user should model one or more of the benchmark problems in this dissertation in order to gain experience selecting time steps.

## 4.2 Analytical 1-D Test Problem

Results have been obtained using TDTORT for a simple, homogeneous, one-group, transient diffusion theory problem with one precursor group. The problem

involves a spatially uniform perturbation made in the fission cross section of a bare homogeneous slab reactor. This problem can, of course, be solved analytically.[16] The one group cross sections for this problem are given in Table I. The shape calculations are performed at 1e-4, 1e-2, and 1 seconds using an $S_2$ quadrature set. Each shape interval is divided into 10 reactivity time intervals, and each reactivity time interval is divided into 10 point kinetics time intervals. The point kinetics parameters evaluated at each shape calculation time are given in Table II. The power versus time results for this calculation are in excellent agreement with the analytical results as shown in Figure 3.

**Table I** One Group Cross Sections for 1-D Analytical Test Problem

| $\Sigma_a$ | $\nu\Sigma_f$ | $\Sigma_t$ | $\Sigma_s^{1\to1}$ |
|---|---|---|---|
| .153 | .155489 | .486 | .333 |

**Table II** Point Kinetics Parameters for 1-D Analytical Test Problem

| Time (sec.) | $\rho$ | $\beta$ | $\Lambda$ |
|---|---|---|---|
| 0 | 1.503E-3 | 7.5E-3 | 6.058E-5 |
| 1E-4 | 1.503E-3 | 7.5E-3 | 6.058E-5 |
| 1E-2 | 1.503E-3 | 7.5E-3 | 6.058E-5 |
| 1E+0 | 1.503E-3 | 7.5E-3 | 6.058E-5 |

## 4.3 Benchmark Problem 16-A1

Benchmark 16-A1 is a 1-D transport theory problem for a liquid metal fast

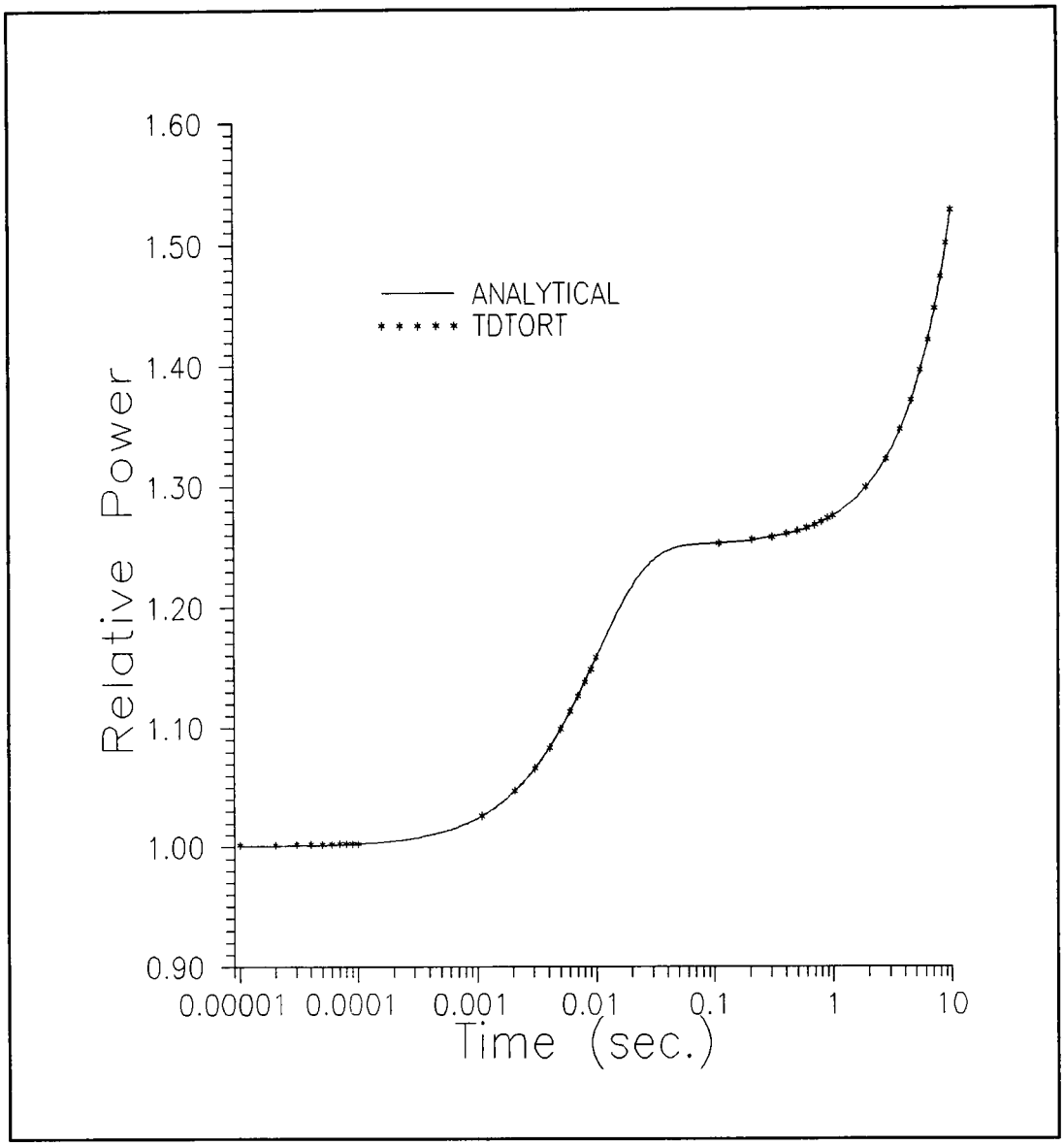



**Figure 3** Relative Power vs. Time Computed by TDTORT for the Analytical one-group Problem.

breeder reactor. The geometry is shown in Figure 4 and a detailed description of this problem is given in Appendix A. The system is perturbed by increasing the density of the material in zone 2 by 5% and decreasing the density of the material in zone 6 by 5% resulting in a step perturbation at time zero. The shape calculations are performed at 1e-5, 1e-4, 1e-3, .1, 5. and 10. seconds using an $S_4$ quadrature set. The pointwise flux convergence, fission convergence and eigenvalue convergence criteria for this problem are 1e-5, 1e-5, and 1e-5, respectively. The point kinetics parameters for this problem calculated by TDTORT at each shape calculation time are given in Table III. Each shape interval is divided into 1000 reactivity time intervals, and each reactivity time interval is divided into ten point kinetics time intervals. The relative power vs. time is given in Figure 5. Again, TDTORT shows excellent agreement with the benchmark results. The normalized flux vs. radial position at time zero and 0.01 second for groups one and two are given in Figures 6 and 7, respectively. The total CPU time for this ten second transient is 33 seconds.

**Table III** Point Kinetics Parameters for Benchmark Problem 16-A1

| Time (sec.) | Reactivity | Total Effective Beta | Generation Time |
|---|---|---|---|
| 0.000E+00 | 4.033E-07 | 3.200E-03 | 3.655E-07 |
| 1.000E-05 | 1.088E-03 | 3.200E-03 | 3.637E-07 |
| 1.000E-04 | 1.110E-03 | 3.200E-03 | 3.642E-07 |
| 9.999E-04 | 1.112E-03 | 3.200E-03 | 3.641E-07 |
| 1.000E-01 | 1.112E-03 | 3.200E-03 | 3.641E-07 |
| 5.000E+00 | 1.114E-03 | 3.200E-03 | 3.641E-07 |

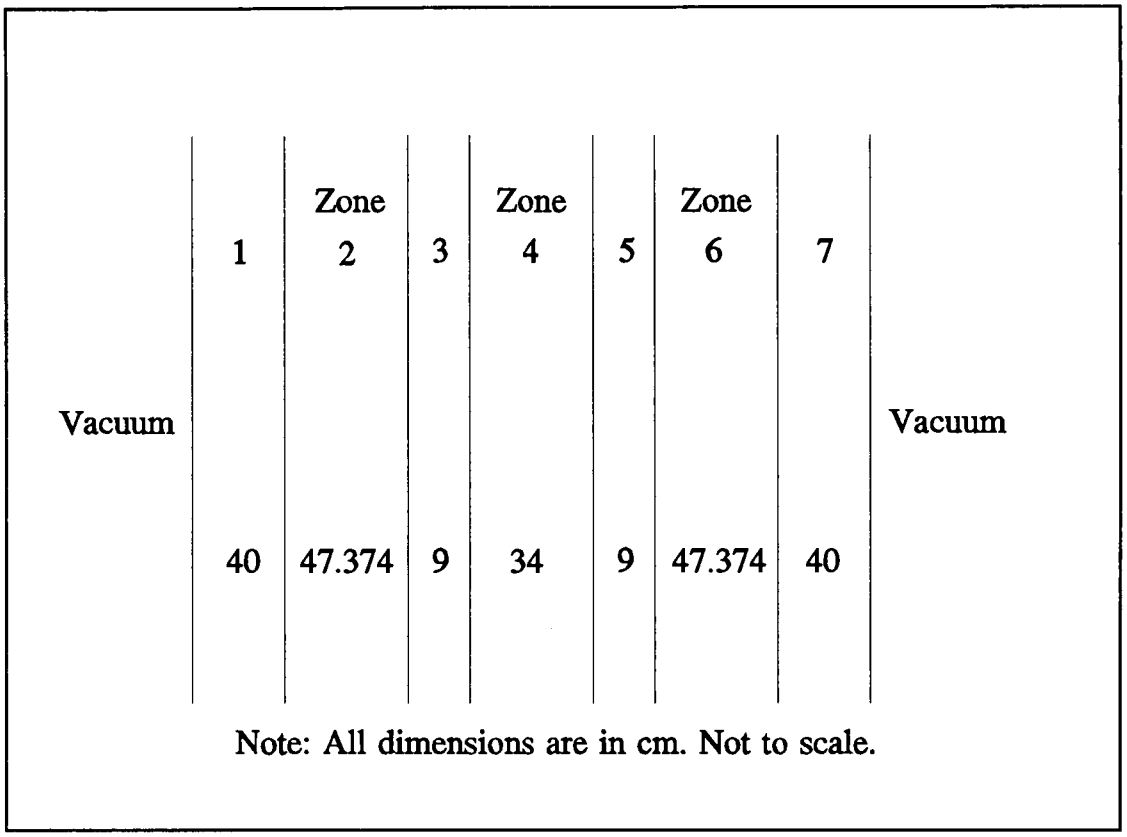



**Figure 4** Schematic Diagram of Benchmark Problem 16.

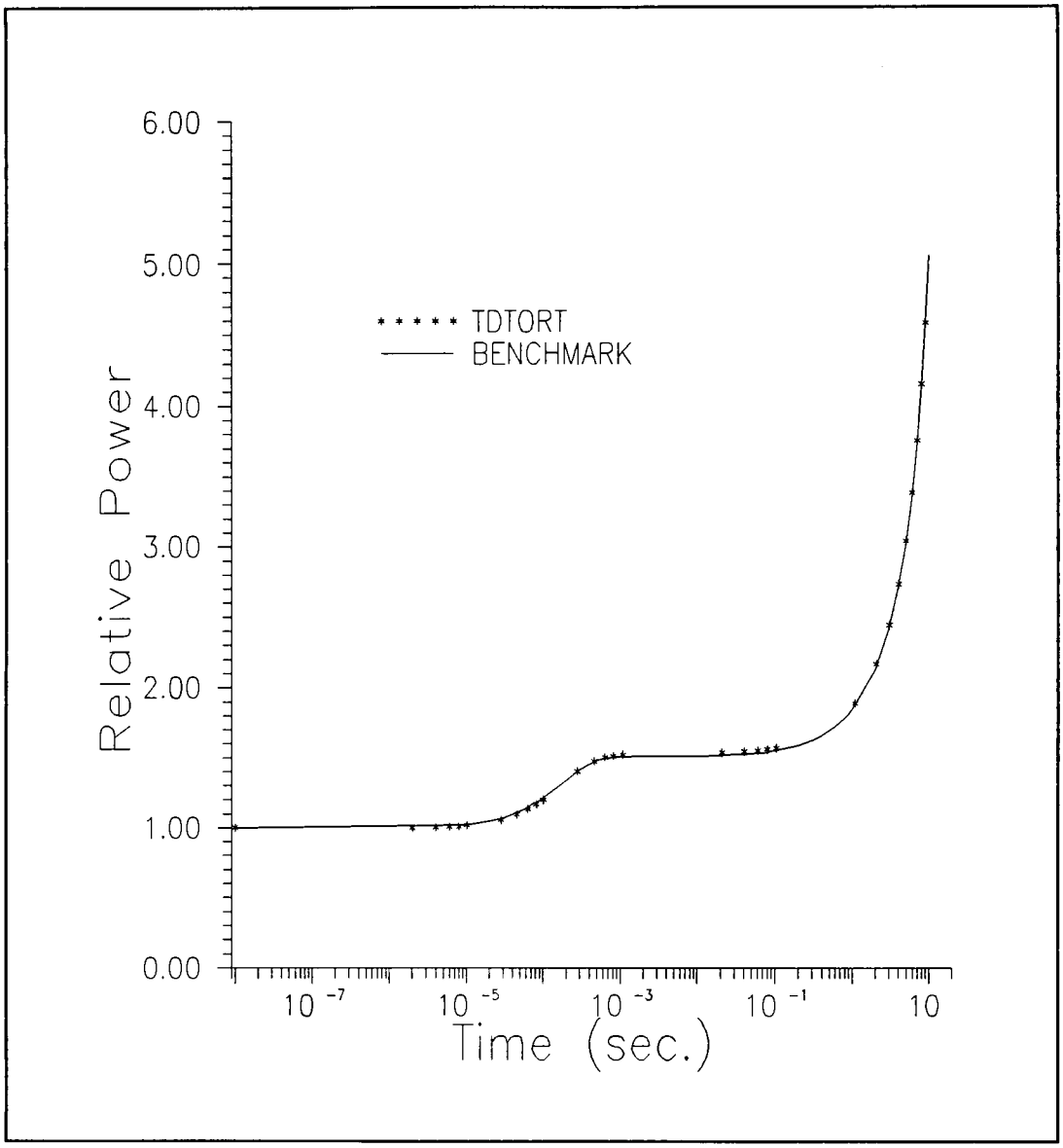



**Figure 5** Relative Power Comparisons of TDTORT and TIMEX(Benchmark) for Problem 16-A1.

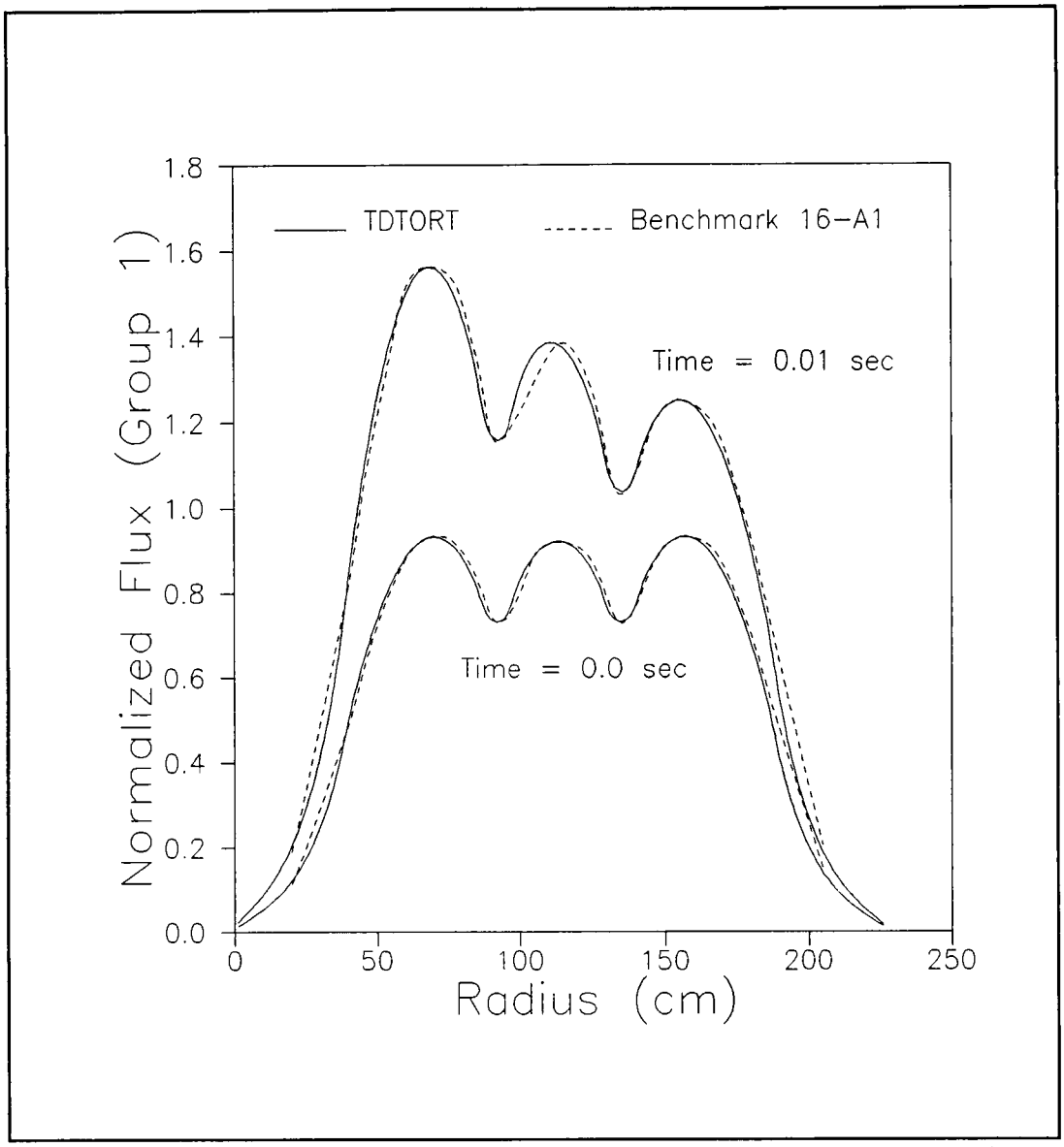



**Figure 6** Normalized Flux vs. Radial Position for Benchmark 16-A1 (Group 1).

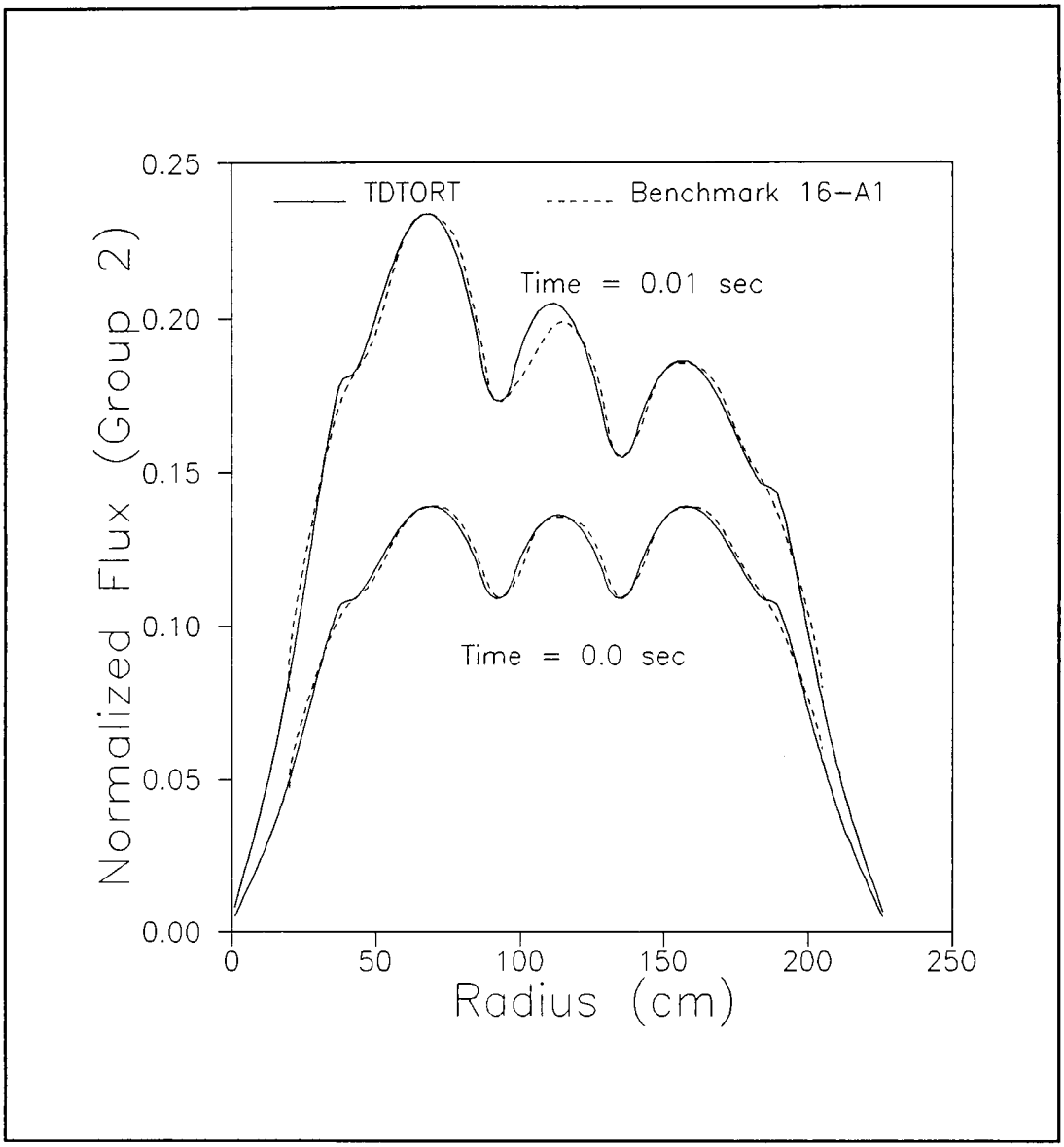



**Figure 7** Normalized Flux vs. Radial Position for Benchmark 16-A1 (Group 2).

## 4.4 Benchmark Problem 16-A2

This problem is the same as 16-A1 except for the initial perturbation of 10% rather than 5%. The shape calculations are performed at 1e-5, 1e-4, 5e-4, 1.1e-3, 2e-3, 2.5e-3 and 3.1e-3 seconds using an $S_4$ quadrature set. Each shape interval is divided into 1000 reactivity time intervals, and each reactivity time interval is divided into ten point kinetics time intervals. The pointwise flux convergence, fission convergence and eigenvalue convergence criteria for this problem are 1e-5, 1e-5, and 1e-5, respectively. The point kinetics parameters for this problem calculated by TDTORT at each shape calculation time are given in Table IV. The results show excellent agreement with the benchmark as shown in Figure 8. The CPU time for this problem is 10 seconds.

**Table IV** Point Kinetics Parameters for Benchmark Problem 16-A2

| Time (sec.) | Reactivity | Total Effective Beta | Generation Time |
|---|---|---|---|
| 0.000E+00 | 3.844E-07 | 3.200E-03 | 3.655E-07 |
| 1.000E-05 | 3.619E-03 | 3.200E-03 | 3.605E-07 |
| 1.000E-04 | 4.174E-03 | 3.200E-03 | 3.592E-07 |
| 4.999E-04 | 4.229E-03 | 3.200E-03 | 3.591E-07 |
| 1.100E-03 | 4.236E-03 | 3.200E-03 | 3.590E-07 |
| 2.000E-03 | 4.260E-03 | 3.200E-03 | 3.593E-07 |
| 2.500E-03 | 4.235E-03 | 3.200E-03 | 3.590E-07 |

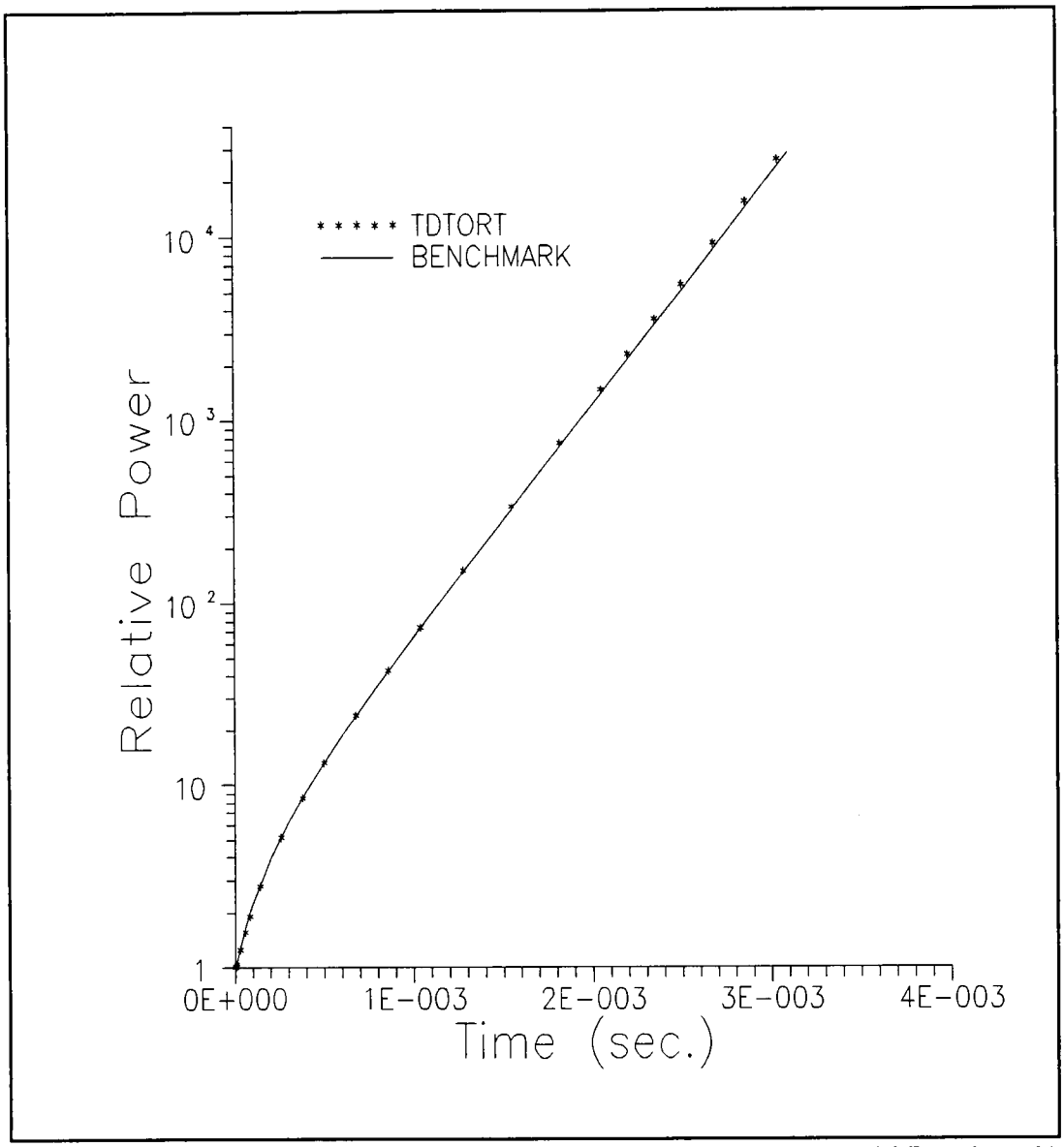



**Figure 8** Relative Power Comparisons of TDTORT and TIMEX(Benchmark) for Problem 16-A2.

## 4.5 Benchmark Problem 16-A3

This problem is also similar to 16-A1. In this problem, the system is first perturbed by changing the material in zone 5 at time zero from a mixture of sodium and control rod materials to 100% sodium (i.e., control rod ejection) resulting in positive reactivity insertion. Then, at time .0001 second 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), resulting in a negative reactivity insertion. The shape calculations are performed at 1e-7, 5e-7, 2e-6, 1e-5, 5e-5, 9e-5, 1e-4, 1.001e-4, 1.01e-4, 1.02e-4, 1.03e-4, 1.05e-4, 1.07e-4, 1.1e-4, 1.15e-4, 1.2e-4, 1.3e-4, 1.5e-4, 2e-4, 6e-4, 1e-3 and 1e-2 seconds using an $S_4$ quadrature set. One hundred reactivity calculations are performed between each shape calculation. Ten point kinetics calculations are performed between each reactivity calculation. The pointwise flux convergence, fission convergence and eigenvalue convergence criteria for this problem are 5e-5, 5e-5, and 5e-5, respectively. The point kinetics parameters at each shape calculation time are presented in Table V. TDTORT results match the benchmark results very well as shown by the relative power comparison in Figure 9. The CPU time for this problem is 32 seconds.

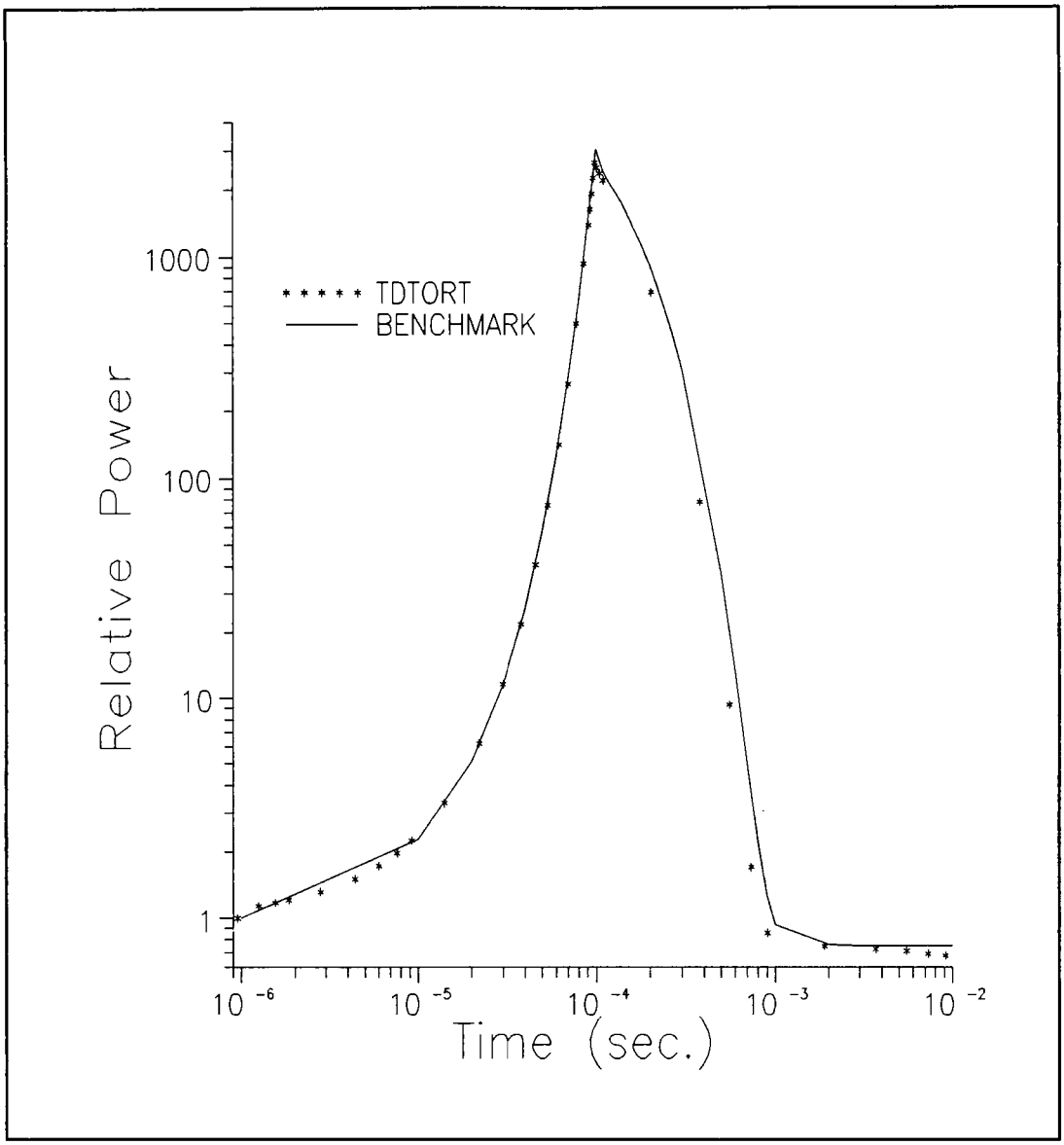



**Figure 9** Relative Power Comparisons of TDTORT and TIMEX(Benchmark) for Problem 16-A3.

**Table V** Point Kinetics Parameters for Benchmark Problem 16-A3

| Time (sec.) | Reactivity | Total Effective Beta | Generation Time |
|---|---|---|---|
| 0.000E+00 | 2.690E-02 | 3.200E-03 | 3.797E-07 |
| 1.000E-07 | 2.808E-02 | 3.200E-03 | 3.785E-07 |
| 5.000E-07 | 2.963E-02 | 3.200E-03 | 3.738E-07 |
| 2.000E-06 | 3.050E-02 | 3.200E-03 | 3.633E-07 |
| 1.000E-05 | 3.047E-02 | 3.200E-03 | 3.541E-07 |
| 5.000E-05 | 3.150E-02 | 3.200E-03 | 3.611E-07 |
| 9.000E-05 | 3.208E-02 | 3.200E-03 | 3.651E-07 |
| 1.000E-04 | -5.036E-02 | 3.200E-03 | 3.583E-07 |
| 1.001E-04 | -3.566E-02 | 3.200E-03 | 3.569E-07 |
| 1.010E-04 | -3.208E-03 | 3.200E-03 | 3.816E-07 |
| 1.020E-04 | -2.625E-03 | 3.200E-03 | 3.817E-07 |
| 1.030E-04 | -2.311E-03 | 3.200E-03 | 3.819E-07 |
| 1.050E-04 | -2.096E-03 | 3.200E-03 | 3.819E-07 |
| 1.070E-04 | -1.896E-03 | 3.200E-03 | 3.820E-07 |
| 1.100E-04 | -1.724E-03 | 3.200E-03 | 3.822E-07 |
| 1.150E-04 | -1.528E-03 | 3.200E-03 | 3.824E-07 |
| 1.000E-03 | -1.578E-03 | 3.200E-03 | 3.824E-07 |

## 4.6 Benchmark Problem 16-A6

This problem simulates material motion in a liquid metal fast breeder reactor. 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 as given in Table A.4. 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., material motion) resulting in final cross sections for each zone at t = 0.5 sec as shown in Table A.5. The shape updates are performed at 1e-6, 1e-4, 1e-3, 3e-3, 4e-3, 1e-2, .1, .5, .8 and 1. seconds using an $S_4$ quadrature set. The pointwise flux convergence, fission convergence and eigenvalue convergence criteria for this problem are 5e-5, 5e-5, and 5e-5, respectively. The point kinetics parameters calculated by TDTORT at each shape calculation time for this problem are given in Table VI. Each shape interval is divided into 100 reactivity time intervals, and each reactivity time interval is divided into ten point kinetics time intervals.The relative power calculated by TDTORT agrees well with the benchmark as shown in Figure 10. The CPU time for this problem is 168 seconds.

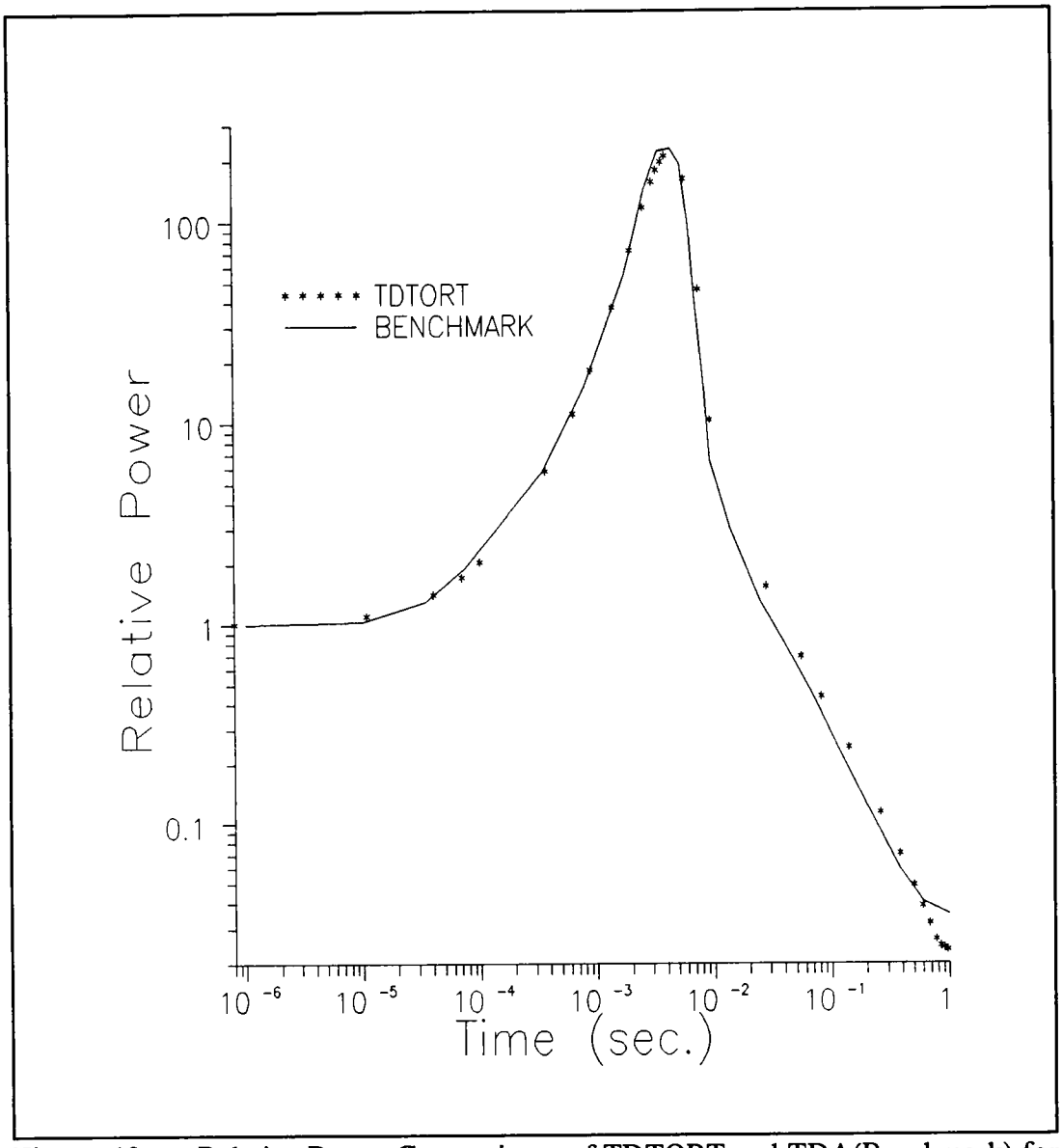



**Figure 10** Relative Power Comparisons of TDTORT and TDA(Benchmark) for Problem 16-A6.

**Table VI** Point Kinetics Parameters for Benchmark Problem 16-A6

| Time (sec.) | Reactivity | Total Effective Beta | Generation Time |
|---|---|---|---|
| 0.000E+00 | 3.642E-03 | 3.200E-03 | 3.810E-07 |
| 1.000E-06 | 3.702E-03 | 3.200E-03 | 3.813E-07 |
| 1.000E-04 | 3.649E-03 | 3.200E-03 | 3.819E-07 |
| 1.000E-03 | 3.385E-03 | 3.200E-03 | 3.824E-07 |
| 3.000E-03 | 3.241E-03 | 3.200E-03 | 3.825E-07 |
| 4.000E-03 | 2.442E-03 | 3.200E-03 | 3.835E-07 |
| 1.000E-02 | -1.016E-02 | 3.200E-03 | 4.010E-07 |
| 1.000E-01 | -8.182E-02 | 3.200E-03 | 5.009E-07 |
| 5.000E-01 | -1.487E-01 | 3.200E-03 | 5.949E-07 |
| 8.000E-01 | -1.409E-01 | 3.200E-03 | 5.848E-07 |

## 4.7 Benchmark Problem 16-A7

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 A.5. The shape calculations are performed at 5e-5, 1e-4, 1e-2, 1.1e-2, 2e-2 and .1 seconds using an $S_4$ quadrature set. The pointwise flux convergence, fission convergence and eigenvalue convergence criteria for this problem are 5e-5, 5e-5, and 5e-5, respectively. The point kinetics parameters for this problem are given in Table VII. Each shape interval is divided into 100 reactivity time intervals, and each reactivity time interval is divided into

10 point kinetics time intervals. The relative power calculated by TDTORT agrees well with the benchmark up to t = 0.01 second. After t = 0.01 seconds TDTORT results are higher than the benchmark as shown in Figure 11. The CPU time for this problem is 650 seconds.

**Table VII** Point Kinetics Parameters for Benchmark Problem 16-A7

| Time (sec.) | Reactivity | Total Effective Beta | Generation Time |
|---|---|---|---|
| 0.000E+00 | 3.809E-03 | 3.200E-03 | 3.809E-07 |
| 5.000E-05 | 3.946E-03 | 3.200E-03 | 3.818E-07 |
| 1.000E-04 | 3.951E-03 | 3.200E-03 | 3.818E-07 |
| 1.000E-03 | 3.952E-03 | 3.200E-03 | 3.817E-07 |
| 1.000E-02 | -1.331E-01 | 3.200E-03 | 6.275E-07 |
| 1.100E-02 | -1.102E-01 | 3.200E-03 | 5.852E-07 |
| 2.000E-02 | -1.102E-01 | 3.200E-03 | 5.853E-07 |
| 1.000E-01 | -1.101E-01 | 3.200E-03 | 5.853E-07 |

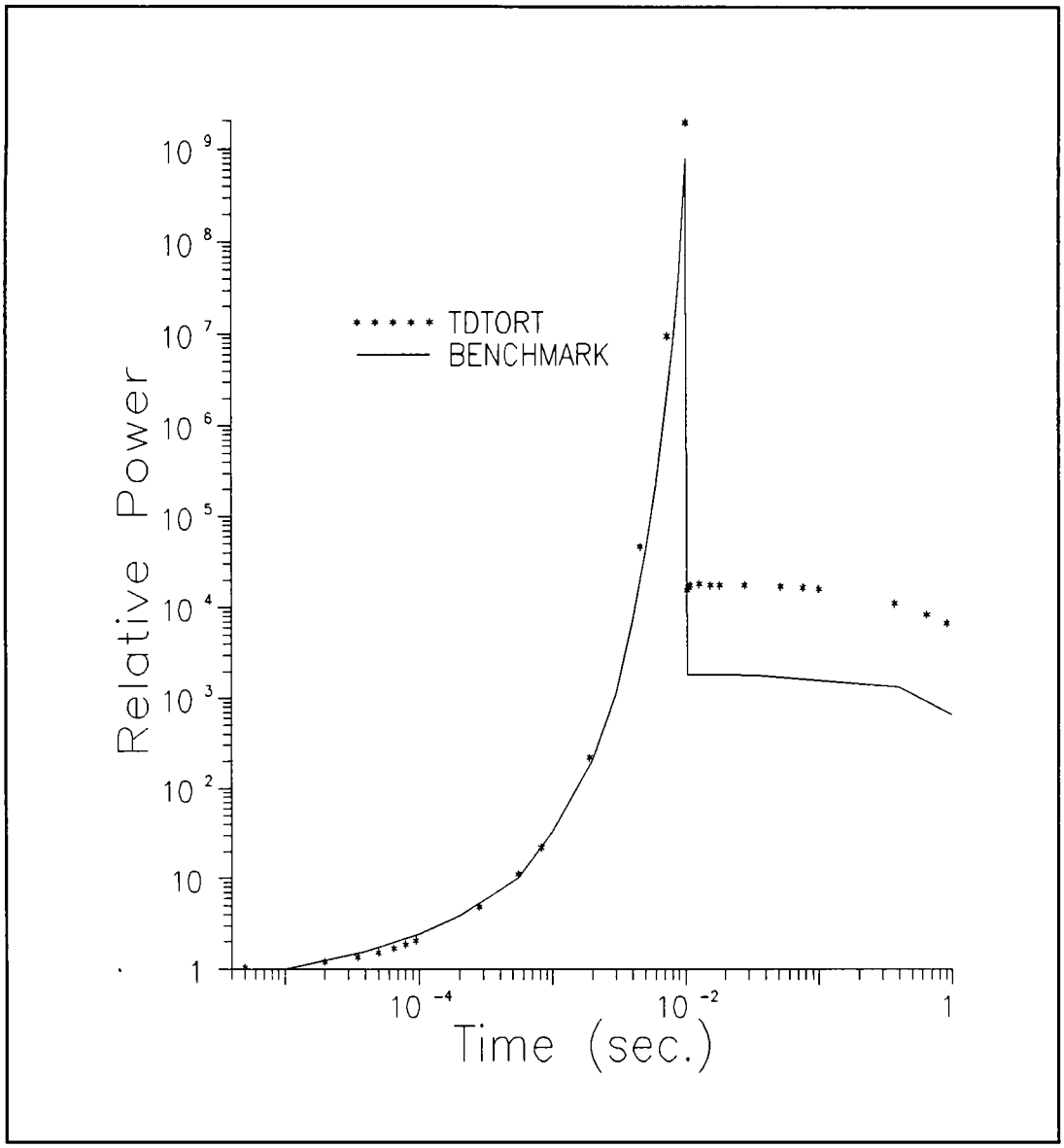



**Figure 11** Relative Power Comparisons of TDTORT and TDA(Benchmark) for Problem 16-A7.

## 4.8 Benchmark Problem 8-A1

Problem 8-A1 is a 2-D (r-z) delayed supercritical transient in a heavy water reactor with two neutron energy groups and six delayed neutron groups. Since this is a diffusion theory problem, the total cross section is calculated by using the relationship between the total scattering cross section and the diffusion coefficient presented previously. Detailed information about this problem is given in Appendix A. The shape calculations are performed at .2, .4, .6, .8, 1., 2., 3. and 4. seconds. Ten reactivity calculations are performed between each shape calculation. Ten point kinetics calculations are performed between each reactivity calculation. An $S_2$ angular quadrature set, 30 radial mesh points, and 28 axial mesh points are used. The pointwise flux convergence, fission convergence and eigenvalue convergence criteria for this problem are 5e-4, 5e-4, and 5e-5, respectively. The normalized flux distributions for group 1 and group 2 at time 0.0 and 4.0 seconds calculated by TWODQD[11] and TDTORT are given in Figures 12 thru 19, respectively. In evaluating these figures, it should be emphasized that TWODQD solves a diffusion theory problem whereas TDTORT solves the corresponding transport theory problem. The point kinetics parameters for this problem at each shape calculation time step are given in Table VIII. The relative power calculated by TDTORT agrees with the benchmark as shown in Figure 20. The maximum difference between relative powers is less than 6%. The CPU time for this problem is 960 seconds.

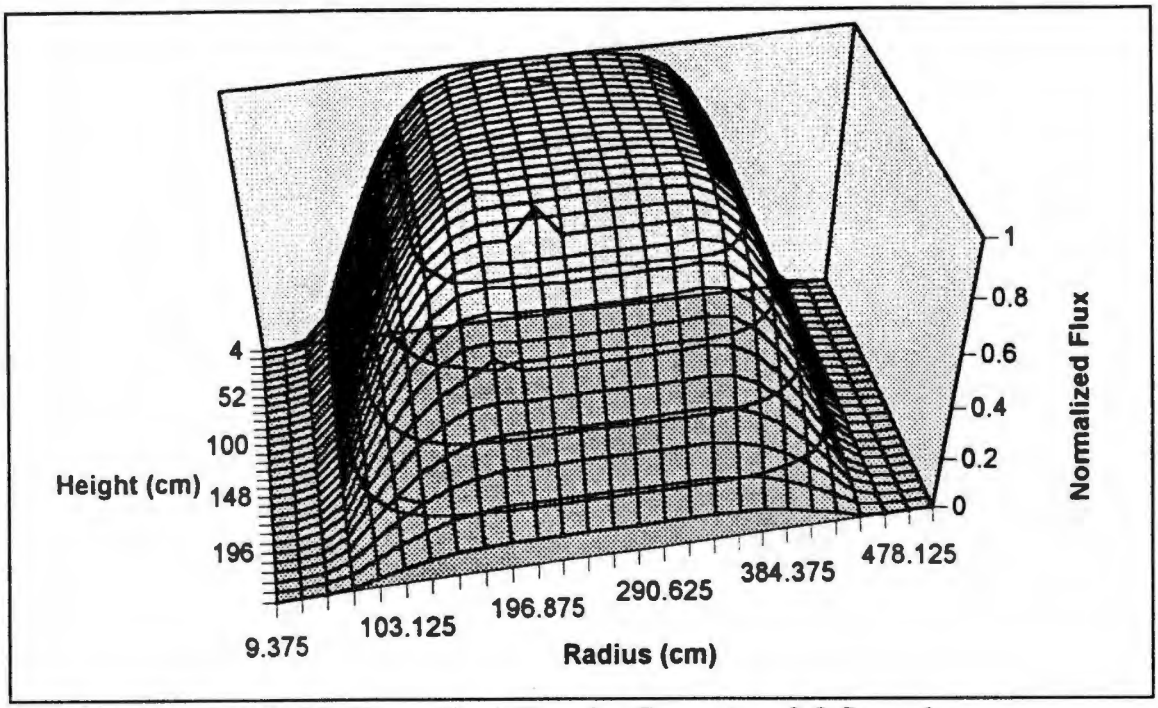



**Figure 12** TWODQD Normalized Flux for Group 1 at 0.0 Seconds.

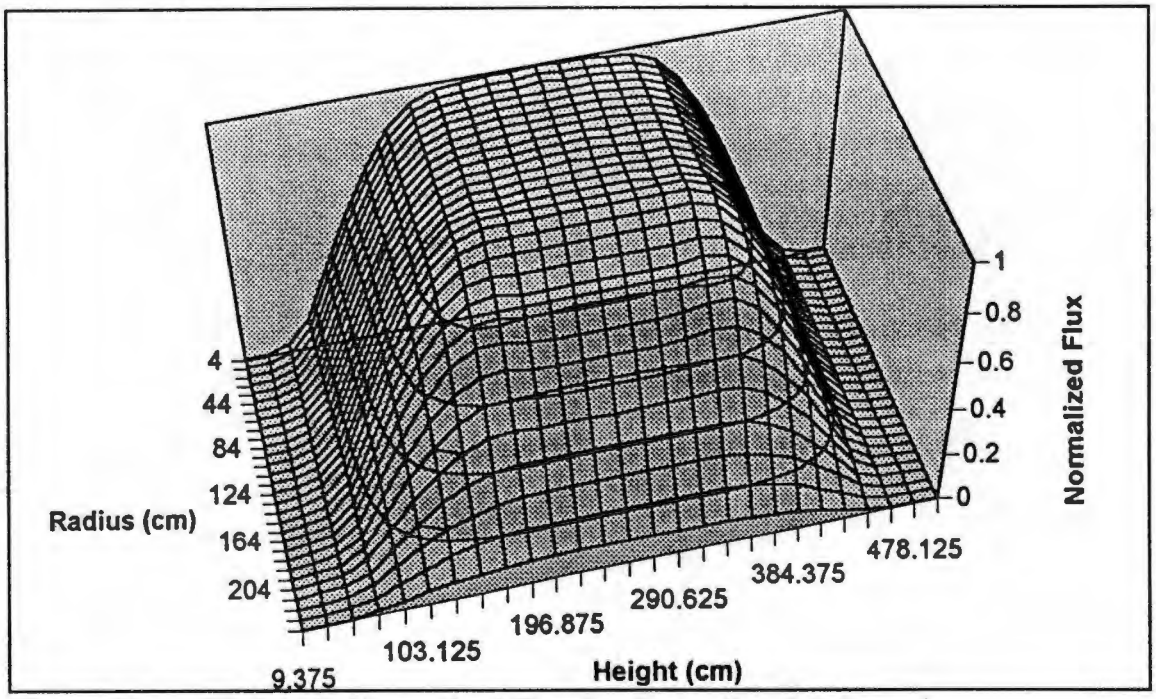



**Figure 13** TWODQD Normalized Flux for Group 2 at 0.0 Seconds.

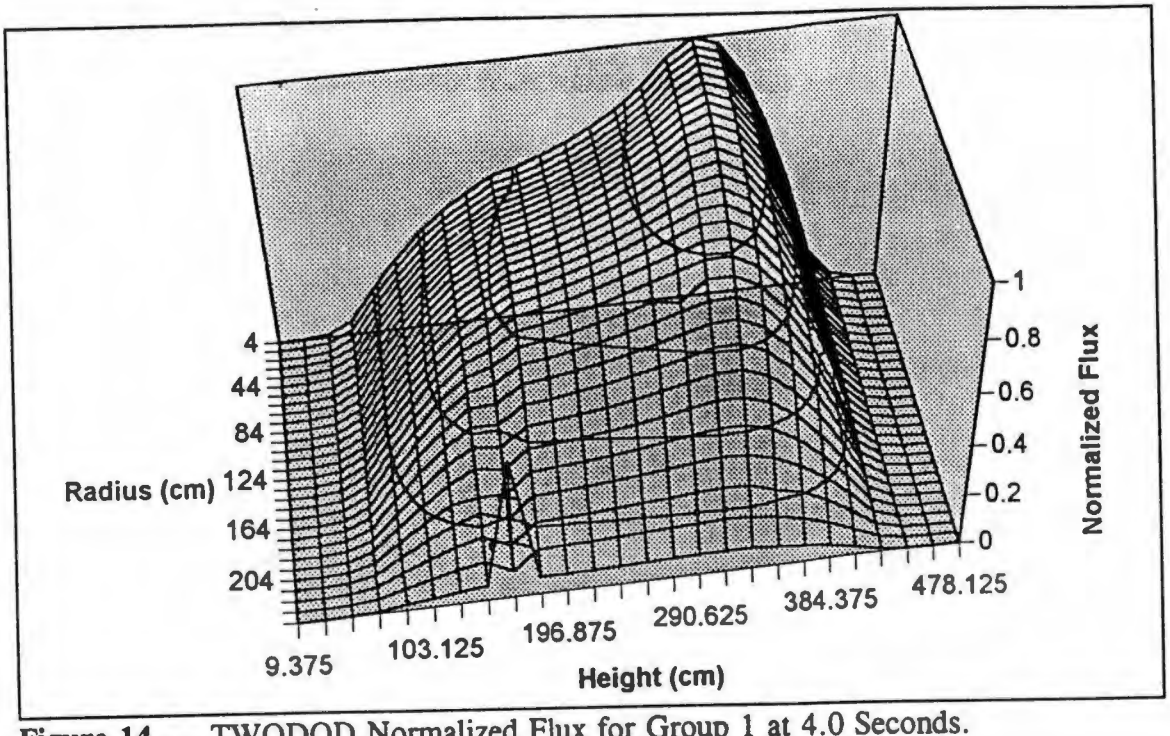



**Figure 14** TWODQD Normalized Flux for Group 1 at 4.0 Seconds.

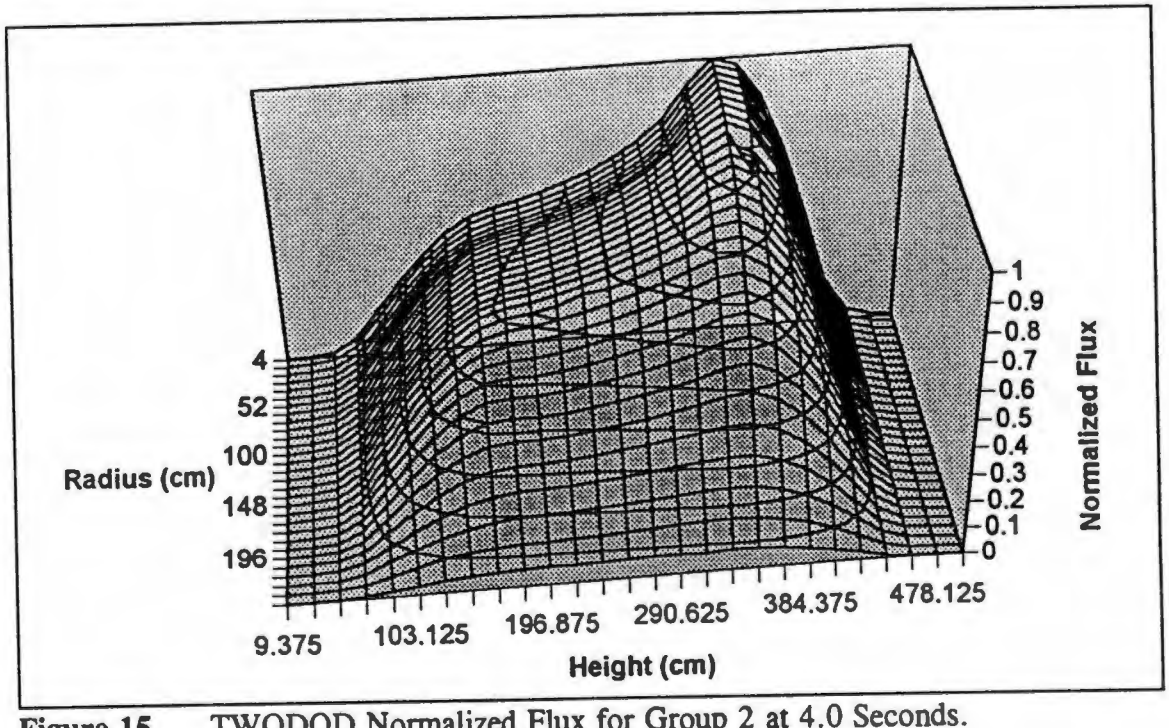



**Figure 15** TWODQD Normalized Flux for Group 2 at 4.0 Seconds.

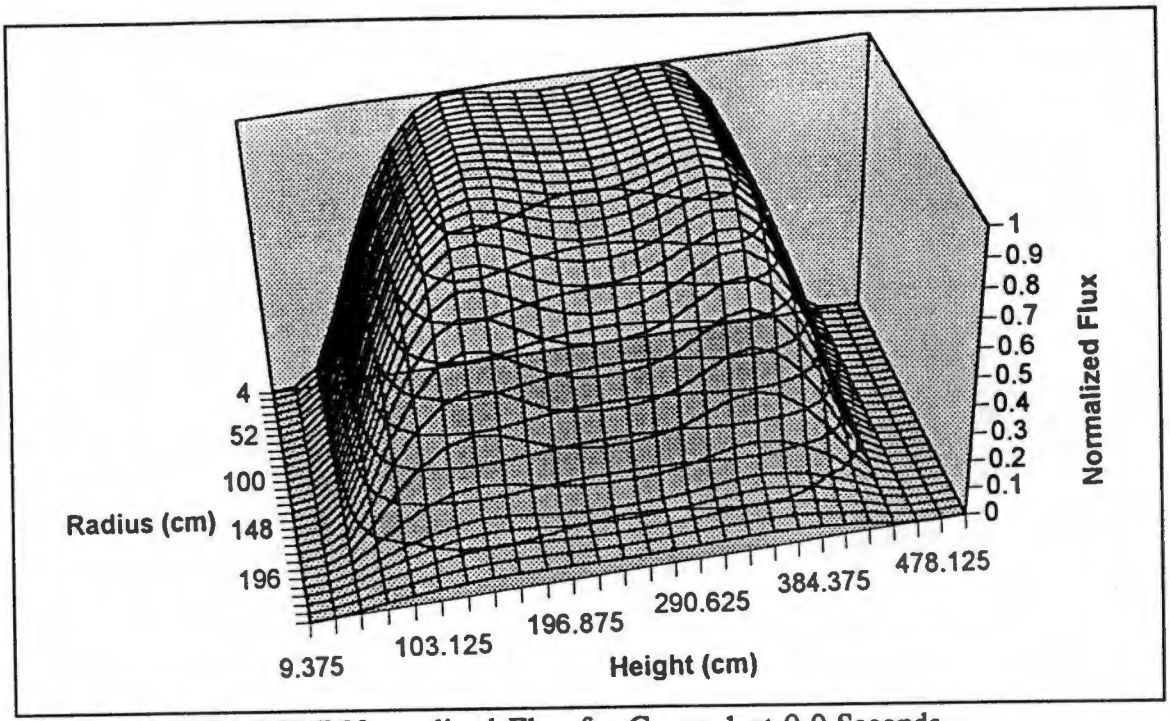



**Figure 16** TDTORT Normalized Flux for Group 1 at 0.0 Seconds.

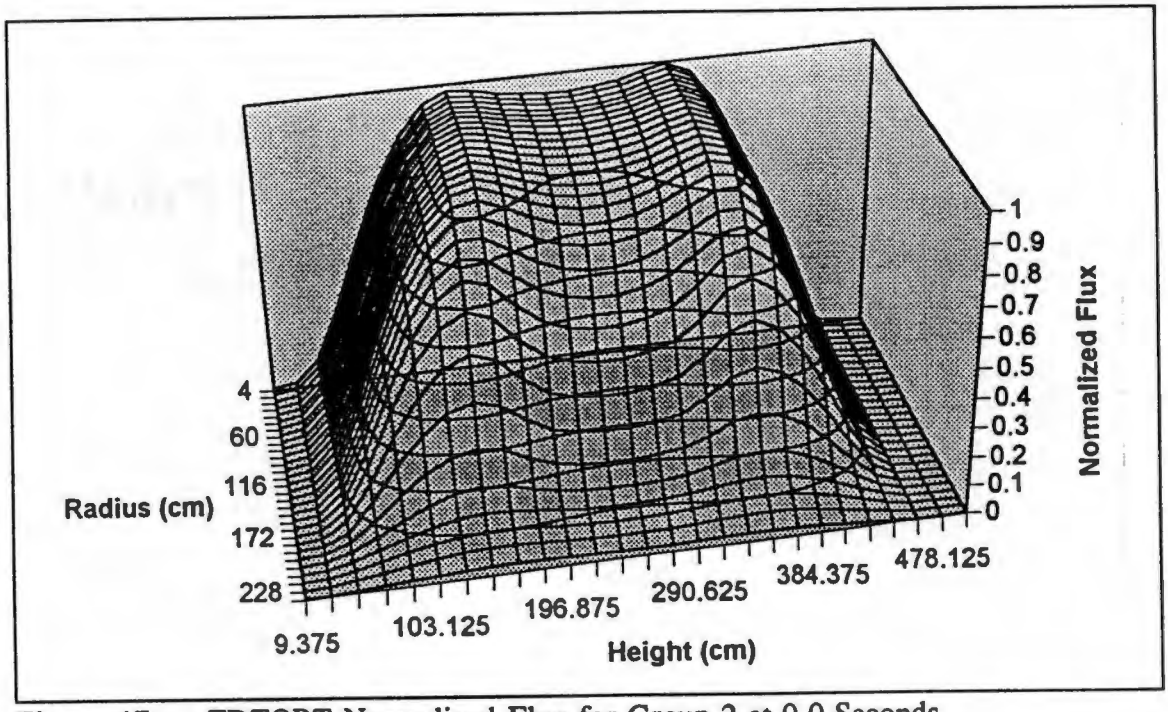



**Figure 17** TDTORT Normalized Flux for Group 2 at 0.0 Seconds.

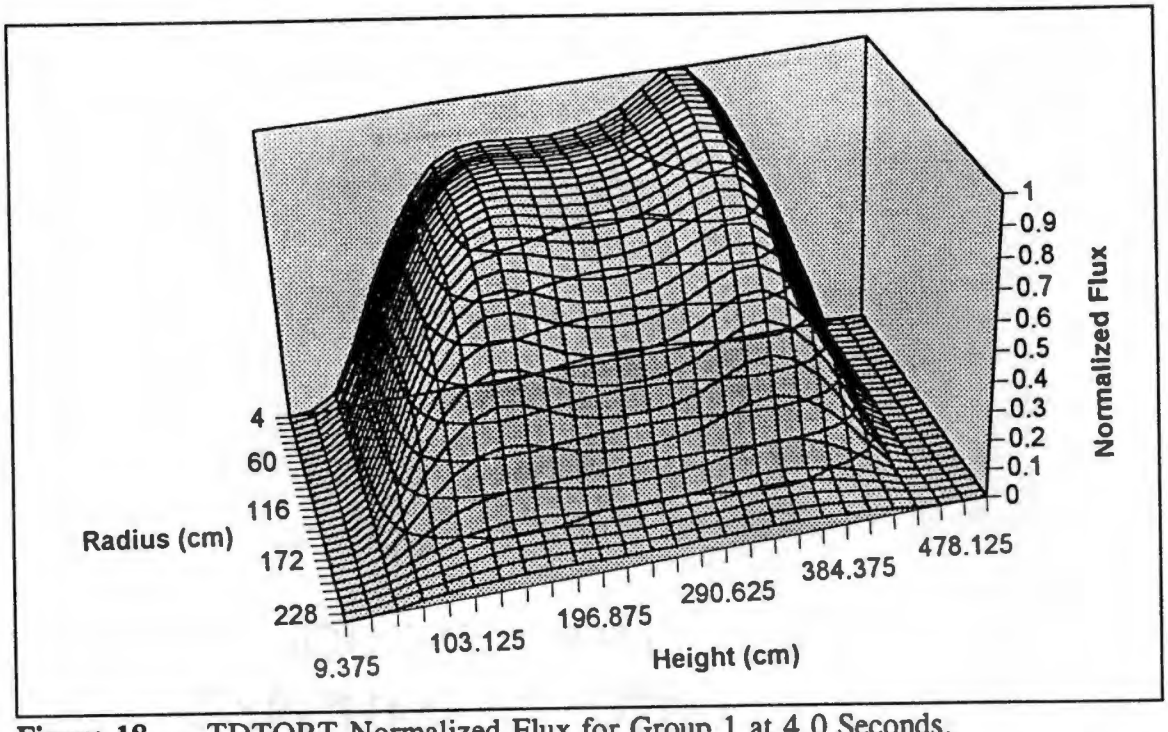



**Figure 18** TDTORT Normalized Flux for Group 1 at 4.0 Seconds.

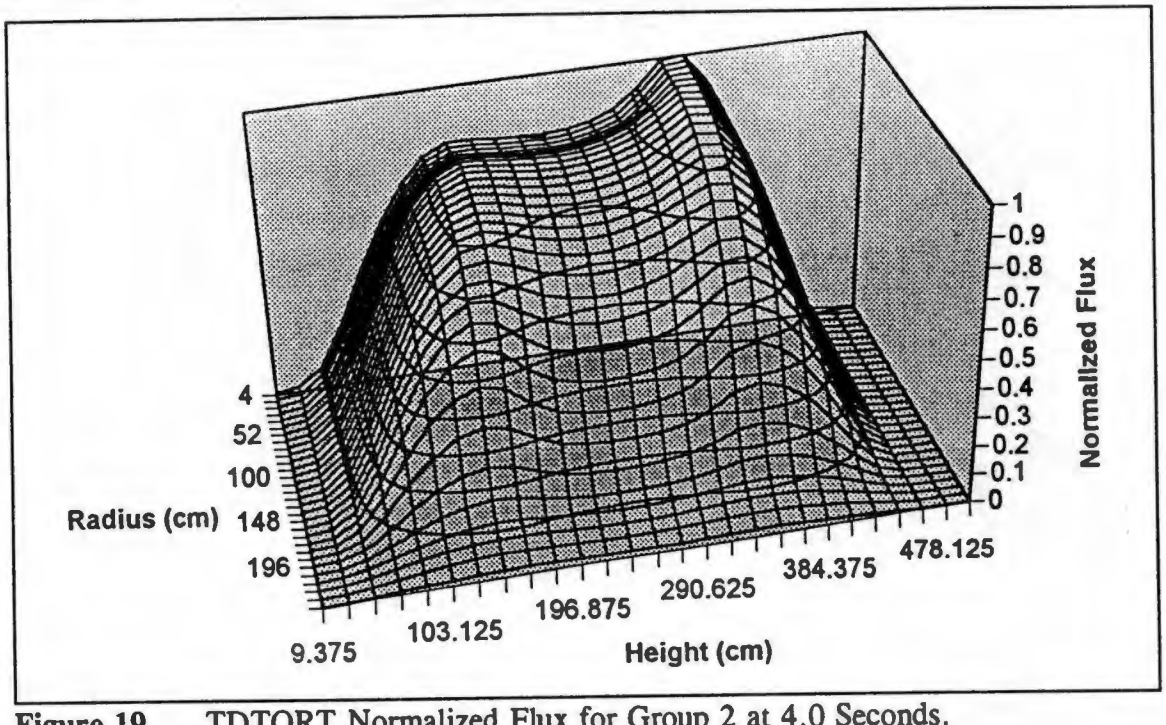



**Figure 19** TDTORT Normalized Flux for Group 2 at 4.0 Seconds.

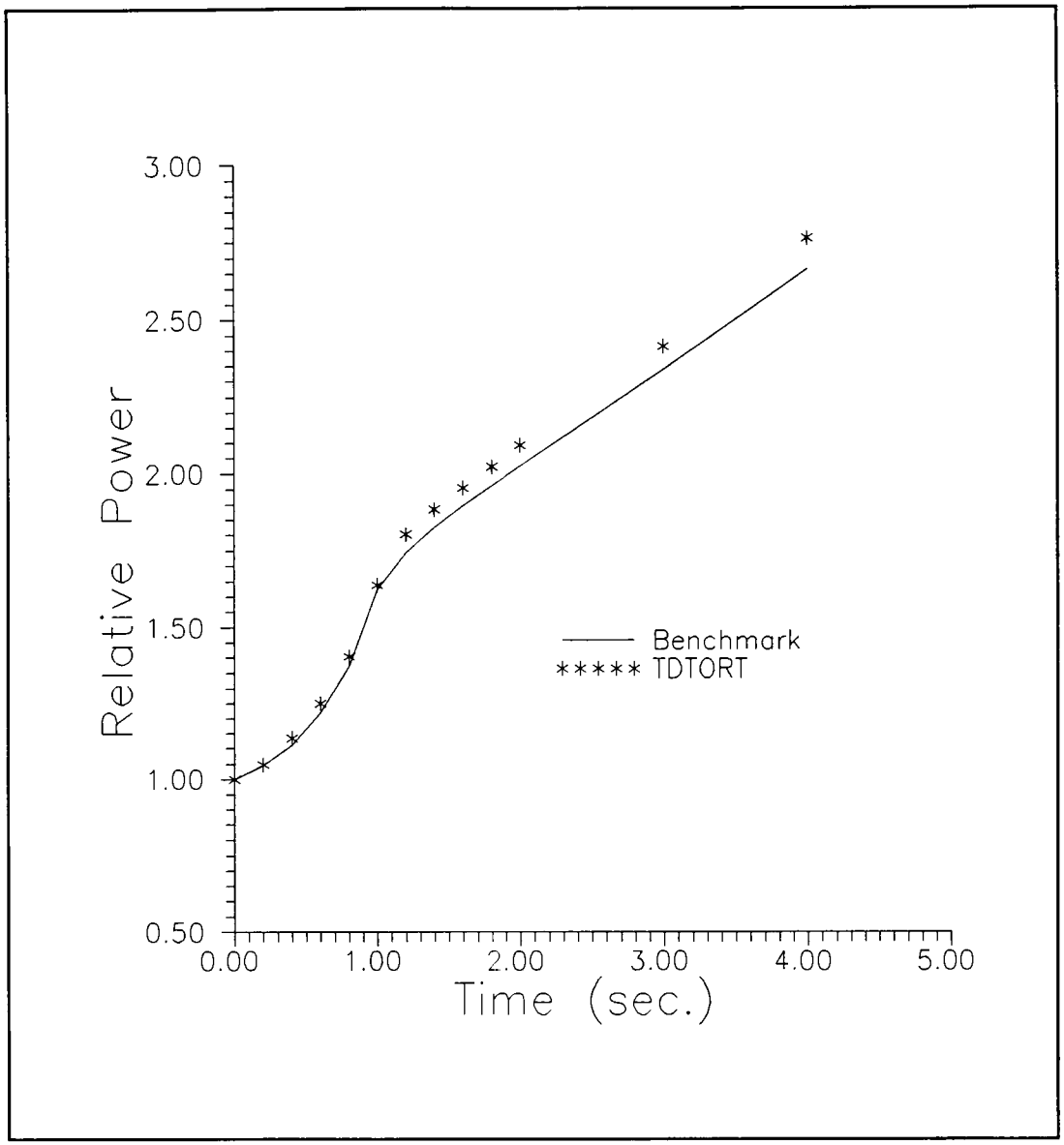



**Figure 20** Relative Power Comparisons of TDTORT and TW0DTA(Benchmark) for Problem 8-A1.

**Table VIII** Point Kinetics Parameters for Benchmark Problem 8-A1

| Time (sec.) | Reactivity | Total Effective Beta | Generation Time |
|---|---|---|---|
| 0.000E+00 | 0.000E+00 | 6.499E-03 | 3.117E-04 |
| 1.000E-01 | 2.201E-04 | 6.499E-03 | 3.117E-04 |
| 2.000E-01 | 4.426E-04 | 6.499E-03 | 3.116E-04 |
| 3.000E-01 | 6.713E-04 | 6.499E-03 | 3.116E-04 |
| 4.000E-01 | 9.048E-04 | 6.499E-03 | 3.116E-04 |
| 5.000E-01 | 1.148E-03 | 6.499E-03 | 3.116E-04 |
| 6.000E-01 | 1.397E-03 | 6.499E-03 | 3.116E-04 |
| 7.000E-01 | 1.682E-03 | 6.499E-03 | 3.116E-04 |
| 8.000E-01 | 1.981E-03 | 6.499E-03 | 3.115E-04 |
| 9.000E-01 | 2.280E-03 | 6.499E-03 | 3.115E-04 |
| 1.000E+00 | 2.591E-03 | 6.499E-03 | 3.115E-04 |
| 1.200E+00 | 2.598E-03 | 6.499E-03 | 3.115E-04 |
| 1.400E+00 | 2.601E-03 | 6.499E-03 | 3.115E-04 |
| 1.600E+00 | 2.605E-03 | 6.499E-03 | 3.115E-04 |
| 1.800E+00 | 2.608E-03 | 6.499E-03 | 3.115E-04 |
| 2.000E+00 | 2.614E-03 | 6.499E-03 | 3.115E-04 |
| 2.500E+00 | 2.613E-03 | 6.499E-03 | 3.115E-04 |
| 3.000E+00 | 2.613E-03 | 6.499E-03 | 3.115E-04 |
| 3.500E+00 | 2.621E-03 | 6.499E-03 | 3.115E-04 |
| 3.900E+00 | 2.627E-03 | 6.499E-03 | 3.115E-04 |

## 4.9 2-D TWIGL Test Problem

The TWIGL test problem is a 2-D (x-y) diffusion theory problem with two neutron energy groups and one delayed neutron group. It is the model of a 165cm by 165cm unreflected seed-blanket reactor problem. The problem is actually two separate problems in that step and ramp perturbations are introduced in the corner seed assembly which result in positive reactivity insertions. The data for this problem is given in Appendix A. The shape updates are performed at .1, .2, .3, .4 and .5 seconds for both types of perturbations. The pointwise flux convergence, fission convergence and eigenvalue convergence criteria for both transients are 5e-4, 5e-4, and 5e-5, respectively. For the step perturbation one hundred reactivity calculations are performed between each shape calculation. Ten point kinetics calculations are performed between each reactivity calculation. For the ramp perturbation ten reactivity calculations are performed between each shape calculation. Ten point kinetics calculations are performed between each reactivity calculation. The point kinetics parameters for step and ramp perturbations using $S_2$ transport theory are given in Tables IX and X. The relative power for step and ramp perturbations calculated by TDTORT agrees very well with the reference as shown in Figures 21 and 22, respectively. The CPU time for the step perturbation is 28 and 56 seconds using $S_2$ and $S_4$ quadrature sets, respectively. The CPU time for the ramp perturbation is 36 and 58 seconds using $S_2$ and $S_4$ quadrature sets, respectively.

**Table IX** Point Kinetics Parameters for TWIGL Test Problem (step perturbation)

| Time (sec.) | Reactivity | Total Effective Beta | Generation Time |
|---|---|---|---|
| 0.000E+00 | 3.701E-03 | 7.500E-03 | 3.803E-05 |
| 5.000E-03 | 3.706E-03 | 7.500E-03 | 3.803E-05 |
| 1.000E-02 | 3.711E-03 | 7.500E-03 | 3.803E-05 |
| 2.000E-02 | 3.722E-03 | 7.500E-03 | 3.803E-05 |
| 3.000E-02 | 3.732E-03 | 7.500E-03 | 3.803E-05 |
| 4.000E-02 | 3.743E-03 | 7.500E-03 | 3.804E-05 |
| 5.000E-02 | 3.754E-03 | 7.500E-03 | 3.804E-05 |
| 6.000E-02 | 3.765E-03 | 7.500E-03 | 3.804E-05 |
| 7.000E-02 | 3.775E-03 | 7.500E-03 | 3.804E-05 |
| 8.000E-02 | 3.786E-03 | 7.500E-03 | 3.804E-05 |
| 9.000E-02 | 3.797E-03 | 7.500E-03 | 3.805E-05 |
| 1.000E-01 | 3.807E-03 | 7.500E-03 | 3.805E-05 |
| 1.400E-01 | 3.808E-03 | 7.500E-03 | 3.805E-05 |
| 1.800E-01 | 3.809E-03 | 7.500E-03 | 3.805E-05 |
| 3.000E-01 | 3.809E-03 | 7.500E-03 | 3.805E-05 |
| 4.000E-01 | 3.809E-03 | 7.500E-03 | 3.805E-05 |
| 4.990E-01 | 3.809E-03 | 7.500E-03 | 3.805E-05 |

**Table X** Point Kinetics Parameters for TWIGL Test Problem (ramp perturbation)

| Time (sec.) | Reactivity | Total Effective Beta | Generation Time |
|---|---|---|---|
| 0.000E+00 | 0.000E+00 | 7.500E-03 | 3.803E-05 |
| 2.000E-04 | 3.694E-06 | 7.500E-03 | 3.805E-05 |
| 4.000E-04 | 7.375E-06 | 7.500E-03 | 3.806E-05 |
| 6.000E-04 | 1.104E-05 | 7.500E-03 | 3.808E-05 |
| 8.000E-04 | 1.470E-05 | 7.500E-03 | 3.810E-05 |
| 1.000E-03 | 1.834E-05 | 7.500E-03 | 3.812E-05 |
| 2.080E-02 | 3.829E-04 | 7.500E-03 | 3.811E-05 |
| 4.060E-02 | 7.503E-04 | 7.500E-03 | 3.811E-05 |
| 6.040E-02 | 1.121E-03 | 7.500E-03 | 3.810E-05 |
| 8.020E-02 | 1.494E-03 | 7.500E-03 | 3.810E-05 |
| 1.000E-01 | 1.870E-03 | 7.500E-03 | 3.809E-05 |
| 1.200E-01 | 2.254E-03 | 7.500E-03 | 3.807E-05 |
| 1.400E-01 | 2.641E-03 | 7.500E-03 | 3.804E-05 |
| 1.600E-01 | 3.032E-03 | 7.500E-03 | 3.802E-05 |
| 1.800E-01 | 3.426E-03 | 7.500E-03 | 3.799E-05 |
| 2.000E-01 | 3.823E-03 | 7.500E-03 | 3.797E-05 |
| 2.500E-01 | 3.816E-03 | 7.500E-03 | 3.801E-05 |
| 3.000E-01 | 3.810E-03 | 7.500E-03 | 3.805E-05 |
| 3.500E-01 | 3.809E-03 | 7.500E-03 | 3.805E-05 |
| 4.000E-01 | 3.809E-03 | 7.500E-03 | 3.805E-05 |
| 4.500E-01 | 3.809E-03 | 7.500E-03 | 3.805E-05 |
| 4.900E-01 | 3.809E-03 | 7.500E-03 | 3.805E-05 |

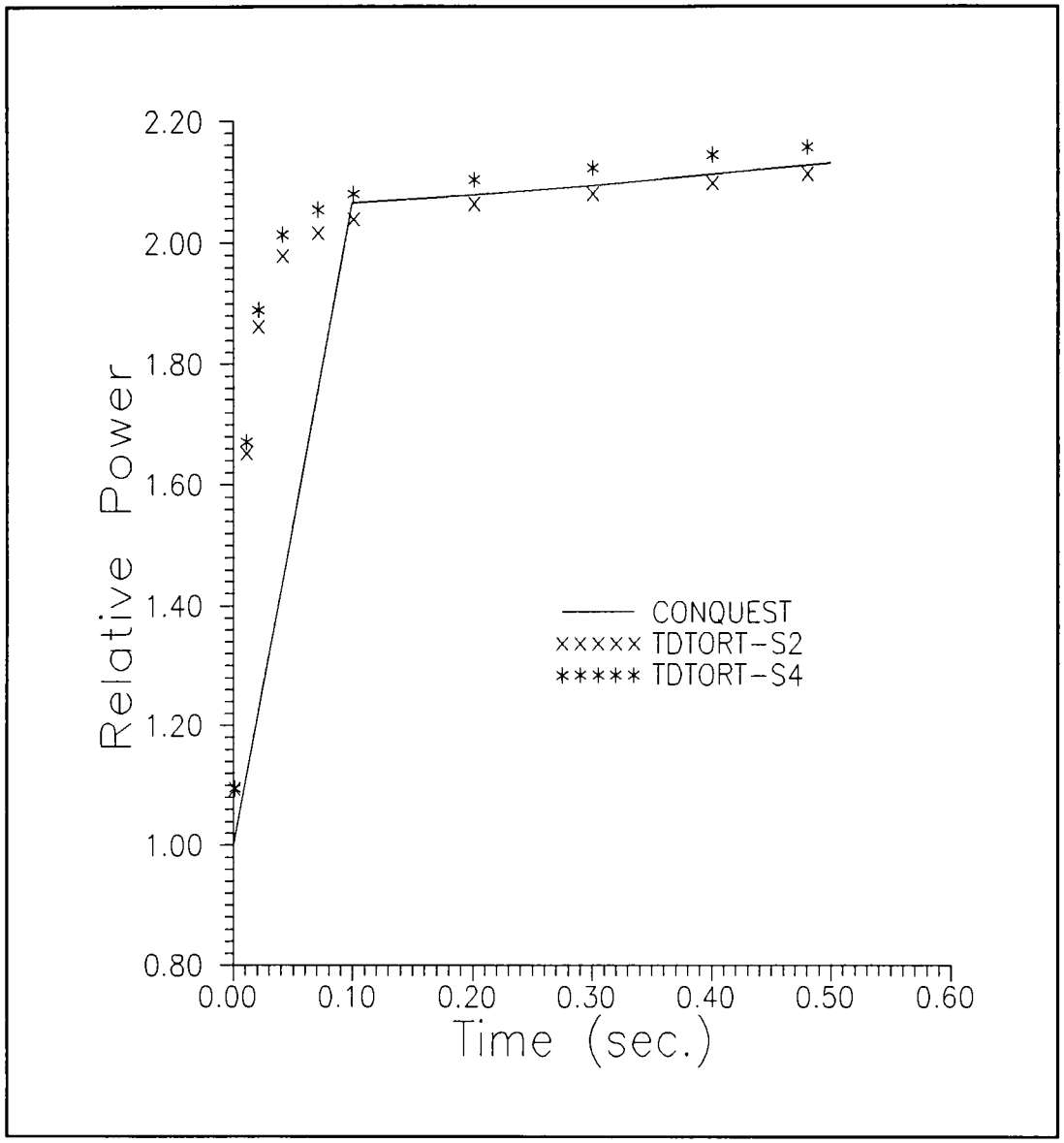



**Figure 21** Relative Power Comparisons of TDTORT and CONQUEST for the TWIGL Test Problem (step perturbation).

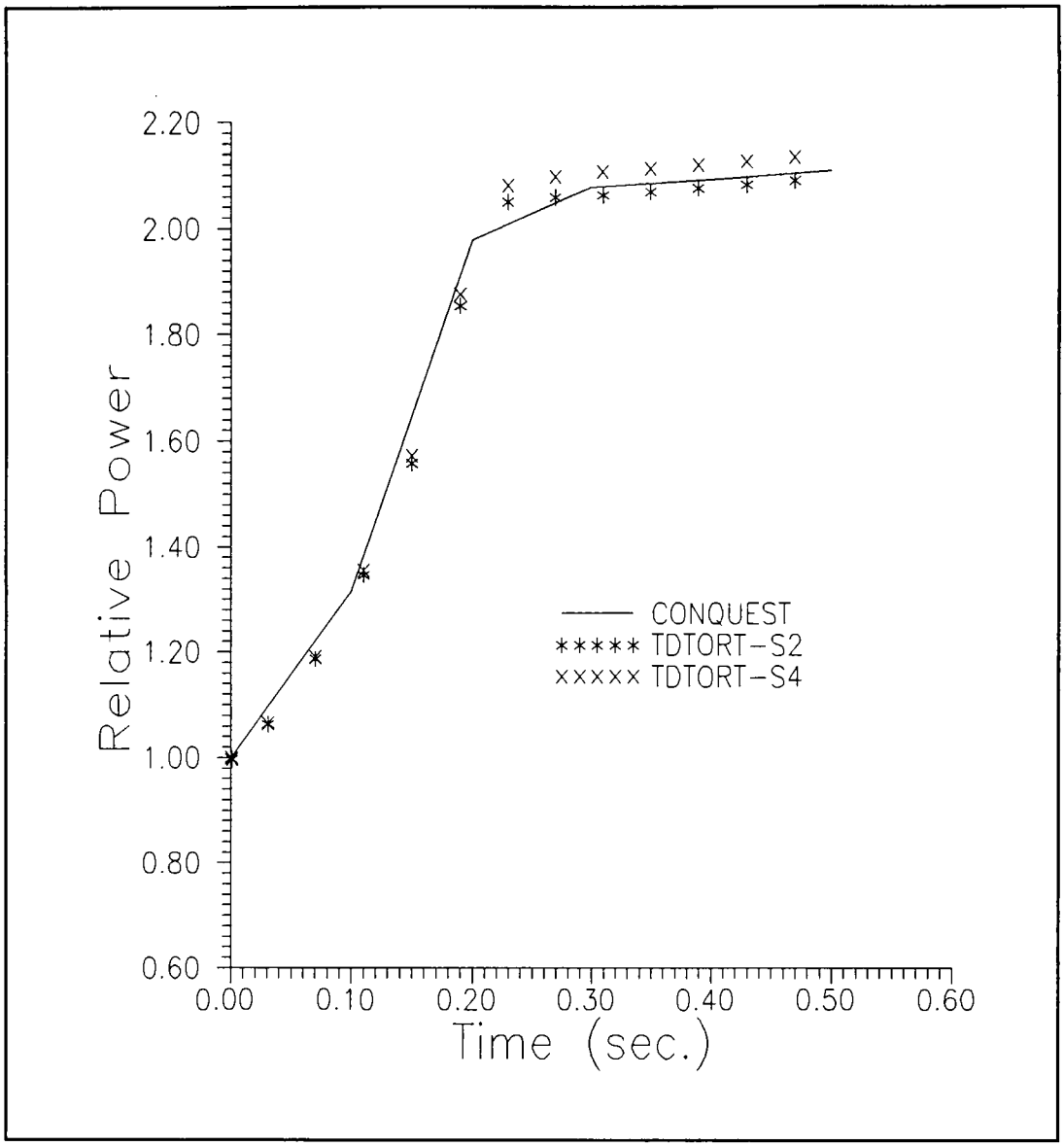



**Figure 22** Relative Power Comparisons of TDTORT and CONQUEST for the TWIGL Test Problem (ramp perturbation).

## 4.10 Benchmark Problem 14-A1

This problem is a 2-D diffusion theory benchmark problem with two energy and two delayed neutron groups. It has a simple thermal feedback model to account for the doppler effect. A detailed description of this problem is given in Appendix A. The pointwise flux convergence, fission convergence and eigenvalue convergence criteria for this problem are 5e-4, 5e-4, and 5e-5, respectively. In all calculations, ten reactivity calculations are performed between each shape calculation. Ten point kinetics calculations are performed in each reactivity time interval. The average core power density comparisons for the benchmark and TDTORT with $S_2$ and $S_4$ quadrature sets and 7.5cm meshes in both x and y directions are given in Figures 23 and 24, respectively. The TDTORT results show good agreement with the benchmark results. The CPU times for the calculations with $S_2$ and $S_4$ quadrature sets are 390 and 1200 minutes, respectively. Furthermore, another calculation is performed to evaluate the effect of the spatial mesh size on the transient. The result of this calculation with an $S_2$ quadrature set and 5cm meshes in both the x and y directions is given in Figure 25. When a finer mesh is used, TDTORT results showed slightly better agreement with the benchmark results. Due to the severity of the transient many shape calculations had to be performed as shown in Table XI. The CPU time for the calculation with finer mesh and $S_2$ quadrature set is 1150 minutes.

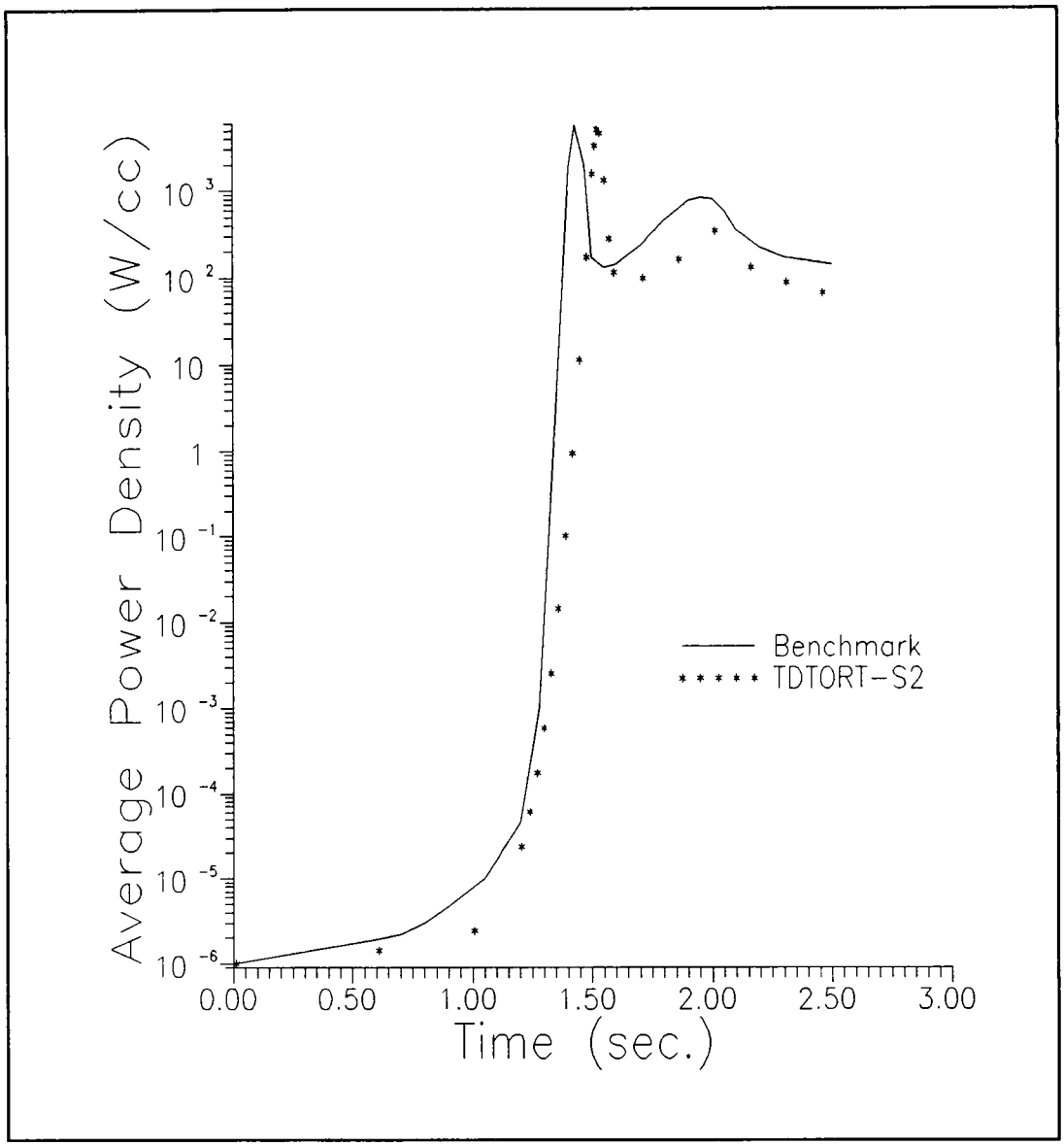



**Figure 23** Power vs. Time Comparisons of TDTORT with an $S_2$ Quadrature and Benchmark for Problem 14-A1.

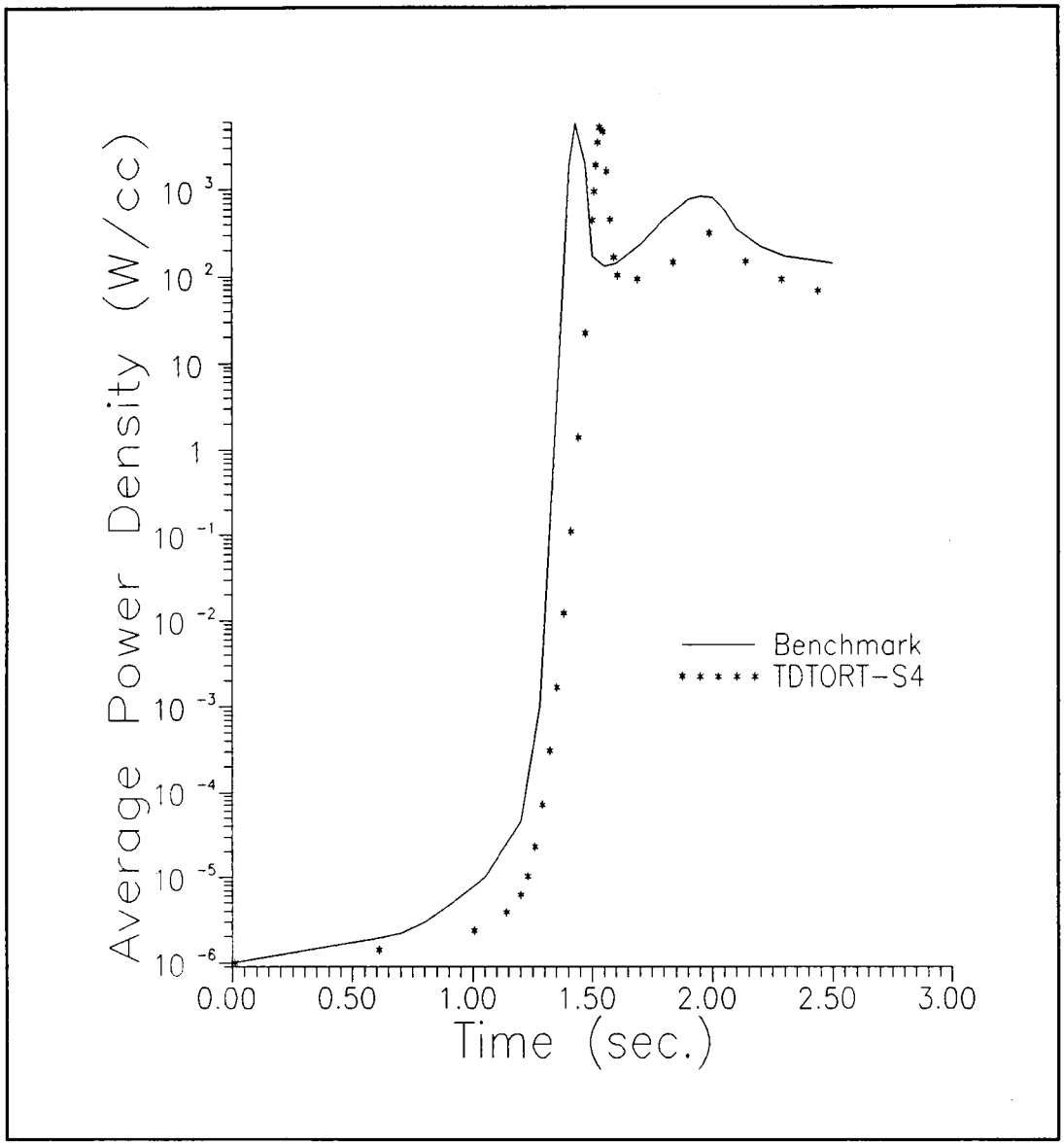



**Figure 24** Power vs. Time Comparisons of TDTORT with an $S_4$ Quadrature and Benchmark for Problem 14-A1.

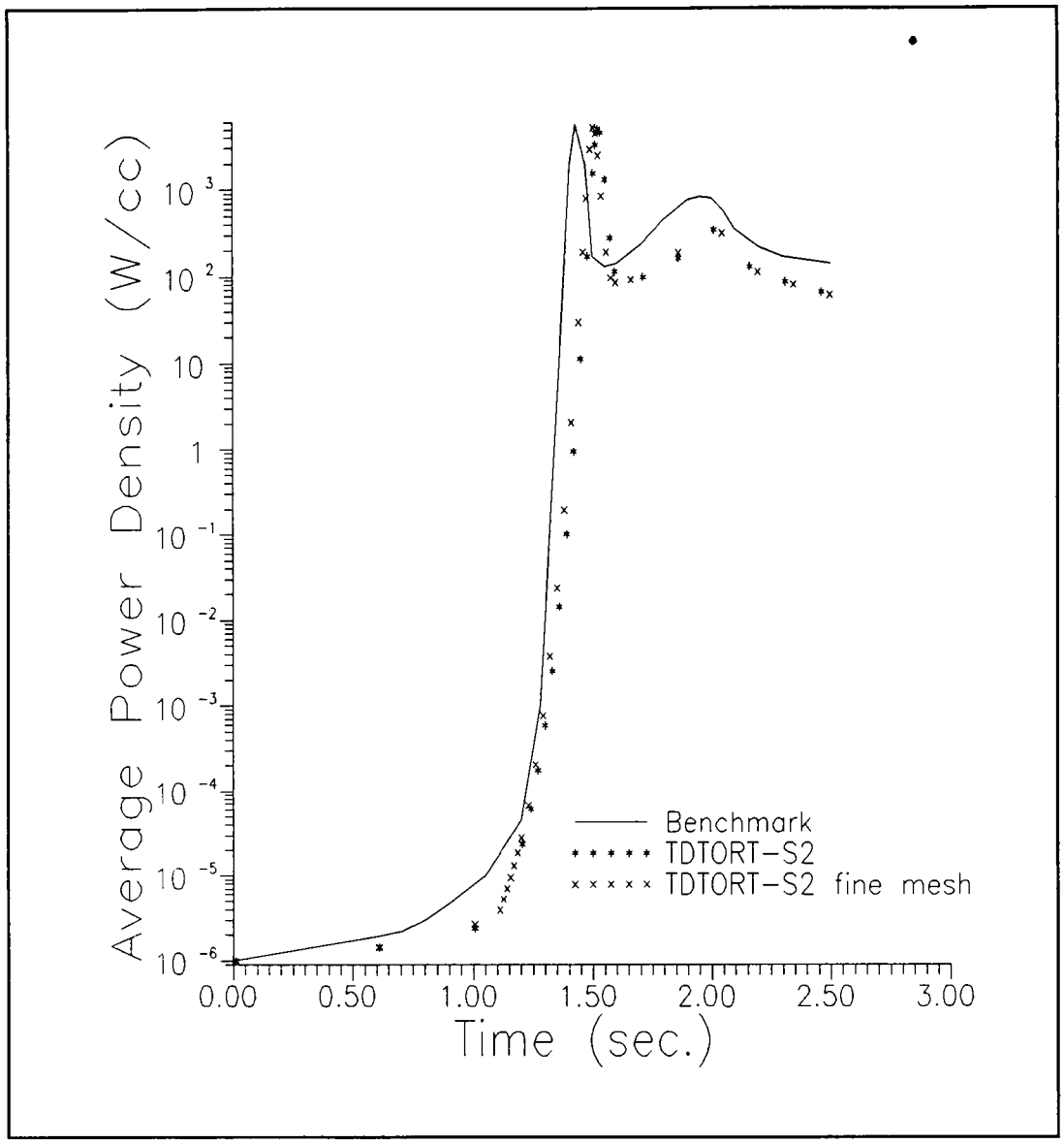



**Figure 25** Power vs. Time Comparisons of TDTORT and Benchmark for Problem 14-A1.

**Table XI** Shape Update Time Intervals for 14-A1.

| Time (sec.) | Shape Update Interval (msec.) |
|---|---|
| $0.0 \leq t \leq 0.8$ | 100 |
| $0.8 \leq t \leq 1.1$ | 50 |
| $1.1 \leq t \leq 1.2$ | 10 |
| $1.2 \leq t \leq 1.5$ | 5 |
| $1.5 \leq t \leq 2.5$ | 1.25 |

## 4.11 Benchmark Problem 14-A2

This problem is similar to problem 14-A1 except that 14-A2 is a 3-D BWR core surrounded by a reflector. The perturbation is introduced by moving one of the control rods out of the reactor in 2 seconds (150cm/sec.). Due to severity of the transient, this problem is widely used to test the codes based on diffusion theory. A comparison between several of these codes is given in Figure 26. Although they all exhibit the same trend, the disagreement between some of the diffusion codes in normalized power density is as much as 50%. TDTORT results with an $S_2$ quadrature set and a 5cm by 5cm by 15cm mesh (in x, y, and z, respectively) show satisfactory agreement with the benchmark results as shown in Figure 27. The shape update time intervals are given in Table XII. The pointwise flux convergence, fission convergence and eigenvalue convergence criteria for this problem are 5e-4, 5e-4, and 5e-5, respectively. The CPU time for this problem is 7760 minutes.

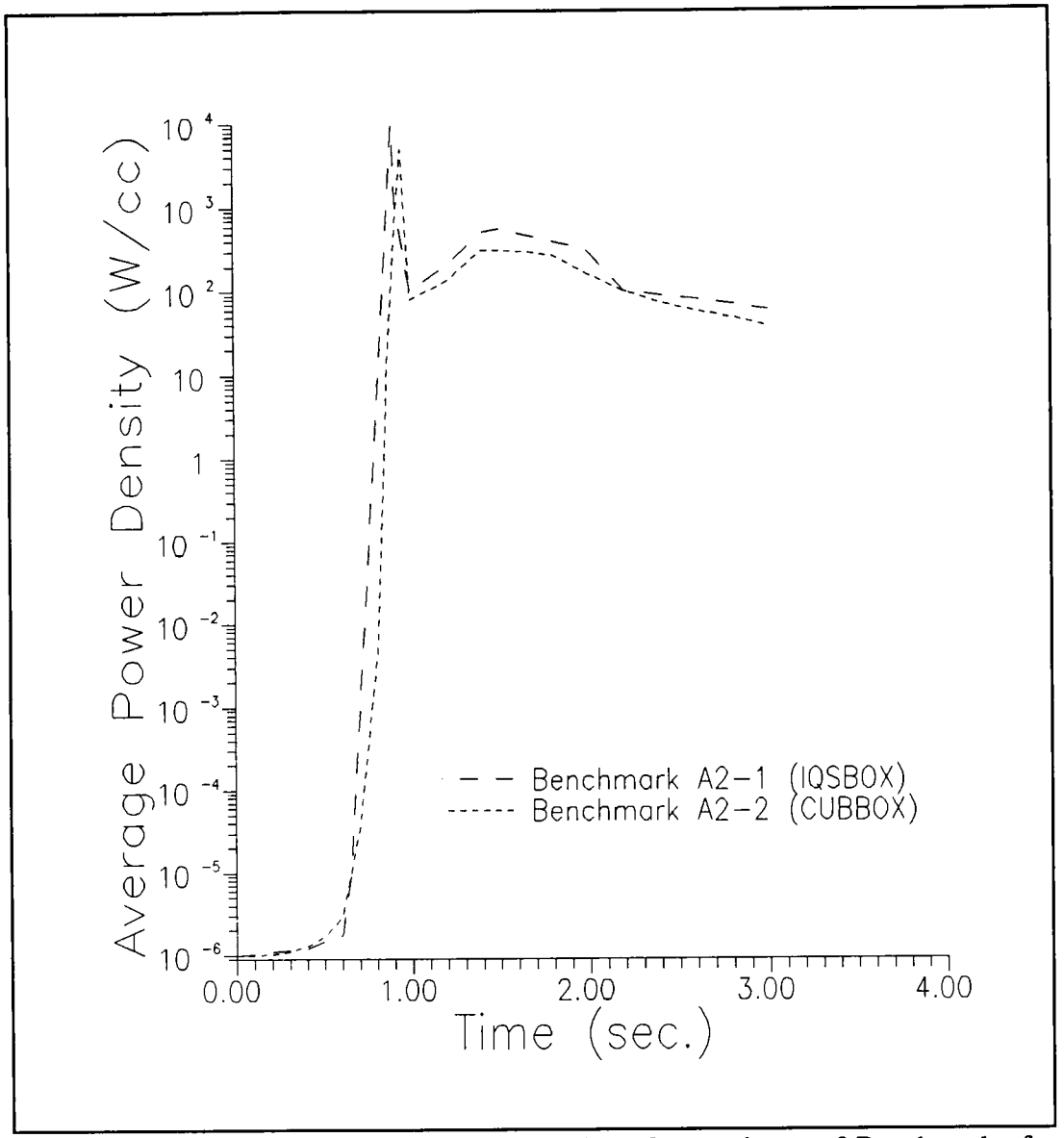



**Figure 26** Average Power Density vs. Time Comparisons of Benchmarks for Problem 14-A2.

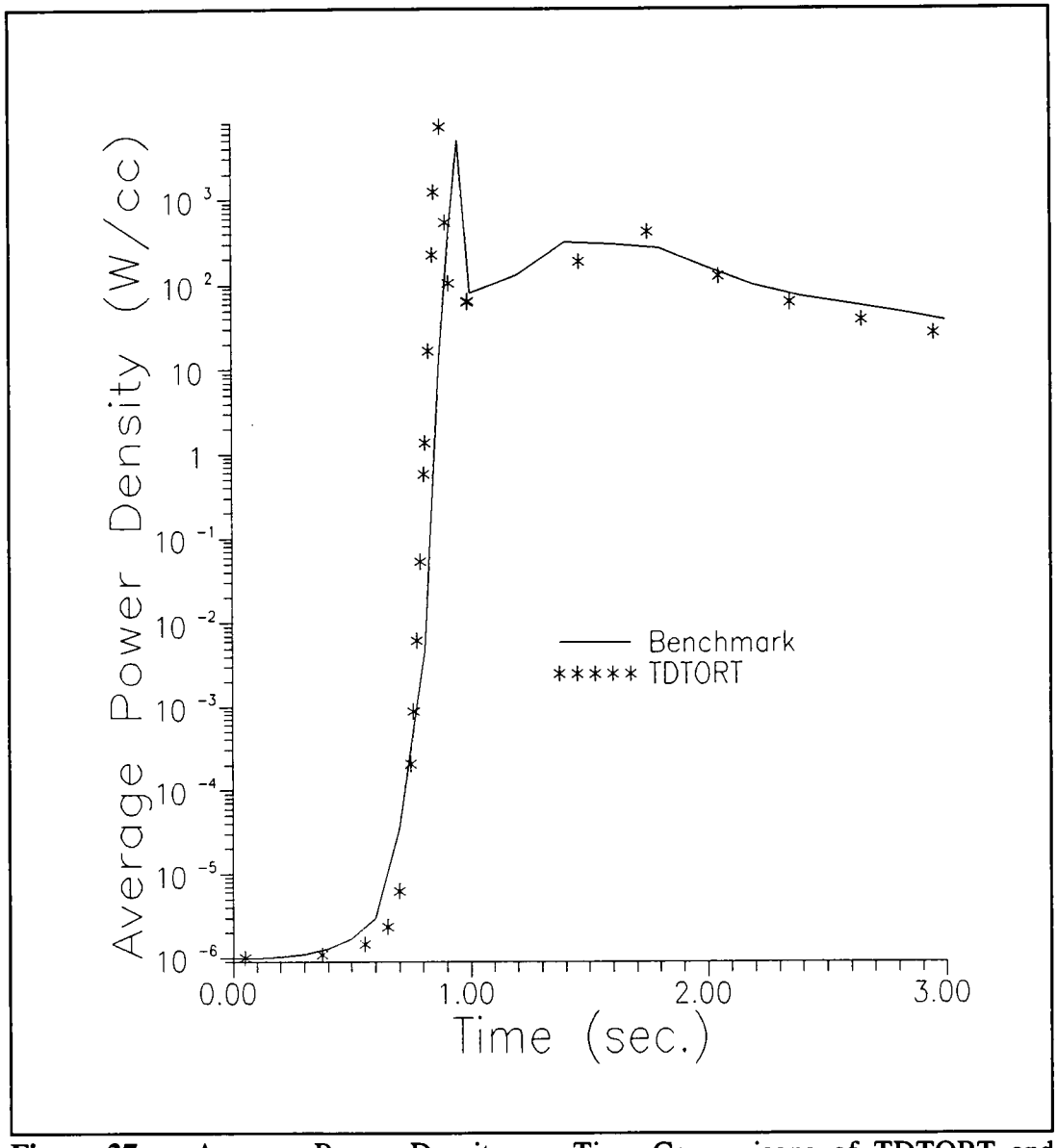



**Figure 27** Average Power Density vs. Time Comparisons of TDTORT and Benchmark for Problem 14-A2.

**Table XII** Shape Update Time Intervals for 14-A2.

| Time (sec.) | Shape Update Interval (msec.) |
|---|---|
| $0.0 \leq t \leq 0.1$ | 100 |
| $0.1 \leq t \leq 0.25$ | 150 |
| $0.25 \leq t \leq 0.5$ | 250 |
| $0.5 \leq t \leq 0.6$ | 50 |
| $0.6 \leq t \leq 0.8625$ | 3.75 |
| $0.8625 \leq t \leq 0.9$ | 1.25 |
| $0.9 \leq t \leq 1.6$ | 1.67 |
| $1.6 \leq t \leq 2.2$ | 1.38 |
| $2.2 \leq t \leq 3.0$ | 800 |

# Chapter Five

# Summary, Conclusions, and Suggestions for Future Work

## 5.1 Summary

A new deterministic methodology is developed in this work for solving the time-dependent, three-dimensional Boltzmann transport equation with explicit representation of delayed neutrons. The methodology used for the time variable is the improved quasi-static (IQS) method. The flux shape is calculated by using TORT which treats the position,energy and angle variables using the discrete ordinates method. The resulting time-dependent, 3-D code is called TDTORT. The flux shape calculated by TORT is linearly interpolated to compute the point kinetics parameters (e.g., reactivity, generation time, etc.) using their inner product definitions. The amplitude function is calculated by solving the point kinetics equations using LSODE and quadratically interpolated point kinetics parameters. The amplitude and shape are then used to compute the flux to obtain the power history of the system.

Transient 1-D benchmark problems 16-A1, -A2, -A3, -A6, -A7, 2-D benchmark problems 8-A1, 14-A1, and 3-D benchmark problem 14-A2 from the ANL benchmark book, and a 2-D TWIGL test problem are used to verify TDTORT. The agreement between TDTORT and the benchmark results for 16-A1 and -A2 is excellent. For 16-A3, -A6 and -A7, the TDTORT results are in good agreement with benchmark results. For the 2-D test problem 8-A1 and the 2-D TWIGL problem, TDTORT results are in

good agreement with the benchmark results. TDTORT results for problems 14-A1 and 14-A2, which have a simple temperature feedback model, exhibit satisfactory agreement with the benchmark results. In conclusion, the IQS methodology implemented in TDTORT for transient, multi-dimensional neutron transport provides satisfactory to excellent results for the benchmark problems considered in this work.

## 5.2 Conclusions

A methodology for solving the time-dependent, three-dimensional Boltzmann transport equation with explicit representation of delayed neutrons has been developed and implemented in a new transient code named TDTORT. TDTORT has been evaluated by comparison with several computational benchmark problems including 1-D, 2-D, and 3-D diffusion and transport theory based problems. All of the benchmark comparisons show that the methodology and code developed in this work have sufficient accuracy and speed to serve as a benchmarking tool for other less accurate models and codes. More importantly, a new computational tool now exists for analyzing the dynamic behavior of complex neutronic systems including voided reactors, small high leakage reactors such as space reactors, and other systems for which the validity of diffusion theory is questionable.

## 5.3 Suggestions for Future Work

With the development of TDTORT, it is now possible to analyze a 3-D system

using either deterministic or stochastic (TDKENO[6]) transport theory. Since both of these codes use the IQS method to treat the time dependance, they could be combined together into one code. The user could then select the type of calculation (deterministic or stochastic) to be used for a specific transient. Alternatively, the user could also specify part of a transient using one approach while another part of the same transient could use a different approach. In addition, THREEDANT[54], another widely used 3-D, discrete ordinates code and MINOTAUR[55], a 3-D discrete ordinates code with generalized geometry, could also be added to such a system. More sophisticated thermal-hydraulic feedback models could also be implemented. Furthermore, in order to show the necessity of using transport theory instead of the diffusion theory for some problems, a computational problem for which the diffusion theory is not valid should be defined and analyzed using both TDTORT and a diffusion code (e.g., NESTLE[1]). Finally, the entire code system including the cross sections should be validated by comparison with results of transient experiments.

# References

# References

1. Rifat M. Al-Chalabi, et al, "NESTLE: A Nodal Kinetics Code," *Trans. Am. Nucl. Soc.*, 68 (1993).

2. T.A. Taiwo, and H.S. Khalil, "DIF3D-K: A Nodal Kinetics Code for Solving the Time-Dependent Diffusion Equation," *Proc. Conf. Mathematics and Computations*, Portland, Oregon (May, 1995).

3. R. Alcouffe, LANL, personal communication.

4. 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 (1993).

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

6. C.L. Bentley, R. DeMeglio, M. Dunn, S. Goluoglu, K. Norton, R. Pevey, I. Suslov and H.L. Dodds, "Development of a Hybrid Stochastic/Deterministic Method for Transient, Three-Dimensional Neutron Transport," *Trans. Am. Nucl. Soc.* (June, 1997).

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

8. W. M. Stacey, Jr., Space-Time Nuclear Reactor Kinetics. Academic Press, New York and London (1969).

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

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

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

12. J. Sanchez, "On the Numerical Solution of the Point Reactor Kinetics Equations by Generalized Runge-Kutta Methods," *Nucl. Sci. Eng.*, 103, 94-99 (1989).

13. K. O. Ott and R. J. Neuhold, *Introductory Nuclear Reactor Dynamics*, ANS, La

Grange Park, Il (1985).

14. W. A. Rhoades and Y. Y. Azmy, "Three Dimensional Sn Calculations with the Oak Ridge TORT code," *Proceedings of the ANS Topical Meeting in Mathematics and Computations*, Portland, Oregon (May, 1995).

15. "Argonne Code Center: Benchmark Problem Book," ANL-7416, Supplement No. 3, Argonne National Laboratory (December, 1985).

16. A. F. Henry, *Nuclear Reactor Analysis*, MIT Press, Cambridge, MA (1975).

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

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

19. W. M. Stacey, Jr., "A General Modal Expansion Method for Obtaining Approximate Equations for Linear Systems," *Nucl. Sci. Eng.* 28, 438 (1967).

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

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

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

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

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

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

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

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

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

29. NEM, personal communication with H.L. Dodds.

30. 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).

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

32. 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 N. E.*, Williamsburgh, Virginia (April, 1979).

33. M. R. Buckner and J. W. Stewart, "Multidimensional Space-Time Nuclear-Reactor Kinetics Studies - Part I : Theoretical," *Nucl. Sci. Eng.* 59, 289-297 (1976).

34. 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).

35. J. B. Yasinsky, M. Natelson, and L. A. Hageman, "TWIGL - A Program to Solve the 2-D, Two-Group, Space-Time Neutron Diffusion Equations with Temperature Feedback," WAPD-TM-743, Bettis Atomic Power Laboratory (1968).

36. D. R. Fergurson, PhD Thesis, MIT, Nucl. Eng. Dept., A EC report MIT-3903-4, U. S. Atomic Energy Commission (August, 1971).

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

38. S. A. Dupree, H. A. Sandmeier, G. E. Hansen, W. W. Engle, Jr., F. R. Mynatt, "Time-Dependent Neutron and Photon Transport Calculations using the Method of Discrete Ordinates," LA-4557 (ORNL-4662) (SAI 70-125) (May, 1971).

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

40. 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).

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

42. W. 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).

43. M. Ash, Nuclear Reactor Kinetics, McGraw-Hill, USA (1979).

44. R. L. Murray, "Reactor Kinetics Pedagogical Insight," *Nucl. Sci. Eng.*, 118, 268-271 (1994).

45. D. A. Meneley, et. al., "A Kinetics Model for Fast Reactor Analysis in Two Dimensions," Dynamics of Nuclear Systems, D. Hetrick, Ed., The University of Arizona Press, Tucson, Arizona, pp. 483-500 (1972).

46. Personal communication with H.L. Dodds.

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

48. S. Goluoglu, C. Bentley, M. Dunn, L. Paschal, R. Pevey, and H.L. Dodds, "Progress in Time-Dependent, Three-Dimensional Neutron Transport Methods Development," *International Conference on the Physics of Reactors - PHYSOR 96*, Mito, Japan (September, 1996).

49. J.E. Morel, J.M. McGhee, and E.W. Larsen, "A 3-D Time-Dependent Unstructured Tetrahedral-Mesh $SP_n$ Method," *Proc. of the ANS Topical Meeting on Mathematics and Computations*, Portland, Oregon (May, 1995).

50. A. Rineiski, and J.Y. Doriath, "KIN3D - A Module That Extends Capabilities of the TGV2 Code for Solving the Time-Dependent Diffusion and Transport Equation," *Proc. Seminar on 3D Deterministic Radiation Transport Computer Programs*, OECD/NEA, Paris (Dec. 1996).

51. K. D. Lathrop, "Discrete Ordinates Methods for the Numerical Solution of the Transport Equation," *Reactor Technology*, Vol 15, No. 2 (Summer, 1972).

52. A. C. Hindmarsh, "LSODE and LSODI - Two New Initial Value Ordinary Differential Equation Solvers," ACM-Signum Newsletter, Vol. 15, No. 4 (1980).

53. W.H. Press, S.A. Teukolsky, W.T. Vetterling, and B.P. Flannery, *Numerical Recipes in FORTRAN*, Cambridge University Press., New York, NY (1992).

54. R.E. Alcouffe, et al., "DANTSYS: A Diffusion Accelerated Neutral Particle Transport Code System," LA-12969-M or UC-705, Los Alamos National Laboratory, (June 1995).

55. I.R. Suslov, R.E. Pevey, C. Bentley, S. Goluoglu, R. DeMeglio, K. Norton, and H.L. Dodds, "Application of the Method of Characteristics to Criticality Safety," *Trans. Am. Nucl. Soc.*, 23, 592 (1997).

# Appendices

# Appendix A: Description of Benchmark Problems

## A.1 Problem 16-A1

| | |
|---|---|
| Identification: | 16-A1 |
| By: | Dr. H.L. Dodds, Jr. (University of Tennessee, Knoxville) |
| Date Submitted: | July, 1980 |
| Descriptive Title: | Delayed Supercritical Transient; 1-D, Two-Group Neutron Transport Problem in a Fast Reactor |
| Suggested Function: | Test 1-D neutron transport transient solutions |
| Reactor Configuration: | See Figure 4 |
| Data: | Initial two-group constants are shown in Table A.1. Delayed neutron parameters, prompt and delayed spectra, and velocities are shown in Table A.2. |

Reduction of Source Situation:

1. 1-D (slab with azimuthal symmetry), two-group neutron transport theory
2. Isotropic scattering
3. Zero return current boundary conditions on external surfaces
4. Steady-state critical 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 of the material in Zone 6 is decreased by 5% (resulting in a step insertion of reactivity at t = 0.0 sec.)

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 A.2 and A.3) 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 A.2 and A.3 if the conventional discrete ordinates method is used.

**Table A.1 Initial Cross Sections for Benchmark Problem 16-A1**

| Zone | Group | $\nu\Sigma_f^g$ | $\Sigma_t^g$ | $\Sigma_s^{g\to g}$ | $\Sigma_s^{g\to g'}$ |
|---|---|---|---|---|---|
| 1,7 | 1 | $8.3441\times10^{-4}$ | $2.411\times10^{-1}$ | $2.336\times10^{-1}$ | $3.598\times10^{-3}$ |
| | 2 | $3.2776\times10^{-4}$ | $4.172\times10^{-1}$ | $4.070\times10^{-1}$ | 0.0 |
| 2,4,6 | 1 | $7.4518\times10^{-3}$ | $1.849\times10^{-1}$ | $1.777\times10^{-1}$ | $2.085\times10^{-3}$ |
| | 2 | $1.1061\times10^{-2}$ | $3.668\times10^{-1}$ | $3.537\times10^{-1}$ | 0.0 |
| 3,5 | 1 | 0.0 | $9.432\times10^{-2}$ | $8.571\times10^{-2}$ | $1.717\times10^{-3}$ |
| | 2 | 0.0 | $1.876\times10^{-1}$ | $1.713\times10^{-1}$ | 0.0 |

Table A.2 $S_4$ Angular Quadrature

| Cosine ($\mu$) | Weight |
|---|---|
| -1.0 | 0.00 |
| -0.788675 | 0.25 |
| -0.211325 | 0.25 |
| +0.211325 | 0.25 |
| +0.788675 | 0.25 |

Table A.3 Spatial Mesh

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

# A.2 Problem 16-A2

| | |
|---|---|
| Identification: | 16-A2 |
| By: | Dr. H.L. Dodds, Jr. (University of Tennessee, Knoxville) |
| Date Submitted: | July, 1980 |
| Descriptive Title: | Prompt Supercritical Transient; 1-D, Two-Group Neutron Transport Problem in a Fast Reactor. Reduction of the source situation, initial data, expected primary results, and possible additional results are the same as in Problem 16-A1. |
| 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. |

# A.3 Problem 16-A3

| | |
|---|---|
| Identification: | 16-A3 |
| By: | Dr. H.L. Dodds, Jr. (University of Tennessee, Knoxville) |
| Date Submitted: | July, 1980 |
| Descriptive Title: | Transient Initially Prompt Supercritical, then Subcritical; 1-D, Two-Group Neutron Transport Problem in a Fast Reactor. Reduction of the source situation, initial data, expected primary results, and possible additional results are the same as in Problem 16-A1. |
| 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 A.4. |

Table A.4 Two-Group Constants for Sodium and Control Rod Materials

| Material | Energy Group | $\Sigma_{tg}^{g}$ | $\Sigma_{s}^{g\rightarrow g}$ | $\Sigma_{s}^{g\rightarrow g'}$ |
|---|---|---|---|---|
| Sodium | 1 | 6.830e-2 | 6.3293e-2 | 1.294e-3 |
| | 2 | 1.257e-1 | 1.21099e-1 | 0.0 |
| Control Rod | 1 | 1.795e-1 | 1.59078e-1 | 3.101e-3 |
| | 2 | 3.903e-1 | 3.35661e-1 | 0.0 |

# A.4 Problem 16-A6

Identification: 16-A6

By: Dr. 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); 1-D, 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. 1-D (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 A.5. 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 A.6.

**Table A.5 Two-Group Constants for Step Perturbation in Zone 5**

| Energy Group | $\Sigma_t^g$ | $\Sigma_s^{g\to g}$ | $\Sigma_s^{g\to g'}$ |
|---|---|---|---|
| 1 | 9.05399e-2 | 8.24499e-2 | 1.6550e-3 |
| 2 | 1.7860e-1 | 1.640e-1 | 0.0 |

**Table A.6 Two-Group Constants for Ramp Perturbations**

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

# A.5 Problem 16-A7

| | |
|---|---|
| Identification: | 16-A7 |
| By: | Dr. 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); 1-D 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. 1-D (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 A.6.

# A.6 Problem 8-A1

| | |
|---|---|
| Identification: | 8-A1 |
| By: | Dr. H.L. Dodds, Jr. (University of Tennessee, Knoxville) |
| Date Submitted: | June, 1977 |
| Descriptive Title: | Two-Dimensional (r-z) reactor model. Delayed supercritical transient; two-dimensional, two-group neutron diffusion problem in thermal reactor |
| Suggested Function: | Test two-dimensional neutron kinetics solutions |
| Reactor Configuration: | |

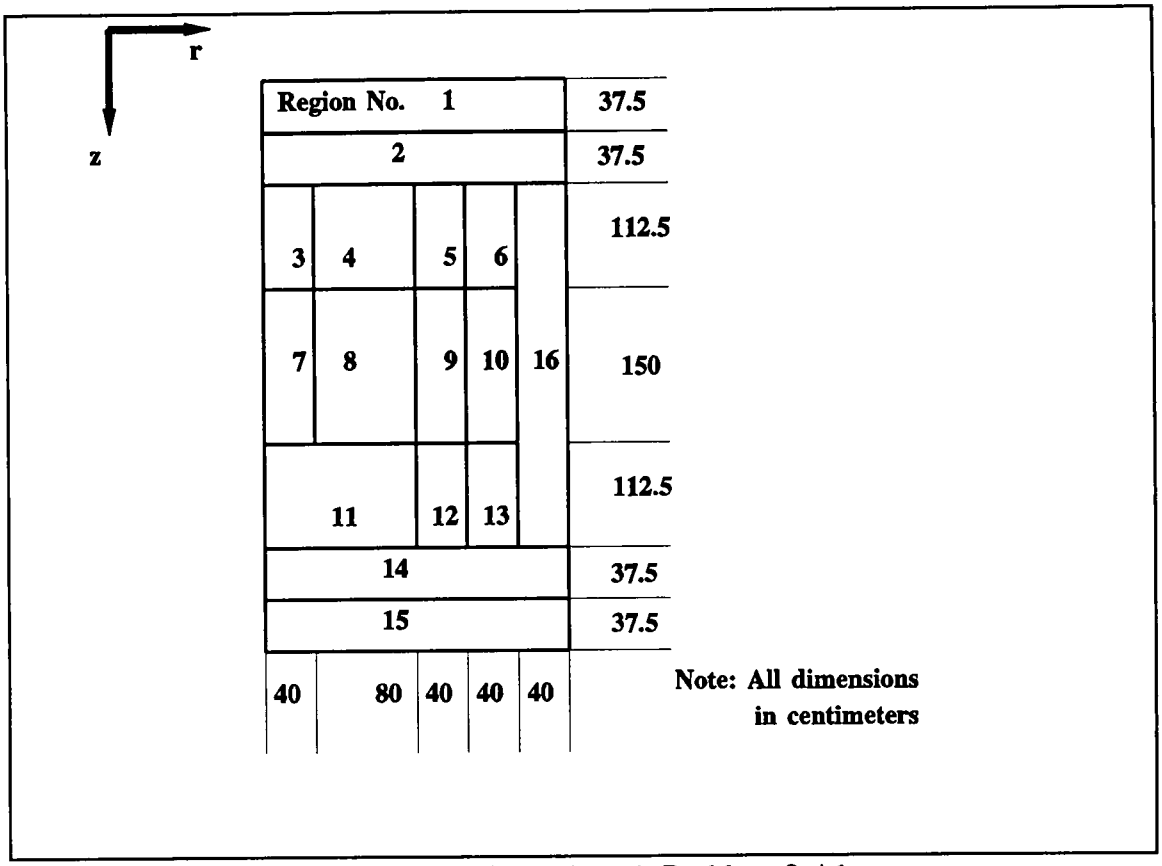



**Figure 28** Schematic Diagram of Benchmark Problem 8-A1.

Data:

v1 = 1e+7 cm/sec

v2 = 2.2e+5 cm/sec

**Table A.7 Initial Cross Sections for Benchmark Problem 8-A1**

| Region | Group | $D_g$ | $\Sigma_g$ | $\nu\Sigma_f$ | $\Sigma_s^{g\to g'}$ |
|---|---|---|---|---|---|
| 1,15 | 1 | 1.0684 | 2.8e-2 | 0 | 2.6e-2 |
| | 2 | .32051 | 3.3e-3 | 0 | 0 |
| 2,14 | 1 | 1.3495 | 1.201e-2 | 0 | 1.2e-2 |
| | 2 | 8.7032e-1 | 1.9e-2 | 0 | 0 |
| 3,4,11 | 1 | 1.3052 | 1.0475e-2 | 1.1776e-3 | 8.0351e-3 |
| | 2 | 8.8857e-1 | 1.3063e-2 | 1.3268e-2 | 0 |
| 5,12 | 1 | 1.3052 | 1.0475e-2 | 1.1776e-3 | 8.0351e-3 |
| | 2 | 8.8857e-1 | 1.2623e-2 | 1.3268e-2 | 0 |
| 6,13 | 1 | 1.3052 | 1.0475e-2 | 1.1776e-3 | 8.0351e-3 |
| | 2 | 8.8857e-1 | 1.1183e-2 | 1.3268e-2 | 0 |
| 7,8 | 1 | 1.3052 | 1.0475e-2 | 1.1776e-3 | 8.0351e-3 |
| | 2 | 8.8857e-1 | 1.3453e-2 | 1.3268e-2 | 0 |
| 9 | 1 | 1.3052 | 1.0475e-2 | 1.1776e-3 | 8.0351e-3 |
| | 2 | 8.8857e-1 | 1.2973e-2 | 1.3268e-2 | 0 |
| 10 | 1 | 1.3052 | 1.0475e-2 | 1.1776e-3 | 8.0351e-3 |
| | 2 | 8.8857e-1 | 1.2933e-2 | 1.3268e-2 | 0 |
| 16 | 1 | 1.2997e+1 | 1.047e-2 | 1.2875e-3 | 7.9061e-3 |
| | 2 | 8.7951e-1 | 1.3065e-2 | 1.4246e-2 | 0 |

Initiating Perturbation: In regions 3 and 7 $\Sigma_2$ is linearly increased by 3% over 1 second. In region 11 $\Sigma_2$ is linearly decreased by 3% in 1 second.

Table A.8 Delayed Neutron Parameters for Benchmark Problem 8-A1.

| Group | Beta | Lambda |
|---|---|---|
| 1 | 2.47e-4 | 1.27e-2 |
| 2 | 1.384e-3 | 3.17e-2 |
| 3 | 1.222e-3 | 1.15e-1 |
| 4 | 2.645e-3 | 3.11e-1 |
| 5 | 8.32e-4 | 1.40 |
| 6 | 1.69e-4 | 3.87 |

# A.7 TWIGL Test Problem

Reactor Configuration:

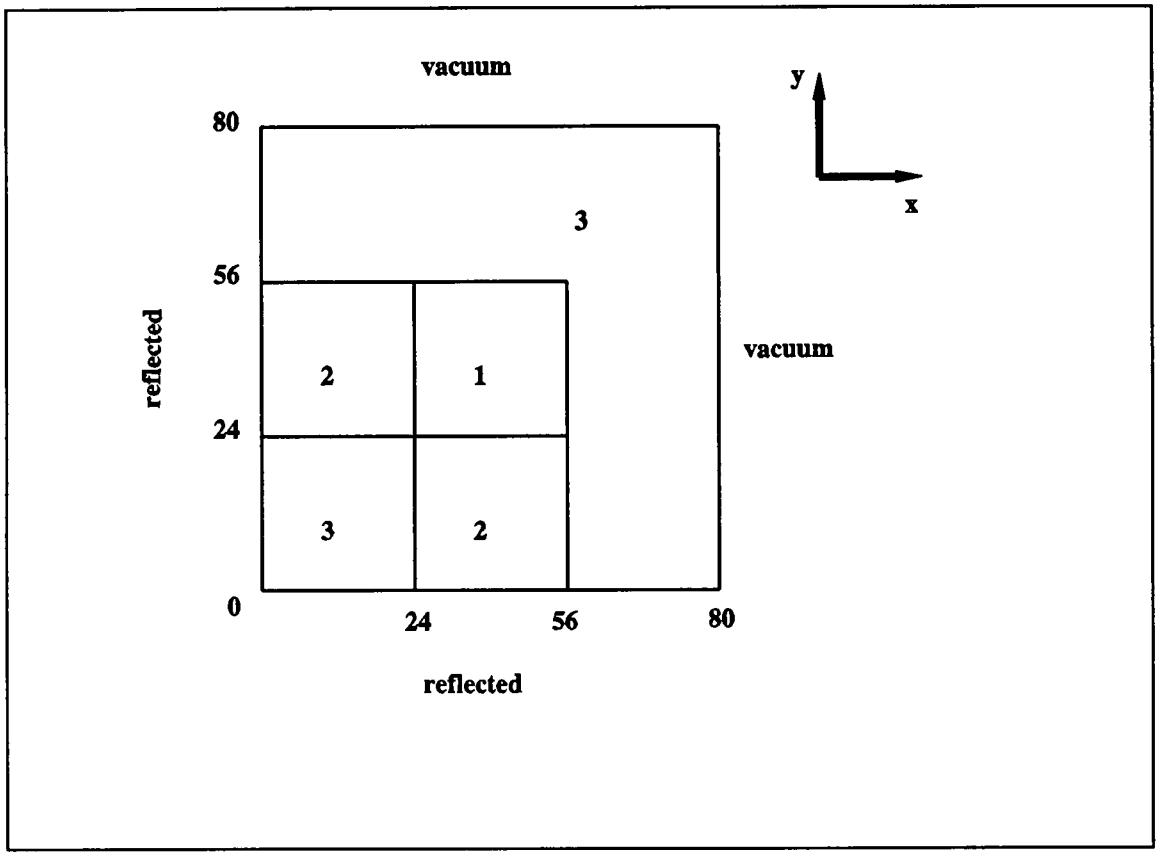



**Figure 29** Schematic Diagram of 2-D TWIGL Test Problem.

Data:

**Table A.9 Initial Cross Sections for 2-D TWIGL Test Problem**

| Composition | Group | $\Sigma_a$ | $\nu\Sigma_f$ | $\Sigma_t$ | $\Sigma_s^{g\to g}$ | $\Sigma_s^{g'\to g}$ |
|---|---|---|---|---|---|---|
| 1 | 1 | 0.01 | 0.007 | 0.2481 | 0.2281 | 0.0 |
| | 2 | 0.15 | 0.2 | 0.9833 | 0.8333 | 0.01 |
| 2 | 1 | 0.01 | 0.007 | 0.2481 | 0.2281 | 0.0 |
| | 2 | 0.15 | 0.2 | 0.9833 | 0.8333 | 0.01 |
| 3 | 1 | 0.008 | 0.003 | 0.2644 | 0.2464 | 0.0 |
| | 2 | 0.05 | 0.06 | 0.7167 | 0.6667 | 0.01 |

v1 = 1e+7 cm/sec
v2 = 2e+5 cm/sec

**Table A.10 Delayed Neutron Parameters for TWIGL Test Problem.**

| Group | Beta | Lambda |
|---|---|---|
| 1 | .0075 | .08 |

Initiating Perturbation: For step perturbation, in composition 1 $\Sigma_{a2}$ is decreased by 0.0035 at time zero. For ramp perturbation in composition 1 $\Sigma_{a2}$ is linearly decreased by 2.333% in 0.2 second.

# A.8 Problems 14-A1 and 14-A2

Identification: 14-A1,14-A2

By: S. Langenbuch (GRS-Munich)

W. Werner (GRS-Munich)

Date Submitted: June, 1976

Descriptive Title: Super Prompt-critical Transient; 2-D, 3-D, Two-group Neutron Diffusion Problem, with Adiabatic Heatup and Doppler Feedback in a Thermal Reactor

Suggested Function: Test 2d, 3d Neutron Kinetics Solution, Especially for Coarse Mesh Methods

Reactor Configuration:

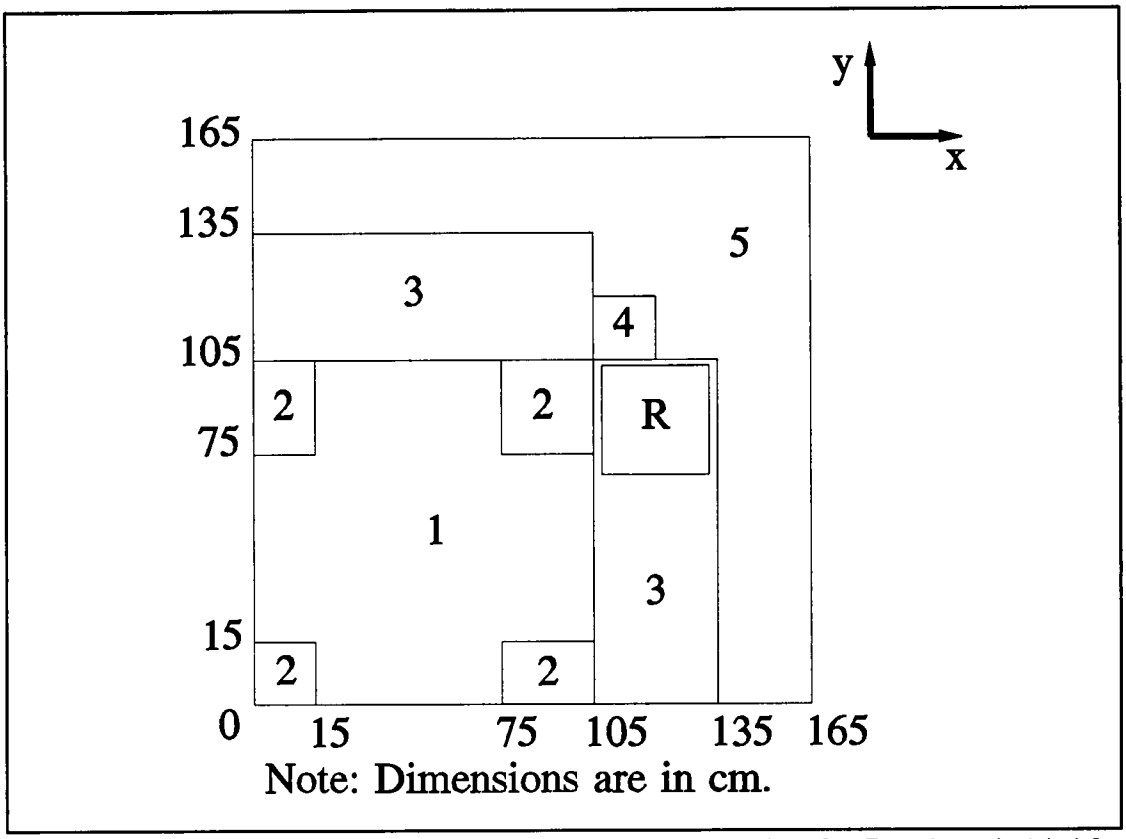



**Figure 30** Quadrant of Reactor Horizontal Cross Section for Benchmark 14-A2.

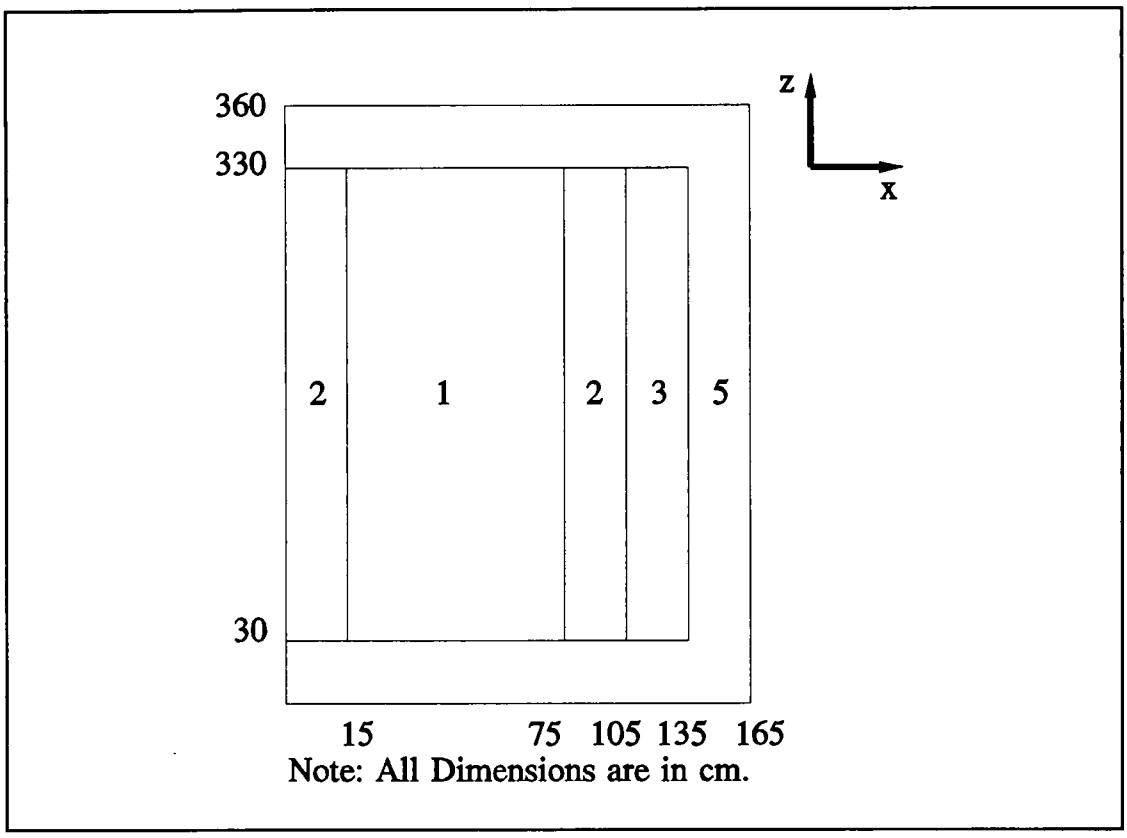



**Figure 31** Reactor Vertical Cross Section for Benchmark 14-A2.

Data: Delayed neutron parameters, prompt and delayed spectra, and speeds are shown in Table A.11. Initial two-group constants are shown in Table A.12.

**Table A.11 Benchmark Problem 14-A2 Delayed Neutron Parameters**

| Delayed Neutron Group | Delayed Neutron Fraction, $\beta$ | Decay Constant, $\lambda$ |
|---|---|---|
| 1 | $5.40 \times 10^{-3}$ | $6.54 \times 10^{-3}$ |
| 2 | $1.087 \times 10^{-3}$ | 1.35 |

Table A.12 Benchmark Problem 14-A2 Initial Two-Group Data

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

Reduction of Source Situation:

1. 3-D (x-y-z), two-group diffusion theory
2. Two delayed neutron precursor groups
3. Adiabatic heatup:

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

4. Doppler feedback:

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

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:

$\alpha$ = 3.83e-11 K cm$^3$ conversion factor

$\gamma$ = 3.034e-3 K$^{1/2}$ feedback constant

$\epsilon$ = 3.204e-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 for 14-A1:

where, t = time (sec).

$$\frac{\Sigma_{a_2}(t)}{\Sigma_{a_2}(t=0)} = \begin{cases} 1 - 0.0606184\,t & t \le 2 \\ 0.8787631 & t \ge 2 \end{cases}$$

Cross Section Perturbation for 14-A2:

Control rod in region R is moved out of the reactor at 150cm/sec. in 2 seconds. This corresponds to replacing material 3 in region R with material 4.

# Appendix B: TDTORT Users Manual

## B.1 Input File Description

TDTORT input is prepared in FIDO[14] format for user friendliness and has two sections. The first section contains the time-dependent information for the transient being modeled, and is entered in three blocks of data. Each data block ends with block terminator "t" in any column after the last data entry. This section of data is placed between the two cards "=transin" and "end", starting in column one. A sample transient input section is given below:

```
=transin
 1$$ 6 1  1 22 1 0 6 2000 52311 1 1 0 1 0 e
 2$$ 2 0 1 1 0 e
t
 4** .0129 .0311 .134 .331 1.26 3.21
 5** .81-4 6.87-4 6.12-4 11.38-4 5.12-4 1.7-4
 6** 1.851-9  1.088-8
 7** 1. 0.
 8** 2.43 0. 1 0   /nu,powini ,rtol,atol
 9** 1e-5 1e-4 1e-3 .1 5 10
t
10$$ 1000   1000 1000  1000 1000 1000
11$$ 10000 10000 10000 10000 10000 10000
t
end
```

The second section contains the input file for the static TORT calculations, and is in the standard TORT format. An example of the TORT format sample input is shown below:

```
=tort
"Benchmark 16: 1-D 2 gr case
61$$  0 1 a5 -3    e                  /flux to 1; scratch source to 3
62$$  50 500 1 0 0  a8 0 1 a12 1 a34 100 e
63$$  21 1 1 0 0   1 1 1 1 20  7 0 32 7  e        /geom
/63$$  114 1 1 0 0   1 1 1 1 20  7 0 32 7  e       /geom
64$$  2 5 0 3 5  e                          /c.s.
```

```
66**  1-5 1-5 100 1-5 a12 .9  a25 0  e      /convg, theta, acc ratio
 t
 t
/ s   4 full symmetric
81**
0 2r416664-7 0 4r416664-7
 3q   8
82**
-4714-4 -33333-5 33333-5 -94281-5 -88192-5 -33333-5 33333-5 88192-5
 3q   8
83**
3r-88192-5 5r-33333-5
  q   8 g  16
 t
 1**  1 f0
 6**  1 f1
 2**  0 40 43 50 59.1 68.2 77.374 84.374 87.374 96.374 101.374
     125.374 130.374 139.374 142.374 149.374 158.5 167.6
      176.748 183.748 186.748 226.748
/ 2**  19i0 23i40 4i87.374 15i96.374 4i130.374
/       23i139.374 19i186.748 226.748
 3**  0  1
 4**  0  1
 8$$  1 2 3 4 5 6 7
 9$$  1 2 3 4 3 5 1
14**  0  40      87.374 96.374  130.374 139.374 186.748
15**  40 87.374 96.374 130.374 139.374 186.748 226.748
17**  f1
19**  f1
 t
96**  f.1
 t
=end
```

These two files appear one after the other, and an "end" card starting in column one indicates the end of the entire input file. This file, must be named *inputa*, has the final form:

```
=transin
 1$$ 6 1  1 22 1 0 6 2000 52311 1 1 0 1 0 e
 2$$ 2 0 1 1 0 e t
 4** .0129 .0311 .134 .331 1.26 3.21
 5** .81-4 6.87-4 6.12-4 11.38-4 5.12-4 1.7-4
 6** 1.851-9  1.088-8
 7** 1. 0.
 8** 2.43 0. 1 0   /nu,powini ,rtol,atol
 9** 1e-5 1e-4 1e-3 .1 5 10
```

```
t
10$$ 1000   1000 1000  1000 1000 1000 1000
11$$ 10000 10000 10000 10000 10000 10000
t
end
=tort
"Benchmark 16: 1-D 2 gr case
61$$  0 1 a5 -3    e                  /flux to 1; scratch source to 3
62$$  50 500 1 0 0  a8 0 1 a12 1 a34 100 e
63$$  21 1 1 0 0   1 1 1 1 20  7 0 32 7  e        /geom
/63$$  114 1 1 0 0   1 1 1 1 20  7 0 32 7  e        /geom
64$$  2 5 0 3 5  e                              /c.s.
66**  1-5 1-5 100 1-5 a12 .9  a25 0  e     /convg, theta, acc ratio
 t
 t
/  s   4 full symmetric
81**
0 2r416664-7 0 4r416664-7
 3q   8
82**
-4714-4 -33333-5 33333-5 -94281-5 -88192-5 -33333-5 33333-5 88192-5
 3q   8
83**
3r-88192-5 5r-33333-5
  q   8 g  16
 t
1**  1 f0
6**  1 f1
2**  0 40 43 50 59.1 68.2 77.374 84.374 87.374 96.374 101.374
    125.374 130.374 139.374 142.374 149.374 158.5 167.6
     176.748 183.748 186.748 226.748
/ 2**  19i0 23i40 4i87.374 15i96.374 4i130.374
/       23i139.374 19i186.748 226.748
3**  0  1
4**  0  1
8$$  1 2 3 4 5 6 7
9$$  1 2 3 4 3 5 1
14**  0  40     87.374 96.374  130.374 139.374 186.748
15**  40 87.374 96.374 130.374 139.374 186.748 226.748
17**  f1
19**  f1
 t
96**  f.1
 t
=end
end
```

The input processor receives the input and creates the files necessary for TORT to perform steady-state forward and adjoint shape calculations, as well as the file needed to calculate the fixed source. The steady-state forward and adjoint shapes are calculated first if they are not readily available from previous runs. TDTORT then proceeds with the transient.

## B.2 Transient Input

BLOCK ONE

1$$ array:

ndg = number of delayed neutron precursor groups.

irepmx = number of iterations over each shape time interval

ixtr = spare, enter 0

n6fi = output unit for fidas messages (default=6)

iout = output flag (0=short,1=long,2=all)

ifix = spare, enter 0

idtnm = number of shape calculations

idtjm = spare, enter 0

idtkm = spare, enter 0

iafl = adjoint flux calculation flag (default=0)

0 = calculate and put adjoint flux on default unit 26,

1 = read the flux from unit 26 in varscl format,

2 = use flat adjoint flux)

iqsf = spare, enter 0

ipint = normalization flag for shape

1 = calculate interpolated shape and apply normalization,

0 = don't

iagrev = adjoint ordering reversal flag

0 = do nothing,

1 = reverse the energy groups for adjoint flux

iextrp = extrapolation flag

0 = don't,

1 = extrapolate, but if negatives are encountered don't extrapolate,

2 = extrapolate, but if negatives are encountered set

them equal to 0.

irstrt = restart flag (start calculating new shape at irstrt+1)

End this array with an "e".

2$$ array:

movem = perturbation type

0 = special case for 14-A1 and -A2,

1 = columnwise material movement

2 = step with cross section file

3 = 2 step perturbations

5 = linear with cross section file

6 = step followed by linear

92= do one forward to determine k_eff and critical cross sections and quit (for 0,1,2 and 5)

93= do one forward to determine k_eff and critical cross sections and quit (for 3)

96= do one forward to determine k_eff and critical cross sections and quit (for 6)

ilpend = shape calculation number at which linear perturbation ends (for movem=5 or 6 only)

shape calculation number at which second step perturabtion occurs (for movem=3 only)

ipkmod = point kinetics mode flag

0 = Bell & Glastone method,

1 = Henry method

iqdmod = delayed neutron source calculation method

0 = approximation,

1 = direct integration

iqs = method flag

1 = traditional iqs shape calculation

0 = Henry's method flux calculation

ifeed = spare

mzm = number of zones that were step perturbed

movem = 6 only, max 10

(mz(i),i=1,mzm) = zone numbers that were step perturbed

movem = 6 only

End this array with an "e".

End this block with a "t".

BLOCK TWO

4** array:

alambda = decay constants (ndg values)

5** array:

beta = delayed neutron fractions (ndg values)

6** array:

vinv = inverse velocities (igm values)

7** array:

chid = delayed neutron spectrum (igm values)

8** array:

rnu = average number of neutrons per fission

powini = inital power

(fluxes will be normalized to achieve this power if not zero)

srtol = tolerance parameter

-ve = use abs(srtol)

+ve = use built in algorithm to generate optimum tolerance parameters

satol = absolute tolerance parameter (enter zero)

9** array:

dtnx = times at which shape calculations to be performed (idtnm values)

End this block with a "t".

BLOCK THREE

10$$ array:

idtj = number of reactivity calculations for each shape calculation (idtnm values)

11$$ array

idtk = total number of point kinetics calculations for each shape calculation (idtnm values)

End this block with a "t".

BLOCK FOUR (Required only for movem = 0 and 1)

70$$ array:

mvmatr = material number that replaces the old material (nonzero for movem=1 only)

mvkl = lowest plane number of material motion

mvku = highest plane number of material motion

mvjl = lowest column number of material motion

mvju = highest column number of material motion

mvil = lowest row number of material motion

mviu = highest row number of material motion

End this array with an "e".

71** array

trmst = time at which rod starts moving

trmnd = time at which rod stops moving

trmsp = speed at which rod moves (if zero, calculated from data)

ieditn = number of power density edits (movem = 0 only, max. 5)

tedit = times of power density edits (ieditn values)

End this array with an "e".

End this block with a "t".

## B.3 Cross Section Input Files

TDTORT uses GIP[14] formatted cross section files which are the standard files use in TORT. The user must create the steady-state adjoint and forward cross section files, as well as the perturbed cross section files. The adjoint and forward cross section files must be named "*fort.18*" and "*fort.8*", respectively. The perturbed cross sections must be place in file "*fort.9*". If there are two perturbations, such as two step perturbations or step followed by linear, then a second perturbed cross section file must be prepared and placed in file "*fort.11*". TDTORT will read the perturbed cross sections from unit 9, and start the calculations. If a second perturbation is initiated, it will read the second perturbed cross sections from unit 11.

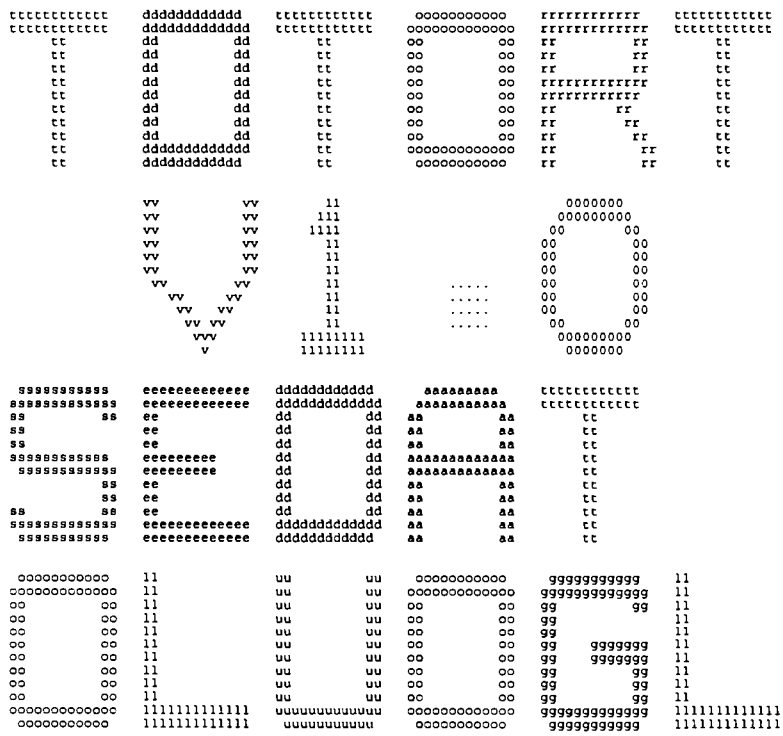
# B.4 Output Files

**transin**

This file contains the information read from the input. In addition to giving a detailed input description, it contains information about the workstation on which the problem is executed. A sample output file is as follows:

```
**************************************************
  Execution started on Tue Dec 31 09:40:46 1996
   Workstation Information
 ------------------------
  Machine   name  = bodrum.engr.utk.edu
  Directory name  = /tnfs/ne2/codes/proj/output/16a3

**************************************************
```



```
 number of neutron energy groups =   2
 number of x intervals           =   114
 number of y intervals           =   1
 number of z intervals           =   1
 number of material zones        =   7
 **************************************************
 ****************** B L O C K  1 ******************
 **************************************************
```

```
 number of precursor groups       =    6
 number of sedat iterations       =    0
 number of tn calculations        =    17
 total number of tj calculations  =    63
 total number of tk calculations  =    52311
 output unit for fidas messages   =    22
 output flag =    1 (0=short,1=long,2=all)
 steady state fixed source flag   =    0
 adjoint calculation flag         =    1  (0 calculate,
                                          1 adjoint flux is at unit 26(tort
output varscl format),
                                          2 adjoint flux is at unit 39(i,f
format),
                                          3 use flat adjoint flux ))
 forward calculation flag         =    1  (0 calculate,
                                          1 forward flux is at unit 27(tort
output varscl format))
 shape flag                       =    0  (0 don't renormalize the shape,
                                          1 do)
 adjoint energy group flag        =    1  (0 don't do anything,
                                          1 reverse group ordering)
 Extrapolation flag               =    0 (0 don't,
                                         1 extrapolate but if negatives are
encountered don't extrapolate,
                                         2 extrapolate but if negatives are
encountered set them equal to 0)
 perturbation type                =    3 (0  linear perturbation#,
                                         1  step perturbation
                                         2  step with cross section file
                                         3  2 steps
                                         5  linear with cross section file
                                         6  step followed by linear
                                        92  do one forward to determine
k_eff and critical cross sections
                                            and quit (for 0,1,2 and 5)
                                        93  do one forward to determine
k_eff and critical cross sections
                                            and quit (for 3)
                                        96  do one forward to determine
k_eff and critical cross sections
                                            and quit (for 6)
 Second step perturbation occurs at time step   8
 Point kinetics method                =    1(0 B&G, 1 Henry)
 Delayed neutron source calc. method =    1(0 sedat, 1 direct integration)
 Method flag                          =    0 (1 = shape calculation
                                             0  =  Henry's  method  flux
calculation)
 Doppler feedback flag                =    0 (0=no, 1=yes)
0     4* array      6 entries read
0     5* array      6 entries read
0     6* array      2 entries read
0     7* array      2 entries read
0     8* array      4 entries read
0     9* array     17 entries read
0     0t
 ************************************************
 ****************** B L O C K  2 ******************
 ************************************************
     lambda         beta
 ---------------------------
```

```
 1.290000E-02   8.100000E-05
 3.110000E-02   6.870000E-04
 1.340000E-01   6.120000E-04
 3.310000E-01   1.138000E-03
 1.260000E+00   5.120000E-04
 3.210000E+00   1.700000E-04
 total beta = 3.200000E-03

  nu      =     2.43000
  powini  =   0. (Reactor power is normalized to number if not zero)
  rtol    =     1.00000E-02
  atol    =   0.
  inverse speeds for   2 groups follows
  1.85100E-09  1.08800E-08
0   10$ array     17 entries read
0   11$ array     17 entries read
0    0t
 ************************************************
 ****************** B L O C K  3 ******************
 ************************************************
   tn time      min. time     total tj       total tk
  interval       interval       calc.          calc.
 ------------------------------------------------
 1.00000E-07  1.00000E-09        10            100
 4.00000E-07  4.00000E-09        10            100
 1.50000E-06  1.50000E-08        10            100
 8.00000E-06  8.00000E-08        10            100
 4.00000E-05  4.00000E-07        10            100
 4.00000E-05  4.00000E-07        10            100
 1.00000E-05  1.00000E-07        10            100
 1.00001E-07  1.00001E-09        10            100
 9.00000E-07  9.00000E-09        10            100
 1.00000E-06  1.00000E-08        10            100
 1.00000E-06  1.00000E-08        10            100
 2.00000E-06  2.00000E-08        10            100
 2.00000E-06  2.00000E-08        10            100
 2.99999E-06  2.99999E-08        10            100
 5.00001E-06  5.00001E-08        10            100
 8.85000E-04  8.85000E-06        10            100
 9.00000E-03  9.00000E-05        10            100
 ------------------------------------------------
   End of container array        =   60533
   Scratch space available       =   1439467
   End of Total Space Used       =   64178
   End of Total Space Available  =   1500000
 ------------------------------------------------
0   61$ array     10 entries read
0    0t
```

**fort.50**

This file contains power vs. time information in two column format. The first entry on each line is time while the second is the total power. A sample fort.50 file is

as follows:

```
1.00000E-07    1.00789
2.00000E-07    1.01595
3.00000E-07    1.02416
4.00000E-07    1.03251
5.00000E-07    1.04102
6.00000E-07    1.04969
7.00000E-07    1.05852
8.00000E-07    1.06751
9.00000E-07    1.07667
1.00000E-06    1.08600
1.90000E-06    1.17670
2.80000E-06    1.27480
3.70000E-06    1.38096
4.60000E-06    1.49599
5.50000E-06    1.62061
6.40000E-06    1.75637
7.30000E-06    1.90370
8.20000E-06    2.06362
9.10000E-06    2.23721
1.00000E-05    2.42596
1.90000E-05    5.41208
2.80000E-05    11.9582
3.70000E-05    26.3407
4.60000E-05    58.0514
5.50000E-05    128.160
6.40000E-05    283.580
7.30000E-05    629.144
8.20000E-05    1399.76
9.10000E-05    3123.15
1.00000E-04    6988.49
1.00100E-04    6921.12
1.00200E-04    6912.38
1.00300E-04    6905.33
1.00400E-04    6898.33
1.00500E-04    6891.39
1.00600E-04    6884.50
1.00700E-04    6877.66
1.00800E-04    6870.89
1.00900E-04    6864.19
1.01000E-04    6857.55
2.40900E-04    1878.75
3.80800E-04    518.255
5.20700E-04    143.483
6.60600E-04    40.5017
8.00500E-04    12.2340
9.40400E-04    4.41320
1.08030E-03    2.23940
1.22020E-03    1.63475
1.36010E-03    1.46517
1.50000E-03    1.42010
2.35000E-03    1.40805
3.20000E-03    1.40739
4.05000E-03    1.40687
4.90000E-03    1.40645
5.75000E-03    1.40607
6.60000E-03    1.40570
7.45000E-03    1.40532
8.30000E-03    1.40495
9.15000E-03    1.40458
1.00000E-02    1.40420
```

**pshort**

"*pshort*" is the same as fort.50 file except that it contains a maximum of ten entries per shape calculation. This is useful when dealing with thousands of data points.

## pkparm

This file contains point kinetics parameters calculated from a shape calculation. A sample file is as follows:

```
time (sec.) reactivity  total beta  gen. time
0.000E+00   4.033E-07   3.200E-03   3.655E-07
1.000E-05   1.088E-03   3.200E-03   3.637E-07
1.000E-04   1.110E-03   3.200E-03   3.642E-07
9.999E-04   1.112E-03   3.200E-03   3.641E-07
1.000E-01   1.112E-03   3.200E-03   3.641E-07
5.000E+00   1.114E-03   3.200E-03   3.641E-07
```

## pkout

This file contains the output of all calculations performed by the code. A sample output file with output flag set equal to "0" follows:

```
=================================================================================================
 Initial power        =    0.392993 fissions/sec
 Initial norm.d power =   0. fissions/sec
 Initial amplitude    =   0.
 Initial shape        =   0. (volume and energy integrated)
 Initial eigenvalue   =   0.
=================================================================================================
  Initial constraint constant =     1.38072E-06
  New constraint constant     =     1.38072E-06
  % Difference                =   0.
=================================================================================================
 Initial power        =    0.392993 fissions/sec
 Initial norm.d power =   0. fissions/sec
 Initial amplitude    =     1.00000
 Initial shape        =     1.38072E-06 (volume and energy integrated)
 Initial eigenvalue     =   0.
=================================================================================================
  Reactivity                                  = 0.000000E+00  (0.000000E+00$)
  Effective Delayed Neutron Fractions         =   7.500E-03
  Effective Total Delayed Neutron Fraction = 7.5000E-03
  Weighted Precursor Densities                =   1.5452E+03
  Generation Time                             = 6.067285E-05
=================================================================================================
 initial amplitude              =     1.00000
 initial normalized amplitude   =     1.00000
 initial weighted precursors  2....  c(  1) =     1545.1719970703
 ***********************************
  rtol =     9.9999994396249D-11 at time interval   1
  atol =   0. at time interval   1
 ***********************************
  Initial constraint constant =     1.38072E-06
  New constraint constant     =     1.38072E-06
  % Difference                =   0.
 ***********************************
  number of steps used in qtrap =   5
 ***********************************
  amplitude derivative at     1.00000E-04 =     24.3902 by delta (P) / delta (t)
  amplitude derivative at     1.00000E-04 =     24.3423 by LSODE
  Initial constraint constant =     1.38072E-06
  New constraint constant     =     1.38355E-06
  % Difference                =    0.204793
 ***********************************
  number of steps used in qtrap =   5
 ***********************************
=================================================================================================
 power              =    0.393349 fissions/sec
 relative power     =     1.00091
 amplitude          =     1.00205
 relative amplitude =     1.00205
 shape              =     1.38072E-06 (volume and energy     integrated)
 eigenvalue         =     1.62067
=================================================================================================
```

```
 Time = 0.000000E+00      Iteration Number =  1
 Reactivity                                = 1.502719E-03  (2.003625E-01$)
 Effective Delayed Neutron Fractions       =   7.500E-03
 Effective Total Delayed Neutron Fraction =  7.500E-03
 Generation Time                           =  6.058E-05
================================================================================
================================================================================
 Time = 1.000000E-05      Iteration Number =  1
 Reactivity                                = 1.502719E-03  (2.003624E-01$)
 Effective Delayed Neutron Fractions       =   7.500E-03
 Effective Total Delayed Neutron Fraction =  7.500E-03
 Generation Time                           =  6.058E-05
================================================================================
================================================================================
 Time = 2.000000E-05      Iteration Number =  1
 Reactivity                                = 1.502719E-03  (2.003625E-01$)
 Effective Delayed Neutron Fractions       =   7.500E-03
 Effective Total Delayed Neutron Fraction =  7.500E-03
 Generation Time                           =  6.058E-05
================================================================================
================================================================================
 Time = 3.000000E-05      Iteration Number =  1
 Reactivity                                = 1.502719E-03  (2.003625E-01$)
 Effective Delayed Neutron Fractions       =   7.500E-03
 Effective Total Delayed Neutron Fraction =  7.500E-03
 Generation Time                           =  6.058E-05
================================================================================
================================================================================
 Time = 4.000001E-05      Iteration Number =  1
 Reactivity                                = 1.502718E-03  (2.003624E-01$)
 Effective Delayed Neutron Fractions       =   7.500E-03
 Effective Total Delayed Neutron Fraction =  7.500E-03
 Generation Time                           =  6.058E-05
================================================================================
================================================================================
 Time = 5.000001E-05      Iteration Number =  1
 Reactivity                                = 1.502719E-03  (2.003625E-01$)
 Effective Delayed Neutron Fractions       =   7.500E-03
 Effective Total Delayed Neutron Fraction =  7.500E-03
 Generation Time                           =  6.058E-05
================================================================================
================================================================================
 Time = 6.000001E-05      Iteration Number =  1
 Reactivity                                = 1.502718E-03  (2.003625E-01$)
 Effective Delayed Neutron Fractions       =   7.500E-03
 Effective Total Delayed Neutron Fraction =  7.500E-03
 Generation Time                           =  6.058E-05
================================================================================
================================================================================
 Time = 7.000002E-05      Iteration Number =  1
 Reactivity                                = 1.502718E-03  (2.003625E-01$)
 Effective Delayed Neutron Fractions       =   7.500E-03
 Effective Total Delayed Neutron Fraction =  7.500E-03
 Generation Time                           =  6.058E-05
================================================================================
================================================================================
 Time = 8.000001E-05      Iteration Number =  1
 Reactivity                                = 1.502719E-03  (2.003625E-01$)
 Effective Delayed Neutron Fractions       =   7.500E-03
 Effective Total Delayed Neutron Fraction =  7.500E-03
 Generation Time                           =  6.058E-05
================================================================================
================================================================================
 Time = 9.000002E-05      Iteration Number =  1
 Reactivity                                = 1.502718E-03  (2.003625E-01$)
 Effective Delayed Neutron Fractions       =   7.500E-03
 Effective Total Delayed Neutron Fraction =  7.500E-03
 Generation Time                           =  6.058E-05
================================================================================
================================================================================
 Time = 1.000000E-04      Iteration Number =  1
 Reactivity                                = 1.502718E-03  (2.003625E-01$)
 Effective Delayed Neutron Fractions       =   7.500E-03
 Effective Total Delayed Neutron Fraction =  7.500E-03
 Generation Time                           =  6.058E-05
================================================================================
 ***********************************
 number of steps used in qtrap =   5
 ***********************************
 ***********************************
 number of steps used in qtrap =   5
 ***********************************
================================================================================
power              =    0.393349 fissions/sec
relative power     =     1.00091
amplitude          =     1.00205
relative amplitude =     1.00205
shape              =     1.38072E-06 (volume and energy     integrated)
eigenvalue         =     1.62067
 ***********************************
 rtol =     9.8999990427728D-09 at time interval   2
 atol =   0. at time interval   2
 ***********************************
 Initial constraint constant =     1.38072E-06
 New constraint constant     =     1.38072E-06
 % Difference                =   0.
 ***********************************
 number of steps used in qtrap =   5
 ***********************************
 amplitude derivative at     9.99999E-03 =     9.61766 by delta (P) / delta (t)
```

```
 amplitude derivative at     9.99999E-03 =     8.36498 by LSODE
 Initial constraint constant =     1.38072E-06
 New constraint constant     =     1.34563E-06
 % Difference                =    -2.54127
 WARNING : Constraint Integral changed too much
*************************************
 number of steps used in qtrap =   5
*************************************
=================================================================================================
power              =    0.393568 fissions/sec
relative power     =     1.00146
amplitude          =    0.974587
relative amplitude =    0.974587
shape              =     1.38072E-06 (volume and energy     integrated)
eigenvalue         =     76.7021
=================================================================================================
 Time = 1.090000E-03      Iteration Number =  1
 Reactivity                                = 1.502718E-03  (2.003625E-01$)
 Effective Delayed Neutron Fractions       =   7.500E-03
 Effective Total Delayed Neutron Fraction =  7.500E-03
 Generation Time                           =  6.058E-05
=================================================================================================
=================================================================================================
 Time = 2.080000E-03      Iteration Number =  1
 Reactivity                                = 1.502718E-03  (2.003624E-01$)
 Effective Delayed Neutron Fractions       =   7.500E-03
 Effective Total Delayed Neutron Fraction =  7.500E-03
 Generation Time                           =  6.058E-05
=================================================================================================
=================================================================================================
 Time = 3.070001E-03      Iteration Number =  1
 Reactivity                                = 1.502719E-03  (2.003625E-01$)
 Effective Delayed Neutron Fractions       =   7.500E-03
 Effective Total Delayed Neutron Fraction =  7.500E-03
 Generation Time                           =  6.058E-05
=================================================================================================
=================================================================================================
 Time = 4.060001E-03      Iteration Number =  1
 Reactivity                                = 1.502719E-03  (2.003625E-01$)
 Effective Delayed Neutron Fractions       =   7.500E-03
 Effective Total Delayed Neutron Fraction =  7.500E-03
 Generation Time                           =  6.058E-05
=================================================================================================
=================================================================================================
 Time = 5.050002E-03      Iteration Number =  1
 Reactivity                                = 1.502719E-03  (2.003625E-01$)
 Effective Delayed Neutron Fractions       =   7.500E-03
 Effective Total Delayed Neutron Fraction =  7.500E-03
 Generation Time                           =  6.058E-05
=================================================================================================
=================================================================================================
 Time = 6.040002E-03      Iteration Number =  1
 Reactivity                                = 1.502718E-03  (2.003625E-01$)
 Effective Delayed Neutron Fractions       =   7.500E-03
 Effective Total Delayed Neutron Fraction =  7.500E-03
 Generation Time                           =  6.058E-05
=================================================================================================
=================================================================================================
 Time = 7.030003E-03      Iteration Number =  1
 Reactivity                                = 1.502718E-03  (2.003625E-01$)
 Effective Delayed Neutron Fractions       =   7.500E-03
 Effective Total Delayed Neutron Fraction =  7.500E-03
 Generation Time                           =  6.058E-05
=================================================================================================
=================================================================================================
 Time = 8.020003E-03      Iteration Number =  1
 Reactivity                                = 1.502719E-03  (2.003625E-01$)
 Effective Delayed Neutron Fractions       =   7.500E-03
 Effective Total Delayed Neutron Fraction =  7.500E-03
 Generation Time                           =  6.058E-05
=================================================================================================
=================================================================================================
 Time = 9.010002E-03      Iteration Number =  1
 Reactivity                                = 1.502719E-03  (2.003625E-01$)
 Effective Delayed Neutron Fractions       =   7.500E-03
 Effective Total Delayed Neutron Fraction =  7.500E-03
 Generation Time                           =  6.058E-05
=================================================================================================
=================================================================================================
 Time = 9.999998E-03      Iteration Number =  1
 Reactivity                                = 1.502718E-03  (2.003625E-01$)
 Effective Delayed Neutron Fractions       =   7.500E-03
 Effective Total Delayed Neutron Fraction =  7.500E-03
 Generation Time                           =  6.058E-05
=================================================================================================
*************************************
 number of steps used in qtrap =   5
*************************************
*************************************
 number of steps used in qtrap =   5
*************************************
=================================================================================================
power              =    0.393568 fissions/sec
relative power     =     1.00146
amplitude          =    0.974587
relative amplitude =    0.974587
shape              =     1.38072E-06 (volume and energy     integrated)
eigenvalue         =     76.7021
*************************************
 rtol =     9.8999998954241D-07 at time interval   3
```

```
 atol =   0. at time interval   3
 ***********************************
 Initial constraint constant =     1.38072E-06
 New constraint constant     =     1.38072E-06
 % Difference                =   0.
 ***********************************
 number of steps used in qtrap =   5
 ***********************************
 amplitude derivative at    0.999999 =     2.54759E-02 by delta (P) / delta (t)
 amplitude derivative at    0.999999 =     2.55509E-02 by LSODE
 Initial constraint constant =     1.38072E-06
 New constraint constant     =     1.39910E-06
 % Difference                =     1.33067
 WARNING : Constraint Integral changed too much
 ***********************************
 number of steps used in qtrap =   5
 ***********************************
================================================================================================================
power              =    0.393584 fissions/sec
relative power     =     1.00151
amplitude          =     1.01331
relative amplitude =     1.01331
shape              =     1.38072E-06 (volume and energy     integrated)
eigenvalue         =     154.879
================================================================================================================
  Time = 1.090000E-01      Iteration Number =  1
  Reactivity                                 = 1.502718E-03  (2.003625E-01$)
  Effective Delayed Neutron Fractions        =   7.500E-03
  Effective Total Delayed Neutron Fraction =  7.500E-03
  Generation Time                            =  6.058E-05
================================================================================================================
================================================================================================================
  Time = 2.080000E-01      Iteration Number =  1
  Reactivity                                 = 1.502719E-03  (2.003625E-01$)
  Effective Delayed Neutron Fractions        =   7.500E-03
  Effective Total Delayed Neutron Fraction =  7.500E-03
  Generation Time                            =  6.058E-05
================================================================================================================
================================================================================================================
  Time = 3.070000E-01      Iteration Number =  1
  Reactivity                                 = 1.502718E-03  (2.003625E-01$)
  Effective Delayed Neutron Fractions        =   7.500E-03
  Effective Total Delayed Neutron Fraction =  7.500E-03
  Generation Time                            =  6.058E-05
================================================================================================================
================================================================================================================
  Time = 4.060001E-01      Iteration Number =  1
  Reactivity                                 = 1.502719E-03  (2.003625E-01$)
  Effective Delayed Neutron Fractions        =   7.500E-03
  Effective Total Delayed Neutron Fraction =  7.500E-03
  Generation Time                            =  6.058E-05
================================================================================================================
================================================================================================================
  Time = 5.050001E-01      Iteration Number =  1
  Reactivity                                 = 1.502718E-03  (2.003625E-01$)
  Effective Delayed Neutron Fractions        =   7.500E-03
  Effective Total Delayed Neutron Fraction =  7.500E-03
  Generation Time                            =  6.058E-05
================================================================================================================
================================================================================================================
  Time = 6.040001E-01      Iteration Number =  1
  Reactivity                                 = 1.502719E-03  (2.003625E-01$)
  Effective Delayed Neutron Fractions        =   7.500E-03
  Effective Total Delayed Neutron Fraction =  7.500E-03
  Generation Time                            =  6.058E-05
================================================================================================================
================================================================================================================
  Time = 7.029998E-01      Iteration Number =  1
  Reactivity                                 = 1.502719E-03  (2.003625E-01$)
  Effective Delayed Neutron Fractions        =   7.500E-03
  Effective Total Delayed Neutron Fraction =  7.500E-03
  Generation Time                            =  6.058E-05
================================================================================================================
================================================================================================================
  Time = 8.019996E-01      Iteration Number =  1
  Reactivity                                 = 1.502718E-03  (2.003624E-01$)
  Effective Delayed Neutron Fractions        =   7.500E-03
  Effective Total Delayed Neutron Fraction =  7.500E-03
  Generation Time                            =  6.058E-05
================================================================================================================
================================================================================================================
  Time = 9.009993E-01      Iteration Number =  1
  Reactivity                                 = 1.502719E-03  (2.003625E-01$)
  Effective Delayed Neutron Fractions        =   7.500E-03
  Effective Total Delayed Neutron Fraction =  7.500E-03
  Generation Time                            =  6.058E-05
================================================================================================================
================================================================================================================
  Time = 9.999990E-01      Iteration Number =  1
  Reactivity                                 = 1.502719E-03  (2.003625E-01$)
  Effective Delayed Neutron Fractions        =   7.500E-03
  Effective Total Delayed Neutron Fraction =  7.500E-03
  Generation Time                            =  6.058E-05
================================================================================================================
 ***********************************
 number of steps used in qtrap =   5
 ***********************************
 ***********************************
 number of steps used in qtrap =   5
 ***********************************
```

```
power              =    0.393584 fissions/sec
relative power     =     1.00151
amplitude          =     1.01331
relative amplitude =     1.01331
shape              =     1.38072E-06 (volume and energy     integrated)
eigenvalue         =     154.879
 ***********************************
 rtol =     9.0000003183377D-06 at time interval   4
 atol =   0. at time interval   4
 ***********************************
 Initial constraint constant =     1.38072E-06
 New constraint constant     =     1.38072E-06
 % Difference                =     1.64677E-05
 ***********************************
 number of steps used in qtrap =   5
 ***********************************
 amplitude derivative at    10.00001 =     3.02633E-02 by delta (P) / delta (t)
 amplitude derivative at    10.00001 =     3.13534E-02 by LSODE
 Initial constraint constant =     1.38072E-06
 New constraint constant     =     1.94970E-06
 % Difference                =     41.2086
 WARNING : Constraint Integral changed too much
 ***********************************
 number of steps used in qtrap =   5
 ***********************************
power              =    0.393584 fissions/sec
relative power     =     1.00151
amplitude          =     1.41209
relative amplitude =     1.41209
shape              =     1.38072E-06 (volume and energy     integrated)
eigenvalue         =     156.081
  Time = 1.899999E+00      Iteration Number =  1
  Reactivity                                 = 1.502719E-03  (2.003625E-01$)
  Effective Delayed Neutron Fractions        =   7.500E-03
  Effective Total Delayed Neutron Fraction =  7.500E-03
  Generation Time                            =  6.058E-05
  Time = 2.799999E+00      Iteration Number =  1
  Reactivity                                 = 1.502718E-03  (2.003624E-01$)
  Effective Delayed Neutron Fractions        =   7.500E-03
  Effective Total Delayed Neutron Fraction =  7.500E-03
  Generation Time                            =  6.058E-05
  Time = 3.699998E+00      Iteration Number =  1
  Reactivity                                 = 1.502719E-03  (2.003624E-01$)
  Effective Delayed Neutron Fractions        =   7.500E-03
  Effective Total Delayed Neutron Fraction =  7.500E-03
  Generation Time                            =  6.058E-05
  Time = 4.599998E+00      Iteration Number =  1
  Reactivity                                 = 1.502718E-03  (2.003624E-01$)
  Effective Delayed Neutron Fractions        =   7.500E-03
  Effective Total Delayed Neutron Fraction =  7.500E-03
  Generation Time                            =  6.058E-05
  Time = 5.499999E+00      Iteration Number =  1
  Reactivity                                 = 1.502718E-03  (2.003624E-01$)
  Effective Delayed Neutron Fractions        =   7.500E-03
  Effective Total Delayed Neutron Fraction =  7.500E-03
  Generation Time                            =  6.058E-05
  Time = 6.400000E+00      Iteration Number =  1
  Reactivity                                 = 1.502718E-03  (2.003624E-01$)
  Effective Delayed Neutron Fractions        =   7.500E-03
  Effective Total Delayed Neutron Fraction =  7.500E-03
  Generation Time                            =  6.058E-05
  Time = 7.300002E+00      Iteration Number =  1
  Reactivity                                 = 1.502719E-03  (2.003625E-01$)
  Effective Delayed Neutron Fractions        =   7.500E-03
  Effective Total Delayed Neutron Fraction =  7.500E-03
  Generation Time                            =  6.058E-05
  Time = 8.200003E+00      Iteration Number =  1
  Reactivity                                 = 1.502719E-03  (2.003625E-01$)
  Effective Delayed Neutron Fractions        =   7.500E-03
  Effective Total Delayed Neutron Fraction =  7.500E-03
  Generation Time                            =  6.058E-05
  Time = 9.100004E+00      Iteration Number =  1
  Reactivity                                 = 1.502719E-03  (2.003625E-01$)
  Effective Delayed Neutron Fractions        =   7.500E-03
  Effective Total Delayed Neutron Fraction =  7.500E-03
  Generation Time                            =  6.058E-05
  Time = 1.000001E+01      Iteration Number =  1
  Reactivity                                 = 1.502719E-03  (2.003625E-01$)
  Effective Delayed Neutron Fractions        =   7.500E-03
```

```
 Effective Total Delayed Neutron Fraction =  7.500E-03
 Generation Time                          =  6.058E-05
=================================================================================================

Problem elapsed time     0 min. 58 seconds

 CPU time (sec.)      1.438 ; user time (sec.)      0.762 ; system time (sec.)     0.676
```

# B.5 TDTORT Code Listing

The following source listing of TDTORT includes only new source coding created in this work and does not include the source listing of the TORT code or LSODE.

```
::::::::::::::
ampil.f
::::::::::::::
      subroutine ampil(shape,phiadj,vinv,index,index1,igm,iqs,ir
     $ ,vr,dth,dz,im,jm,km,amplit,shap,indexm,mm,weight,it,n6pk,itn)
      real*4 shape(indexm),phiadj(indexm),vinv(igm),vr(im)
     $ ,dth(jm),dz(km),amplit,shap,weight(320),ampl,vol,shapl
c
      include 'comsdc.F'
c
      real*4 zero
      zero = 0.e0
c
c    if called after extrapolation return
c
      if(ir.eq.0.and.itn.ge.1) return
c
      constnew = zero
      do 55 m=1,mm
      ampl = zero
      do 510 ig=1, igm
      do 520 k =1,km
      do 530 j =1,jm
      do 540 i =1,im
       vol = vr(i)*dth(j)*dz(k)
       i1 = i + im*((j-1) + jm*((k-1) + km*((ig-1)+igm*(m-1))))
c    from eq 5 of Henry paper, eq. 9.6 of B&G
       ampl = ampl + phiadj(i1)*shape(i1)*vol*vinv(ig)
540    continue
530    continue
520    continue
510    continue
      constnew = constnew + ampl*weight(m)
55     continue
c
c    compute shape
c set initial amplitute equal to 1. and constraint const. to constnew
c
      if(it.eq.0) const = constnew
      itx = itn
      if(it.eq.0) itx = 1
      write(n6pk,*) ' Initial constraint constant = ',const
      write(n6pk,*) ' Previous constraint constant = ',ratio(itx)*const
      write(n6pk,*) ' New      constraint constant = ',constnew
       perc = (constnew-ratio(itn)*const)/constnew*100.
      write(n6pk,*) ' % Difference                = ',perc
      if(abs(perc).gt.1.)
     $ write(n6pk,*) ' WARNING : Constraint Integral changed too much'
      ratio(itn+1)  = constnew/const   ! initial is at pos 1
      amplit = constnew/const
      do 550 i=1,indexm
       shape(i) = shape(i)/ratio(itn+1)
550    continue
c
c    check volume integrated shape
      shap = zero
      do 105 m=1,mm
      shapl = zero
      do 110 ig=1,igm
      do 120 k =1,km
      do 130 j =1,jm
      do 140 i =1,im
       vol = vr(i)*dth(j)*dz(k)
       i1 = i + im*((j-1) + jm*((k-1) + km*((ig-1)+igm*(m-1))))
       shapl = shapl + phiadj(i1)*shape(i1)*vol*vinv(ig)
140   continue
130   continue
120   continue
110   continue
      shap = shap + shapl*weight(m)
105   continue
```

```
c
      return
      end
::::::::::::::
ampilh.f
::::::::::::::
      subroutine ampilh(shape,phiadj,vinv,index,index1,igm,iqs,ir
     $ ,vr,dth,dz,im,jm,km,amplit,shap,it,n6pk,itn)
      real*4 shape(index),phiadj(index),vinv(igm),vr(im)
     $ ,dth(jm),dz(km),amplit,shap,vol
c
      include 'comsdc.F'
c
      real*4 zero
      zero = 0.e0
c
c    if called after extrapolation return
c
      if(ir.eq.0.and.itn.ge.1) return
c
      constnew = zero
      do 510 ig=1, igm
      do 520 k =1,km
      do 530 j =1,jm
      do 540 i =1,im
       vol = vr(i)*dth(j)*dz(k)
       i1 = i + im*((j-1) + jm*((k-1) + km*(ig-1)))
c     from eq 5 of Henry paper, eq. 9.6 of B&G
       constnew = constnew + phiadj(i1)*shape(i1)*vol*vinv(ig)
540    continue
530    continue
520    continue
510    continue
c
c    compute shape
c set initial amplitute equal to 1. and constraint const. to constnew
c
      if(it.eq.0) const = constnew
      itx = itn
      if(it.eq.0) itx = 1
      write(n6pk,*) ' Initial  constraint constant = ',const
      write(n6pk,*) ' Previous constraint constant = ',ratio(itx)*const
      write(n6pk,*) ' New      constraint constant = ',constnew
       perc = (constnew-ratio(itx)*const)/constnew*100.
      write(n6pk,*) ' % Difference                 = ',perc
      if(abs(perc).gt.1.)
     $ write(n6pk,*) ' WARNING : Constraint Integral changed too much'
      ratio(itn+1)  = constnew/const  ! initial is at pos 1
      amplit = constnew/const
      if(it.ne.0) then
      do 550 i=1,index
       shape(i) = shape(i)/ratio(itn+1)
550    continue
      endif
c
c    check volume integrated shape
      shap = zero
      do 110 ig=1,igm
      do 120 k =1,km
      do 130 j =1,jm
      do 140 i =1,im
       vol = vr(i)*dth(j)*dz(k)
       i1 = i + im*((j-1) + jm*((k-1) + km*(ig-1)))
       shap = shap + phiadj(i1)*shape(i1)*vol*vinv(ig)
140   continue
130   continue
120   continue
110   continue
c
      return
      end
::::::::::::::
avpedit.f
::::::::::::::
      subroutine avpedit
      character*10 infile
      dimension tedit(100)
      dimension iedit(100)
c      write(6,'(" Input filename = ",$)')
c      read(5,*) infile
      infile ='avpinp'
      open(20,file=infile,status='old')
      open(21,file='avp',status='old')
c      open(22,file='avp.OUT',status='unknown')
      open(22,file='dene.dat',status='unknown')
      open(23,file='skips',status='old')
      read(23,*) iskp
      open(25,file='delete',status='unknown')
      read(20,*) maxedit
      do i=1,maxedit
      read(20,*) tedit(i),iedit(i)
      enddo
c
      icx = 0
      do i=1,maxedit
      write(6,*) ' processing ',i
      ic = icx
20      read(21,*,end=30) time,power
      ic = ic + 1
      goto 25
```

```
30        time = tedit(i)+1
        ic = ic +1
25        if(time.gt.tedit(i)) then
        write(6,*) ' processing ',i
        ic = ic -1
        rewind(25)
        ics = iedit(i)*10
        write(6,*) 'ic= ',ic,'ics= ',ics
c
50        do ix2=1,10
        read(25,*,end=60) timex,powerx
        if(ix2.eq.iskp) then
        write(22,*) timex,powerx
        endif
        enddo
c skip next iedit sets
        do ix=1,ics
        read(25,*,end=60) timex,powerx
        enddo
        goto 50
60        continue
c
        close(25,status='delete')
        open(25,file='delete',status='unknown')
        write(25,*) time,power
        icx = 1
        goto 10
        endif
        write(25,*) time,power
        goto 20
10        continue
        enddo
c
        end
::::::::::::::
avpowd.f
::::::::::::::
      subroutine avpowd(powxt,index1,vr,dth,dz,im,jm,km,itn,itj,time
     $ ,xsfp,index,irep,irepmx,powzt,volz,mtm)
c
      real*4 powxt(index1),vr(im),dth(jm),dz(km),vol,voltot,powdav,time
     $ ,xsfp(index),powzt(mtm),volz(mtm)
      parameter(eps=3.204e-11)
      real*4 zero
      zero = 0.e0
c
c   xsfp is used only to determine whether the zone contibutes
c   to the total power
c
      if(itn.eq.1.and.itj.eq.1) then
       open(35,file='avpowd',status='unknown')
       open(36,file='avpowdz',status='unknown')
      else
       open(35,file='avpowd',status='old',access='append')
       open(36,file='avpowdz',status='old',access='append')
      endif
c
      voltot = zero
      powdav = zero
      do k=1,km
      do j=1,jm
      do i=1,im
       i1 = i + im*((j-1) + jm*(k-1))
       if(xsfp(i1).gt.0.0) then
        vol = vr(i)*dth(j)*dz(k)
        voltot = voltot + vol
c
c   powxt has energy conversion factor in it
c
         powdav = powdav +powxt(i1)*vol
       endif
      enddo
      enddo
      enddo
      powdav = powdav/voltot    ! power is for 1/4 core
      write(35,*) time,4.0*powdav,itn,itj,voltot
      close(35)
c
      powdavz = zero
      voltz = zero
      do i=1,mtm
        if(powzt(i).gt.0.) then
          voltz = voltz + volz(i)
          powdavz = powdavz + powzt(i)*volz(i)
        endif
      enddo
      powdavz = powdavz / voltz
      write(36,*) time,4.0*powdavz,itn,itj,voltz
      close(36)
c
      if(irep.eq.irepmx) then
      if(itn.eq.1.and.itj.eq.1) then
       open(35,file='avp',status='unknown')
       open(36,file='avpz',status='unknown')
      else
       open(35,file='avp',status='old',access='append')
       open(36,file='avpz',status='old',access='append')
      endif
      write(35,*) time,4.0*powdav,itn,itj
      close(35)
```

```
      write(36,*) time,4.0*powdavz,itn,itj
      close(36)
      endif
c
      return
      end
::::::::::::::
avtemp.f
::::::::::::::
      subroutine avtemp(tempold,index1,vr,dth,dz,im,jm,km,itn,itj,time
     $ ,xsfp,index,irep,irepmx,tempiz,mtm,volz,powzt)
c
      real*4 tempold(index1),vr(im),dth(jm),dz(km),vol,voltot,tempav
     $ ,time,xsfp(index),tempiz(mtm),volz(mtm),powzt(mtm)
      real*4 zero
      zero = 0.e0
c
c   xsfp is used only to determine whether the zone contibutes
c   to the total power and therefore to av. temp
c
      if(itn.eq.1.and.itj.eq.1) then
       open(35,file='avtemp',status='unknown')
       open(36,file='avtempz',status='unknown')
      else
       open(35,file='avtemp',status='old',access='append')
       open(36,file='avtempz',status='old',access='append')
      endif
c
      voltot = zero
      tempav = zero
      do k=1,km
      do j=1,jm
      do i=1,im
       i1 = i + im*((j-1) + jm*(k-1))
       if(xsfp(i1).gt.0.0) then
        vol = vr(i)*dth(j)*dz(k)
        voltot = voltot + vol
        tempav = tempav +tempold(i1)*vol
       endif
      enddo
      enddo
      enddo
      tempav = tempav/voltot
      write(35,*) time,tempav,itn,itj
      close(35)
c
      do i=1,mtm
        if(powzt(i).gt.0.) then
          tempavz = tempavz + tempiz(i)*volz(i)
        endif
      enddo
      tempavz = tempavz / voltot
      write(36,*) time,tempavz,itn,itj
      close(36)
c
      if(irep.eq.irepmx) then
        if(itn.eq.1.and.itj.eq.1) then
          open(35,file='avt',status='unknown')
          open(36,file='avtz',status='unknown')
        else
          open(35,file='avt',status='old',access='append')
          open(36,file='avtz',status='old',access='append')
        endif
        write(35,*) time,tempav,itn,itj
        close(35)
        write(36,*) time,tempavz,itn,itj
        close(36)
      endif
c
      return
      end
::::::::::::::
betcal.f
::::::::::::::
      subroutine betcal(betaef,fterm,idim,dume1,shape,xsfp,dume2
     $ ,scadj,vr,dth,dz,index,index1,im,jm,km,igm)
c     betaef is batasubi(t)
c     fterm is F(t)
c
      include 'comsd1.F'
c
      real*4 betaef(idim),fterm,dume1(index1),shape(index),xsfp(index)
     $ ,dume2(index),scadj(index),vr(im),dth(jm),dz(km),res
c
c     start delayed group loop
c
      do 100 idelay=1,ndg
c
c     clear space
      call clearx( dume1,1,index1)
      call clearx( dume2,1,index)
c
c    perform the integral over all other energies first
c    and load the result onto dume1
      call gtermr(dume1,shape,xsfp,index,index1,igm)
c     note gofr does not depend on energy
c
c     multiply by delayed spectrum,betai and adjoint flux
c     and load onto dume2
      do 20 ig=1,igm
```

```
      do 30 i=1,index1
       i1    = i + index1*(ig-1)
       dume2(i1) = dume1(i)*chid(ig)*scadj(i1)*beta(idelay)
30    continue
20    continue
c
c    perform integration over volume and energy
c
      call vgintg(dume2,vr,dth,dz,index,igm,im,jm,km,res)
      betaef(idelay) = res / fterm
100   continue
c
c
      return
      end
::::::::::::::
chitr.f
::::::::::::::
      subroutine chitr(chit, igmw, betat, chid, beta, ndg, chi)
      real*4 chit(igmw),betat,chid(100),beta(ndg),chi(igmw)
      real*4 zero
      zero = 0.e0
      do 10 ig=1, igmw
      chit(ig) = zero
      do 20 in=1, ndg
      chit(ig) = chit(ig) + chid(ig)*beta(in)
20    continue
      chit(ig) = chit(ig) + chi(ig)*(1.-betat)
10    continue
c  overwrite
      return
      do ig=1,igmw
        chit(ig) = chi(ig)
      enddo
c
      return
      end
::::::::::::::
cigint.f
::::::::::::::
      subroutine cigint(flux,beta,alambda,ndg,igm,index1,cig0
     $ ,cig1,index,xsfs)
      real*4 flux(index),beta(10),alambda(10),cig0(index1,ndg)
     $ ,cig1(index1,ndg),xsfs(index),sum
      real*4 zero
      zero = 0.e0
c   this routine calculates initial steady-state precursor densities cig0
c
      do 10 j=1,ndg
      do 20 in=1,index1
       sum = zero
       do igp=1,igm
        i1 = in + index1*(igp-1)
        sum = sum + xsfs(i1)*flux(i1)
       enddo
       cig0(in,j) = beta(j)/alambda(j)*sum
       cig1(in,j) = beta(j)/alambda(j)*sum
20    continue
10    continue
c
      return
      end
::::::::::::::
clearx.f
::::::::::::::
      subroutine clearx (a,n1,n2)
      real*4 a(1)
      real*4 zero
      zero = 0.e0
c
c - - - - - sets (a(i),i=n1,n2) to 0 if n1.le.n2
      if(n2.lt.n1) go to 9900
      do 10 n=n1,n2
   10 a(n) = zero
 9900 return
      end
::::::::::::::
delsor2.f
::::::::::::::
      subroutine delsor2(cig1,cig2,alambda,beta,index,index1,ndg
     $ ,igmw,xsfp,xsfs,shap1,shap2,amp,ldamp,idtj,dume1,dume2,dtn,dtnx
     $ ,amp0,itn,irestx,it,dtj,xsf0,xsf1,ihtw,mtmw,izmw,z1c,cp,imw,jmw
     $ ,kmw,ihpw,tx0,tx1,ampold,indexn,dume,izart,n6pk)
c
      real*4 cig1(indexn),cig2(indexn)
     $ ,alambda(10),beta(10),xsfp(index),xsfs(index)
     $ ,shap1(index),shap2(index),amp(ldamp+1)
     $ ,dume1(2*index),dume2(2*index*ndg),dtn(2000),dtnx,amp0,dtj
     $ ,xsf0(izart),xsf1(izart),z1c(index1),cp(izmw)
     $ ,tx0,tx1,ampold,dume(irestx)
      dimension idtj(2000)
c
      include 'comsdc.F'
c
      real*4 zero
       zero = 0.e0
c  sets up arrays to be used in qtstrp
c  can be conbined with delsor5 by adjusting tx0 and tx1
c  to do what delsor5 does
c
```

```
      l4 = 1
      l5 = l4 + index1
      l6 = l5 + index
      l7 = l6 + index
      lt1 = l7 + idtj(itn) + 1
      lt2 = lt1 + index
      lt3 = lt2 + index
c     send unnormalized shapes
      open(35,file='shapet',status='old',form='unformatted')
c    skip unneded records
      do i1 = 2 , itn
      read(35)
      enddo
      if(it.eq.0) then
        read(35) (dume(lt1+i-1),i=1,index)
        do i=1,index
         dume(lt2+i-1) = dume(lt1+i-1)
        enddo
      elseif(it.eq.1) then
        read(35) (dume(lt1+i-1),i=1,index)
        read(35) (dume(lt2+i-1),i=1,index)
      endif
      close(35)
c     send unnormalized shape,   another way
c       do i=1,index
c          dume(lt2+i-1) = shap1(i)*ratio(itn+1)
c          dume(lt1+i-1) = shap2(i)*ratio(itn)
c       enddo
c
c     send normalized shape
       do i=1,index
          dume(lt2+i-1) = shap1(i)
          dume(lt1+i-1) = shap2(i)
       enddo
c
       call qtstrp2(idtj(itn),dtj,index,alambda,amp(2),ndg,xsf0,xsf1
     $ ,dume(lt2),dume(lt1),index1,igmw,dume(l4),cig1,beta,cig2
     $ ,dume(l5),dume(l6),ihtw-1,mtmw,izmw,z1c,cp,imw,jmw,kmw,ihpw
     $ ,tx0,tx1,ampold,dume(l7),dtn(itn),jlast)
c
       write(n6pk,*) '***********************************'
       write(n6pk,*) ' number of steps used in qtrap = ',jlast
       write(n6pk,*) '***********************************'
c
c
c       call wout(cig1,indexn,6,1)
c
       return
       end
c234567890123456789012345678901234567890123456789012345678901234567890123456789012
:::::::::::::::
delsor6.f
:::::::::::::::
       subroutine delsor6(cig1,cig2,alambda,beta,index,index1,ndg
     $ ,igmw,xsfp,xsfs,shap1,shap2,amp,ldamp,idtj,dume1,dume2,dtn,dtnx
     $ ,amp0,itn,irestx,it,dtj,xsf0,xsf1,ihtw,mtmw,izmw,z1c,cp,imw,jmw
     $ ,kmw,ihpw,tx0,tx1,ampold,tltot,indexn,dume,izart,n6pk
     $ ,xsf2,mz,mzm)
c
       real*4 cig1(indexn),cig2(indexn)
     $ ,alambda(10),beta(10),xsfp(index),xsfs(index)
     $ ,shap1(index),shap2(index),amp(ldamp+1)
     $ ,dume1(2*index),dume2(2*index*ndg),dtn(2000),dtnx,amp0,dtj
     $ ,xsf0(izart),xsf1(izart),z1c(index1),cp(izmw)
     $ ,tx0,tx1,ampold,tltot,dume(irestx),xsf2(izart)
       dimension idtj(2000),mz(10)
c
       real*4 zero
        zero = 0.e0
c  sets up arrays to be used in qtstrp
c  can be conbined with delsor5 by adjusting tx0 and tx1
c  to do what delsor5 does
c
       l4 = 1
       l5 = l4 + index1
       l6 = l5 + index
       l7 = l6 + index
       lt1 = l7 + idtj(itn) + 1
       lt2 = lt1 + index
       lt3 = lt2 + index
c     send unnormalized shapes
       open(35,file='shapet',status='old',form='unformatted')
c    skip unneded records
       do i1 = 2 , itn
       read(35)
       enddo
       if(it.eq.0) then
         read(35) (dume(lt1+i-1),i=1,index)
         do i=1,index
          dume(lt2+i-1) = dume(lt1+i-1)
         enddo
       elseif(it.eq.1) then
         read(35) (dume(lt1+i-1),i=1,index)
         read(35) (dume(lt2+i-1),i=1,index)
       endif
       close(35)
c
c send normalized shape
c       do i=1,index
c          dume(lt2+i-1) = shap1(i)
```

```
c         dume(lt1+i-1) = shap2(i)
c       enddo
       call qtstrp6(idtj(itn),dtj,index,alambda,amp(2),ndg,xsf0,xsf1
     $ ,dume(lt2),dume(lt1),index1,igmw,dume(l4),cig1,beta,cig2
     $ ,xsf2,mzm,mz
     $ ,dume(l5),dume(l6),ihtw-1,mtmw,izmw,z1c,cp,imw,jmw,kmw,ihpw
     $ ,tx0,tx1,ampold,dume(l7),dtn(itn),jlast)
c
       write(n6pk,*) '***********************************'
       write(n6pk,*) ' number of steps used in qtrap = ',jlast
       write(n6pk,*) '***********************************'
c
c
c       call wout(cig1,indexn,6,1)
c
       return
       end
c23456789012345678901234567890123456789012345678901234567890123456789012
::::::::::::::
dopler.f
::::::::::::::
c23456789012345678901234567890123456789012345678901234567890123456789012
       subroutine dopler(xsts,xstp,xstd,xsap,index,igm,index1,temp)
c    make adjustment to zone xsecs,  called after dxsec or dxlsec
       real*4 xsts(index),xstp(index),xstd(index),xsap(index)
      $,temp(index1),gamma,sigap
c
       parameter(gamma = 2.034e-3)
c
       ig = 1     ! feedback for group 1 only
       do i =1,index1
       j = i + index1*(ig-1)
        sigap  = xsap(j)*(1.+gamma*(sqrt(temp(i)) - sqrt(300.0)))
        xstp(j) = xstp(j) - xsap(j) + sigap
        xstd(j) = xstp(j) - xsts(j)
       enddo
c
       return
       end
c
c
       subroutine doplerz(uxsfp,ihp,mtm,igm,iht,tempz,izm,n6pk)
c    make adjustment to zone xsecs,  called after dxsec or dxlsec
       real*4 uxsfp(ihp,mtm,igm),tempz(mtm),gamma,sigao,sigap
c
       parameter(gamma = 2.034e-3)
c
       if(mtm.ne.izm) then
         write(n6pk,*) 'Error : doplerz : mtm.ne.izm '
         write(n6pk,*) '                  cannot perform feedback!!'
         stop
       endif
c
       ig=1
       do mt =1, mtm
        sigao = uxsfp(1,mt,ig)
        sigap = uxsfp(1,mt,ig)*(1.+gamma*(sqrt(tempz(mt))-sqrt(300.0)))
        uxsfp(1,mt,ig) = sigap
        uxsfp(iht,mt,ig) = uxsfp(iht,mt,ig) - sigao + sigap
       enddo
c
       return
       end
::::::::::::::
dxsec.f
::::::::::::::
c23456789012345678901234567890123456789012345678901234567890123456789012
       subroutine dxsec(xsts,xstp,xstd,xsfs,xsfp,xsfd,xsss,xssp,xssd,xsf0
      $,xsf1,index,zorg,zmat,igm,index1,ihp,iht,mtm,izm
      $,ir,dum0,dum1,dum)
c    ir = 0 signals first call
       real*4  xsts(index),xstp(index),xstd(index),xsfs(index)
      $       ,xsfp(index),xsfd(index)
      $ ,xsss(index1,igm,igm),xssp(index1,igm,igm),xssd(index1,igm,igm)
      $ ,xsf0(ihp,mtm,igm),xsf1(ihp,mtm,igm),dum0(ihp,mtm,igm)
      $ ,dum1(ihp,mtm,igm),dum(igm,igm,mtm),zorg(index1),zmat(izm)
c
       if(ir.eq.0) then
c  copy ss xsecs onto dum0
       call clearx(dum0,1,ihp*mtm*igm)
       do ig=1,igm
       do mt=1,mtm
       do ih=1,ihp
       dum0(ih,mt,ig) = xsf0(ih,mt,ig)
       enddo
       enddo
       enddo
c       goto 100
c  adjust scat in dum0
       call clearx(dum,1,mtm*igm*igm)
       do mt=1,mtm
       do i5=1,igm
       il = 0
       do i6=igm,1,-1
       if(i6.le.i5) then
       il = il + 1
       dum(il,i5,mt) = xsf0(iht+i6,mt,i5)
       endif
       enddo
       enddo
```

```
      enddo
c
      do mt=1,mtm
      do i5=1,igm
      do i6=1,igm
      dum0(iht+i6,mt,i5) = dum(i6,i5,mt)
      enddo
      enddo
      enddo
c
c100   continue
c first call generate ss cross sections
      do ig=1,igm
      do i =1,index1
      j = i + index1*(ig-1)
      morg   = int(zorg(i))
      matorg = int(zmat(morg))
      xsts(j) = xsf0(iht   ,matorg,ig)
      xstp(j) = xsf0(iht   ,matorg,ig)
      xsfs(j) = xsf0(iht-1,matorg,ig)
      xsfp(j) = xsf0(iht-1,matorg,ig)
      do ih = 1,igm
      xsss(i,ig,ih) = dum0(iht+ig,matorg,ih)
      xssp(i,ig,ih) = dum0(iht+ig,matorg,ih)
      enddo
      enddo
      enddo
c  make sure differences are zero
      call clearx(xssd,1,index*igm)
      call clearx(xstd,1,index)
      call clearx(xsfd,1,index)
c
      else
c  copy pert xsecs onto dum1
      call clearx(dum1,1,ihp*mtm*igm)
      do ig=1,igm
      do mt=1,mtm
      do ih=1,ihp
      dum1(ih,mt,ig) = xsf1(ih,mt,ig)
      enddo
      enddo
      enddo
c      goto 200
c  adjust scat in dum1
      call clearx(dum,1,mtm*igm*igm)
      do mt=1,mtm
      do i5=1,igm
      il = 0
      do i6=igm,1,-1
      if(i6.le.i5) then
      il = il + 1
      dum(il,i5,mt) = xsf1(iht+i6,mt,i5)
      endif
      enddo
      enddo
      enddo
c
      do mt=1,mtm
      do i5=1,igm
      do i6=1,igm
      dum1(iht+i6,mt,i5) = dum(i6,i5,mt)
      enddo
      enddo
      enddo
c
c200   continue
      do ig=1,igm
      do i =1,index1
      j = i + index1*(ig-1)
      morg   = int(zorg(i))
      matorg = int(zmat(morg))
      xstp(j) = xsf1(iht   ,matorg,ig)
      xstd(j) = xstp(j) - xsts(j)
      xsfp(j) = xsf1(iht-1,matorg,ig)
      xsfd(j) = xsfp(j) - xsfs(j)
      do ih = 1,igm
      xssp(i,ig,ih) = dum1(iht+ig,matorg,ih)
      xssd(i,ig,ih) = xssp(i,ig,ih) - xsss(i,ig,ih)
      enddo
      enddo
      enddo
      endif
c
      return
      end
::::::::::::::
dxsec0.f
::::::::::::::
c234567890123456789012345678901234567890123456789012345678901234567890123456789012
      subroutine dxsec0(xsts,xstp,xstd,xsas,xsap,xsf0,index,zlast,zmat
     $ ,igm,index1,ihp,iht,mtm,izm,itn,itj,im,jm,km,telaps,dz
     $ ,n6pk,uxsfp,ntsigf,itjm,dtns,vr,dth)
c
      real*4 xsts(index),xstp(index),xstd(index),zlast(index1),zmat(izm)
     $ ,xsf0(ihp,mtm,igm),xsas(index),xsap(index),telaps,dz(km),dkl,dku
     $ ,uxsfp(ihp,mtm,igm),dtns(2000),vr(im),dth(jm)
c
      include 'comsdx.F'
c
      parameter(sigao = .08344, sigan = .073324)
```

```
      real*4 dsumz(100),cpos
      real*4 zero
      zero = 0.e0
c     steady state xsecs must already be generated
c
c     scat. and fission x-secs donot change over this transient
c     xssp and xsfp are set equal to xsss and xsfs in dxsec, respectively
c
c     if start time has not been reached, return
c
      if(telaps.lt.trmst) goto 30
      if(trmst.eq.trmnd) goto 30
c
c    set time to be used
c
      tuse = telaps
c
      if(itn.eq.1.and.itj.eq.1) then  !first call set absorbtion
c
       do ig=1,igm
       do i =1,index1
       j = i + index1*(ig-1)
       mnew   = int(zlast(i))
       matnew = int(zmat(mnew))
       xsas(j) = xsf0(1,matnew,ig)
       xsap(j) = xsas(j)
       enddo
       enddo
c    convert kl and ku to absolute dimensions
       dkl = zero
       do k=1,mvkl-1    ! lower plane included
        dkl = dkl + dz(k)
       enddo
       dku = zero
       do k=1,mvku      ! upper plane included
        dku = dku + dz(k)
       enddo
c    convert jl and ju to absolute dimensions
       djl = zero
       do j=1,mvjl - 1
        djl = djl + dth(j)
       enddo
       dju = zero
       do j=1,mvju
        dju = dju + dth(j)
       enddo
c    convert il and iu to absolute dimensions
       dil = zero
       do i=1,mvil - 1
        dil = dil + vr(i)
       enddo
       diu = zero
       do i=1,mviu
        diu = diu + vr(i)
       enddo
       write(n6pk,*) '************************************************'
       write(n6pk,*) ' Rod moves from k=',dku,' to ',dkl
       write(n6pk,*) ' Rod moves from j=',dju,' to ',djl
       write(n6pk,*) ' Rod moves from i=',diu,' to ',dil
       write(n6pk,*) '************************************************'
c
c    calculate heights
c
       call clearx(dsumz,1,100)
       dsumz(1) = dz(1)
       do k=2,km
         dsumz(k) = dsumz(k-1) + dz(k)
       enddo
c
c    if speed is zero, calculate
c
       if(trmsp.eq.0.0) then
        trmsp = (dku - dkl) / (trmnd - trmst)
        write(n6pk,*) '****************************************'
        write(n6pk,*) ' Calculated Speed = ',trmsp
        write(n6pk,*) '****************************************'
       endif
c
      endif
c
      if(itn.eq.1) then
        tltot = zero
        tjtot = dtns(1)
        tjtot = tjtot*itj/itjm
        tltot = tltot + tjtot
      elseif(itn.gt.1) then
        tltot = dtns(itn-1)
        tjtot = dtns(itn) - dtns(itn-1)
        tjtot = tjtot*itj/itjm
        tltot = tltot + tjtot
      endif
c
c    tltot must be equal to telaps
c
      if(tltot.ne.telaps)  then
        write(n6pk,*) '************************************************'
        write(n6pk,*) ' warning : current time passed to dxsec0 is '
        write(n6pk,*) '           different from calculated one'
        write(n6pk,*) ' current = ',telaps,' calculated = ',tltot
        write(n6pk,*) ' using calculated value!!!!!!'
```

```
        write(n6pk,*) '************************************************'
        tuse = tltot
      endif
c
      kml = mvkl
      kmu = mvku
c
c   determine current position
c
      if(tuse.le.trmnd) then
        cpos = -trmsp*tuse + dku
       else
        cpos = dkl
      endif
c
c   make sure perturbation does not go past lowest pertuebation point
c
      if(cpos.lt.dkl) then
        write(n6pk,*) '************************************************'
        write(n6pk,*) 'warning : current position is lower than lowest'
        write(n6pk,*) '          position allowed'
        write(n6pk,*) 'current = ',cpos,' lowest allowed = ',dkl
        write(n6pk,*) 'resetting current position to lowest allowed'
        write(n6pk,*) '************************************************'
        cpos = dkl
      endif
c
c   determine current max. plane
c
      kmuc = kmu
      do k=kml,kmu
       if(cpos.lt.dsumz(k)) then
          kmuc = k              ! current max. plane
          goto 10
       endif
      enddo
10    continue
c
      write(n6pk,*) '************************************************'
      write(n6pk,*) '  Current                        Plane'
      write(n6pk,*) '   Time    Position  Plane   Height'
      write(n6pk,'(1x,e10.4,1x,f8.4,4x,i2,3x,f8.4)') tuse,cpos,kmuc
     $    ,dz(kmuc)
      write(n6pk,*) '************************************************'
c
c   if first pass, reset abs and total (for irep.gt.0) to ss
c
      if(itj.eq.1) then
        do ig=1,igm
        do i =1,index1
          j = i + index1*(ig-1)
          mnew   = int(zlast(i))
          matnew = int(zmat(mnew))
          xsap(j) = xsf0(1,matnew,ig)
          xstp(j) = xsf0(iht,matnew,ig)
          xstd(j) = zero
        enddo
        enddo
      endif
c
      jl = mvjl
      ju = mvju
      il = mvil
      iu = mviu
      ig=2       ! only group 2 changes
c
c first fill all previous regions (above current plane)
c
      if(kmuc.ge.kmu) goto 20
      do l=kmu,kmuc+1,-1
       do j=jl,ju
        do i=il,iu
          i1 = i + im*((j-1)+jm*((l-1)+km*(ig-1)))
          xsap(i1) = sigan
          xstp(i1) = xsts(i1) - sigao + sigan
          xstd(i1) = sigan - sigao
        enddo
       enddo
      enddo
20    continue
c
c   now fill this region with weighted sum of x-secs
c
      dzc = dz(kmuc)
      dkin = dsumz(kmuc) - cpos
      if(dkin.gt.dzc) dkin = dzc
      do j=jl,ju
      do i=il,iu
       i1 = i + im*((j-1)+jm*((kmuc-1)+km*(ig-1)))
       xsap(i1) = sigan*dkin/dzc + sigao*(1.0-dkin/dzc)
       xstp(i1) = xsts(i1) - sigao + xsap(i1)
       xstd(i1) = xsap(i1) - sigao
      enddo
      enddo
c
c    read in ss xsecs and modify (for doplerz and rdgip0)
c
30    continue
      if(itj.eq.itjm) then
        call xsecr(uxsfp,ihp,mtm,igm,ntsigf)
```

```
        if(telaps.lt.trmst) return
        if(trmst.eq.trmnd) return
c
      ig=2     ! group 2 only
      do l=kmu,kmuc,-1
       do j=jl,ju
        do i=il,iu
          i2 = i + im*((j-1)+jm*(l-1))
          morg = int(zlast(i2))
          mat  = int(zmat(morg))
c
c   replace the material from il to iu, jl to ju, kmuc to kmu
c   either completely or partially
c   note that every perturbed plane must have
c            a different material id number
c
          if(l.gt.kmuc) then
c
            uxsfp(l,mat,ig) = sigan
            uxsfp(iht,mat,ig) = xsf0(iht,mat,ig) -sigao + sigan
c
          elseif(l.eq.kmuc) then
c
            uxsfp(l,mat,ig) = sigan*dkin/dzc + sigao*(1.0-dkin/dzc)
            uxsfp(iht,mat,ig) = xsf0(iht,mat,ig) -sigao +
     $                          uxsfp(l,mat,ig)
          endif
        enddo
       enddo
      enddo
c
      endif
c
      return
      end
::::::::::::::
dxsec1.f
::::::::::::::
c234567890123456789012345678901234567890123456789012345678901234567890123456789012
      subroutine dxsec1(xsts,xstp,xstd,xsfs,xsfp,xsfd,xsss,xssp,xssd
     $ ,uxsfp,xsf0,index,zlast,zmat
     $ ,igm,index1,ihp,iht,mtm,izm,itn,itj,im,jm,km,telaps,dz
     $ ,n6pk,xsfnew,uxsfnew,dum0,dumx,ntsigf,itjm,dtns,vr,dth)
c
      real*4 xsts(index),xstp(index),xstd(index),xsfs(index),xsfp(index)
     $ ,xsfd(index),xsss(index1,igm,igm),xssp(index1,igm,igm)
     $ ,xssd(index1,igm,igm),uxsfp(ihp,mtm,igm),zlast(index1)
     $ ,zmat(izm),xsf0(ihp,mtm,igm),telaps,dz(km),xsfnew(ihp,1,igm)
     $ ,uxsfnew(ihp,1,igm),dum0(ihp,mtm,igm),dumx(igm,igm,mtm),dkl,dku
     $ ,dtns(2000),vr(im),dth(jm)
c
      include 'comsdx.F'
c
      real*4 dsumz(100),dum(50,50,1),cpos
      real*4 zero
      zero = 0.e0
c     steady state xsecs must already be generated
c
c     xssp and xsfp are set equal to xsss and xsfs in dxsec, respectively
c
c     if start time has not been reached, return
c
      if(telaps.lt.trmst) return
c
c    set time to be used
c
      tuse = telaps
c
      if(itn.eq.1.and.itj.eq.1) then  !first call
c
c     read replacing material
c
      call xsecr(xsfnew,ihp,1,igm,mvmatr)
      call xsecr(uxsfnew,ihp,1,igm,mvmatr)
c
c     adjust scattering permanantly in new xsecs
c
      call clearx(dum,1,igm*igm)
      do i5=1,igm
      il = 0
      do i6=igm,1,-1
      if(i6.le.i5) then
      il = il + 1
      dum(il,i5,1) = xsfnew(iht+i6,1,i5)
      endif
      enddo
      enddo
c
      do i5=1,igm
      do i6=1,igm
      xsfnew(iht+i6,1,i5) = dum(i6,i5,1)
      enddo
      enddo
c
c    convert kl and ku to absolute dimensions
c
       dkl = zero
       do k=1,mvkl-1
        dkl = dkl + dz(k)
       enddo
```

```
      dku = zero
      do k=1,mvku     ! upper plane included
       dku = dku + dz(k)
      enddo
c   convert jl and ju to absolute dimensions
      djl = zero
      do j=1,mvjl - 1
       djl = djl + dth(j)
      enddo
      dju = zero
      do j=1,mvju
       dju = dju + dth(j)
      enddo
c   convert il and iu to absolute dimensions
      dil = zero
      do i=1,mvil - 1
       dil = dil + vr(i)
      enddo
      diu = zero
      do i=1,mviu
       diu = diu + vr(i)
      enddo
      write(n6pk,*) '************************************************'
      write(n6pk,*) ' Rod moves from k=',dku,' to ',dkl
      write(n6pk,*) ' Rod moves from j=',dju,' to ',djl
      write(n6pk,*) ' Rod moves from i=',diu,' to ',dil
      write(n6pk,*) '************************************************'
c
c   calculate heights
c
       call clearx(dsumz,1,100)
      do k=1,km
       do l=1,k
        dsumz(k) = dsumz(k) + dz(l)
       enddo
c      if(dsumz(k).eq.dkl) kml = k+1    ! perturbation boundaries
c      if(dsumz(k).eq.dku) kmu = k
      enddo
c
c   if speed is zero, calculate
c
       if(trmsp.eq.0.0) then
        trmsp = (dku - dkl) / (trmnd - trmst)
        write(n6pk,*) '****************************************'
        write(n6pk,*) ' Calculated Speed = ',trmsp
        write(n6pk,*) '****************************************'
       endif
c
      endif
c
      if(itn.eq.1) then
        tltot = zero
        tjtot = dtns(1)
        tjtot = tjtot*itj/itjm
        tltot = tltot + tjtot
      elseif(itn.gt.1) then
        tltot = dtns(itn-1)
        tjtot = dtns(itn) - dtns(itn-1)
        tjtot = tjtot*itj/itjm
        tltot = tltot + tjtot
      endif
c
c   tltot must be equal to telaps
c
      if(tltot.ne.telaps)  then
        write(n6pk,*) '************************************************'
        write(n6pk,*) ' warning : current time passed to dxsec1 is '
        write(n6pk,*) '           different from calculated one'
        write(n6pk,*) ' current = ',telaps,' calculated = ',tltot
        write(n6pk,*) ' using calculated value!!!!!!'
        write(n6pk,*) '************************************************'
        tuse = tltot
      endif
c
      kml = mvkl
      kmu = mvku
c
c   copy ss xsecs onto dum0
c
      call clearx(dum0,1,ihp*mtm*igm)
      do ig=1,igm
      do mt=1,mtm
      do ih=1,ihp
      dum0(ih,mt,ig) = xsf0(ih,mt,ig)
      enddo
      enddo
      enddo
c
c   adjust scattering in dum0
c
      call clearx(dumx,1,mtm*igm*igm)
      do mt=1,mtm
      do i5=1,igm
      il = 0
      do i6=igm,1,-1
      if(i6.le.i5) then
      il = il + 1
      dumx(il,i5,mt) = xsf0(iht+i6,mt,i5)
      endif
      enddo
```

```
      enddo
      enddo
c
      do mt=1,mtm
      do i5=1,igm
      do i6=1,igm
      dum0(iht+i6,mt,i5) = dumx(i6,i5,mt)
      enddo
      enddo
      enddo
c
c   determine current position
c
      if(tuse.le.trmnd) then
        cpos = -trmsp*tuse + dku
       else
        cpos = dkl
      endif
c
c   make sure perturbation does not go past lowest pertuebation point
c
      if(cpos.lt.dkl) then
        write(n6pk,*) '************************************************'
        write(n6pk,*) 'warning : current position is lower than lowest'
        write(n6pk,*) '          position allowed'
        write(n6pk,*) 'current = ',cpos,' lowest allowed = ',dkl
        write(n6pk,*) 'resetting current position to lowest allowed'
        write(n6pk,*) '************************************************'
        cpos = dkl
      endif
c
c   determine current max. plane
c
      kmuc = kmu
      do k=kml,kmu
       if(cpos.lt.dsumz(k)) then
          kmuc = k              ! current max. plane
          goto 10
       endif
      enddo
10    continue
c
      write(n6pk,*) '  Current                        Plane'
      write(n6pk,*) '   Time    Position  Plane    Height'
      write(n6pk,'(1x,e10.4,1x,f8.4,4x,i2,3x,f8.4)') tuse,cpos,kmuc
     $    ,dz(kmuc)
c
      jl = mvjl
      ju = mvju
      il = mvil
      iu = mviu
c
c first fill all previous regions (above current plane)
c
      if(kmuc.ge.kmu) goto 20
      do ig=1,igm
      do l=kmu,kmuc+1,-1
       do j=jl,ju
        do i=il,iu
          i1 = i + im*((j-1)+jm*((l-1)+km*(ig-1)))
          xstp(i1) = xsfnew(iht,1,ig)
          xstd(i1) = xstp(i1) - xsts(i1)
          xsfp(i1) = xsfnew(iht-1,1,ig)
          xsfd(i1) = xsfp(i1) - xsfs(i1)
          i2 = i + im*((j-1)+jm*(l-1))
          do ih=1,igm
            xssp(i2,ig,ih) = xsfnew(iht+ig,1,ih)
            xssd(i2,ig,ih) = xssp(i2,ig,ih)-xsss(i2,ig,ih)
          enddo
        enddo
       enddo
      enddo
      enddo
20    continue
c
c   now fill this region with weighted sum of x-secs
c
      dzc = dz(kmuc)
      dkin = dsumz(kmuc) - cpos
      if(dkin.gt.dzc) dkin = dzc
      do ig=1,igm
      do j=jl,ju
      do i=il,iu
       i1 = i + im*((j-1)+jm*((kmuc-1)+km*(ig-1)))
       i2 = i + im*((j-1)+jm*(kmuc-1))
      morg = int(zlast(i2))
      mat  = int(zmat(morg))
       xstp(i1) = xsfnew(iht,1,ig)*dkin/dzc + xsf0(iht,mat,ig)*
     $            (1.0-dkin/dzc)
       xstd(i1) = xstp(i1) - xsts(i1)
       xsfp(i1) = xsfnew(iht-1,1,ig)*dkin/dzc + xsf0(iht-1,mat,ig)*
     $            (1.0-dkin/dzc)
       xsfd(i1) = xsfp(i1) - xsfs(i1)
          do ih=1,igm
            xssp(i2,ig,ih) = xsfnew(iht+ig,1,ih)*dkin/dzc +
     $                       dum0(iht+ig,mat,ih)*(1.0-dkin/dzc)
            xssd(i2,ig,ih) = xssp(i2,ig,ih)-xsss(i2,ig,ih)
          enddo
      enddo
      enddo
```

```
      enddo
c
c     read in ss xsecs if last pass over tj (for rdgip1)
c
      if(itj.eq.itjm) then
        call xsecr(uxsfp,ihp,mtm,igm,ntsigf)
c
      do ig=1,igm
      do l=kmu,kmuc,-1
       do j=jl,ju
        do i=il,iu
          i2 = i + im*((j-1)+jm*(l-1))
          morg = int(zlast(i2))
          mat  = int(zmat(morg))
c
c   replace the material from il to iu, jl to ju, kmuc to kmu
c   either completely or partially
c   note that every perturbed plane must have
c            a different material id number
c
           if(l.gt.kmuc) then
            do ih=1,ihp
             uxsfp(ih,mat,ig) = uxsfnew(ih,1,ig)
            enddo
           elseif(l.eq.kmuc) then
            do ih=1,ihp
             uxsfp(ih,mat,ig) = uxsfnew(ih,1,ig)*dkin/dzc +
     $                          xsf0(ih,mat,ig)*(1.0-dkin/dzc)
            enddo
           endif
        enddo
       enddo
      enddo
      enddo
      endif
c
c
      return
      end
:::::::::::::::
dxsec2.f
:::::::::::::::
c234567890123456789012345678901234567890123456789012345678901234567890123456789012
      subroutine dxsec2(xsts,xstp,xstd,xsfs,xsfp,xsfd,xsss,xssp,xssd
     $ ,xsf0,xsf1,index,zorg,zmat,igm,index1,ihp,iht,mtm,izm
     $ ,ir,dum0,dum1,dum)
c   ir = 0 signals first call
      real*4  xsts(index),xstp(index),xstd(index),xsfs(index)
     $       ,xsfp(index),xsfd(index)
     $ ,xsss(index1,igm,igm),xssp(index1,igm,igm),xssd(index1,igm,igm)
     $ ,xsf0(ihp,mtm,igm),xsf1(ihp,mtm,igm),dum0(ihp,mtm,igm)
     $ ,dum1(ihp,mtm,igm),dum(igm,igm,mtm),zorg(index1),zmat(izm)
c
      if(ir.eq.0) then
c  copy ss xsecs onto dum0
      call clearx(dum0,1,ihp*mtm*igm)
      do ig=1,igm
      do mt=1,mtm
      do ih=1,ihp
      dum0(ih,mt,ig) = xsf0(ih,mt,ig)
      enddo
      enddo
      enddo
c      goto 100
c  adjust scat in dum0
      call clearx(dum,1,mtm*igm*igm)
      do mt=1,mtm
      do i5=1,igm
      il = 0
      do i6=igm,1,-1
      if(i6.le.i5) then
      il = il + 1
      dum(il,i5,mt) = xsf0(iht+i6,mt,i5)
      endif
      enddo
      enddo
      enddo
c
      do mt=1,mtm
      do i5=1,igm
      do i6=1,igm
      dum0(iht+i6,mt,i5) = dum(i6,i5,mt)
      enddo
      enddo
      enddo
c
c100   continue
c first call generate ss cross sections
      do ig=1,igm
      do i =1,index1
      j = i + index1*(ig-1)
      morg   = int(zorg(i))
      matorg = int(zmat(morg))
      xsts(j) = xsf0(iht  ,matorg,ig)
      xstp(j) = xsf0(iht  ,matorg,ig)
      xsfs(j) = xsf0(iht-1,matorg,ig)
      xsfp(j) = xsf0(iht-1,matorg,ig)
      do ih = 1,igm
      xsss(i,ig,ih) = dum0(iht+ig,matorg,ih)
      xssp(i,ig,ih) = dum0(iht+ig,matorg,ih)
```

```
      enddo
      enddo
      enddo
c  make sure differences are zero
      call clearx(xssd,1,index*igm)
      call clearx(xstd,1,index)
      call clearx(xsfd,1,index)
c
      else
c  copy pert xsecs onto dum1
      call clearx(dum1,1,ihp*mtm*igm)
      do ig=1,igm
      do mt=1,mtm
      do ih=1,ihp
      dum1(ih,mt,ig) = xsf1(ih,mt,ig)
      enddo
      enddo
      enddo
c      goto 200
c  adjust scat in dum1
      call clearx(dum,1,mtm*igm*igm)
      do mt=1,mtm
      do i5=1,igm
      il = 0
      do i6=igm,1,-1
      if(i6.le.i5) then
      il = il + 1
      dum(il,i5,mt) = xsf1(iht+i6,mt,i5)
      endif
      enddo
      enddo
      enddo
c
      do mt=1,mtm
      do i5=1,igm
      do i6=1,igm
      dum1(iht+i6,mt,i5) = dum(i6,i5,mt)
      enddo
      enddo
      enddo
c
c200   continue
      do ig=1,igm
      do i =1,index1
      j = i + index1*(ig-1)
      morg   = int(zorg(i))
      matorg = int(zmat(morg))
      xstp(j) = xsf1(iht  ,matorg,ig)
      xstd(j) = xstp(j) - xsts(j)
      xsfp(j) = xsf1(iht-1,matorg,ig)
      xsfd(j) = xsfp(j) - xsfs(j)
      do ih = 1,igm
      xssp(i,ig,ih) = dum1(iht+ig,matorg,ih)
      xssd(i,ig,ih) = xssp(i,ig,ih) - xsss(i,ig,ih)
      enddo
      enddo
      enddo
      endif
c
      return
      end
::::::::::::::
dxsec5.f
::::::::::::::
c234567890123456789012345678901234567890123456789012345678901234567890123456789012
      subroutine dxsec5(xsts,xstp,xstd,xsfs,xsfp,xsfd,xsss,xssp,xssd
     $,xsf1,index,zorg,zmat,igm,index1,ihp,iht,mtm,izm
     $,dum0,dum1,dum,tx)
c   ir = 1 signals perturbation ended use latest
      real*4 xsts(index),xstp(index),xstd(index),xsfs(index)
     $ ,xsfp(index),xsfd(index)
     $ ,xsss(index1,igm,igm),xssp(index1,igm,igm),xssd(index1,igm,igm)
     $ ,xsf1(ihp,mtm,igm),zorg(index1),zmat(izm),dum0(ihp,mtm,igm)
     $ ,dum1(ihp,mtm,igm),dum(igm,igm,mtm),tx
c
      call clearx(dum1,1,ihp*mtm*igm)
      do ig=1,igm
      do mt=1,mtm
      do ih=1,ihp
      dum1(ih,mt,ig) = xsf1(ih,mt,ig)
      enddo
      enddo
      enddo
c  adjust scat in dum1
      call clearx(dum,1,mtm*igm*igm)
      do mt=1,mtm
      do i5=1,igm
      il = 0
      do i6=igm,1,-1
      if(i6.le.i5) then
      il = il + 1
      dum(il,i5,mt) = xsf1(iht+i6,mt,i5)
      endif
      enddo
      enddo
      enddo
c
      do mt=1,mtm
      do i5=1,igm
      do i6=1,igm
```

```
      dum1(iht+i6,mt,i5) = dum(i6,i5,mt)
      enddo
      enddo
      enddo
c
c  calculate differences
c
      do ig=1,igm
      do i =1,index1
      j = i + index1*(ig-1)
      morg   = int(zorg(i))
      matorg = int(zmat(morg))
      xstp(j) = (xsf1(iht,matorg,ig)-xsts(j))*tx + xsts(j)
      xstd(j) = (xsf1(iht  ,matorg,ig)-xsts(j))*tx
      xsfp(j) = (xsf1(iht-1,matorg,ig)-xsfs(j))*tx + xsfs(j)
      xsfd(j) = (xsf1(iht-1,matorg,ig)-xsfs(j))*tx
c
      do ih = 1,igm
      xssp(i,ig,ih) = (dum1(iht+ig,matorg,ih)-xsss(i,ig,ih))*tx +
     $                 xsss(i,ig,ih)
      xssd(i,ig,ih) = (dum1(iht+ig,matorg,ih)-xsss(i,ig,ih))*tx
      enddo
c
      enddo
      enddo
c
      return
      end
:::::::::::::::
dxsec6.f
:::::::::::::::
c234567890123456789012345678901234567890123456789012345678901234567890123456789012
      subroutine dxsec6(xsts,xstp,xstd,xsfs,xsfp,xsfd,xsss,xssp,xssd
     $,xsf1,xsf2,index,zorg,zmat,igm,index1,ihp,iht,mtm,izm
     $,dum0,dum1,dum,tx,mzm,mz,it)
c
      real*4 xsts(index),xstp(index),xstd(index),xsfs(index)
     $ ,xsfp(index),xsfd(index),xsf2(ihp,mtm,igm)
     $ ,xsss(index1,igm,igm),xssp(index1,igm,igm),xssd(index1,igm,igm)
     $ ,xsf1(ihp,mtm,igm),zorg(index1),zmat(izm),dum0(ihp,mtm,igm)
     $ ,dum1(ihp,mtm,igm),dum(igm,igm,mtm),tx
      dimension mz(10)
c
      call clearx(dum1,1,ihp*mtm*igm)
      do ig=1,igm
      do mt=1,mtm
      do ih=1,ihp
      dum1(ih,mt,ig) = xsf1(ih,mt,ig)
      enddo
      enddo
      enddo
c  adjust scat in dum1
      call clearx(dum,1,mtm*igm*igm)
      do mt=1,mtm
      do i5=1,igm
      il = 0
      do i6=igm,1,-1
      if(i6.le.i5) then
      il = il + 1
      dum(il,i5,mt) = xsf1(iht+i6,mt,i5)
      endif
      enddo
      enddo
      enddo
c
      do mt=1,mtm
      do i5=1,igm
      do i6=1,igm
      dum1(iht+i6,mt,i5) = dum(i6,i5,mt)
      enddo
      enddo
      enddo
c
c  calculate differences
c
      do ig=1,igm
      do i =1,index1
      j = i + index1*(ig-1)
      morg   = int(zorg(i))
      matorg = int(zmat(morg))
      xstp(j) = (xsf1(iht,matorg,ig)-xsts(j))*tx + xsts(j)
      xstd(j) = (xsf1(iht  ,matorg,ig)-xsts(j))*tx
      xsfp(j) = (xsf1(iht-1,matorg,ig)-xsfs(j))*tx + xsfs(j)
      xsfd(j) = (xsf1(iht-1,matorg,ig)-xsfs(j))*tx
c
      do ih = 1,igm
      xssp(i,ig,ih) = (dum1(iht+ig,matorg,ih)-xsss(i,ig,ih))*tx +
     $                 xsss(i,ig,ih)
      xssd(i,ig,ih) = (dum1(iht+ig,matorg,ih)-xsss(i,ig,ih))*tx
      enddo
c
      enddo
      enddo
c
      if(it.eq.0) return
c
c   overwrite the unchanging zone
c
      call clearx(dum1,1,ihp*mtm*igm)
      do ig=1,igm
```

```
      do mt=1,mtm
      do ih=1,ihp
      dum1(ih,mt,ig) = xsf2(ih,mt,ig)
      enddo
      enddo
      enddo
c  adjust scat in dum1
      call clearx(dum,1,mtm*igm*igm)
      do mt=1,mtm
      do i5=1,igm
      il = 0
      do i6=igm,1,-1
      if(i6.le.i5) then
      il = il + 1
      dum(il,i5,mt) = xsf2(iht+i6,mt,i5)
      endif
      enddo
      enddo
      enddo
c
      do mt=1,mtm
      do i5=1,igm
      do i6=1,igm
      dum1(iht+i6,mt,i5) = dum(i6,i5,mt)
      enddo
      enddo
      enddo
c
c
      do ig=1,igm
      do i =1,index1
      j = i + index1*(ig-1)
      morg   = int(zorg(i))
      matorg = int(zmat(morg))
       do iz = 1, mzm
      if(morg.eq.mz(iz)) then
      xstp(j) = xsf2(iht,matorg,ig)
      xstd(j) = xstp(j)-xsts(j)
      xsfp(j) = xsf2(iht-1,matorg,ig)
      xsfd(j) = xsfp(j)-xsfs(j)
c
      do ih = 1,igm
      xssp(i,ig,ih) = dum1(iht+ig,matorg,ih)
      xssd(i,ig,ih) = xssp(i,ig,ih)-xsss(i,ig,ih)
      enddo
c
      endif
       enddo
c
      enddo
      enddo
      return
      end
::::::::::::::
extrap.f
::::::::::::::
      subroutine extrap(flux,fluxp,fluxa,h1,h2,index,iextrp,n6pk)
      real*4 flux(index),fluxp(index),fluxa(index),h1,h2
      real*4 zero
      zero = 0.e0
c     h1=0 signals for constant extrapolation
c
c
c
c      note that x2=0, |x2|-|x1|=h1, x3-x2=h2
c       x1,y1     x2,y2      x3,y3
c         |         |          |
c        -x1        0          x3
c       <---------><---------->
c            h1           h2
c
c      h1 = x2 - x1
c      h2 = x3 - x2
c
c
      if(h1.eq.0.) goto 200
      if(iextrp.eq.0.) goto 200
      do 20 i=1,index
      flux(i) = (fluxp(i)-fluxa(i))*(h1+h2)/h1 + fluxa(i)
       if(flux(i).le.0.) then
         if(iextrp.eq.1) then
        write(n6pk,*) "WARNING : Negative Values Occured During
     + Extrapolation. Skipping Extrapolation Process !"
           goto 200  !if negatives occur do not extrapolate
         elseif(iextrp.eq.2) then
          flux(i) = zero !if negatives occur set them equal to zero
         endif
        write(n6pk,*) "WARNING : Negative Values Occured During
     + Extrapolation. Setting the Negatives Equal to Zero !"
       endif
20    continue
      goto 900
200   do 220 i=1,index
      flux(i) = fluxp(i)
220   continue
c
900   return
      end
::::::::::::::
f.f
```

```
::::::::::::::
      subroutine f(neq,t,y,ydot)
c
c     f is called by lsode to determine derivatives of the
c     dependent variables
c
c
      include 'comsdl.F'
      include 'comsd8.F'
c
      double precision y,ydot,t,sum
      dimension y(neq),ydot(neq)
      real*4 zero
      zero = 0.e0
c
c     y(1) is total energy release
c     y(2) is power (or amplitude normalized to power)
c     y(3) thru y(2+ndg) are precursors
c
      sum = zero
      do 100 i=1,ndg
          sum = sum + alambda(i)*y(i+1)
  100 continue
      ydot(1) = (rhop-betp)*y(1)/rlamp + sum + extp
      dampndtp = ydot(1)
c
c    equations 1 thru ndg+1: precursors
c
      do 200 i=1,ndg
          ydot(i+1) = betap(i)*y(1)/rlamp - alambda(i)*y(i+1)
  200 continue
c
c
      return
      end
::::::::::::::
ffread.f
::::::::::::::
*deck        ffread
      subroutine ffread(in,k,v,nf,n5,n6,iprtrg)
      dimension ny(80),in(2),k(2),v(2),n(10)
csw
      character*4  ny, k, n, nsa, nsq, nyncc
     1, nt, nbb, ndp, npl, npa, nmi, nss, ndl, ns, ne,  nq,  nm
     1, nn, nr,  ni,  nff, na,  nl,  nk,  nsl, ng, neq, nbs
      data nt, nbb, ndp, npl, npa, nmi, nss, ndl,  ns,  ne,  nq,  nm
     1  / 1ht, 1h , 1h., 1h+, 1h&, 1h-, 1hs, 1h$, 1h*, 1he, 1hq, 1hm   /
      data nn,  nr,  ni, nff,  na,  nl,  nk, nsl,  ng, neq, nbs, ifree
     1  / 1hn, 1hr, 1hi, 1hf, 1ha, 1hl, 1hk, 1h/, 1hg, 1h=, 1hb, 0     /
      data nsa /1h'/, nsq /1h"/
      data n(1), n(2), n(3), n(4), n(5), n(6), n(7), n(8), n(9), n(10)
     1   /  1h0,  1h1,  1h2,  1h3,  1h4,  1h5,  1h6,  1h7,  1h8,  1h9 /
csw
clw
c     character*8  ny, k, n, nsa, nsq, nyncc
c     1, nt, nbb, ndp, npl, npa, nmi, nss, ndl, ns, ne,  nq,  nm
c     1, nn, nr,  ni,  nff, na,  nl,  nk,  nsl, ng, neq, nbs
c      data nt, nbb, ndp, npl, npa, nmi, nss, ndl,  ns,  ne,  nq,  nm
c     1  / 't', ' ', '.', '+', '&', '-', 's', '$', '*', 'e', 'q', 'm'   /
c      data nn,  nr,  ni, nff,  na,  nl,  nk, nsl,  ng, neq, nbs, ifree
c     1  / 'n', 'r', 'i', 'f', 'a', 'l', 'k', '/', 'g', '=', 'b', 0     /
c      data nsa /''''/, nsq /'"'/
c      data n(1), n(2), n(3), n(4), n(5), n(6), n(7), n(8), n(9), n(10)
c     1   /  '0',  '1',  '2',  '3',  '4',  '5',  '6',  '7',  '8',  '9' /
clw
c
c - - - - - free/fixed card image translator
c           based on original by w.w.engle, ornl
c
 1998 do 1 i=1,37
      in(i)=0
      v(i)=0.0
    1 k(i)=nbb
c           number is isgn*num**(iex*iabs(iexp) + nscl)
      isgn=1
      num=0
      iex=0
      iexp=0
c           nscl=no. places past .     ndpn=1 after decimal is past
      nscl=0
      ndpn=0
c           iblnk=1 after non (blank, +, &, -)
      iblnk=0
c           nf=field  ncc=card column  ifc=field column  ids=1 after $*
      nf=1
      ncc=1
      ifc=0
      ids=0
c
  112 read (n5,10) ny
   10 format(80a1)
c
      if(ny(1).eq.nsa) go to 110
c
      if(ny(1).eq.nsq) go to 110
c
      if(ny(1).ne.nsl .and. (ny(1).ne.npl.or.ny(2).ne.npl)) go to 114
  110 write(n6,30) ny
      go to 112
  114 if(iprtrg.gt.0)write(n6,30)ny
```

```
c
c - - - - - start column loop
  111 nyncc=ny(ncc)
      if(nyncc.eq.nsl) go to 999
      if(ifree.eq.0)go to 122
c
c           fixed field only
      ifc=ifc+1
      if(ifc.gt.12)ifc=1
      if(ifc.eq.3)go to 124
      if(ifc.eq.11 .and. iex.ne.0 .and. nyncc.eq.nbb)go to 106
      if(nyncc.eq.ne .and. ifc.gt.4)go to 203
c
c - - - - - was symbol blank, +, &, -, or exponent?
  122 if(nyncc.eq.nbb)go to 101
      if(nyncc.eq.npl .or. nyncc.eq.npa)go to 103
      if(nyncc.eq.nmi)go to 104
      if(nyncc.eq.ne .and. iblnk.eq.1)go to 203
      iblnk=1
      if(nyncc.eq.ndp)go to 102
c
c - - - - - interpret as a digit
  124 do 1024 i=1,10
      if(nyncc.ne.n(i))go to 1024
      numm=i-1
c           was this an exponent or fraction digit?
      if(iex.ne.0)go to 107
      go to 105
 1024 continue
  121 if(k(nf).ne.nbb .and. nyncc.eq.nbb)go to 101
c           field 3 has started
      k(nf)=nyncc
      in(nf)=num
      num=0
      iblnk=0
      if(nyncc.eq.nbb)go to 106
      if(k(nf).ne.ndl .and. k(nf).ne.ns .and. k(nf).ne.neq) go to 113
c           $ or * or = encountered
      ifree=0
      ids=1
      if(nf.eq.1)go to 108
      if(k(nf).ne.k(nf-1))go to 108
c           this begins a free-field array
      k(nf)=nbb
      ids=0
      if(k(nf-1).eq.neq) nf = nf - 1
      go to 106
  113 if(ifree.eq.0 .and.
     1   nyncc.ne.nr .and. nyncc.ne.ni .and. nyncc.ne.nff .and.
     1   nyncc.ne.na .and. nyncc.ne.nl .and. nyncc.ne.nk .and.
     2   nyncc.ne.nq .and. nyncc.ne.nm .and. nyncc.ne.nn .and.
     3   nyncc.ne.ng )   go to 108
      if(ifree.eq.1 .and. (nyncc.eq.nss.or.nyncc.eq.nbs))go to 108
      if(ncc.eq.72)write(n6,20)ny
      go to 106
c           build number
  105 num=num*10 + numm
      nscl=nscl + ndpn
      go to 206
c           decimal point
  102 ndpn=-1
      go to 206
c           build exponent
  107 iexp=iexp*10 + iex*numm
  206 if(ifc.eq.12)go to 301
      go to 106
c           + sign
  103 if(num.ne.0)go to 203
      isgn=1
      go to 106
c           start + exponent
  203 iex=1
      go to 106
c           - sign
  104 if(num.ne.0)go to 204
      isgn=-1
      go to 106
c           start - exponent
  204 iex=-1
      go to 106
c           set fixed field - comes here if $ or * followed by blank
  123 ifree=1
      ids=0
      go to 109
c           blank or end of card encountered
  101 if(ids.eq.1)go to 123
      if(ifree.eq.1 .and. ifc.eq.12 .and. k(nf).ne.nbb)go to 301
c           leading blanks or fixed field?
      if(iblnk.eq.0)go to 106
      if(ifree.eq.1 .and. ifc.ne.12)go to 106
c           a blank in free-field with iblank=1 stops field
c           assemble entire number
  301 vnum=isgn*num
      v(nf)=vnum*10.0**(iexp+nscl)
      ndpn=0
      iex=0
      iexp=0
      nscl=0
      iblnk=0
      num=0
```

```
      isgn=1
c
c           advance field
  108 nf=nf + 1
      if(nyncc.eq.nss .or. nyncc.eq.nbs)go to 106
c           come here if field advance or $ or * followed by blank
  109 ifc=0
      if(ifree.eq.1)ncc=((ncc+11)/12)*12
      if(k(nf-1).eq.nt)go to 999
c           advance column counter, continue field unless card is ended
  106 if(ncc.ge.73)go to 999
      ncc=ncc+1
      if(ncc.le.72)go to 111
c           .end of column loop
c
      go to 101
c           pass card if it had no data fields
  999 nf=nf-1
      if(nf.le.0)go to 1998
   20 format(63h0 ****** incomplete field at end of card, some data may
     1be lost /27h         card image follows/ 1x,80a1)
   30 format( 1x, 80a1)
      return
      end
:::::::::::::::
fgues.f
:::::::::::::::
      subroutine fgues(qmom,sor,lm,im,jm,km,igm,n6,n6o)
c
c    qmom  is moments
c    use only group 1 moments
      real*4 qmom(im,jm,km,lm,igm),sor(im*jm*km)
      character*80 cny
c
      call clearx(sor,1,index1)
      do 15  k=1,km
      do 20  j=1,jm
      do 25  i=1,im
      i1 = i + im*((j-1)+jm*(k-1))
      sor(i1) = qmom(i,j,k,1,1)
25    continue
20    continue
15    continue
c
      open(n6o,file='n6osil',status='unknown')
      open(n6 ,file='n6sor',status='unknown')
      index1 = im*jm*km
      jmax=index1/5
      if(jmax*5.ne.index1) jmax=jmax+1
      do j=1, jmax
       k=5*(j-1)+1
       kmax= k+4
       if(kmax.gt.index1) kmax=index1
       if(j.eq.1)
     $ write(n6o,85) (sor(i),i=k,kmax)
       if(j.ne.1)
     $ write(n6o,86) (sor(i),i=k,kmax)
      end do
      rewind (n6o)
23    read(n6o,'(a80)',end=24) cny
      do i=1,80
       if(cny(i:i).eq.'E') cny(i:i)='e'
      enddo
      write(n6,'(a80)') cny
      goto 23
24    write(n6,*) 't'
      write(n6,*) '=end'
c
85    format('96** ',5(1pe12.5,2x))
86    format('     ',5(1pe12.5,2x))
c
      close(n6)
      close(n6o,status='delete')
      return
      end
:::::::::::::::
fhlpr.f
:::::::::::::::
      subroutine fhlpr ( cc, a, nou )
      external lttr
c#########################################################################
c
c               audit trail information
c
c     date the module was last permanently updated:     93/02/24
c     time the module was last permanently updated:     07:54:44
c     programmer name:                                  l.m.petrie
c     module name:                                      s7fhlpr
c     current archiving level number:                   00008
c     current number of permanent updates:              00008
c     date of last access by librarian:                 93/02/24
c     dataset name:  x4s.scale4.master
c
c#########################################################################
      character*1 cc,s(50),xchar,a*8,chr(12,51)*16
      common /letter/ chr
      dimension nc(8)
      data s/'a','b','c','d','e','f','g','h','i','j','k','l','m','n',
     *   'o','p','q','r','s','t','u','v','w','x','y','z',' ','+','-',
     *   '/','0','1','2','3','4','5','6','7','8','9','=','.',':',')',
```

```
     *   '*','.',',','(','''','&','$'/
      do 120 ic=1,8
         xchar = a(ic:ic)
         do 100 l=1,50
            if (xchar.eq.s(l)) go to 110
  100       continue
         l = 51
  110    continue
         nc(ic) = l
  120    continue
      write(nou,10000) cc,((chr(j,nc(k)),k=1,8),j=1,12)
      return
10000 format(a1,2x,8a16/(3x,8a16))
      end
::::::::::::::
fidas.f
::::::::::::::
*deck       fidas
      subroutine fidas (d,ldtk,loc,l12,j3,n5,n6)
c - - - - - *** warning -- d and ldtk are equivalent
c           this subroutine was modified from the anisn code
      dimension in(37),k(37),v(37),ldtk(1),vmt(18),
     1  d(1), loc(1)
      dimension prt(3)
      equivalence (bb,lbb)
      data iprtrg /-1/
      data ldl,  ls, lr1,  li,  lt, lss,  lf,  la, lap, lpl, lmi
     1   / lh$, lh*, lhr, lhi, lht, lhs, lhf, lha, lh&, lh+, lh-       /
      data  lz,  lv,  lu,  bb,  lg,  lm,  ln, lki,  lo,  lc,  le, lli
     1   / lhz, lhv, lhu, lh , lhq, lhm, lhn, lhk, lho, lhc, lhe, lhl  /
      data  lp,  lg
     1   / lhp, lhg                                                    /
      data prt(1), prt(2), prt(3)
     1   /  3hoff,  3hprt,   2hon                                      /
c
c - - - - - machine-independent input array processor
      j=0
      j3=0
      iii=0
      ivmt=0
      ncount=0
    1 call ffread(in,k,v,nf,n5,n6,iprtrg)
      do 27 i=1,nf
      if(k(i).ne.lpl .and. k(i).ne.lap .and. k(i).ne.lmi)go to 27
c   *** exponentiation (&,+,-)
      l=in(i)
      if(l.eq.0)go to 27
      e=10.0**l
      if(k(i).eq.lmi)go to 28
      v(i)=v(i)*e
      go to 27
   28 v(i)=v(i)/e
   27 continue
 1002 do 2 i=1,nf
      if(iii.eq.0)go to 4
      if(iii-2)3,44,48
    4 iii=0
      if(k(i).eq.lbb)go to 14
      if(k(i).eq.ldl)go to 5
      if(k(i).eq.ls )go to 6
      if(k(i).eq.lr1)go to 7
      if(k(i).eq.li )go to 8
      if(k(i).eq.lt )go to 9
      if(k(i).eq.lss)go to 11
      if(k(i).eq.lf)go to 23
      if(k(i).eq.la)go to 26
      if(k(i).eq.lz)go to 31
      if(k(i).eq.le)go to 39
      if(k(i).eq.lq)go to 35
      if(k(i).eq.lg)go to 35
      if(k(i).eq.lm .or. k(i).eq.ln)go to 36
      if(k(i).eq.lu)go to 33
      if(k(i).eq.lv)go to 34
      if(k(i).eq.lli)go to 42
      if(k(i).eq.lki)go to 46
      if(k(i).eq.lo)go to 55
      if(k(i).eq.lp)go to 57
      if(k(i).eq.lc)go to 56
      go to 14
c   *** terminate (t)
    9 j2=0
      itest=in(i)
      if(j.eq.0)go to 16
      go to 12
c   *** variable format control (u,v)
   33 read(n5,80)vmt
   34 ivmt=1
c   *** begin new array (*,$)
    6 kkk=0
      go to 13
    5 kkk=1
   13 if(ncount+j .eq. 0)go to 15
      j2=1
   12 write (n6,20) l1,lb,j
      if(j.eq.ncount)go to 16
      imax=j1 + j - 1
      if(kk.eq.1)go to 22
      write (n6,222) (d(ii),ii=j1,imax)
  222 format( 1x, 1p,10e12.5 )
      go to 422
```

```
   22 write (n6,322) (ldtk(ii),ii=j1,imax)
  322 format( 1x,   10i12   )
  422 j3=j3 + 1
      write (n6,30) ncount,ll,lb
   16 if(j2.ne.0)go to 15
      write(n6,50)in(i),lt
  999 return
   15 ll=in(i)
      lll=ll + ll2 - 1
      kk=kkk
      lb=k(i)
      j=v(i)
      j1 = loc(lll)+j
      ncount = loc(lll+1)-loc(lll)-j
   41 j=0
      if(ivmt.eq.0)go to 2
c   *** variable format control (u,v)
   32 if(ncount.eq.0)go to 1
      j=ncount
      ncc=j1 + ncount - 1
      read(n5,vmt) (d(j2),j2=j1,ncc)
      ivmt=0
      go to 1
c   *** fill array (f)
   23 if(j.ge.ncount)go to 25
      ncc=j+1
      do 24 ii=ncc,ncount
      j2=j1 + ii -1
      d(j2)=v(i)
   24 if(kk.ne.0)ldtk(j2)=v(i)
      j=ncount
      go to 2
   25 write (n6,60) ll,lb
      go to 2
c   *** end array (e)
   39 if(j.le.ncount)j=ncount
      go to 2
c   *** address modification (a)
   26 j=v(i)
      if(j.gt.ncount .or. j.le.0)write(n6,70)j,ll,lb
      j=j-1
      go to 2
c   *** skip (s)
   11 j=j + in(i)
      go to 2
c   *** turn print trigger on/off
   55 iprtrg = 1
      go to 58
   57 iprtrg =-1
   58 write(n6,110)prt(2),prt(iprtrg+2),in(i),lo
      go to 2
c   *** print count in current array
   56 write(n6,90)ll,lb,j,in(i),lc
      go to 2
c   *** no modification
   14 continue
   17 j2=j1 + j
      d(j2)=v(i)
      if(kk.ne.0)ldtk(j2)=v(i)
      j=j + 1
      go to 2
c   *** zero (z)
   31 in(i)=v(i) + in(i)
      v(i)=0.0
c   *** repeat (r)
    7 l=in(i)
      do 18 ii=1,l
      j2=j1 + j
      d(j2)=v(i)
      if(kk.ne.0)ldtk(j2)=v(i)
   18 j=j + 1
      go to 2
c   *** sequence repeat (q) or inverted repeat (g)
   35 l=v(i) + in(i)
      lseq=1
      if(v(i).eq.0.0  .or.  in(i).eq.0)go to 51
      l=v(i)
      lseq=in(i)
   51 do 52 lsq=1,lseq
      do 37 ii=1,l
      j2=j1 + j
      j4=j2 - 1
      d(j2)=d(j4)
      if(k(i).ne.1g)go to 37
      if(kk.ne.0) go to 59
      d(j2) =-d(j4)
      go to 37
   59 ldtk(j2) =-ldtk(j4)
   37 j=j + 1
   52 continue
      go to 2
c   *** inverted sequence repeat (n)
   36 l=v(i) + in(i)
      lseq=1
      if(v(i).eq.0.0  .or.  in(i).eq.0)go to 53
      l=v(i)
      lseq=in(i)
cwar
        if(lseq.gt.1) then
      write (n6,553)
```

```
  553 format( '0fido problem -- n1.gt.0 with m or n option')
      j3 = j3 + 1
        endif
cwar
   53 do 54 lsq=1,lseq
      do 38 ii=1,l
      j2=j1 + j
      j4=j2 - ii - ii + 1
      d(j2)=d(j4)
c    *** reversed sign inverted sequence repeat (m)
      if(k(i).ne.lm)go to 38
      if(kk.ne.0) go to 40
      d(j2) =-d(j4)
      go to 38
   40 ldtk(j2) =-ldtk(j4)
   38 j=j + 1
   54 continue
      go to 2
c    *** logarithmic interpolation (l)
   42 l=in(i) + 1
      vv=v(i)
      iii=2
      go to 2
   44 del=exp(log(v(i)/vv)/l)
      j2=j1 + j
      d(j2)=vv
      j=j + 1
      do 45 ii=2,l
      j2=j1 + j
      d(j2)=del*d(j2-1)
   45 j=j + 1
      if(iii.ne.0)go to 4
      go to 2
c    *** fixed 10 log interp.  (k)
   46 l=in(i) + 1
      vv=v(i)
      iii=3
      go to 2
   48 dif=(v(i)-vv)*.1111111
      del=exp(2.302585/l)
      j2=j1 + j
      d(j2)=vv
      vv=vv-dif
      j=j+1
      do 49 ii=2,l
      j2=j1 + j
      dif=dif*del
      d(j2)=vv + dif
   49 j=j+1
      if(iii.ne.0)go to 4
      go to 2
c    *** interpolate (i)
    8 l=in(i) + 1
      vv=v(i)
      iii=1
      go to 2
    3 del=(v(i)-vv)/float(l)
      do 21 ii=1,l
      j2=j1 + j
      delt=del*float(ii-1)
      d(j2)=vv + delt
        if(kk.ne.0) then
      i1 = vv
      i2 = delt + sign(0.5,del)
      ldtk(j2) = i1 + i2
        endif
   21 j=j + 1
      if(iii.ne.0)go to 4
    2 continue
      go to 1
   20 format( '0', i5, a1,' array', i7,' entries read' )
   30 format( '0****** error ', i7,' entries required in', i3, a1,
     1   ' array' / '0 data edit continues' )
   50 format( '0', i5, a1 )
   60 format( '0****** fill option ignored in', i3, a1, ' array' )
   70 format( '0****** warning  address', i7, ' is beyond limits of',
     1  i3, a1, ' array' )
   80 format( 18a4 )
   90 format( 90x, i5, a1, ' array', i7,' entries read at', i4, a1 )
  110 format( 53x, 2a4, i7, a1 )
      end
:::::::::::::::
fluxr.f
:::::::::::::::
      subroutine fluxr(flux,zcc,iafl,it,cp,cx,cy,cz,enr,denr
     $ ,index,index1,izmw,imw,jmw,kmw,igmw)
c    this routine is to read scalar flux file in varscl format
c    for adjoint the file is fort.26
c    for forward the file is fort.27
c
c    also loads the zone map ijzn into container array
c    future perturbations can be performed by changing the
c    material number of a zone in this array
c
      real*4 flux(index),zcc(index1),cp(izmw),cx(imw+1),cy(jmw+1)
     $ ,cz(kmw+1),enr(igmw+2),denr(igmw)
c
      character hname*8,huse(2)*8,ivers*8
      character date*8,user*8,charge*8,case*8,times*8,titl(9)*8
      dimension iparm(25)
```

```
      dimension izcomp(2000)
      dimension dampg(100)
      dimension ijzn(10000)
c
      character*8 infil
c
      real*4 zero
      zero = 0.e0
c
      if(it.eq.0) infil="fort.26"
      if(it.gt.0) infil="fort.27"
      ntdirf=4
      open(ntdirf,file=infil,status='old',form='unformatted')
      rewind (ntdirf)
      read(ntdirf) hname,(huse(i),i=1,2),ivers
      read(ntdirf) date,user,charge,case, times,(titl(i),i=1,9)
      read(ntdirf) (iparm(i),i=1,25)
c
      if(iparm(5).gt.2000) stop 'fluxr : array size too small .1'
      if(iparm(1).gt.100) stop 'fluxr : array size too small .2'
        id23 = iparm(2)*iparm(3)
      if(id23.gt.10000) stop 'fluxr : array size too small .3'

c
c    discontinuous mesh is not allowed
c
c    integer data is loaded to s array as real.
c    they need to be converted to integer before use.
c
       if(iparm(2).lt.0) then
        write(6,*) 'ERROR: reading varscl format file'
        write(6,*) '       discontinuos mesh is not allowed !!!'
        stop
       endif
       if(it.eq.0.or.(it.gt.0.and.iafl.gt.1)) then
       read(ntdirf) (izcomp(i),i=1, iparm(5))
        do i = 1 , iparm(5)
          cp(i) = float(izcomp(i))
        enddo
c  skip remaining of the record, don't need it.
c     +,(iznrg(i),i=1,iparm(5))
c
c  skip indexing arrays
c
        read(ntdirf)
c       read(ntdirf) (imbis(is),is=1,iparm(7))
c     +,(jmbjs(js),js=1,iparm(8))
c     +,((iset(j,k),j=1,iparm(10)),k=1,iparm(4))
c     +,(jset(k),k=1,iparm(4))
c
c  read mesh info on first call only
c
      read(ntdirf) (cx(i), i = 1, iparm(2)+1)
     $,            (cy(j), j = 1, iparm(3)+1)
     $,            (cz(k), k = 1, iparm(4)+1)
     $,(enr(ig),ig=1,iparm(1)+1)
     $,eneut,(dampg(ig),ig=1,2*iparm(1))
c
c    calculate dener
       do ig=1,iparm(1)
        denr(ig) = enr(ig) - enr(ig+1)
       enddo
       else
c    in proceeding calls make dummy reads (mesh info already available)
       read(ntdirf)
       read(ntdirf)
       read(ntdirf)
       endif

       int23 = iparm(2) * iparm(3)
       n = 0
       do 10 k=1,iparm(4)
       read(ntdirf) (ijzn(i),i=1,int23)
         do i = 1, int23
         n = n + 1
         zcc(n) = float(ijzn(i))
         enddo
10     continue
       int910 = iparm(9) * iparm(10)
        i1 = 1
        i2 = int910
       do 20 ig=1,iparm(1)
        do 30 k=1,iparm(4)
        read(ntdirf) (flux(n) ,n = i1 ,i2)
        i1 = i1 + int910
        i2 = i2 + int910
30     continue
20     continue
       close(4)
       return
       end
::::::::::::::
flxgen.f
::::::::::::::
       subroutine flxgen(flux,shap,ind,amplit)
       real*4 flux(ind),shap(ind),amplit
c
       do i=1,ind
        flux(i) = shap(i)*amplit
       enddo
```

```
c
       return
       end
::::::::::::::
fmgra.f
::::::::::::::
      subroutine fmgra
c     manages input files, sets up array pointers
      common /nacom/ n,ndoub
c
      common /ivcf1/ ivcf1g
      include 'comsd1.F'
      include 'comsd2.F'
      include 'comsd4.F'
      include 'comsd5.F'
      include 'comsd6.F'
      character*1 star
      character*20 usr
      data star/"*"/
c
c   1                              is the pointer for adjoint flux
c   locaa  (1+index)  is the pointer for adjoint angular flux
c   loccp (=locaa + indexm   ) is the pointer for zone data
c   loccx (=loccp + izmw      ) is the pointer for cell mesh data
c   lxsf0 (=loccx + im+jm+km+3) is the pointer for X-sec data (original forward)
c   lxsf1 (=lxsf0 + izart) is the pointer for perturbed forward X-sec data
c   lxsf2 (=lxsf1 + izart) is the pointer for perturbed forward X-sec data
c   locxs (=lxsf2 + izart) is the pointer for adjoint xsecs
c   lxsts (=locxs + izart) is the pointer for ss,total,zone xsecs
c   lxstp (=lxsts + index) is the pointer for ss,total,zone xsecs
c   lxstd (=lxstp + index) is the pointer for ss,total,zone xsecs
c   lxsfs (=lxstd + index) is the pointer for ss,total,zone xsecs
c   lxsfp (=lxsfs + index) is the pointer for ss,total,zone xsecs
c   lxsfd (=lxsfp + index) is the pointer for ss,total,zone xsecs
c   lxsss (=lxsfd + index) is the pointer for ss,total,zone xsecs
c   lxssp (=lxsss + index*igm) is the pointer for ss,total,zone xsecs
c   lxssd (=lxssp + index*igm) is the pointer for ss,total,zone xsecs
c   ldum0 (=lxssd + index*igm) is the pointer used for xsecs in dxsec (ss)
c   ldum1 (=ldum0+ ihp*mtm*igm) is the pointer used for xsecs in dxsec (pert)
c   ldumx (=ldum1+ ihp*mtm*igm) is the pointer used for dummy array in dxsec
c   locvo (=ldumx+ igm*igm ) is the pointer for volume vectors;
c                              izart is length of xsec data
c   locah (=locvo           ) is the pointer for volume vector ah
c   locav (=locah + imw     ) is the pointer for volume vector av
c   locvr (=locav + imw + 1 ) is the pointer for volume vector vr
c   locdth(=locvr + imw     ) is the pointer for volume vector dth
c   locdz (=locdth+ jmw     ) is the pointer for volume vector dz
c   locchi(=locdz + kmw     ) is the pointer for fission spectrum
c   locchs(=locchi+ igmw    ) is the pointer for fixed source spectrum
c   locenr(=locchs+ igmw    ) is the pointer for energy boundaries
c   ldenr (=locenr+ igmw+2  ) is the pointer for group energy width
c   local (=locenr+ igmw    ) is the pointer for adjoint leakage
c   lsnpl(=local + (lmw-1)*mmw+1 ) is the pointer for expansion coefficients
c   loc1  (=lsnpl+ index    ) is the pointer for forward flux
c   loc2  (=loc1  + index   ) is the pointer for forward flux (new)
c   loc3  (=loc2  + index   ) is the pointer for forward flux (new)
c            loc3 has third flux needed for quadraticc interpolation
c   loc4  (=loc3  + index   ) is the pointer for forward flux (new)
c            loc4 has the interpolated flux
c   locsh1(=loc4  + index   ) is the pointer for forward flux shape
c   locsh2(=locsh1+ index   ) is the pointer for forward flux shape (new)
c   locsh3(=locsh2+ index   ) is the pointer for forward flux shape
c   locsh4(=locsh3+ index ) is the pointer for forward flux shape (interpolated)
c   loc1a thru loc4a same as loc1 thru loc4 but contains angular data
c   locsa1 thru locsa4 same as locsh1 thru locsh4 but contains angular data
c   loc1c (=locsh4 + index ) is the pointer for mat. # by cell
c   loc2c (=loc1c + index1) is the pointer for mat. # by cell (new)
c   loc3c (=loc2c + index1) is the pointer for mat. # by cell (new)
c   loc4c (=loc3c + index1) is the pointer for mat. # by cell (new)
c   lchit (=loc4c + index1) is the pointer for chit
c   locpd (=lchit + igmw ) is the pointer for power densities
c   lrhot3 (=locpd + index1) is the poinetr for space used in calculation of rho
c   lrhot4 (=lrhot3 + index) is the poinetr for space used in calculation of rho
c   lrhot5 (=lrhot4 + index) is the poinetr for space used in calculation of rho
c   lcig0  (=lrhot5 + index) is the pointer for initial prec. densities cig0
c   lcig1  (=lcig0 + index1*ndg) is the pointer for prec. densities
c   lcig2  (=lcig1 + index1*ndg) is the pointer for prec. densities
c   lcig4  (=lcig2 + index1*ndg) is the pointer for interpolated prec. densities
c   lrp1   (=lrhot5 + index1*ndg) is the pointer for space used in calculation of rp1
c   lrp2   (=lrp1 + ndg) is the pointer for space used in calculation of rp2
c   lrp3   (=lrp2 + ndg) is the pointer for space used in calculation of rp3
c            rp1,rp2,and rp3 appear in precursor equation
c   locamp (=lrp3 +ndg ) is the pointer for amplitutes calculated
c   lrxsec (=locamp + idtkm+1) is the poineter for xsecs in rdlgip
c   lchi0  (=lrxsec + izart) is the pointer for original chi
c   locq   (=lchi0 + igmw) is the pointer for space used for distributed source
c   locqs  (=locq  + ) is the pointer for space used for modified
c                          scalar distributed source
c   locqm  (=locqs + index) is the pointer for space used for modified
c                          distributed source moments
c   locpp (=lfluxa + index*mm) is the pointer for power histories
c   locap (=locpp + gak) is the pointer for amplitute histories
c   loce  (=locap + gak) is the end of array
c
c
c   ivcflg is flag for volume vectors transfer
c
c   index  is (xinterval*yinterval*zinterval*numberofgroups)
c   index1 is (index/numberofgroups)
c   indexm is (index*lm)
```

```
c
c    rdnp is the common block name for real delayed neutron parameters
c    idnp is the common block name for integer delayed neutron parameters
c    ijkmgw is the common block name for integer # of mesh intervals in
c       x, y, z directions.
c
c   n6 , output unit
c   n5in, input unit
c   n6sed, output unit for transient file
c   n6tor, output unit for tort (forward/adjoint) input file
c   n6fi, output unit to redirect fidas output to
c   n6sc, scratch
c   n6pk, point kinetics data output unit
c
c  set defaults
c
      ivcflg = 0
      n6     = 6
      n5in   = 21
      n6sed  = 22
      n6tor  = 23
      n6fi   = 6
      n6sc   = 20
      n6pk   = 30
      ifxprtw = 1
      ntfoga = 0
      ntfogf = 0
      ntflxf = 0
      ntsigf = 8
      ntsigp = 9
      ntsigp2 = 11
      nts    = 10
      ntsiga = 18
      ntscla = 26
      ntsclf = 27
      ntdira = 0
      ntdirf = 0
      ntdsif = 31
      ntzonf = 28
      idjlek = 47
      modepw = 1
c   open output file and write execution information
      open(n6sed,file='transin',status='unknown')
      call info(n6sed,usr)
      write(n6sed,*)
      write(n6sed,*)
      call fhlpr(' ',' tdtort ',n6sed)
      write(n6sed,*)
      write(n6sed,*)
      call fhlpr(' ','  v1.0  ',n6sed)
      write(n6sed,*)
      write(n6sed,*)
c   find username length
      do i=1,20
       if(usr(i:i).eq.' ') then
        ilen = i
       goto 10
       endif
      enddo
c
10    if(i.gt.8) goto 20
      il = (8 - ilen)/2
c   shift
      if(il.gt.0) then
        do i=1,8-il
         usr(8-i+1:8-i+1) = usr(8-i+1-il:8-i+1-il)
        enddo
c   clean the first il characters
        do i=1,il
         usr(i:i) = ' '
        enddo
      endif
20    call fhlpr(' ',usr(1:8),n6sed)
      write(n6sed,*)
c      write(n6sed,*)
c      call fhlpr(' ','goluoglu',n6sed)
c      write(n6sed,*)
c
c
c   the next call is to read input file inputa and create two output
c   files which will be input files for tort and transsient calculations
c
      j3=0
      call reader(j3)
        if(j3.eq.98) then
        write(n6,*) 'error: abnormal file inputa please fix it....'
        goto 999
        endif
c
c   files inputs and tortinf are created
c
c    the next call is to read the file tortinf and from that to create
c    the adjoint calculation input file tortina.
c
      write(n6,*) 'creating tortina .......'
      j3=0
      call readet(j3,0)
       if(j3.eq.99) then
        write(n6,*) 'error: a forward calculation must be specified
     $ to resume execution.....'
```

```
        goto 999
       endif
       if(j3.eq.98) then
        write(n6,*) 'error: abnormal file tortinf please fix it....'
        goto 999
       endif
       if(j3.ne.0) stop17
c
c     the file tortina is created
c
c  the next call is to do same changes to tortinf
      j3=0
      call readet(j3,1)
c
c     the next call is to get the number of neutron energy groups
c     from the file tortinf and add it to the file transin (niff)
c
      write(n6,*) 'creating tortinf .......'
      j3=0
      call readev(j3)
       if(j3.eq.98) then
        write(n6,*) 'error: abnormal file tortinf please fix it....'
        goto 999
       endif
       if(j3.ne.0) stop17
c
c    the next call is to read inputs and write the transient info
c    to the file transin (not in fido format=niff)
c    it also returns lambdas and betas.
c
      j3=0
      call reades(j3,ierrfl)
       if(j3.ne.0) then
        write(n6sed,900) (star,j=1,103)
        write(n6,900) (star,j=1,103)
        write(n6,*) 'error in reading the file inputs.....'
        write(n6,*) ' in input BLOCK ',ierrfl
        write(n6sed,900) (star,j=1,103)
        write(n6,900) (star,j=1,103)
        goto 999
       endif
c
900   format(1x,103(a1))
c
c   file transin is created
c
c  set array pointers
c
      index1 = imw * jmw * kmw
      index  = index1 * igmw
c   indexm is not needed
      indexm = index * mmw
      ndoub = 1
      n     = index  + 1
      locaa = na(indexm)
      loccp = na(izmw)
      loccx = na(imw+1)
      loccy = na(jmw+1)
      loccz = na(kmw+1)
      ihpw  = ihmw + 1
c   if iups=0 then ihpw does not exist
      if(iupsw.eq.0) ihpw=ihmw
      izart = ihpw*ncrxw*(nsctmw+1)*igmw
c
      lxsf0 = na(izart)
      lxsf1 = na(izart)
      lxsf2 = na(izart)    ! to save old perturbed xsecs for movem=6
      locxs = na(izart)
      lxsts = na(index)
      lxstp = na(index)
      lxstd = na(index)
      lxsfs = na(index)
      lxsfp = na(index)
      lxsfd = na(index)
      lxsss = na(index*igmw)
      lxssp = na(index*igmw)
      lxssd = na(index*igmw)
      lxsas = na(index)
      lxsap = na(index)
c
      mtmw = ncrxw*(nsctmw+1)
      ldum0 = na(ihpw*mtmw*igmw)
      ldum1 = na(ihpw*mtmw*igmw)
      ldumx = na(mtmw*igmw*igmw)
c
c     ihpw is
c     ncrxw is # of matls
c     each matl has nsctmw+1 components
c     xsec data is read in adjoint and first forward calculation only
c     izart is the length of xsec data for all groups
c     first  izart is adjoint xsecs
c     second izart is forward xsecs (most recent)
c     first  izart/igmw is group1
c     second izart/igmw is group2
c     etc.
c
      locvo = na(imw)
      locah = locvo
      locav = na(imw+1)
      locvr = na(imw)
```

```
      locdth= na(jmw)
      locdz = na(kmw)
      locchi= na(igmw)
      locchs= na(igmw)
      locenr= na(igmw+2)
      ldenr = na(igmw)
      local = na(index)
c      idimsnpl = mmw*(lmw-1) + 1   ! 1 extra is to make sure snpl array
      idimsnpl = mmw*lmw ! only mmw*(lmw-1) terms are used
      lsnpl= na(idimsnpl)           ! has nonzero dimension
c   note that volume(i,j,k) = vr(i)*dth(j)*dz(k)
c
c  the remaining is disposable
c  (new data replaces old as soon as calculated)
      if(movem.ne.0.and.movem.ne.1) then
      loc1  = na(index)
      loc2  = na(index)
      endif
      locsh1= na(index)
      locsh2= na(index)
      locsh3= na(index)
      locsh4= na(index)
c   angular flux and shape arrays
      if(movem.ne.0.and.movem.ne.1) then
      loc1a = na(index*mmw)
      loc2a = na(index*mmw)
      endif
      locsa1= na(index*mmw)
      locsa2= na(index*mmw)
      locsa3= na(index*mmw)
      locsa4= na(index*mmw)
c
      loc1c = na(index1)
c      loc2c = na(index1)
c      loc3c = na(index1)
c      loc4c = na(index1)
c
c    set up kinetics working array pointers
c
      lchit  = na(igmw)
      locpd  = na(index1)
      lrhot3 = na(index)
      lrhot4 = na(index)
      lrhot5 = na(index)
      indexn = index1*ndg
      lcig0  = na(indexn)
      lcig1  = na(indexn)
      lcig2  = na(indexn)
      lddum = 0
      do ii=1,idtnm
      ldamp = max(idtj(ii),lddum)
      lddum = ldamp
      enddo
      idtjm = ldamp
      locamp = na(ldamp+1)
      lrxsec = na(izart)
      lchi0  = na(igmw)
      locqm  = na(index*lmw)
c   determine max tk calculation
      lddum = 0
      do ii=1,idtnm
       if(movem.eq.0.or.movem.eq.1) then
      mdamp = max(idtk(ii)/idtj(ii),lddum)
       else
      mdamp = max(idtk(ii),lddum)
       endif
      lddum = mdamp
      enddo
      mdamp = mdamp + 1
c  this is for feedback
      if(movem.eq.0) then
      ltempold  = na(index1)
      ltemprp   = na(index1)
      ltmpz     = na(mtmw)
      ltmpzrp   = na(mtmw)
c      lpowxtl = na(index1)
      lpowxt  = na(index1)
      lpowxb  = na(index1)
      lpowzt  = na(mtmw)
      lpowzb  = na(mtmw)
       luxsfp  = na(izart)
      lvolz   = na(mtmw)
      endif
c
      if(movem.eq.1) then
       lxsfnew = na(ihpw*igmw)
       luxsfn  = na(ihpw*igmw)
       luxsfp  = na(izart)
      endif
c
      loce   = n
c
c   create input file for running with zone map file
c
      imany = 0
      j3 = 0
      call readw(j3,imany)
      if(j3.ne.0) then
       write(n6,*) 'error in reading file tortinf or writing tortinw'
       goto 999
```

```
      endif
      return
999   stop
      end
::::::::::::::
giper.f
::::::::::::::
      subroutine giper(xsecsa,xsecs,xsecp,ihp,mtm,igm,evw)
      real*4 xsecs(ihp,mtm,igm),xsecsa(ihp,mtm,igm),xsecp(ihp,mtm,igm)
c
      iht=3
      open(56,form='unformatted',status='unknown')
      open(57,form='unformatted',status='unknown')
      open(58,form='unformatted',status='unknown')
c
      do ig=1,igm
      do mt=1,mtm
c   adjust fission  to maintain criticality
      xsecs(iht-1,mt,ig)    = xsecs(iht-1,mt,ig)/evw
      xsecsa(iht-1,mt,ig)   = xsecsa(iht-1,mt,ig)/evw
      xsecp(iht-1,mt,ig)    = xsecp(iht-1,mt,ig)/evw
      enddo
      enddo
c
      do 10 ig=1,igm
       write(56) ((xsecsa(ih,mt,ig),ih=1,ihp),mt=1,mtm)
       write(57) ((xsecs(ih,mt,ig),ih=1,ihp),mt=1,mtm)
       write(58) ((xsecp(ih,mt,ig),ih=1,ihp),mt=1,mtm)
10    continue
c
      return
      end
::::::::::::::
giper2.f
::::::::::::::
      subroutine giper2(xsecp,ihp,mtm,igm,evw)
      real*4 xsecp(ihp,mtm,igm)
c
      iht=3
      open(55,form='unformatted',status='unknown')
      do ig=1,igm
      do mt=1,mtm
c   adjust fission  to maintain criticality
      xsecp(iht-1,mt,ig)    = xsecp(iht-1,mt,ig)/evw
      enddo
      enddo
c
      do 10 ig=1,igm
       write(55) ((xsecp(ih,mt,ig),ih=1,ihp),mt=1,mtm)
10    continue
c
      return
      end
::::::::::::::
gtermr.f
::::::::::::::
      subroutine gtermr(gofr,phi,sigfs,index,index1,igmw)
c     performs integration over energy only
c     sigfs = sigf  if called from rhocal
c           = dsigf if called from rhot5r
c           = dsigs if called from rhot4r
      real*4 gofr(index1),phi(index),sigfs(index)
      real*4 zero
      zero = 0.e0
c     note gofr does not depend on energy
c     and i1 goes from 1 to index
      do 70 i=1,index1
       gofr(i) = zero
       do 80 ig=1,igmw
        i1 = (ig-1)*index1+i
        gofr(i) = gofr(i)+phi(i1)*sigfs(i1)
80     continue
70    continue
      return
      end
::::::::::::::
info.f
::::::::::::::
      subroutine info(n6,usr)
      integer hostnm,getcwd,getlog,status,n,time
      character*24 ctime,t
      character*100 dir,mac
      character*20  usr
c
      n = time()
      t = ctime( n )
      status = hostnm( mac )
      status = getcwd( dir )
      status = getlog( usr )
      write(n6,'("************************************************")')
      write(n6,'("  Execution started on ",a24)') t
      write(n6,*)'  Workstation Information '
      write(n6,*)'------------------------'
      write(n6,'("  Machine   name  = ",a30)') mac
      write(n6,'("  Directory name  = ",a100)') dir
      write(n6,'("  User      name  = ",a20)') usr
      write(n6,'("************************************************")')
      write(6,'("************************************************")')
      write(6,'("  Execution started on ",a24)') t
      write(6,*)'  Workstation Information '
```

```
      write(6,*)'-------------------------'
      write(6,'(" Machine   name  = ",a30)') mac
      write(6,'(" Directory name  = ",a100)') dir
      write(6,'(" User      name  = ",a20)') usr
      write(6,'("*************************************************")')
c
      return
      end
::::::::::::::
inical.f
::::::::::::::
      subroutine inical(xsfs,vr,dth,dz,pd,cig0,cig1,anshl
     $ ,scadj,anadj,shap1,cx,cy,cz,xsf0,rxsec,qmom,dume,chi
     $ ,enr,z1c,cp,denr,xstd,xssd,xsfd,chit,xsfp,dume1,dume2,dume3
     $ ,dume4,rhot3,rhot4,rhot5)
c
      common sweight(320)
      real*4 sweight
c
      include 'comsd1.F'
      include 'comsd2.F'
      include 'comsd4.F'
      include 'comsd5.F'
      include 'comsd7.F'
c
      real*4 xsfs(index),vr(imw),dth(jmw),dz(kmw)
     $      ,pd(index1),cig0(indexn),cig1(indexn)
     $      ,anshl(indexm),scadj(index),anadj(indexm),shap1(index)
     $      ,cx(imw+1),cy(jmw+1),cz(kmw+1),xsf0(izart),rxsec(izart)
     $      ,qmom(index*lmw),dume(irest),chi(igmw),enr(igmw+2)
     $      ,z1c(index1),cp(izmw),denr(igmw)
c   for pkvar
      real*4 xstd(index),xssd(index*igmw),xsfd(index),chit(igmw)
     $      ,xsfp(index)
     $      ,dume1(index1),dume2(index),dume3(index1),dume4(index)
     $      ,rhot3(index),rhot4(index),rhot5(index)
c
      character*1 equal
      logical outflg
c
       character*10 xnam
c
      real*4 zero
      zero = 0.e0
c
       xnam = 'tortnew '
c
c      calculate initial total power
c
       equal = '='
c
      evw = evnew
      outflg = .false.
      if(iout.eq.1) outflg = .true.
      pow0 = zero
      call powcal(pow0,shap1,index,index1,igmw,xsfs
     $ ,vr,dth,dz,imw,jmw,kmw,rnu,pd)
c
c     normalize the fluxes to give total power=powini
      if(powini.gt.0.) then
      call normaf(shap1,index,anshl,indexm,pow0,powini)
      call powcal(pown,shap1,index,index1,igmw,xsfs
     $ ,vr,dth,dz,imw,jmw,kmw,rnu,pd)
      pow0 = pown
c  also update the interface files if not restart
      if(irstrt.eq.0) then
      open(35,file='shapet',status='unknown',form='unformatted')
      write(35) (shap1(i),i=1,index)
      close(35)
      open(35,file='ashapet',status='unknown',form='unformatted')
      write(35) (anshl(i),i=1,indexm)
      close(35)
      endif
      endif
      call cigint(shap1,beta,alambda,ndg,igmw,index1
     $ , cig0, cig1,index, xsfs)
c       call wout(cig0,indexn,6,0)
c
c     normalize the shape
c
        if(ipkmod.eq.0) then
      call ampil(anshl,anadj,vinv,index,index1,igmw,iqs,0
     $ ,vr,dth,dz,imw,jmw,kmw,amplit0,shap0,indexm,mmw,sweight,0,n6pk,0)
      call ampilh(shap1,scadj,vinv,index,index1,igmw,iqs,0
     $ ,vr,dth,dz,imw,jmw,kmw,amplit0,shap0,0,n6pk,0)
        elseif(ipkmod.eq.1) then
      call ampilh(shap1,scadj,vinv,index,index1,igmw,iqs,0
     $ ,vr,dth,dz,imw,jmw,kmw,amplit0,shap0,0,n6pk,0)
        endif
c
c   print flux shape to a file
      call woutfl(shap1,index,0,48,"fs0.0",cx,cy,cz,imw,jmw,kmw,igmw
     $ ,vr,dth,dz)
      call woutfl(pd,index1,1,49,"pd0.0",cx,cy,cz,imw,jmw,kmw,igmw
     $ ,vr,dth,dz)
c
      write(n6pk,'(120(a1))') (equal,itest=1,115)
      write(n6pk,*)'Initial power        = ',pow0,' fissions/sec'
      write(n6pk,*)'Initial norm.d power = ',pown,' fissions/sec'
      write(n6pk,*)'Initial amplitude    = ',amplit0
```

```
      write(n6pk,*)'Initial shape        = ',shap0
     $ , ' (volume and energy integrated)'
      write(n6pk,*)'Initial eigenvalue     = ',evnew
c
c    now have f0, c0, p0=powini
c
      idim   = ndg
      call pkvar(rho0,fterm0,betaef0,betaeft,rlamda0,extsrc0
     $ ,idim,shap1,ansh1,rhot3,rhot4,rhot5,anadj,xstd,scadj,xssd
     $ ,dume1,xsfd,chit,dume2,vr,dth,dz,dume3,xsfp,dume4
     $ ,index,index1,indexm,imw,jmw,kmw,igmw,mmw,ipkmod)
c
      call pdcal(csubi0,rlamda0,fterm0,idim,cig0,scadj,dume2,vr,dth,dz
     $ ,imw,jmw,kmw,igmw,index,indexn,index1,chid)
c
      write(n6pk,'(120(a1))') (equal,itest=1,115)
      write(n6pk,900) rho0,rho0/betaeft
      write(n6pk,910) (betaef0(i),i=1,ndg)
      write(n6pk,920) betaeft
      write(n6pk,930) (csubi0(i),i=1,ndg)
      write(n6pk,940) rlamda0
c      write(n6pk,*) ' fterm        = ',fterm0
      write(n6pk,'(120(a1))') (equal,itest=1,115)
900   format(1x,44h Reactivity                                    = ,1pe13.6
     $ ,2x,1h(,e13.6,2h$))
910   format(1x,44h Effective Delayed Neutron Fractions           = ,1p
     $ ,6(2x,e10.3))
920   format(1x,44h Effective Total Delayed Neutron Fraction      = ,1pe11.4)
930   format(1x,44h Weighted Precursor Densities                  = ,1p
     $ ,6(2x,e11.4))
940   format(1x,44h Generation Time                               = ,1pe13.6)
c
c  write original zone info to file
      open(35,file='ozones',status='unknown',form='unformatted')
      do k=1,kmw
      i1 = (k-1)*(imw*jmw) +1
      i2 = k*imw*jmw
      write(35) (z1c(i),i=i1,i2)
      enddo
      close(35)
c
      return
      end
c234567890123456789012345678901234567890123456789012345678901234567890123456789012
::::::::::::::
jobw0.f
::::::::::::::
c234567890123456789012345678901234567890123456789012345678901234567890123456789012
      subroutine jobw0(iot,cig0,scadj,vr,dth,dz,z1c,xsts,xstp,xstd,xsfs
     $ ,xsfp,xsfd,xsss,xssp,xssd,xsf0,xsf1,cp,dum0,dum1,dumx
     $ ,rxsec,chi,chit,qmom,cx,cy,cz,enr,denr,anadj,snpl,pd,amp
     $ ,shap1,shap2,shap3,shap4,ansh1,ansh2,ansh3
     $ ,ansh4,cig1,cig2,dume1,dume2,amptk,dumep1,dumep2,dumep3
     $ ,dumep4,dume,rhot3,rhot4,rhot5,xsf2,tempold,xsas,xsap
     $ ,powxt,powxb,uxsfp,temprp,tempiz,tempizrp,powzt,powzb,volz)
c
      include 'comsd1.F'
      include 'comsd2.F'
      include 'comsd4.F'
      include 'comsd5.F'
      include 'comsd6.F'
      include 'comsd7.F'
      include 'comsd8.F'
      include 'comsd0.F'
      include 'comsdc.F'
      include 'comsdx.F'
c
c234567890123456789012345678901234567890123456789012345678901234567890123456789012
      real*4 cig0(indexn),scadj(index),vr(imw),dth(jmw),dz(kmw)
     $ ,z1c(index1),xsts(index),xstp(index),xstd(index),xsfs(index)
     $ ,xsfp(index),xsfd(index),xsss(index*igmw),xssp(index*igmw)
     $ ,xssd(index*igmw),xsf0(izart),xsf1(izart),cp(izmw),dum0(izart)
     $ ,dum1(izart),dumx(igmw*igmw*mtmw)
     $ ,rxsec(izart),chi(igmw),chit(igmw),qmom(index*lmw),cx(imw+1)
     $ ,cy(jmw+1),cz(kmw+1),enr(igmw+2),denr(igmw),anadj(indexm)
     $ ,snpl(idimsnpl),pd(index1),amp(ldamp+1)
     $ ,shap1(index),shap2(index),shap3(index),shap4(index)
     $ ,ansh1(indexm),ansh2(indexm)
     $ ,ansh3(indexm),ansh4(indexm),cig1(indexn),cig2(indexn)
     $ ,dume1(2*index),dume2(2*index*ndg),amptk(mdamp+1)
     $ ,dume(irestx),xsf2(izart),tempold(index1)
     $ ,xsas(index),xsap(index),powxt(index1),powxb(index1)
     $ ,uxsfp(izart),temprp(index1),tempiz(mtmw),tempizrp(mtmw)
     $ ,powzt(mtmw),powzb(mtmw),volz(mtmw)
c
      character*7 filned
      character*8 filnedz
c
c   for pkvar
      real*4 dumep1(index1),dumep2(index),dumep3(index1)
     $ ,dumep4(index),rhot3(index),rhot4(index),rhot5(index)
c
      real*4 telaps,rho,rlamda,extsrc,fterm,power,bett,hhi1
     $ ,rho2rp,rho1rp,rhonrp,rlam2rp,rlam1rp,rlamnrp,bet2rp,bet1rp
     $ ,ampn,dampndt
      real*4 zero
      zero = 0.e0
      iot = n6pk
      xnam = 'tortnew '
c
```

```
c
c    iquad =  quadratic interpolation flag for flux
c    jquad =  quadratic interpolation flag for rho
c
c   note that z1c is for initial zone map
c    idtnm = # of tn calc.
c
       equal = '='
c   open output unit
      open(n6pk,file='pkout',status='unknown')
      call xsetun(n6pk)
c
c   rhonew is newest
c   rho1   is previous
c   rho2   is oldest
c
c    betaeft is sum(betaef(i))
c
c    initialize
c
      ijamp = 0
      neq = ndg+1
      itol = 1
      ilsrto = 0
      if(srtol.lt.0..or.satol.lt.0.) then
       ilsrto = 1
      endif
      rtol = 1.0d0*abs(srtol)
      atol = 1.0d0*abs(satol)
      itask = 1
      istate = 1
      iopt = 0
      lrw = 500
      liw = 100
      mf = 22
      t    = zero
      tout= zero
      telaps = zero
c
c     reset irepmx so that 1 extra iteration is performed by default
      irepmx = irepmx + 1
      evw = evnew
      outflg = .false.
      if(iout.gt.0) outflg = .true.
      rho1    = zero
      rho2    = zero
      rhonew = zero
      rho     = zero
      rlam1    = zero
      rlam2    = zero
      rlamnew = zero
      rlamda   = zero
      q1       = zero
      q2       = zero
      qnew    = zero
      extsrc = zero
      fterm   = zero
      power   = zero
      do i=1,ndg
       betae1(i) = zero
       betae2(i) = zero
       betnew(i) = zero
       betaef(i) = zero
       csubi(i) = zero
      enddo
      bet1   = zero
      bet2   = zero
      betn   = zero
      bett   = zero
c
c   set initial temp dist.
c
      do i=1,index1
       tempold(i) = 300.0
      enddo
c
      do i=1,mtmw
       tempiz(i) = 300.0
      enddo
c
      call zoneinfo(xsfs,z1c,cp,imw,jmw,kmw,index,index1,igmw,izmw)
c   normalize and save initial zone power
      econfr = 3.204e-11
c   assume initial amplitude is 1.0 and normalize to get power=1e-6
      call normap(shap1,ansh1,xsfs,imw,jmw,kmw,igmw,indexm
     $ ,index,rnu,econfr,vr,dth,dz)
      call powcal(pown,shap1,index,index1,igmw,xsfs
     $ ,vr,dth,dz,imw,jmw,kmw,rnu,pd)
      pown = pown*econfr
      pow0 = pown
c   also update the interface files if not restart
      if(irstrt.eq.0) then
      open(35,file='shapet',status='unknown',form='unformatted')
      write(35) (shap1(i),i=1,index)
      close(35)
      open(35,file='ashapet',status='unknown',form='unformatted')
      write(35) (ansh1(i),i=1,indexm)
      close(35)
      endif
      call cigint(shap1,beta,alambda,ndg,igmw,index1
```

```
     $ , cig0, cig1,index, xsfs)
c
       if(ipkmod.eq.0) then
      call ampil(ansh1,anadj,vinv,index,index1,igmw,iqs,0
     $ ,vr,dth,dz,imw,jmw,kmw,amplit0,shap0,indexm,mmw,sweight,0,n6pk
     $ ,0)
      call ampilh(shap1,scadj,vinv,index,index1,igmw,iqs,0
     $ ,vr,dth,dz,imw,jmw,kmw,amplit0,shap0,0,n6pk,0)
         elseif(ipkmod.eq.1) then
      call ampilh(shap1,scadj,vinv,index,index1,igmw,iqs,0
     $ ,vr,dth,dz,imw,jmw,kmw,amplit0,shap0,0,n6pk,0)
c
c   also normalize angular shape for qaddj
c
       do i=1,indexm
         ansh1(i) = ansh1(i)/ratio(1)
       enddo
         endif
c
       call woutfl(shap1,index,0,48,"fs0.0",cx,cy,cz,imw,jmw,kmw,igmw
     $ ,vr,dth,dz)
       call woutfl(pd,index1,1,49,"pd0.0",cx,cy,cz,imw,jmw,kmw,igmw
     $ ,vr,dth,dz)
c
       write(n6pk,*)'Initial power        = ',pow0,' fissions/sec'
       write(n6pk,*)'Initial norm.d power = ',pown,' fissions/sec'
       write(n6pk,*)'Initial amplitude    = ',amplit0
       write(n6pk,*)'Initial shape        = ',shap0
     $ , ' (volume and energy integrated)'
       write(n6pk,*)'Initial eigenvalue     = ',evnew
c
c    now have f0, c0, p0=powini
c
       idim   = ndg
       call pkvar(rho0,fterm0,betaef0,betaeft,rlamda0,extsrc0
     $ ,idim,shap1,ansh1,rhot3,rhot4,rhot5,anadj,xstd,scadj,xssd
     $ ,dumep1,xsfd,chit,dumep2,vr,dth,dz,dumep3,xsfp,dumep4
     $ ,index,index1,indexm,imw,jmw,kmw,igmw,mmw,ipkmod)
c
       call pdcal(csubi0,rlamda0,fterm0,idim,cig0,scadj,dume2,vr,dth,dz
     $ ,imw,jmw,kmw,igmw,index,indexn,index1,chid)
c
c
       write(n6pk,'(120(a1))') (equal,itest=1,115)
       write(n6pk,900) rho0,rho0/betaeft
       write(n6pk,910) (betaef0(i),i=1,ndg)
       write(n6pk,920) betaeft
       write(n6pk,930) (csubi0(i),i=1,ndg)
       write(n6pk,940) rlamda0
c       write(n6pk,*) ' fterm          = ',fterm0
       write(n6pk,'(120(a1))') (equal,itest=1,115)
900    format(1x,44h Reactivity                                    = ,1pe13.6
     $ ,2x,1h(,e13.6,2h$))
910    format(1x,44h Effective Delayed Neutron Fractions           = ,1p
     $ ,6(2x,e10.3))
920    format(1x,44h Effective Total Delayed Neutron Fraction      = ,1pe11.4)
930    format(1x,44h Weighted Precursor Densities                  = ,1p
     $ ,6(2x,e11.4))
940    format(1x,44h Generation Time                               = ,1pe13.6)
c
       call rkld0(betaef0,alambda,rlamda0,n6pk,ndg,amplit0,y)
c   this is needed to keep initial values of y vector
c   the last arg. is : 1= y to resrep , otherwise resrep to y
       call rkldrp(y,resrep,neq,1)
c
c   convert the edit times to edit shape calc number (itn)
c
       do ixe=1,ieditn
         do ix=1,idtnm
           if(tedit(ixe).le.dtns(ix)) then
             nedit(ixe) = ix
             goto 10
           endif
         enddo
10       continue
       enddo
c
       rhonew = rho0
       rlamnew = rlamda0
       betn = zero
       do ii=1,ndg
       betnew(ii) = betaef0(ii)
       betn = betn + betnew(ii)
       enddo
c
       iquad = .false.
c      start dtn time loop
       do 800 itn = 1, idtnm
c    set lsode parameters
       if(ilsrto.eq.0) then
        rtol = dtn(itn)/idtk(itn)*srtol
        atol = dtn(itn)/idtk(itn)*satol
       endif
       if(ilsrto.eq.0.or.itn.eq.1) then
        write(n6pk,*) '*********************************'
        write(n6pk,*) ' rtol = ',rtol,' at time interval ',itn
        write(n6pk,*) ' atol = ',atol,' at time interval ',itn
        write(n6pk,*) '*********************************'
       endif
c
```

```
      irep = 0      ! this is to repeat pk calc. and update
      hi1 = dtn(itn)
      hi2 = dtn(itn)
c
c   shift fluxes
      call shftfl(shap1,shap2,shap3,ansh1,ansh2,ansh3,index,indexm)
c
      if(itn.eq.1) then
        amptk(1) = amplit0
        amp(1) = amplit0
        rlamda = rlamda0
        fterm  = fterm0
       hhi1=zero
c  save last amplitude as first for next time step
       ampold = amplit0
       amp0   = amplit0
      else
        rlamda = rlamold
        fterm  = ftermold
       hhi1 = dtn(itn-1)
      endif
c    generate precursor densities using last calculated delayed
c    neutron source and load onto y array
c
c   this next call causes discontinuities in y b/c csubi does not
c   match the value calculated by lsode
c   use with caution
      call rkld(y,rlamda,fterm,n6pk,ndg,cig2,scadj,imw,jmw,kmw,igmw
     $ ,vr,dth,dz,indexn,index,itn,index1,dume1,chid,ampold,amp0,amplit)
c
c   extrapolate to calculate first flux shape and normalize
c   extrapolate angular and scalar shape
c
        if(ipkmod.eq.0) then
      call extrap(ansh1,ansh2,ansh3,hhi1,hi2,indexm,iextrp,n6pk)
      call extrap(shap1,shap2,shap3,hhi1,hi2,index,iextrp,n6pk)
      call ampil(ansh1,anadj,vinv,index,index1,igmw,iqs,0
     $ ,vr,dth,dz,imw,jmw,kmw,amplit,shap,indexm,mmw,sweight,1,n6pk,itn)
      call ampilh(shap1,scadj,vinv,index,index1,igmw,iqs,0
     $ ,vr,dth,dz,imw,jmw,kmw,amplit,shap,1,n6pk,itn)
        elseif(ipkmod.eq.1) then
      call extrap(shap1,shap2,shap3,hhi1,hi2,index,iextrp,n6pk)
      call ampilh(shap1,scadj,vinv,index,index1,igmw,iqs,0
     $ ,vr,dth,dz,imw,jmw,kmw,amplit,shap,1,n6pk,itn)
        endif
c
c    idtj = # of tj calc.
710   jquad = .false.
c
c    clear rwork for lsode
      do itt2=1,500
         rwork(itt2) = 0.0d0
      enddo
      do itt2=1,100
         iwork(itt2) = 0
      enddo
      if(irep.eq.0) then
c   svae old react.,gen. time and effective betas for second pass
      call rkldrp(y,resrep,ndg+1,1)
        do ix=1,index1
          temprp(ix) = tempold(ix)
        enddo
        do ix=1,mtmw
          tempizrp(ix) = tempiz(ix)
        enddo
        do ix=1,indexn
          cig0(ix) = cig1(ix)
        enddo
        amptkrp = amptk(1)
        rho2rp = rho2
        rho1rp = rho1
        rhonrp = rhonew
        rlam2rp = rlam2
        rlam1rp = rlam1
        rlamnrp = rlamnew
        bet2rp = zero
        bet1rp = zero
        betnrp = betn
        do i=1,ndg
        bete2rp(i) = betae2(i)
        bete1rp(i) = betae1(i)
        betenrp(i) = betnew(i)
        bet2rp     = bet2rp + bete2rp(i)
        bet1rp     = bet1rp + bete1rp(i)
        enddo
      elseif(irep.gt.0) then
c
c   put old data back
       call rkldrp(y,resrep,neq,0)
        do ix=1,index1
          tempold(ix) = temprp(ix)
        enddo
        do ix=1,mtmw
          tempiz(ix) = tempizrp(ix)
        enddo
        do ix=1,indexn
          cig1(ix) = cig0(ix)
        enddo
        amptk(1) = amptkrp
        rho2 = rho2rp
```

```
        rho1 = rho1rp
        rhonew = rhonrp
        rlam2 = rlam2rp
        rlam1 = rlam1rp
        rlamnew = rlamnrp
        bet2 = zero
        bet1 = zero
        betn = betnrp
        do i=1,ndg
        betae2(i) = bete2rp(i)
        betae1(i) = bete1rp(i)
        betnew(i) = betenrp(i)
        bet2    = bet2 + betae2(i)
        bet1    = bet1 + betae1(i)
        enddo
      endif
c
c     start dtj time loop
      lamp = 0
      dtj = zero
      dtj = dtn(itn)/idtj(itn)
c   open and close fort.45 and fort.46 to force buffer dump
      if(itn.eq.1.and.irep.eq.0) then
        open(45,status='unknown')
        open(46,status='unknown')
      else
        open(45,status='old',access='append')
        open(46,status='old',access='append')
      endif
c
      dttot = zero
      if(itn.gt.1) dttot = dtns(itn-1)
c
      do 700 itj=1, idtj(itn)
c   transfer the times for irep = 0 case
      if(irep.eq.0.and.itj.eq.1.and.irepmx.gt.0) then
        ttrep1 = telaps
        ttrep2 = tout
        ttrep3 = t
      elseif(irep.gt.0.and.itj.eq.1.and.irepmx.gt.0) then
        telaps = ttrep1
        tout   = ttrep2
        t      = ttrep3
      endif
      hj1    = dtj
      hj2    = dtj
      ijamp = ijamp + 1
      if(itj.gt.2)    jquad = .true.
      if(itj.eq.1) istate = 1
c
      tnewj = itj*dtj
c
c   linear interp.
      if(ipkmod.eq.0) then
      call linter(ansh2,ansh1,ansh4,hi2,tnewj,indexm)
      call linter(shap2,shap1,shap4,hi2,tnewj,index)
      elseif(ipkmod.eq.1) then
      call linter(shap2,shap1,shap4,hi2,tnewj,index)
      endif
c
      idim  = ndg
c
      if(itj.eq.idtj(itn).and.irep.eq.irepmx) then
c   this is for recalculation of csubi in rkld, save the last values
      rlamold = rlamda
      ftermold = fterm
      endif
c
c     determine new xsecs, calculate differences and calc. rho
c
      call dxsec0(xsts,xstp,xstd,xsas,xsap,xsf0,index,z1c,cp,igmw,index1
     $ ,ihpw,ihtw,mtmw,izmw,itn,itj,imw,jmw,kmw,telaps+dtj,dz
     $ ,n6pk,uxsfp,ntsigf,idtj(itn),dtns,vr,dth)
c
      call dopler(xsts,xstp,xstd,xsap,index,igmw,index1,tempold)
c
      call pkvar(rho,fterm,betaef,betaeft,rlamda,extsrc
     $ ,idim,shap4,ansh4,rhot3,rhot4,rhot5,anadj,xstd,scadj,xssd
     $ ,dumep1,xsfd,chit,dumep2,vr,dth,dz,dumep3,xsfp,dumep4
     $ ,index,index1,indexm,imw,jmw,kmw,igmw,mmw,ipkmod)
c
      if(itn.eq.1.and.itj.eq.1.and.irep.eq.irepmx) then
        open(35,file='pkparm',status='unknown')
        write(35,*) " elapsed time    rho       betaeft     rlamda"
        write(35,'(1p,2x,4(e10.3,2x))') telaps+dtj,rho,betaeft,rlamda
        close(35)
      endif
      if((itn.gt.1.and.itj.eq.1.and.irep.eq.irepmx).or.
     $ (iout.gt.0.and.irep.eq.irepmx)) then
        open(35,file='pkparm',status='old',access='append')
        write(35,'(1p,2x,4(e10.3,2x))') telaps+dtj,rho,betaeft,rlamda
        close(35)
      endif
c
c         call over(rho,itn,itj,idtj(itn))
c
      if(iout.ne.0) then
      do ig=1,igmw
      do ix=1,imw
      do jx=1,jmw
```

```
      do kx=1,kmw
       il = ix + imw*((jx-1) + jmw*((kx-1) + kmw*(ig-1)))
       if(xstd(il).ne.0.0.or.xsfd(il).ne.0.0) then
      write(n6pk,*) 'zone ',ix,jx,kx,' grp. ',ig,' contributed by '
     $ ,xstd(il),xsfd(il),xsas(il),xsap(il)
       endif
      enddo
      enddo
      enddo
      enddo
      endif
c
      call mvpk(rho2,rho1,rhonew,rho,bett,betae2,betae1,betnew
     $ ,betaef,bet2,bet1,betn,rlam2,rlam1,rlamnew,rlamda,q2,q1,qnew
     $ ,extsrc,n6pk,telaps+dtj,irep,irepmx,fterm,ndg,iout)
c
c    idtk = # of tk calc.
c
      dtk = zero
      dtk  = dtn(itn)/idtk(itn)
      time = zero
      do 600 itk=1,idtk(itn)/idtj(itn)
c      telaps = telaps + dtk
      tout = t + dtk
c
      call pkcal(rho,betaef,rlamda,extsrc,betaeft,ndg,time)
      rhop = rho
      do i=1,ndg
      betap(i) = betaef(i)
      enddo
      rlamp = rlamda
      betp = betaeft
      extp = extsrc
c
      call lsode(f,neq,y,t,tout,itol,rtol,atol,itask,istate,
     $    iopt,rwork,lrw,iwork,liw,jex,mf)
c
      if (istate .lt. 0) then
      write(n6pk,90)istate
90    format(///22h error halt.. istate =,i3)
c      stop ' solvem : lsode '
      write(6,*) ' stop .... solvem : lsode '
       goto 9999
      endif
c
       call rkunld(y,csubi,ndg,amplit0,n6pk,itn,itj,tout,irep,iout)
       write(41,*) telaps+(itk-1)*dtk,rhop,time
c      if(irep.eq.irepmx) then
c       call rkunld(y,csubi,ndg,amplit0,n6pk,itn,itj,tout,irep,iout)
c      endif
c
        amptk(1+itk) = y(1)
      time = time + dtk
c
600   continue
c
      lamp = lamp + 1
      amp(lamp+1) = y(1)
c
      if(itn.eq.1) then
        tltot = zero
        tjtot = dtns(1)
        tjtot = tjtot*itj/idtj(itn)
        tltot = tltot + tjtot
      elseif(itn.gt.1) then
        tltot = dtns(itn-1)
        tjtot = dtns(itn) - dtns(itn-1)
        tjtot = tjtot*itj/idtj(itn)
        tltot = tltot + tjtot
      endif
      telaps = tltot
c
      write(45,*) telaps,amp(lamp+1)/amplit0,irep
      itkm = idtk(itn)/idtj(itn)
      call pownc0(pow0,shap1,index,igmw,xsfp,itn,vr,dth,dz,imw,jmw,kmw
     $ ,rnu,dtn(itn),51,amptk(2),itkm,dtk,shap2,dume,index1,powxt,mtmw
     $ ,itj,dtj,irep,irepmx,dttot,mdamp,volz,izmw,z1c,cp,powzt)
c
c    if edit time is reached save the power files bpower and bpowerz
c
      do ixe=1,ieditn
        if(itn.eq.1) then
          open(35,file='volz',status='unknown',form='unformatted')
          write(35) mtmw
          write(35) volz
          close(35)
          open(35,file='vol',status='unknown',form='unformatted')
          write(35) imw,jmw,kmw
          write(35) vr,dth,dz
          close(35)
        endif
        if(itn.eq.nedit(ixe)) then
          write(filned,'(a6,i1)') 'xpower',ixe
          write(filnedz,'(a7,i1)') 'xpowerz',ixe
          call system('cp bpower '//filned)
          call system('cp bpowerz '//filnedz)
        endif
      enddo
c
c   rxtemp reads the power into powxt
```

```
c
      call rxtemp(tempold,index1,itn,itkm,powxt,dtk,powxb,tempiz
     $ ,mtmw,powzt,powzb,volz)
c
c     find max./min. power and temp change
c
      call out0(powxt,tempold,index1,xsfp,index,itn,itj,telaps,n6pk)
c
      call avpowd(powxt,index1,vr,dth,dz,imw,jmw,kmw,itn,itj,telaps
     $ ,xsfp,index,irep,irepmx,powzt,volz,mtmw)
c
      call avtemp(tempold,index1,vr,dth,dz,imw,jmw,kmw,itn,itj,telaps
     $ ,xsfp,index,irep,irepmx,tempiz,mtmw,volz,powzt)
c
      call shftcg(cig1,cig2,indexn)
c
c    note : cig1 is latest (cig^(n+1)) & cig2 is prev. (cig^(n))
c
      idatk = mdamp + 1
      call qtstrp0(itkm+1,index,alambda,amptk,ndg,shap1,shap2,index1
     $ ,igmw,dume,cig1,beta,cig2,xsfp,dtj,jlast,itn,dtn(itn),itj,idatk)
c
      if(iout.ne.0) then
      write(n6pk,*) '************************************'
      write(n6pk,*) ' number of steps used in qtrap = ',jlast
      write(n6pk,*) '************************************'
      endif
c
      amptk(1) = amptk(1+itkm)
c
c     end dtj time loop
700   continue
      close(45)
      close(46)
c
      ampn = amp(lamp+1)
      call qaddj(ansh2,cig1,vinv,dtn(itn),alambda,mmw,lmw
     $ ,imw,jmw,kmw,igmw,ndg,index1,qmom,sweight,snpl,chid,iqs,ampn)
c
      call wdistr(igmw,imw,jmw,kmw,lmw,mmw,ntdsif,qmom,cx,cy,cz,denr)
c
      dampndt = (amptk(itkm+1) - amptk(itkm))/ dtk
      write(n6pk,*) ' amplitude derivative at ',telaps,' = ',dampndt
     $,' by delta (P) / delta (t)'
      write(n6pk,*) ' amplitude derivative at ',telaps,' = ',dampndtp
     $,' by LSODE'
      dampndt = dampndtp
c
c      ! need to calc. shape with latest pert. and dop. fed xsecs
c
      call doplerz(uxsfp,ihpw,mtmw,igmw,ihtw,tempiz,izmw,n6pk)
c
c    skip shape calc. only if already calculated at least one
c
      if(irep.eq.irepmx.and.irepmx.ne.0) goto 880
c
      call rdgip0(igmw,ihtw,ihpw,mtmw,betat,ampn,dampndt,iqs,izmw
     $ ,vinv,dtn(itn),nts,rxsec,1,ntsigf,index1,z1c,cp,uxsfp)
c
c   set input and output units for flux guess
c   note that they must be duifferent???? not sure ....
      if(itn.eq.1) then
      j3 = 0
      call rwinpw(j3,igmw,ndg,chi)
      if(j3.ne.0) then
      write(n6pk,*) 'error readw:jobw ....'
      goto 9999
      endif
      endif
c
c   set convergence parameters for fixed source if not done so before
      if(ifix.eq.1.and.itn.eq.1) then
      j3 = 0
      call read66(j3)
       if(j3.ne.0) then
       write(n6pk,*) 'error read66:jobw0 ....'
       goto 9999
       endif
      endif
c
         if(itn.gt.irstrt) then
c
      evold = evnew
      if(imany.le.99) then
        write(filnam,'(a,i2.2)') 'to',imany
        fils = ' <tortinw >'
        call system(xnam//fils//filnam)
      else
        write(filnam2,'(a,i3.3)') 'to',imany
        fils = ' <tortinw >'
        call system(xnam//fils//filnam2)
      endif
       open(35,file='evfil',status='old')
       read(35,*) evdum
       close(35)
        evnew = evdum
c
c     read the new shape in
      call fluxr(shap1,z1c,iafl,1,cp,cx,cy,cz,enr,denr
     $ ,index,index1,izmw,imw,jmw,kmw,igmw)
```

```
      open(35,file='anflux',status='old',form='unformatted')
       read(35) (ansh1(iw),iw=1,indexm)
      close(35)
c     write to interface files
       if(irstrt.gt.0.and.itn.eq.irstrt+1) then
      open(35,file='shapet',status='old',form='unformatted')
      do iix=1,itn
       read(35)
      enddo
      write(35) (shap1(i),i=1,index)
      close(35)
      open(35,file='ashapet',status='old',form='unformatted')
      do iix=1,itn
       read(35)
      enddo
      write(35) (ansh1(i),i=1,indexm)
      close(35)
       else
      open(35,file='shapet',status='old',form='unformatted'
     $ ,access='append')
      write(35) (shap1(i),i=1,index)
      close(35)
      open(35,file='ashapet',status='old',form='unformatted'
     $ ,access='append')
      write(35) (ansh1(i),i=1,indexm)
      close(35)
       endif
c
         else
c   restart, read from interface files
c
      open(35,file='shapet',status='old',form='unformatted')
      do iix=1,itn
       read(35)   ! skip earlier records
      enddo
      read(35) (shap1(i),i=1,index)
      close(35)
      open(35,file='ashapet',status='old',form='unformatted')
      do iix=1,itn
       read(35)   ! skip earlier records
      enddo
      read(35) (ansh1(i),i=1,indexm)
      close(35)
c
         endif
c
        if(ipkmod.eq.0) then
      call ampil(ansh1,anadj,vinv,index,index1,igmw,iqs,1
     $ ,vr,dth,dz,imw,jmw,kmw,amplit,shap,indexm,mmw,sweight,1,n6pk,itn)
       call ampilh(shap1,scadj,vinv,index,index1,igmw,iqs,1
     $ ,vr,dth,dz,imw,jmw,kmw,amplit,shap,1,n6pk,itn)
         elseif(ipkmod.eq.1) then
       call ampilh(shap1,scadj,vinv,index,index1,igmw,iqs,1
     $ ,vr,dth,dz,imw,jmw,kmw,amplit,shap,1,n6pk,itn)
c
c   also normalize angular shape for qaddj
c
      do i=1,indexm
        ansh1(i) = ansh1(i)/ratio(itn+1)
      enddo
        endif
c
         if(irstrt.eq.0.or.itn.gt.irstrt) then
c
c   print flux shape to a file
c
      if(imany.le.99) then
      call woutfl(shap1,index,0,48,filnam//"s"
     $ ,cx,cy,cz,imw,jmw,kmw,igmw,vr,dth,dz)
      else
      call woutfl(shap1,index,0,48,filnam2//"s"
     $ ,cx,cy,cz,imw,jmw,kmw,igmw,vr,dth,dz)
      endif
c
         endif
c
880   continue
c
      if(iout.eq.3) call outshp(n6pk,equal,irep,index,shap1,shap2,igmw)
c
c      calculate total power
c
      call powcal(power,shap1,index,index1,igmw,xsfp
     $ ,vr,dth,dz,imw,jmw,kmw,rnu,pd)
      power = power*econfr
c
c
      write(n6pk,'(120(a1))') (equal,itest=1,115)
       write(n6pk,*)'power                = ',power,' fissions/sec'
       write(n6pk,*)'relative power       = ',power/pow0
      write(n6pk,*)'amplitude            = ',amplit
      write(n6pk,*)'relative amplitude = ',amplit/amplit0
      write(n6pk,*)'shape                = ',shap,' (volume and energy
     $ integrated)'
      write(n6pk,*)'eigenvalue           = ',evnew
c
      close(52,status='delete')
       if(irep.eq.irepmx) goto 890
       irep = irep+1
       ijamp = ijamp - idtj(itn)
```

```
       goto 710
c      endif
890    continue
c
       imany = imany +1
       ampold = amp(lamp+1)
       amp0   = amp(lamp+1)
c
c      end dtn time loop
800    continue
c
c    done , edit
c
       call avpedit
c
9999   return
       end
:::::::::::::::
jobw1.f
:::::::::::::::
c23456789012345678901234567890123456789012345678901234567890123456789012
       subroutine jobw1(iot,cig0,scadj,vr,dth,dz,z1c,xsts,xstp,xstd,xsfs
      $ ,xsfp,xsfd,xsss,xssp,xssd,uxsfp,xsf0,xsf1,cp,dum0,dum1,dumx
      $ ,rxsec,chi,chit,qmom,cx,cy,cz,enr,denr,anadj,snp1,pd,amp
      $ ,shap1,shap2,shap3,shap4,ansh1,ansh2,ansh3
      $ ,ansh4,cig1,cig2,dume1,dume2,amptk,dumep1,dumep2,dumep3
      $ ,dumep4,dume,rhot3,rhot4,rhot5,xsf2,xsfnew,uxsfnew)
c
       include 'comsd1.F'
       include 'comsd2.F'
       include 'comsd4.F'
       include 'comsd5.F'
       include 'comsd6.F'
       include 'comsd7.F'
       include 'comsd8.F'
       include 'comsd0.F'
       include 'comsdc.F'
c
c23456789012345678901234567890123456789012345678901234567890123456789012
       real*4 cig0(indexn),scadj(index),vr(imw),dth(jmw),dz(kmw)
      $ ,z1c(index1),xsts(index),xstp(index),xstd(index),xsfs(index)
      $ ,xsfp(index),xsfd(index),xsss(index*igmw),xssp(index*igmw)
      $ ,xssd(index*igmw),xsf0(izart),xsf1(izart),cp(izmw),dum0(izart)
      $ ,dum1(izart),dumx(igmw*igmw*mtmw),uxsfp(izart)
      $ ,rxsec(izart),chi(igmw),chit(igmw),qmom(index*1mw),cx(imw+1)
      $ ,cy(jmw+1),cz(kmw+1),enr(igmw+2),denr(igmw),anadj(indexm)
      $ ,snp1(idimsnp1),pd(index1),amp(ldamp+1)
      $ ,shap1(index),shap2(index),shap3(index),shap4(index)
      $ ,ansh1(indexm),ansh2(indexm)
      $ ,ansh3(indexm),ansh4(indexm),cig1(indexn),cig2(indexn)
      $ ,dume1(2*index),dume2(2*index*ndg),amptk(mdamp+1)
      $ ,dume(irestx),xsf2(izart),xsfnew(ihpw*igmw),uxsfnew(ihpw*igmw)
c
c    for pkvar
       real*4 dumep1(index1),dumep2(index),dumep3(index1)
      $ ,dumep4(index),rhot3(index),rhot4(index),rhot5(index)
c
       real*4 telaps,rho,rlamda,extsrc,fterm,power,bett,hhi1
      $ ,rho2rp,rho1rp,rhonrp,rlam2rp,rlam1rp,rlamnrp,bet2rp,bet1rp
      $ ,ampn,dampndt
       real*4 zero
       zero = 0.e0
       iot = n6pk
       xnam = 'tortnew '
c
c
c     iquad =  quadratic interpolation flag for flux
c     jquad =  quadratic interpolation flag for rho
c
c    note that z1c is for initial zone map
c     idtnm = # of tn calc.
c
        equal = '='
c    open output unit
       open(n6pk,file='pkout',status='unknown')
       call xsetun(n6pk)
c
c    rhonew is newest
c    rho1   is previous
c    rho2   is oldest
c
c     betaeft is sum(betaef(i))
c
c     initialize
c
       ijamp = 0
       neq = ndg+1
       itol = 1
       ilsrto = 0
       if(srtol.lt.0..or.satol.lt.0.) then
        ilsrto = 1
       endif
       rtol = 1.0d0*abs(srtol)
       atol = 1.0d0*abs(satol)
       itask = 1
       istate = 1
       iopt = 0
       lrw = 500
       liw = 100
       mf = 22
```

```
      t   = zero
      tout= zero
      telaps = zero
c
c     reset irepmx so that 1 extra iteration is performed by default
      irepmx = irepmx + 1
      evw = evnew
      outflg = .false.
      if(iout.gt.0) outflg = .true.
      rho1   = zero
      rho2   = zero
      rhonew = zero
      rho    = zero
      rlam1   = zero
      rlam2   = zero
      rlamnew = zero
      rlamda  = zero
      q1     = zero
      q2     = zero
      qnew   = zero
      extsrc = zero
      fterm  = zero
      power  = zero
      do i=1,ndg
       betae1(i) = zero
       betae2(i) = zero
       betnew(i) = zero
       betaef(i) = zero
       csubi(i) = zero
      enddo
      bet1  = zero
      bet2  = zero
      betn  = zero
      bett  = zero
      call zoneinfo(xsfs,z1c,cp,imw,jmw,kmw,index,index1,igmw,izmw)
c  normalize and save initial zone power
c
      write(n6pk,*)'Initial power        = ',pow0,' fissions/sec'
      write(n6pk,*)'Initial norm.d power = ',pown,' fissions/sec'
      write(n6pk,*)'Initial amplitude    = ',amplit0
      write(n6pk,*)'Initial shape        = ',shap0
     $ , ' (volume and energy integrated)'
      write(n6pk,*)'Initial eigenvalue     = ',evnew
c
c    now have f0, c0, p0=powini
c
      idim   = ndg
      call pkvar(rho0,fterm0,betaef0,betaeft,rlamda0,extsrc0
     $ ,idim,shap1,ansh1,rhot3,rhot4,rhot5,anadj,xstd,scadj,xssd
     $ ,dumep1,xsfd,chit,dumep2,vr,dth,dz,dumep3,xsfp,dumep4
     $ ,index,index1,indexm,imw,jmw,kmw,igmw,mmw,ipkmod)
c
      call pdcal(csubi0,rlamda0,fterm0,idim,cig0,scadj,dume2,vr,dth,dz
     $ ,imw,jmw,kmw,igmw,index,indexn,index1,chid)
c
c
      write(n6pk,'(120(a1))') (equal,itest=1,115)
      write(n6pk,900) rho0,rho0/betaeft
      write(n6pk,910) (betaef0(i),i=1,ndg)
      write(n6pk,920) betaeft
      write(n6pk,930) (csubi0(i),i=1,ndg)
      write(n6pk,940) rlamda0
c      write(n6pk,*) ' fterm         = ',fterm0
      write(n6pk,'(120(a1))') (equal,itest=1,115)
900   format(1x,44h Reactivity                                  = ,1pe13.6
     $ ,2x,1h(,e13.6,2h$))
910   format(1x,44h Effective Delayed Neutron Fractions         = ,1p
     $ ,6(2x,e10.3))
920   format(1x,44h Effective Total Delayed Neutron Fraction = ,1pe11.4)
930   format(1x,44h Weighted Precursor Densities               = ,1p
     $ ,6(2x,e11.4))
940   format(1x,44h Generation Time                             = ,1pe13.6)
c
      call rkld0(betaef0,alambda,rlamda0,n6pk,ndg,amplit0,y)
c   this is needed to keep initial values of y vector
c   the last arg. is : 1= y to resrep , otherwise resrep to y
      call rkldrp(y,resrep,neq,1)
c
      rhonew = rho0
      rlamnew = rlamda0
      betn = zero
      do ii=1,ndg
      betnew(ii) = betaef0(ii)
      betn = betn + betnew(ii)
      enddo
c
c
c   also normalize angular shape for qaddj
c
      do i=1,indexm
        ansh1(i) = ansh1(i)/ratio(1)
      enddo
c
      iquad = .false.
c     start dtn time loop
      do 800 itn = 1, idtnm
c   set lsode parameters
      if(ilsrto.eq.0) then
       rtol = dtn(itn)/idtk(itn)*srtol
       atol = dtn(itn)/idtk(itn)*satol
```

```
      endif
      if(ilsrto.eq.0.or.itn.eq.1) then
       write(n6pk,*) '**********************************'
       write(n6pk,*) ' rtol = ',rtol,' at time interval ',itn
       write(n6pk,*) ' atol = ',atol,' at time interval ',itn
       write(n6pk,*) '**********************************'
      endif
c
      irep = 0      ! this is to repeat pk calc. and update
      hi1 = dtn(itn)
      hi2 = dtn(itn)
c
c   shift fluxes
      call shftfl(shap1,shap2,shap3,ansh1,ansh2,ansh3,index,indexm)
c
      if(itn.eq.1) then
        amptk(1) = amplit0
        amp(1) = amplit0
        rlamda = rlamda0
        fterm  = fterm0
       hhi1=zero
c  save last amplitude as first for next time step
       ampold = amplit0
       amp0   = amplit0
      else
        rlamda = rlamold
        fterm  = ftermold
       hhi1 = dtn(itn-1)
      endif
c    generate precursor densities using last calculated delayed
c    neutron source and load onto y array
c
c   this next call causes discontinuities in y b/c csubi does not
c   match the value calculated by lsode
c   use with caution
      call rkld(y,rlamda,fterm,n6pk,ndg,cig2,scadj,imw,jmw,kmw,igmw
     $ ,vr,dth,dz,indexn,index,itn,index1,dume1,chid,ampold,amp0,amplit)
c
c   extrapolate to calculate first flux shape and normalize
c   extrapolate angular and scalar shape
c
        if(ipkmod.eq.0) then
       call extrap(ansh1,ansh2,ansh3,hhi1,hi2,indexm,iextrp,n6pk)
       call extrap(shap1,shap2,shap3,hhi1,hi2,index,iextrp,n6pk)
       call ampil(ansh1,anadj,vinv,index,index1,igmw,iqs,0
     $ ,vr,dth,dz,imw,jmw,kmw,amplit,shap,indexm,mmw,sweight,1,n6pk,itn)
       call ampilh(shap1,scadj,vinv,index,index1,igmw,iqs,0
     $ ,vr,dth,dz,imw,jmw,kmw,amplit,shap,1,n6pk,itn)
        elseif(ipkmod.eq.1) then
       call extrap(shap1,shap2,shap3,hhi1,hi2,index,iextrp,n6pk)
       call ampilh(shap1,scadj,vinv,index,index1,igmw,iqs,0
     $ ,vr,dth,dz,imw,jmw,kmw,amplit,shap,1,n6pk,itn)
        endif
c
c    idtj = # of tj calc.
710   jquad = .false.
c
c    clear rwork for lsode
      do itt2=1,500
         rwork(itt2) = 0.0d0
      enddo
      do itt2=1,100
         iwork(itt2) = 0
      enddo
      if(irep.eq.0) then
c   svae old react.,gen. time and effective betas for second pass
       call rkldrp(y,resrep,ndg+1,1)
         rho2rp = rho2
         rho1rp = rho1
         rhonrp = rhonew
         rlam2rp = rlam2
         rlam1rp = rlam1
         rlamnrp = rlamnew
         bet2rp = zero
         bet1rp = zero
         betnrp = betn
         do i=1,ndg
         bete2rp(i) = betae2(i)
         bete1rp(i) = betae1(i)
         betenrp(i) = betnew(i)
         bet2rp    = bet2rp + bete2rp(i)
         bet1rp    = bet1rp + bete1rp(i)
         enddo
       elseif(irep.gt.0) then
c
c   put old data back
        call rkldrp(y,resrep,neq,0)
         rho2 = rho2rp
         rho1 = rho1rp
         rhonew = rhonrp
         rlam2 = rlam2rp
         rlam1 = rlam1rp
         rlamnew = rlamnrp
         bet2 = zero
         bet1 = zero
         betn = betnrp
         do i=1,ndg
         betae2(i) = bete2rp(i)
         betae1(i) = bete1rp(i)
         betnew(i) = betenrp(i)
```

```
        bet2    = bet2 + betae2(i)
        bet1    = bet1 + betae1(i)
        enddo
      endif
c
c     start dtj time loop
      lamp = 0
      dtj = zero
      dtj = dtn(itn)/idtj(itn)
c  open and close fort.45 and fort.46 to force buffer dump
      if(itn.eq.1.and.irep.eq.0) then
        open(45,status='unknown')
        open(46,status='unknown')
      else
        open(45,status='old',access='append')
        open(46,status='old',access='append')
      endif
c
      dttot = zero
      if(itn.gt.1) dttot = dtns(itn-1)
c
      do 700 itj=1, idtj(itn)
c  transfer the times for irep = 0 case
      if(irep.eq.0.and.itj.eq.1.and.irepmx.gt.0) then
        ttrep1 = telaps
        ttrep2 = tout
        ttrep3 = t
      elseif(irep.gt.0.and.itj.eq.1.and.irepmx.gt.0) then
        telaps = ttrep1
        tout   = ttrep2
        t      = ttrep3
      endif
      hj1    = dtj
      hj2    = dtj
      ijamp = ijamp + 1
      if(itj.gt.2)    jquad = .true.
      if(itj.eq.1) istate = 1
c
      tnewj = itj*dtj
c
c  linear interp.
      if(ipkmod.eq.0) then
      call linter(ansh2,ansh1,ansh4,hi2,tnewj,indexm)
      call linter(shap2,shap1,shap4,hi2,tnewj,index)
      elseif(ipkmod.eq.1) then
      call linter(shap2,shap1,shap4,hi2,tnewj,index)
      endif
c
      idim  = ndg
c
      if(itn.eq.1.and.itj.eq.1.and.irep.eq.irepmx) then
        open(35,file='pkparm',status='unknown')
        write(35,*) " elapsed time    rho       betaeft     rlamda"
        write(35,'(1p,2x,4(e10.3,2x))') telaps,rho,betaeft,rlamda
        close(35)
      endif
      if((itn.gt.1.and.itj.eq.1.and.irep.eq.irepmx).or.
     $ (iout.gt.0.and.irep.eq.irepmx)) then
        open(35,file='pkparm',status='old',access='append')
        write(35,'(1p,2x,4(e10.3,2x))') telaps,rho,betaeft,rlamda
        close(35)
      endif
c
      if(itj.eq.idtj(itn).and.irep.eq.irepmx) then
c  this is for recalculation of csubi in rkld, save the last values
      rlamold = rlamda
      ftermold = fterm
      endif
c
c     determine new xsecs, calculate differences and calc. rho
c
      call dxsec1(xsts,xstp,xstd,xsfs,xsfp,xsfd,xsss,xssp,xssd,uxsfp
     $ ,xsf0,index,z1c,cp,igmw,index1
     $ ,ihpw,ihtw,mtmw,izmw,itn,itj,imw,jmw,kmw,telaps+dtj,dz
     $ ,n6pk,xsfnew,uxsfnew,dum0,dumx,ntsigf,idtj(itn),dtns,vr,dth)
c
      call pkvar(rho,fterm,betaef,betaeft,rlamda,extsrc
     $ ,idim,shap4,ansh4,rhot3,rhot4,rhot5,anadj,xstd,scadj,xssd
     $ ,dumep1,xsfd,chit,dumep2,vr,dth,dz,dumep3,xsfp,dumep4
     $ ,index,index1,indexm,imw,jmw,kmw,igmw,mmw,ipkmod)
c
      if(iout.ne.0) then
      do ig=1,igmw
      do ix=1,imw
      do jx=1,jmw
      do kx=1,kmw
       i1 = ix + imw*((jx-1) + jmw*((kx-1) + kmw*(ig-1)))
       if(xstd(i1).ne.0.0.or.xsfd(i1).ne.0.0) then
      write(n6pk,*) 'zone ',ix,jx,kx,' contributed to rho for group',ig
     $ ,'by ',xstd(i1),xsfd(i1)
       endif
      enddo
      enddo
      enddo
      enddo
      endif
c
      call mvpk(rho2,rho1,rhonew,rho,bett,betae2,betae1,betnew
     $ ,betaef,bet2,bet1,betn,rlam2,rlam1,rlamnew,rlamda,q2,q1,qnew
     $ ,extsrc,n6pk,telaps+dtj,irep,irepmx,fterm,ndg,iout)
```

```
c
c    idtk = # of tk calc.
c
      dtk = zero
      dtk  = dtn(itn)/idtk(itn)
      time = zero
      do 600 itk=1,idtk(itn)/idtj(itn)
c      telaps = telaps + dtk
      tout = t + dtk
c
      call pkcal(rho,betaef,rlamda,extsrc,betaeft,ndg,time)
      rhop = rho
      do i=1,ndg
      betap(i) = betaef(i)
      enddo
      rlamp = rlamda
      betp = betaeft
      extp = extsrc
c
      call lsode(f,neq,y,t,tout,itol,rtol,atol,itask,istate,
     $    iopt,rwork,lrw,iwork,liw,jex,mf)
c
      if (istate .lt. 0) then
      write(n6pk,90)istate
90    format(///22h error halt.. istate =,i3)
c      stop ' solvem : lsode '
      write(6,*) ' stop .... solvem : lsode '
       goto 9999
      endif
c
       call rkunld(y,csubi,ndg,amplit0,n6pk,itn,itj,tout,irep,iout)
c       write(41,*) telaps+(itk-1)*dtk,rhop,time
c      if(irep.eq.irepmx) then
c       call rkunld(y,csubi,ndg,amplit0,n6pk,itn,itj,tout,irep,iout)
c      endif
c
        amptk(1+itk) = y(1)
      time = time + dtk
c
600   continue
c
      lamp = lamp + 1
      amp(lamp+1) = y(1)
c
      if(itn.eq.1) then
        tltot = zero
        tjtot = dtns(1)
        tjtot = tjtot*itj/idtj(itn)
        tltot = tltot + tjtot
      elseif(itn.gt.1) then
        tltot = dtns(itn-1)
        tjtot = dtns(itn) - dtns(itn-1)
        tjtot = tjtot*itj/idtj(itn)
        tltot = tltot + tjtot
      endif
      telaps = tltot
c
      write(45,*) telaps,amp(lamp+1)/amplit0,irep
      itkm = idtk(itn)/idtj(itn)
      call pownc1(pow0,shap1,index,igmw,xsfp,itn,vr,dth,dz,imw,jmw,kmw
     $,rnu,dtn(itn),51,amptk(2),itkm,dtk,shap2,dume,itj,dtj,irep,irepmx
     $,dttot)
c
      call shftcg(cig1,cig2,indexn)
c
c   note : cig1 is latest (cig^(n+1)) & cig2 is prev. (cig^(n))
c
      call qtstrp0(itkm+1,index,alambda,amptk,ndg,shap1,shap2,index1
     $ ,igmw,dume,cig1,beta,cig2,xsfp,dtj,jlast,itn)
c
      if(iout.ne.0) then
      write(n6pk,*) '***********************************'
      write(n6pk,*) ' number of steps used in qtrap = ',jlast
      write(n6pk,*) '***********************************'
      endif
c
      amptk(1) = amptk(1+itkm)
c
c     end dtj time loop
700   continue
      close(45)
      close(46)
c
      ampn = amp(lamp+1)
      call qaddj(ansh2,cig1,vinv,dtn(itn),alambda,mmw,lmw
     $ ,imw,jmw,kmw,igmw,ndg,index1,qmom,sweight,snpl,chid,iqs,ampn)
c
      call wdistr(igmw,imw,jmw,kmw,lmw,mmw,ntdsif,qmom,cx,cy,cz,denr)
c
      dampndt = (amp(lamp+1) - amp(lamp))/ dtj
      write(n6pk,*) ' amplitude derivative at ',telaps,' = ',dampndt
     $,' by delta (P) / delta (t)'
      write(n6pk,*) ' amplitude derivative at ',telaps,' = ',dampndtp
     $,' by LSODE'
c       dampndt = dampndtp
c
c    skip shape calc. only if already calculated at least one
      if(irep.eq.irepmx.and.irepmx.ne.0) goto 880
c
c      ! need to calc. shape with latest pert. and dop. fed xsecs
```

```
c
      call rdgip1(igmw,ihtw,ihpw,mtmw,betat,ampn,dampndt,iqs,izmw
     $ ,vinv,dtn(itn),nts,rxsec,1,ntsigf,index1,uxsfp,z1c,cp)
c
c   set input and output units for flux guess
c   note that they must be duifferent???? not sure ....
      if(itn.eq.1) then
      j3 = 0
      call rwinpw(j3,igmw,ndg,chi)
      if(j3.ne.0) then
      write(n6pk,*) 'error readw:jobw ....'
      goto 9999
      endif
      endif
c
c   set convergence parameters for fixed source if not done so before
      if(ifix.eq.1.and.itn.eq.1) then
      j3 = 0
      call read66(j3)
       if(j3.ne.0) then
       write(n6pk,*) 'error read66:jobw1 ....'
       goto 9999
       endif
      endif
c
         if(itn.gt.irstrt) then
c
      evold = evnew
      if(imany.le.99) then
        write(filnam,'(a,i2.2)') 'to',imany
        fils = ' <tortinw >'
        call system(xnam//fils//filnam)
      else
        write(filnam2,'(a,i3.3)') 'to',imany
        fils = ' <tortinw >'
        call system(xnam//fils//filnam2)
      endif
       open(35,file='evfil',status='old')
       read(35,*) evdum
       close(35)
        evnew = evdum
c
c     read the new shape in
      call fluxr(shap1,z1c,iafl,1,cp,cx,cy,cz,enr,denr
     $ ,index,index1,izmw,imw,jmw,kmw,igmw)
      open(35,file='anflux',status='old',form='unformatted')
       read(35) (ansh1(iw),iw=1,indexm)
      close(35)
c     write to interface files
        if(irstrt.gt.0.and.itn.eq.irstrt+1) then
      open(35,file='shapet',status='old',form='unformatted')
      do iix=1,itn
       read(35)
      enddo
      write(35) (shap1(i),i=1,index)
      close(35)
      open(35,file='ashapet',status='old',form='unformatted')
      do iix=1,itn
       read(35)
      enddo
      write(35) (ansh1(i),i=1,indexm)
      close(35)
       else
      open(35,file='shapet',status='old',form='unformatted'
     $ ,access='append')
      write(35) (shap1(i),i=1,index)
      close(35)
      open(35,file='ashapet',status='old',form='unformatted'
     $ ,access='append')
      write(35) (ansh1(i),i=1,indexm)
      close(35)
       endif
c
         else
c   restart, read from interface files
c
      open(35,file='shapet',status='old',form='unformatted')
      do iix=1,itn
       read(35)   ! skip earlier records
      enddo
      read(35) (shap1(i),i=1,index)
      close(35)
      open(35,file='ashapet',status='old',form='unformatted')
      do iix=1,itn
       read(35)   ! skip earlier records
      enddo
      read(35) (ansh1(i),i=1,indexm)
      close(35)
c
         endif
c
        if(ipkmod.eq.0) then
      call ampil(ansh1,anadj,vinv,index,index1,igmw,iqs,1
     $ ,vr,dth,dz,imw,jmw,kmw,amplit,shap,indexm,mmw,sweight,1,n6pk,itn)
       call ampilh(shap1,scadj,vinv,index,index1,igmw,iqs,1
     $ ,vr,dth,dz,imw,jmw,kmw,amplit,shap,1,n6pk,itn)
        elseif(ipkmod.eq.1) then
      call ampilh(shap1,scadj,vinv,index,index1,igmw,iqs,1
     $ ,vr,dth,dz,imw,jmw,kmw,amplit,shap,1,n6pk,itn)
c
```

```
c   also normalize angular shape for qaddj
c
      do i=1,indexm
        ansh1(i) = ansh1(i)/ratio(itn+1)
      enddo
        endif
c
         if(irstrt.eq.0.or.itn.gt.irstrt) then
c
c   print flux shape to a file
c
      if(imany.le.99) then
      call woutfl(shap1,index,0,48,filnam//"s"
     $ ,cx,cy,cz,imw,jmw,kmw,igmw,vr,dth,dz)
      else
      call woutfl(shap1,index,0,48,filnam2//"s"
     $ ,cx,cy,cz,imw,jmw,kmw,igmw,vr,dth,dz)
      endif
c
         endif
c
880   continue
c
      if(iout.eq.3) call outshp(n6pk,equal,irep,index,shap1,shap2,igmw)
c
c      calculate total power
c
      call powcal(power,shap1,index,index1,igmw,xsfp
     $ ,vr,dth,dz,imw,jmw,kmw,rnu,pd)
c
c
      write(n6pk,'(120(a1))') (equal,itest=1,115)
       write(n6pk,*)'power                = ',power,' fissions/sec'
       write(n6pk,*)'relative power       = ',power/pow0
      write(n6pk,*)'amplitude          = ',amplit
      write(n6pk,*)'relative amplitude = ',amplit/amplit0
      write(n6pk,*)'shape                = ',shap,' (volume and energy
     $ integrated)'
      write(n6pk,*)'eigenvalue           = ',evnew
c
      close(52,status='delete')
       if(irep.eq.irepmx) goto 890
       irep = irep+1
       ijamp = ijamp - idtj(itn)
       goto 710
c      endif
890   continue
c
      imany = imany +1
      ampold = amp(lamp+1)
      amp0   = amp(lamp+1)
c
c     end dtn time loop
800   continue
9999  return
      end
::::::::::::::
jobw2.f
::::::::::::::
c23456789012345678901234567890123456789012345678901234567890123456789012
      subroutine jobw2(iot,cig0,scadj,vr,dth,dz,z1c,xsts,xstp,xstd,xsfs
     $ ,xsfp,xsfd,xsss,xssp,xssd,xsf0,xsf1,cp,dum0,dum1,dumx
     $ ,rxsec,chi,chit,qmom,cx,cy,cz,enr,denr,anadj,snp1,pd,amp,flux1
     $ ,flux2,shap1,shap2,shap3,shap4,anf11,anf12,ansh1,ansh2,ansh3
     $ ,ansh4,cig1,cig2,dume1,dume2,dume3,dumep1,dumep2,dumep3
     $ ,dumep4,dume,rhot3,rhot4,rhot5,xsf2)
c
       include 'comsd1.F'
       include 'comsd2.F'
       include 'comsd4.F'
       include 'comsd5.F'
       include 'comsd6.F'
       include 'comsd7.F'
       include 'comsd8.F'
       include 'comsd0.F'
       include 'comsdc.F'
c
c23456789012345678901234567890123456789012345678901234567890123456789012
       real*4 cig0(indexn),scadj(index),vr(imw),dth(jmw),dz(kmw)
     $ ,z1c(index1),xsts(index),xstp(index),xstd(index),xsfs(index)
     $ ,xsfp(index),xsfd(index),xsss(index*igmw),xssp(index*igmw)
     $ ,xssd(index*igmw),xsf0(izart),xsf1(izart),cp(izmw),dum0(izart)
     $ ,dum1(izart),dumx(igmw*igmw*mtmw)
     $ ,rxsec(izart),chi(igmw),chit(igmw),qmom(index*1mw),cx(imw+1)
     $ ,cy(jmw+1),cz(kmw+1),enr(igmw+2),denr(igmw),anadj(indexm)
     $ ,snp1(idimsnp1),pd(index1),amp(ldamp+1),flux1(index)
     $ ,flux2(index),shap1(index),shap2(index),shap3(index),shap4(index)
     $ ,anf11(indexm),anf12(indexm),ansh1(indexm),ansh2(indexm)
     $ ,ansh3(indexm),ansh4(indexm),cig1(indexn),cig2(indexn)
     $ ,dume1(2*index),dume2(2*index*ndg),dume3(mdamp)
     $ ,dume(irestx),xsf2(izart)
c
c   for pkvar
      real*4 dumep1(index1),dumep2(index),dumep3(index1)
     $ ,dumep4(index),rhot3(index),rhot4(index),rhot5(index)
c
       real*4 telaps,rho,rlamda,extsrc,fterm,power,bett,hhi1
     $ ,rho2rp,rho1rp,rhonrp,rlam2rp,rlam1rp,rlamnrp,bet2rp,bet1rp
     $ ,tx0,tx1,ampn,dampndt
      real*4 zero
```

```
      zero = 0.e0
      iot = n6pk
      xnam = 'tortnew '
c
c     iquad =  quadratic interpolation flag for flux
c     jquad =  quadratic interpolation flag for rho
c
c    note that z1c is for initial zone map
c     idtnm = # of tn calc.
c
       equal = '='
c    open output unit
       open(n6pk,file='pkout',status='unknown')
       call xsetun(n6pk)
c
c    rhonew is newest
c    rho1   is previous
c    rho2   is oldest
c
c     betaeft is sum(betaef(i))
c
c     initialize
c
      ijamp = 0
      neq = ndg+1
      itol = 1
      ilsrto = 0
      if(srtol.lt.0..or.satol.lt.0.) then
       ilsrto = 1
      endif
      rtol = 1.0d0*abs(srtol)
      atol = 1.0d0*abs(satol)
      itask = 1
      istate = 1
      iopt = 0
      lrw = 500
      liw = 100
      mf = 22
      t    = zero
      tout= zero
      telaps = zero
c
c     reset irepmx so that 1 extra iteration is performed by default
      irepmx = irepmx + 1
      evw = evnew
      outflg = .false.
      if(iout.gt.0) outflg = .true.
      rho1    = zero
      rho2    = zero
      rhonew = zero
      rho     = zero
      rlam1    = zero
      rlam2    = zero
      rlamnew = zero
      rlamda  = zero
      q1      = zero
      q2      = zero
      qnew    = zero
      extsrc = zero
      fterm  = zero
      power  = zero
      do i=1,ndg
       betae1(i) = zero
       betae2(i) = zero
       betnew(i) = zero
       betaef(i) = zero
       csubi(i) = zero
      enddo
      bet1  = zero
      bet2  = zero
      betn  = zero
      bett  = zero
c
       call rkld0(betaef0,alambda,rlamda0,n6pk,ndg,amplit0,y)
c    this is needed to keep initial values of y vector
c    the last arg. is : 1= y to resrep , otherwise resrep to y
       call rkldrp(y,resrep,neq,1)
c
      rhonew = rho0
      rlamnew = rlamda0
      betn = zero
      do ii=1,ndg
      betnew(ii) = betaef0(ii)
      betn = betn + betnew(ii)
      enddo
c
c    also normalize angular shape for qaddj
c
      do i=1,indexm
        anshl(i) = anshl(i)/ratio(1)
      enddo
c
      iquad = .false.
c     start dtn time loop
      do 800 itn = 1, idtnm
c    set lsode parameters
       if(ilsrto.eq.0) then
        rtol = dtn(itn)/idtk(itn)*srtol
        atol = dtn(itn)/idtk(itn)*satol
       endif
```

```
      if(ilsrto.eq.0.or.itn.eq.1) then
       write(n6pk,*) '**********************************'
       write(n6pk,*) ' rtol = ',rtol,' at time interval ',itn
       write(n6pk,*) ' atol = ',atol,' at time interval ',itn
       write(n6pk,*) '**********************************'
      endif
c
      irep = 0      ! this is to repeat pk calc. and update
      hi1 = dtn(itn)
      hi2 = dtn(itn)
c
c   shift fluxes
      call shftfl(shap1,shap2,shap3,ansh1,ansh2,ansh3,index,indexm)
      call shftcg(cig1,cig2,indexn)
c   note : cig1 is latest (cig^(n+1)) & cig2 is prev. (cig^(n))
c
      if(itn.eq.1) then
        amp(1) = amplit0
        rlamda = rlamda0
        fterm  = fterm0
       hhi1=zero
c  save last amplitude as first for next time step
       ampold = amplit0
       amp0   = amplit0
      else
        rlamda = rlamold
        fterm  = ftermold
       hhi1 = dtn(itn-1)
      endif
c    generate precursor densities using last calculated delayed
c    neutron source and load onto y array
c
c   this next call causes discontinuities in y b/c csubi does not
c   match the value calculated by lsode
c   use with caution
      call rkld(y,rlamda,fterm,n6pk,ndg,cig2,scadj,imw,jmw,kmw,igmw
     $ ,vr,dth,dz,indexn,index,itn,index1,dume1,chid,ampold,amp0,amplit)
c
c   extrapolate to calculate first flux shape and normalize
c
c   extrapolate angular and scalar shape
c
        if(ipkmod.eq.0) then
      call extrap(ansh1,ansh2,ansh3,hhi1,hi2,indexm,iextrp,n6pk)
      call extrap(shap1,shap2,shap3,hhi1,hi2,index,iextrp,n6pk)
      call ampil(ansh1,anadj,vinv,index,index1,igmw,iqs,0
     $ ,vr,dth,dz,imw,jmw,kmw,amplit,shap,indexm,mmw,sweight,1,n6pk,itn)
      call ampilh(shap1,scadj,vinv,index,index1,igmw,iqs,0
     $ ,vr,dth,dz,imw,jmw,kmw,amplit,shap,1,n6pk,itn)
        elseif(ipkmod.eq.1) then
      call extrap(shap1,shap2,shap3,hhi1,hi2,index,iextrp,n6pk)
      call ampilh(shap1,scadj,vinv,index,index1,igmw,iqs,0
     $ ,vr,dth,dz,imw,jmw,kmw,amplit,shap,1,n6pk,itn)
        endif
c
c    idtj = # of tj calc.
710   jquad = .false.
c    clear rwork for lsode
      do itt2=1,500
         rwork(itt2) = 0.0d0
      enddo
      do itt2=1,100
         iwork(itt2) = 0
      enddo
      if(irep.eq.0) then
c   svae old react.,gen. time and effective betas for second pass
      call rkldrp(y,resrep,ndg+1,1)
        rho2rp = rho2
        rho1rp = rho1
        rhonrp = rhonew
        rlam2rp = rlam2
        rlam1rp = rlam1
        rlamnrp = rlamnew
        bet2rp = zero
        bet1rp = zero
        betnrp = betn
        do i=1,ndg
        bete2rp(i) = betae2(i)
        bete1rp(i) = betae1(i)
        betenrp(i) = betnew(i)
        bet2rp    = bet2rp + bete2rp(i)
        bet1rp    = bet1rp + bete1rp(i)
        enddo
      elseif(irep.gt.0) then
c   put old data back
       call rkldrp(y,resrep,neq,0)
        rho2 = rho2rp
        rho1 = rho1rp
        rhonew = rhonrp
        rlam2 = rlam2rp
        rlam1 = rlam1rp
        rlamnew = rlamnrp
        bet2 = zero
        bet1 = zero
        betn = betnrp
        do i=1,ndg
        betae2(i) = bete2rp(i)
        betae1(i) = bete1rp(i)
        betnew(i) = betenrp(i)
        bet2    = bet2 + betae2(i)
```

```
        bet1    = bet1 + betae1(i)
        enddo
      endif
c
c     start dtj time loop
      lamp = 0
      dtj = zero
      dtj = dtn(itn)/idtj(itn)
c  open and close fort.45 and fort.46 to force buffer dump
      if(itn.eq.1.and.irep.eq.0) then
        open(45,status='unknown')
        open(46,status='unknown')
      else
        open(45,status='old',access='append')
        open(46,status='old',access='append')
      endif
      do 700 itj=1, idtj(itn)
c  transfer the times for irep = 0 case
      if(irep.eq.0.and.itj.eq.1.and.irepmx.gt.0) then
        ttrep1 = telaps
        ttrep2 = tout
        ttrep3 = t
      elseif(irep.gt.0.and.itj.eq.1.and.irepmx.gt.0) then
        telaps = ttrep1
        tout   = ttrep2
        t      = ttrep3
      endif
      hj1    = dtj
      hj2    = dtj
      ijamp = ijamp + 1
      if(itj.gt.2)    jquad = .true.
      if(itj.eq.1) istate = 1
c
      tnewj = itj*dtj
c
c  linear interp.
      if(ipkmod.eq.0) then
      call linter(ansh2,ansh1,ansh4,hi2,tnewj,indexm)
      call linter(shap2,shap1,shap4,hi2,tnewj,index)
      elseif(ipkmod.eq.1) then
      call linter(shap2,shap1,shap4,hi2,tnewj,index)
      endif
c
      idim  = ndg
c  calculate initial reac. at time 0. with pert. xsecs if
c  step perturbation and first pass
      if(itn.eq.1.and.itj.eq.1) then
c
      call dxsec2(xsts,xstp,xstd,xsfs,xsfp,xsfd
     $,xsss,xssp,xssd,xsf0,xsf1,index
     $,z1c,cp,igmw,index1,ihpw,ihtw,mtmw,izmw
     $,1,dum0,dum1,dumx)
c
      call pkvar(rho,fterm,betaef,betaeft,rlamda,extsrc
     $ ,idim,shap2,ansh2,rhot3,rhot4,rhot5,anadj,xstd,scadj,xssd
     $ ,dumep1,xsfd,chit,dumep2,vr,dth,dz,dumep3,xsfp,dumep4
     $ ,index,index1,indexm,imw,jmw,kmw,igmw,mmw,ipkmod)
c
      call mvpk(rho2,rho1,rhonew,rho,bett,betae2,betae1,betnew
     $ ,betaef,bet2,bet1,betn,rlam2,rlam1,rlamnew,rlamda,q2,q1,qnew
     $ ,extsrc,n6pk,telaps,irep,irepmx,fterm,ndg,iout)
      endif
c
      if(itn.eq.1.and.itj.eq.1.and.irep.eq.irepmx) then
        open(35,file='pkparm',status='unknown')
        write(35,*) " elapsed time    rho       betaeft     rlamda"
        write(35,'(1p,2x,4(e10.3,2x))') telaps,rho,betaeft,rlamda
        close(35)
      endif
      if((itn.gt.1.and.itj.eq.1.and.irep.eq.irepmx).or.
     $ (iout.gt.0.and.irep.eq.irepmx)) then
        open(35,file='pkparm',status='old',access='append')
        write(35,'(1p,2x,4(e10.3,2x))') telaps,rho,betaeft,rlamda
        close(35)
      endif
c
      if(itj.eq.idtj(itn).and.irep.eq.irepmx) then
c  this is for recalculation of csubi in rkld, save the last values
      rlamold = rlamda
      ftermold = fterm
      endif
c
      call pkvar(rho,fterm,betaef,betaeft,rlamda,extsrc
     $ ,idim,shap4,ansh4,rhot3,rhot4,rhot5,anadj,xstd,scadj,xssd
     $ ,dumep1,xsfd,chit,dumep2,vr,dth,dz,dumep3,xsfp,dumep4
     $ ,index,index1,indexm,imw,jmw,kmw,igmw,mmw,ipkmod)
c
      call mvpk(rho2,rho1,rhonew,rho,bett,betae2,betae1,betnew
     $ ,betaef,bet2,bet1,betn,rlam2,rlam1,rlamnew,rlamda,q2,q1,qnew
     $ ,extsrc,n6pk,telaps+dtj,irep,irepmx,fterm,ndg,iout)
c
c     idtk = # of tk calc.
c
      dtk = zero
      dtk  = dtn(itn)/idtk(itn)
      time = zero
      do 600 itk=1,idtk(itn)/idtj(itn)
c      telaps = telaps + dtk
      tout = t + dtk
c
```

```
      call pkcal(rho,betaef,rlamda,extsrc,betaeft,ndg,time)
      rhop = rho
      do i=1,ndg
      betap(i) = betaef(i)
      enddo
      rlamp = rlamda
      betp = betaeft
      extp = extsrc
c
      call lsode(f,neq,y,t,tout,itol,rtol,atol,itask,istate,
     $    iopt,rwork,lrw,iwork,liw,jex,mf)
c
      if (istate .lt. 0) then
      write(n6pk,90)istate
90    format(///22h error halt.. istate =,i3)
c      stop ' solvem : lsode '
      write(6,*) ' stop .... solvem : lsode '
       goto 9999
      endif
c
       call rkunld(y,csubi,ndg,amplit0,n6pk,itn,itj,tout,irep,iout)
      if(irep.eq.irepmx) then
c       call rkunld(y,csubi,ndg,amplit0,n6pk,itn,itj,tout,irep,iout)
c       write(41,*) telaps+(itk-1)*dtk,rhop,time
        sdump = y(1)
       write(52,*) sdump
      endif
c
      time = time + dtk
c
600   continue
c
      lamp = lamp + 1
      amp(lamp+1) = y(1)
c
      if(itn.eq.1) then
        tltot = zero
        tjtot = dtns(1)
        tjtot = tjtot*itj/idtj(itn)
        tltot = tltot + tjtot
      elseif(itn.gt.1) then
        tltot = dtns(itn-1)
        tjtot = dtns(itn) - dtns(itn-1)
        tjtot = tjtot*itj/idtj(itn)
        tltot = tltot + tjtot
      endif
      telaps = tltot
c
      write(45,*) telaps,amp(lamp+1)/amplit0,irep
c     end dtj time loop
700   continue
      close(45)
      close(46)
c  if last pass over this dtn, save the amplitute
      if(irep.eq.irepmx) then
      rewind(52)
      i1 = 1
740   read(52,*,err=750,end=750) dume3(i1)
      i1 = i1 + 1
      goto 740
750   i1 = i1 - 1
      if(itn.eq.1) then
       open(35,file='amplitute',status='unknown',form='unformatted')
      else
       open(35,file='amplitute',status='old',form='unformatted'
     $ ,access='append')
      endif
      write(35) (dume3(i1i),i1i=1,i1)
      close(35)
      endif
c   if this is the last dtn time interval then
c    there is no need to update shape and precursors, just exit.
      if(itn.eq.idtnm.and.irep.eq.irepmx) goto 890
c   test
      write(68,*) 'itn = ',itn,' lamp = ',lamp
      write(68,*) ' amp'
      do izgak=1,lamp+1
      write(68,*) amp(izgak)
      enddo
c end test
c
c  update precursors , calculate fixed source and update shape with
c  new tort calculation.
      dtnx = dtn(itn)/idtj(itn)
c
      tx0 = 1.
      tx1 = 1.
      call delsor2(cig1,cig2,alambda,beta,index,index1,ndg
     $,igmw,xsfp,xsfs,shap1,shap2,amp,ldamp,idtj,dume1,dume2,dtn,dtnx
     $,amp0,itn,irestx,0,dtj,xsf0,xsf1,ihtw,mtmw,izmw,z1c,cp,imw,jmw
     $,kmw,ihpw,tx0,tx1,ampold,indexn,dume,izart,n6pk)
c
      ampn = amp(lamp+1)
      call qaddj(ansh2,cig1,vinv,dtn(itn),alambda,mmw,lmw
     $ ,imw,jmw,kmw,igmw,ndg,index1,qmom,sweight,snp1,chid,iqs,ampn)
c
      call wdistr(igmw,imw,jmw,kmw,lmw,mmw,ntdsif,qmom,cx,cy,cz,denr)
c
      dampndt = (amp(lamp+1) - amp(lamp))/ dtj
      write(n6pk,*) ' amplitude derivative at ',telaps,' = ',dampndt
```

```
     $,' by delta (P) / delta (t)'
      write(n6pk,*) ' amplitude derivative at ',telaps,' = ',dampndtp
     $,' by LSODE'
      dampndt = dampndtp
c
      if(irep.eq.irepmx) goto 880
c
c   xsec file on fort.9
      call rdgip2(xsf1,igmw,ihtw,ihpw,mtmw,betat,ampn,dampndt,iqs
     $ ,vinv,dtn(itn),nts,rxsec,1)
c
c   set input and output units for flux guess
c   note that they must be duifferent???? not sure ....
      if(itn.eq.1) then
      j3 = 0
      call rwinpw(j3,igmw,ndg,chi)
      if(j3.ne.0) then
      write(n6pk,*) 'error readw:jobw ....'
      goto 9999
      endif
      endif
c   set convergence parameters for fixed source if not done so before
c      if(ifix.ne.1.and.itn.eq.1) then
c      j3 = 0
c      call read66(j3)
c       if(j3.ne.0) then
c       write(n6pk,*) 'error read66:jobw ....'
c       goto 9999
c       endif
c      endif
c
        if(itn.gt.irstrt) then
c
      evold = evnew
      if(imany.le.99) then
        write(filnam,'(a,i2.2)') 'to',imany
        fils = ' <tortinw >'
        call system(xnam//fils//filnam)
      else
        write(filnam2,'(a,i3.3)') 'to',imany
        fils = ' <tortinw >'
        call system(xnam//fils//filnam2)
      endif
       open(35,file='evfil',status='old')
       read(35,*) evdum
       close(35)
        evnew = evdum
c
c     read the new shape in
      call fluxr(shap1,z1c,iafl,1,cp,cx,cy,cz,enr,denr
     $ ,index,index1,izmw,imw,jmw,kmw,igmw)
      open(35,file='anflux',status='old',form='unformatted')
       read(35) (ansh1(iw),iw=1,indexm)
      close(35)
c     write to interface files
       if(irstrt.gt.0.and.itn.eq.irstrt+1) then
      open(35,file='shapet',status='old',form='unformatted')
      do iix=1,itn
       read(35)
      enddo
      write(35) (shap1(i),i=1,index)
      close(35)
      open(35,file='ashapet',status='old',form='unformatted')
      do iix=1,itn
       read(35)
      enddo
      write(35) (ansh1(i),i=1,indexm)
      close(35)
       else
      open(35,file='shapet',status='old',form='unformatted'
     $ ,access='append')
      write(35) (shap1(i),i=1,index)
      close(35)
      open(35,file='ashapet',status='old',form='unformatted'
     $ ,access='append')
      write(35) (ansh1(i),i=1,indexm)
      close(35)
       endif
c
        else
c   restart, read from interface files
c
      open(35,file='shapet',status='old',form='unformatted')
      do iix=1,itn
       read(35)    ! skip earlier records
      enddo
      read(35) (shap1(i),i=1,index)
      close(35)
      open(35,file='ashapet',status='old',form='unformatted')
      do iix=1,itn
       read(35)    ! skip earlier records
      enddo
      read(35) (ansh1(i),i=1,indexm)
      close(35)
c
        endif
c
       if(ipkmod.eq.0) then
      call ampil(ansh1,anadj,vinv,index,index1,igmw,iqs,1
     $ ,vr,dth,dz,imw,jmw,kmw,amplit,shap,indexm,mmw,sweight,1,n6pk,itn)
```

```
      call ampilh(shap1,scadj,vinv,index,index1,igmw,iqs,1
     $ ,vr,dth,dz,imw,jmw,kmw,amplit,shap,1,n6pk,itn)
        elseif(ipkmod.eq.1) then
      call ampilh(shap1,scadj,vinv,index,index1,igmw,iqs,1
     $ ,vr,dth,dz,imw,jmw,kmw,amplit,shap,1,n6pk,itn)
c
c   also normalize angular shape for qaddj
c
      do i=1,indexm
        anshl(i) = anshl(i)/ratio(itn+1)
      enddo
        endif
c
880    continue
c
      if(iout.eq.3) call outshp(n6pk,equal,irep,index,shap1,shap2,igmw)
c
      tx0 = 1.
      tx1 = 1.
      call delsor2(cig1,cig2,alambda,beta,index,index1,ndg
     $,igmw,xsfp,xsfs,shap1,shap2,amp,ldamp,idtj,dume1,dume2,dtn,dtnx
     $,amp0,itn,irestx,1,dtj,xsf0,xsf1,ihtw,mtmw,izmw,z1c,cp,imw,jmw
     $,kmw,ihpw,tx0,tx1,ampold,indexn,dume,izart,n6pk)
c
c      calculate total power
c
      call powcal(power,shap1,index,index1,igmw,xsfp
     $ ,vr,dth,dz,imw,jmw,kmw,rnu,pd)
c
c
         if(irstrt.eq.0.or.itn.gt.irstrt) then
c
c   print flux shape to a file
      if(imany.le.99) then
      call woutfl(shap1,index,0,48,filnam//"s"
     $ ,cx,cy,cz,imw,jmw,kmw,igmw,vr,dth,dz)
      else
      call woutfl(shap1,index,0,48,filnam2//"s"
     $ ,cx,cy,cz,imw,jmw,kmw,igmw,vr,dth,dz)
      endif
c
         endif
c
      write(n6pk,'(120(a1))') (equal,itest=1,115)
       write(n6pk,*)'power              = ',power,' fissions/sec'
       write(n6pk,*)'relative power     = ',power/pow0
      write(n6pk,*)'amplitude          = ',amplit
      write(n6pk,*)'relative amplitude = ',amplit/amplit0
      write(n6pk,*)'shape              = ',shap,' (volume and energy
     $ integrated)'
      write(n6pk,*)'eigenvalue         = ',evnew
c
c
       if(irep.eq.irepmx) goto 890
       irep = irep+1
       ijamp = ijamp - idtj(itn)
      close(52)    ! need to close so that in next loop it will be overriden
       goto 710
c      endif
890    continue
c   this is needed to keep initial values of y vector for next itn loop
c   the last arg. is : 1= y to resrep , otherwise resrep to y
      if(ipkmod.eq.0) then
      call flxgen(anfl1,anshl,indexm,amp(lamp+1))
      call scalit(anfl1,indexm,flux1,index,mmw,sweight)
      elseif(ipkmod.eq.1) then
      call flxgen(flux1,shap1,index,amp(lamp+1))
      endif
c
        call pownc2(pow0,shap1,index,igmw,xsfp,50,itn,vr,dth,dz,imw
     $ ,jmw,kmw,rnu,dtj,dtn(itn),amp(2),idtj(itn),51,dume3
     $ ,idtk(itn),52,dtk,shap2,dume)
c
      ampold = amp(lamp+1)
      amp0   = amp(lamp+1)
c
      imany = imany +1
      close(52,status='delete')
c     end dtn time loop
800    continue
9999   return
       end
::::::::::::::
jobw3.f
::::::::::::::
c234567890123456789012345678901234567890123456789012345678901234567890123456789012
      subroutine jobw3(iot,cig0,scadj,vr,dth,dz,z1c,xsts,xstp,xstd,xsfs
     $ ,xsfp,xsfd,xsss,xssp,xssd,xsf0,xsf1,cp,dum0,dum1,dumx
     $ ,rxsec,chi,chit,qmom,cx,cy,cz,enr,denr,anadj,snp1,pd,amp,flux1
     $ ,flux2,shap1,shap2,shap3,shap4,anfl1,anfl2,anshl,ansh2,ansh3
     $ ,ansh4,cig1,cig2,dume1,dume2,dume3,dumep1,dumep2,dumep3
     $ ,dumep4,dume,rhot3,rhot4,rhot5,xsf2)
c
      include 'comsd1.F'
      include 'comsd2.F'
      include 'comsd4.F'
      include 'comsd5.F'
      include 'comsd6.F'
      include 'comsd7.F'
      include 'comsd8.F'
```

```
      include 'comsd0.F'
      include 'comsdc.F'
c
c234567890123456789012345678901234567890123456789012345678901234567890123456789012
      real*4 cig0(indexn),scadj(index),vr(imw),dth(jmw),dz(kmw)
     $ ,z1c(index1),xsts(index),xstp(index),xstd(index),xsfs(index)
     $ ,xsfp(index),xsfd(index),xsss(index*igmw),xssp(index*igmw)
     $ ,xssd(index*igmw),xsf0(izart),xsf1(izart),cp(izmw),dum0(izart)
     $ ,dum1(izart),dumx(igmw*igmw*mtmw)
     $ ,rxsec(izart),chi(igmw),chit(igmw),qmom(index*lmw),cx(imw+1)
     $ ,cy(jmw+1),cz(kmw+1),enr(igmw+2),denr(igmw),anadj(indexm)
     $ ,snpl(idimsnpl),pd(index1),amp(ldamp+1),flux1(index)
     $ ,flux2(index),shap1(index),shap2(index),shap3(index),shap4(index)
     $ ,anfl1(indexm),anfl2(indexm),ansh1(indexm),ansh2(indexm)
     $ ,ansh3(indexm),ansh4(indexm),cig1(indexn),cig2(indexn)
     $ ,dume1(2*index),dume2(2*index*ndg),dume3(mdamp)
     $ ,dume(irestx),xsf2(izart)
c
c   for pkvar
      real*4 dumep1(index1),dumep2(index),dumep3(index1)
     $ ,dumep4(index),rhot3(index),rhot4(index),rhot5(index)
c
      real*4 telaps,rho,rlamda,extsrc,fterm,power,bett,hhi1
     $ ,rho2rp,rho1rp,rhonrp,rlam2rp,rlam1rp,rlamnrp,bet2rp,bet1rp
     $ ,tx0,tx1,ampn,dampndt
      real*4 zero
      zero = 0.e0
      iot = n6pk
      xnam = 'tortnew '
c
c    iquad =  quadratic interpolation flag for flux
c    jquad =  quadratic interpolation flag for rho
c
c   note that z1c is for initial zone map
c    idtnm = # of tn calc.
c
       equal = '='
c   open output unit
      open(n6pk,file='pkout',status='unknown')
      call xsetun(n6pk)
c
c   rhonew is newest
c   rho1   is previous
c   rho2   is oldest
c
c    betaeft is sum(betaef(i))
c
c    initialize
c
      ijamp = 0
      neq = ndg+1
      itol = 1
      ilsrto = 0
      if(srtol.lt.0..or.satol.lt.0.) then
       ilsrto = 1
      endif
      rtol = 1.0d0*abs(srtol)
      atol = 1.0d0*abs(satol)
      itask = 1
      istate = 1
      iopt = 0
      lrw = 500
      liw = 100
      mf = 22
      t    = zero
      tout= zero
      telaps = zero
c
c     reset irepmx so that 1 extra iteration is performed by default
      irepmx = irepmx + 1
      evw = evnew
      outflg = .false.
      if(iout.gt.0) outflg = .true.
      rho1    = zero
      rho2    = zero
      rhonew = zero
      rho     = zero
      rlam1    = zero
      rlam2    = zero
      rlamnew = zero
      rlamda   = zero
      q1       = zero
      q2       = zero
      qnew    = zero
      extsrc = zero
      fterm   = zero
      power   = zero
      do i=1,ndg
       betae1(i) = zero
       betae2(i) = zero
       betnew(i) = zero
       betaef(i) = zero
       csubi(i) = zero
      enddo
      bet1   = zero
      bet2   = zero
      betn   = zero
      bett   = zero
c
      call rkld0(betaef0,alambda,rlamda0,n6pk,ndg,amplit0,y)
```

```
c   this is needed to keep initial values of y vector
c   the last arg. is : 1= y to resrep , otherwise resrep to y
      call rkldrp(y,resrep,neq,1)
c
      rhonew = rho0
      rlamnew = rlamda0
      betn = zero
      do ii=1,ndg
      betnew(ii) = betaef0(ii)
      betn = betn + betnew(ii)
      enddo
c
c   also normalize angular shape for qaddj
c
      do i=1,indexm
        anshl(i) = anshl(i)/ratio(1)
      enddo
c
      iquad = .false.
c     start dtn time loop
      do 800 itn = 1, idtnm
c   set lsode parameters
      if(ilsrto.eq.0) then
       rtol = dtn(itn)/idtk(itn)*srtol
       atol = dtn(itn)/idtk(itn)*satol
      endif
      if(ilsrto.eq.0.or.itn.eq.1) then
       write(n6pk,*) '**********************************'
       write(n6pk,*) ' rtol = ',rtol,' at time interval ',itn
       write(n6pk,*) ' atol = ',atol,' at time interval ',itn
       write(n6pk,*) '**********************************'
      endif
c
      irep = 0      ! this is to repeat pk calc. and update
      hi1 = dtn(itn)
      hi2 = dtn(itn)
c
c   shift fluxes
      call shftf1(shap1,shap2,shap3,anshl,ansh2,ansh3,index,indexm)
      call shftcg(cig1,cig2,indexn)
c   note : cig1 is latest (cig^(n+1)) & cig2 is prev. (cig^(n))
c
      if(itn.eq.1) then
        amp(1) = amplit0
        rlamda = rlamda0
        fterm  = fterm0
       hhi1=zero
c  save last amplitude as first for next time step
       ampold = amplit0
       amp0   = amplit0
      else
        rlamda = rlamold
        fterm  = ftermold
       hhi1 = dtn(itn-1)
      endif
c
      if(itn.eq.ilpend) then
        call xsecr(xsf1,ihpw,mtmw,igmw,ntsigp2)
      endif
c
c
c    generate precursor densities using last calculated delayed
c    neutron source and load onto y array
c
c   this next call causes discontinuities in y b/c csubi does not
c   match the value calculated by lsode
c   use with caution
      call rkld(y,rlamda,fterm,n6pk,ndg,cig2,scadj,imw,jmw,kmw,igmw
     $ ,vr,dth,dz,indexn,index,itn,index1,dume1,chid,ampold,amp0,amplit)
c
c   extrapolate to calculate first flux shape and normalize
c
c   extrapolate angular and scalar shape
c
        if(ipkmod.eq.0) then
      call extrap(anshl,ansh2,ansh3,hhi1,hi2,indexm,iextrp,n6pk)
      call extrap(shap1,shap2,shap3,hhi1,hi2,index,iextrp,n6pk)
      call ampil(anshl,anadj,vinv,index,index1,igmw,iqs,0
     $ ,vr,dth,dz,imw,jmw,kmw,amplit,shap,indexm,mmw,sweight,1,n6pk,itn)
      call ampilh(shap1,scadj,vinv,index,index1,igmw,iqs,0
     $ ,vr,dth,dz,imw,jmw,kmw,amplit,shap,1,n6pk,itn)
        elseif(ipkmod.eq.1) then
      call extrap(shap1,shap2,shap3,hhi1,hi2,index,iextrp,n6pk)
      call ampilh(shap1,scadj,vinv,index,index1,igmw,iqs,0
     $ ,vr,dth,dz,imw,jmw,kmw,amplit,shap,1,n6pk,itn)
        endif
c
c    idtj = # of tj calc.
710   jquad = .false.
c     clear rwork for lsode
      do itt2=1,500
         rwork(itt2) = 0.0d0
      enddo
      do itt2=1,100
         iwork(itt2) = 0
      enddo
      if(irep.eq.0) then
c   svae old react.,gen. time and effective betas for second pass
      call rkldrp(y,resrep,ndg+1,1)
        rho2rp = rho2
```

```
        rho1rp = rho1
        rhonrp = rhonew
        rlam2rp = rlam2
        rlam1rp = rlam1
        rlamnrp = rlamnew
        bet2rp = zero
        bet1rp = zero
        betnrp = betn
        do i=1,ndg
        bete2rp(i) = betae2(i)
        bete1rp(i) = betae1(i)
        betenrp(i) = betnew(i)
        bet2rp     = bet2rp + bete2rp(i)
        bet1rp     = bet1rp + bete1rp(i)
        enddo
      elseif(irep.gt.0) then
c   put old data back
       call rkldrp(y,resrep,neq,0)
        rho2 = rho2rp
        rho1 = rho1rp
        rhonew = rhonrp
        rlam2 = rlam2rp
        rlam1 = rlam1rp
        rlamnew = rlamnrp
        bet2 = zero
        bet1 = zero
        betn = betnrp
        do i=1,ndg
        betae2(i) = bete2rp(i)
        betae1(i) = bete1rp(i)
        betnew(i) = betenrp(i)
        bet2     = bet2 + betae2(i)
        bet1     = bet1 + betae1(i)
        enddo
      endif
c
c     start dtj time loop
      lamp = 0
      dtj = zero
      dtj = dtn(itn)/idtj(itn)
c  open and close fort.45 and fort.46 to force buffer dump
      if(itn.eq.1.and.irep.eq.0) then
        open(45,status='unknown')
        open(46,status='unknown')
      else
        open(45,status='old',access='append')
        open(46,status='old',access='append')
      endif
      do 700 itj=1, idtj(itn)
c  transfer the times for irep = 0 case
      if(irep.eq.0.and.itj.eq.1.and.irepmx.gt.0) then
        ttrep1 = telaps
        ttrep2 = tout
        ttrep3 = t
      elseif(irep.gt.0.and.itj.eq.1.and.irepmx.gt.0) then
        telaps = ttrep1
        tout   = ttrep2
        t      = ttrep3
      endif
      hj1    = dtj
      hj2    = dtj
      ijamp = ijamp + 1
      if(itj.gt.2)    jquad = .true.
      if(itj.eq.1) istate = 1
c
      tnewj = itj*dtj
c
c  linear interp.
      if(ipkmod.eq.0) then
      call linter(ansh2,ansh1,ansh4,hi2,tnewj,indexm)
      call linter(shap2,shap1,shap4,hi2,tnewj,index)
      elseif(ipkmod.eq.1) then
      call linter(shap2,shap1,shap4,hi2,tnewj,index)
      endif
c
      idim  = ndg
c  calculate initial reac. at time 0. with pert. xsecs if
c  step perturbation and first pass
      if((itn.eq.1.and.itj.eq.1).or.
     $   (itn.eq.ilpend.and.itj.eq.1)) then
c
      call dxsec2(xsts,xstp,xstd,xsfs,xsfp,xsfd
     $,xsss,xssp,xssd,xsf0,xsf1,index
     $,z1c,cp,igmw,index1,ihpw,ihtw,mtmw,izmw
     $,1,dum0,dum1,dumx)
c
      if((itn.eq.1.and.itj.eq.1)) then
      call pkvar(rho,fterm,betaef,betaeft,rlamda,extsrc
     $ ,idim,shap2,ansh2,rhot3,rhot4,rhot5,anadj,xstd,scadj,xssd
     $ ,dumep1,xsfd,chit,dumep2,vr,dth,dz,dumep3,xsfp,dumep4
     $ ,index,index1,indexm,imw,jmw,kmw,igmw,mmw,ipkmod)
c
      call mvpk(rho2,rho1,rhonew,rho,bett,betae2,betae1,betnew
     $ ,betaef,bet2,bet1,betn,rlam2,rlam1,rlamnew,rlamda,q2,q1,qnew
     $ ,extsrc,n6pk,telaps,irep,irepmx,fterm,ndg,iout)
      endif
      endif
c
      if(itn.eq.1.and.itj.eq.1.and.irep.eq.irepmx) then
        open(35,file='pkparm',status='unknown')
```

```
        write(35,*) " elapsed time     rho        betaeft      rlamda"
        write(35,'(1p,2x,4(e10.3,2x))') telaps,rho,betaeft,rlamda
        close(35)
       endif
       if((itn.gt.1.and.itj.eq.1.and.irep.eq.irepmx).or.
      $ (iout.gt.0.and.irep.eq.irepmx)) then
         open(35,file='pkparm',status='old',access='append')
         write(35,'(1p,2x,4(e10.3,2x))') telaps,rho,betaeft,rlamda
         close(35)
       endif
c
       if(itj.eq.idtj(itn).and.irep.eq.irepmx) then
c  this is for recalculation of csubi in rkld, save the last values
       rlamold = rlamda
       ftermold = fterm
       endif
c
       call pkvar(rho,fterm,betaef,betaeft,rlamda,extsrc
      $ ,idim,shap4,ansh4,rhot3,rhot4,rhot5,anadj,xstd,scadj,xssd
      $ ,dumep1,xsfd,chit,dumep2,vr,dth,dz,dumep3,xsfp,dumep4
      $ ,index,index1,indexm,imw,jmw,kmw,igmw,mmw,ipkmod)
c
       call mvpk(rho2,rho1,rhonew,rho,bett,betae2,betae1,betnew
      $ ,betaef,bet2,bet1,betn,rlam2,rlam1,rlamnew,rlamda,q2,q1,qnew
      $ ,extsrc,n6pk,telaps+dtj,irep,irepmx,fterm,ndg,iout)
c
c    idtk = # of tk calc.
c
       dtk = zero
       dtk  = dtn(itn)/idtk(itn)
       time = zero
       do 600 itk=1,idtk(itn)/idtj(itn)
c       telaps = telaps + dtk
       tout = t + dtk
c
       call pkcal(rho,betaef,rlamda,extsrc,betaeft,ndg,time)
       rhop = rho
       do i=1,ndg
       betap(i) = betaef(i)
       enddo
       rlamp = rlamda
       betp = betaeft
       extp = extsrc
c
       call lsode(f,neq,y,t,tout,itol,rtol,atol,itask,istate,
      $    iopt,rwork,lrw,iwork,liw,jex,mf)
c
       if (istate .lt. 0) then
       write(n6pk,90)istate
90     format(///22h error halt.. istate =,i3)
c       stop ' solvem : lsode '
       write(6,*) ' stop .... solvem : lsode '
        goto 9999
       endif
c
        call rkunld(y,csubi,ndg,amplit0,n6pk,itn,itj,tout,irep,iout)
       if(irep.eq.irepmx) then
c        call rkunld(y,csubi,ndg,amplit0,n6pk,itn,itj,tout,irep,iout)
c        write(41,*) telaps+(itk-1)*dtk,rhop,time
         sdump = y(1)
        write(52,*) sdump
       endif
c
       time = time + dtk
c
600    continue
c
       lamp = lamp + 1
       amp(lamp+1) = y(1)
c
       if(itn.eq.1) then
         tltot = zero
         tjtot = dtns(1)
         tjtot = tjtot*itj/idtj(itn)
         tltot = tltot + tjtot
       elseif(itn.gt.1) then
         tltot = dtns(itn-1)
         tjtot = dtns(itn) - dtns(itn-1)
         tjtot = tjtot*itj/idtj(itn)
         tltot = tltot + tjtot
       endif
       telaps = tltot
c
       write(45,*) telaps,amp(lamp+1)/amplit0,irep
c      end dtj time loop
700    continue
       close(45)
       close(46)
c  if last pass over this dtn, save the amplitute
       if(irep.eq.irepmx) then
       rewind(52)
       i1 = 1
740    read(52,*,err=750,end=750) dume3(i1)
       i1 = i1 + 1
       goto 740
750    i1 = i1 - 1
       if(itn.eq.1) then
        open(35,file='amplitute',status='unknown',form='unformatted')
       else
        open(35,file='amplitute',status='old',form='unformatted'
```

```
     $ ,access='append')
      endif
      write(35) (dume3(ili),ili=1,il)
      close(35)
      endif
c   if this is the last dtn time interval then
c    there is no need to update shape and precursors, just exit.
      if(itn.eq.idtnm.and.irep.eq.irepmx) goto 890
c   test
      write(68,*) 'itn = ',itn,' lamp = ',lamp
      write(68,*) ' amp'
      do izgak=1,lamp+1
      write(68,*) amp(izgak)
      enddo
c end test
c
c  update precursors , calculate fixed source and update shape with
c  new tort calculation.
      dtnx = dtn(itn)/idtj(itn)
c
      tx0 = 1.
      tx1 = 1.
      call delsor2(cig1,cig2,alambda,beta,index,index1,ndg
     $,igmw,xsfp,xsfs,shap1,shap2,amp,ldamp,idtj,dume1,dume2,dtn,dtnx
     $,amp0,itn,irestx,0,dtj,xsf0,xsf1,ihtw,mtmw,izmw,z1c,cp,imw,jmw
     $,kmw,ihpw,tx0,tx1,ampold,indexn,dume,izart,n6pk)
c
      ampn = amp(lamp+1)
      call qaddj(ansh2,cig1,vinv,dtn(itn),alambda,mmw,lmw
     $ ,imw,jmw,kmw,igmw,ndg,index1,qmom,sweight,snp1,chid,iqs,ampn)
c
      call wdistr(igmw,imw,jmw,kmw,lmw,mmw,ntdsif,qmom,cx,cy,cz,denr)
c
      dampndt = (amp(lamp+1) - amp(lamp))/ dtj
      write(n6pk,*) ' amplitude derivative at ',telaps,' = ',dampndt
     $,' by delta (P) / delta (t)'
      write(n6pk,*) ' amplitude derivative at ',telaps,' = ',dampndtp
     $,' by LSODE'
      dampndt = dampndtp
c
      if(irep.eq.irepmx) goto 880
c
c   xsec file on fort.9
      call rdgip2(xsf1,igmw,ihtw,ihpw,mtmw,betat,ampn,dampndt,iqs
     $ ,vinv,dtn(itn),nts,rxsec,1)
c
c   set input and output units for flux guess
c   note that they must be duifferent???? not sure ....
      if(itn.eq.1) then
      j3 = 0
      call rwinpw(j3,igmw,ndg,chi)
      if(j3.ne.0) then
      write(n6pk,*) 'error readw:jobw ....'
      goto 9999
      endif
      endif
c   set convergence parameters for fixed source if not done so before
c      if(ifix.ne.1.and.itn.eq.1) then
c      j3 = 0
c      call read66(j3)
c       if(j3.ne.0) then
c       write(n6pk,*) 'error read66:jobw ....'
c       goto 9999
c       endif
c      endif
c
        if(itn.gt.irstrt) then
c
      evold = evnew
      if(imany.le.99) then
        write(filnam,'(a,i2.2)') 'to',imany
        fils = ' <tortinw >'
        call system(xnam//fils//filnam)
      else
        write(filnam2,'(a,i3.3)') 'to',imany
        fils = ' <tortinw >'
        call system(xnam//fils//filnam2)
      endif
       open(35,file='evfil',status='old')
       read(35,*) evdum
       close(35)
        evnew = evdum
c
c     read the new shape in
      call fluxr(shap1,z1c,iafl,1,cp,cx,cy,cz,enr,denr
     $ ,index,index1,izmw,imw,jmw,kmw,igmw)
       open(35,file='anflux',status='old',form='unformatted')
       read(35) (ansh1(iw),iw=1,indexm)
      close(35)
c     write to interface files
       if(irstrt.gt.0.and.itn.eq.irstrt+1) then
      open(35,file='shapet',status='old',form='unformatted')
      do iix=1,itn
       read(35)
      enddo
      write(35) (shap1(i),i=1,index)
      close(35)
      open(35,file='ashapet',status='old',form='unformatted')
      do iix=1,itn
       read(35)
```

```
      enddo
      write(35) (anshl(i),i=1,indexm)
      close(35)
       else
      open(35,file='shapet',status='old',form='unformatted'
     $ ,access='append')
      write(35) (shap1(i),i=1,index)
      close(35)
      open(35,file='ashapet',status='old',form='unformatted'
     $ ,access='append')
      write(35) (anshl(i),i=1,indexm)
      close(35)
       endif
c
         else
c   restart, read from interface files
c
      open(35,file='shapet',status='old',form='unformatted')
      do iix=1,itn
       read(35)   ! skip earlier records
      enddo
      read(35) (shap1(i),i=1,index)
      close(35)
      open(35,file='ashapet',status='old',form='unformatted')
      do iix=1,itn
       read(35)   ! skip earlier records
      enddo
      read(35) (anshl(i),i=1,indexm)
      close(35)
c
         endif
c
        if(ipkmod.eq.0) then
      call ampil(anshl,anadj,vinv,index,index1,igmw,iqs,1
     $ ,vr,dth,dz,imw,jmw,kmw,amplit,shap,indexm,mmw,sweight,1,n6pk,itn)
       call ampilh(shap1,scadj,vinv,index,index1,igmw,iqs,1
     $ ,vr,dth,dz,imw,jmw,kmw,amplit,shap,1,n6pk,itn)
         elseif(ipkmod.eq.1) then
      call ampilh(shap1,scadj,vinv,index,index1,igmw,iqs,1
     $ ,vr,dth,dz,imw,jmw,kmw,amplit,shap,1,n6pk,itn)
c
c   also normalize angular shape for qaddj
c
      do i=1,indexm
        anshl(i) = anshl(i)/ratio(itn+1)
      enddo
        endif
c
880    continue
c
      if(iout.eq.3) call outshp(n6pk,equal,irep,index,shap1,shap2,igmw)
c
      tx0 = 1.
      tx1 = 1.
      call delsor2(cig1,cig2,alambda,beta,index,index1,ndg
     $,igmw,xsfp,xsfs,shap1,shap2,amp,ldamp,idtj,dume1,dume2,dtn,dtnx
     $,amp0,itn,irestx,1,dtj,xsf0,xsf1,ihtw,mtmw,izmw,zlc,cp,imw,jmw
     $,kmw,ihpw,tx0,tx1,ampold,indexn,dume,izart,n6pk)
c
c      calculate total power
c
      call powcal(power,shap1,index,index1,igmw,xsfp
     $ ,vr,dth,dz,imw,jmw,kmw,rnu,pd)
c
c
         if(irstrt.eq.0.or.itn.gt.irstrt) then
c
c   print flux shape to a file
      if(imany.le.99) then
      call woutfl(shap1,index,0,48,filnam//"s"
     $ ,cx,cy,cz,imw,jmw,kmw,igmw,vr,dth,dz)
      else
      call woutfl(shap1,index,0,48,filnam2//"s"
     $ ,cx,cy,cz,imw,jmw,kmw,igmw,vr,dth,dz)
      endif
c
         endif
c
      write(n6pk,'(120(a1))') (equal,itest=1,115)
       write(n6pk,*)'power                = ',power,' fissions/sec'
       write(n6pk,*)'relative power       = ',power/pow0
      write(n6pk,*)'amplitude            = ',amplit
      write(n6pk,*)'relative amplitude = ',amplit/amplit0
      write(n6pk,*)'shape                = ',shap,' (volume and energy
     $ integrated)'
      write(n6pk,*)'eigenvalue           = ',evnew
c
c
       if(irep.eq.irepmx) goto 890
       irep = irep+1
       ijamp = ijamp - idtj(itn)
      close(52)   ! need to close so that in next loop it will be overriden
       goto 710
c      endif
890   continue
c   this is needed to keep initial values of y vector for next itn loop
c   the last arg. is : 1= y to resrep , otherwise resrep to y
      if(ipkmod.eq.0) then
      call flxgen(anfl1,anshl,indexm,amp(lamp+1))
      call scalit(anfl1,indexm,flux1,index,mmw,sweight)
```

```
      elseif(ipkmod.eq.1) then
      call flxgen(flux1,shap1,index,amp(lamp+1))
      endif
c
        call pownc2(pow0,shap1,index,igmw,xsfp,50,itn,vr,dth,dz,imw
     $ ,jmw,kmw,rnu,dtj,dtn(itn),amp(2),idtj(itn),51,dume3
     $ ,idtk(itn),52,dtk,shap2,dume)
c
      ampold = amp(lamp+1)
      amp0   = amp(lamp+1)
c
      close(52,status='delete')
      imany = imany +1
c     end dtn time loop
800   continue
9999  return
      end
::::::::::::::
jobw5.f
::::::::::::::
c234567890123456789012345678901234567890123456789012345678901234567890123456789012
      subroutine jobw5(iot,cig0,scadj,vr,dth,dz,z1c,xsts,xstp,xstd,xsfs
     $ ,xsfp,xsfd,xsss,xssp,xssd,xsf0,xsf1,cp,dum0,dum1,dumx
     $ ,rxsec,chi,chit,qmom,cx,cy,cz,enr,denr,anadj,snp1,pd,amp,flux1
     $ ,flux2,shap1,shap2,shap3,shap4,anfl1,anfl2,ansh1,ansh2,ansh3
     $ ,ansh4,cig1,cig2,dume1,dume2,dume3,dumep1,dumep2,dumep3
     $ ,dumep4,dume,rhot3,rhot4,rhot5,xsf2)
c
      include 'comsd1.F'
      include 'comsd2.F'
      include 'comsd4.F'
      include 'comsd5.F'
      include 'comsd6.F'
      include 'comsd7.F'
      include 'comsd8.F'
      include 'comsd0.F'
      include 'comsdc.F'
c
c234567890123456789012345678901234567890123456789012345678901234567890123456789012
      real*4 cig0(indexn),scadj(index),vr(imw),dth(jmw),dz(kmw)
     $ ,z1c(index1),xsts(index),xstp(index),xstd(index),xsfs(index)
     $ ,xsfp(index),xsfd(index),xsss(index*igmw),xssp(index*igmw)
     $ ,xssd(index*igmw),xsf0(izart),xsf1(izart),cp(izmw),dum0(izart)
     $ ,dum1(izart),dumx(igmw*igmw*mtmw)
     $ ,rxsec(izart),chi(igmw),chit(igmw),qmom(index*lmw),cx(imw+1)
     $ ,cy(jmw+1),cz(kmw+1),enr(igmw+2),denr(igmw),anadj(indexm)
     $ ,snp1(idimsnp1),pd(index1),amp(ldamp+1),flux1(index)
     $ ,flux2(index),shap1(index),shap2(index),shap3(index),shap4(index)
     $ ,anfl1(indexm),anfl2(indexm),ansh1(indexm),ansh2(indexm)
     $ ,ansh3(indexm),ansh4(indexm),cig1(indexn),cig2(indexn)
     $ ,dume1(2*index),dume2(2*index*ndg),dume3(mdamp)
     $ ,dume(irestx),xsf2(izart)
c
c   for pkvar
      real*4 dumep1(index1),dumep2(index),dumep3(index1)
     $ ,dumep4(index),rhot3(index),rhot4(index),rhot5(index)
c
      real*4 telaps,rho,rlamda,extsrc,fterm,power,bett,tltot,tx,hhi1
     $ ,rho2rp,rho1rp,rhonrp,rlam2rp,rlam1rp,rlamnrp,bet2rp,bet1rp
     $ ,tx0,tx1,ampn,dampndt
      real*4 zero
      zero = 0.e0
      iot = n6pk
      xnam = 'tortnew '
c
c    iquad =  quadratic interpolation flag for flux
c    jquad =  quadratic interpolation flag for rho
c
c   note that z1c is for initial zone map
c    idtnm = # of tn calc.
c
       equal = '='
c   open output unit
      open(n6pk,file='pkout',status='unknown')
      call xsetun(n6pk)
c
c   rhonew is newest
c   rho1   is previous
c   rho2   is oldest
c
c    betaeft is sum(betaef(i))
c
c    initialize
c
      ijamp = 0
      neq = ndg+1
      itol = 1
      ilsrto = 0
      if(srtol.lt.0..or.satol.lt.0.) then
       ilsrto = 1
      endif
      rtol = 1.0d0*abs(srtol)
      atol = 1.0d0*abs(satol)
      itask = 1
      istate = 1
      iopt = 0
      lrw = 500
      liw = 100
      mf = 22
      t   = zero
```

```
      tout= zero
      telaps = zero
c
c     reset irepmx so that 1 extra iteration is performed by default
      irepmx = irepmx + 1
      evw = evnew
      outflg = .false.
      if(iout.gt.0) outflg = .true.
      rho1   = zero
      rho2   = zero
      rhonew = zero
      rho    = zero
      rlam1   = zero
      rlam2   = zero
      rlamnew = zero
      rlamda  = zero
      q1     = zero
      q2     = zero
      qnew   = zero
      extsrc = zero
      fterm  = zero
      power  = zero
      do i=1,ndg
       betae1(i) = zero
       betae2(i) = zero
       betnew(i) = zero
       betaef(i) = zero
       csubi(i) = zero
      enddo
      bet1  = zero
      bet2  = zero
      betn  = zero
      bett  = zero
c
      call rkld0(betaef0,alambda,rlamda0,n6pk,ndg,amplit0,y)
c   this is needed to keep initial values of y vector
c   the last arg. is : 1= y to resrep , otherwise resrep to y
      call rkldrp(y,resrep,neq,1)
c
      rhonew = rho0
      rlamnew = rlamda0
      betn = zero
      do ii=1,ndg
      betnew(ii) = betaef0(ii)
      betn = betn + betnew(ii)
      enddo
c
c   also normalize angular shape for qaddj
c
      do i=1,indexm
        ansh1(i) = ansh1(i)/ratio(1)
      enddo
c
      iquad = .false.
c     start dtn time loop
      do 800 itn = 1, idtnm
c   set lsode parameters
      if(ilsrto.eq.0) then
       rtol = dtn(itn)/idtk(itn)*srtol
       atol = dtn(itn)/idtk(itn)*satol
      endif
      if(ilsrto.eq.0.or.itn.eq.1) then
       write(n6pk,*) '**********************************'
       write(n6pk,*) ' rtol = ',rtol,' at time interval ',itn
       write(n6pk,*) ' atol = ',atol,' at time interval ',itn
       write(n6pk,*) '**********************************'
      endif
c
      irep = 0      ! this is to repeat pk calc. and update
      hi1 = dtn(itn)
      hi2 = dtn(itn)
c
c   shift fluxes
      call shftf1(shap1,shap2,shap3,ansh1,ansh2,ansh3,index,indexm)
      call shftcg(cig1,cig2,indexn)
c   note : cig1 is latest (cig^(n+1)) & cig2 is prev. (cig^(n))
c
      if(itn.eq.1) then
        amp(1) = amplit0
        rlamda = rlamda0
        fterm  = fterm0
       hhi1=zero
c  save last amplitude as first for next time step
       ampold = amplit0
       amp0   = amplit0
      else
        rlamda = rlamold
        fterm  = ftermold
       hhi1 = dtn(itn-1)
      endif
c     generate precursor densities using last calculated delayed
c     neutron source and load onto y array
c
c   this next call causes discontinuities in y b/c csubi does not
c   match the value calculated by lsode
c   use with caution
      call rkld(y,rlamda,fterm,n6pk,ndg,cig2,scadj,imw,jmw,kmw,igmw
     $ ,vr,dth,dz,indexn,index,itn,index1,dume1,chid,ampold,amp0,amplit)
c
c
```

```
c   find total linear pert. elapse time
       if(itn.eq.1) then
        tltot = zero
        do i=1,ilpend-1
         tltot = tltot + dtn(i)
        enddo
       endif
c
c
c   extrapolate to calculate first flux shape and normalize
c
c   extrapolate angular and scalar shape
c
         if(ipkmod.eq.0) then
       call extrap(ansh1,ansh2,ansh3,hhi1,hi2,indexm,iextrp,n6pk)
       call extrap(shap1,shap2,shap3,hhi1,hi2,index,iextrp,n6pk)
       call ampil(ansh1,anadj,vinv,index,index1,igmw,iqs,0
      $ ,vr,dth,dz,imw,jmw,kmw,amplit,shap,indexm,mmw,sweight,1,n6pk,itn)
       call ampilh(shap1,scadj,vinv,index,index1,igmw,iqs,0
      $ ,vr,dth,dz,imw,jmw,kmw,amplit,shap,1,n6pk,itn)
         elseif(ipkmod.eq.1) then
       call extrap(shap1,shap2,shap3,hhi1,hi2,index,iextrp,n6pk)
       call ampilh(shap1,scadj,vinv,index,index1,igmw,iqs,0
      $ ,vr,dth,dz,imw,jmw,kmw,amplit,shap,1,n6pk,itn)
         endif
c
c    idtj = # of tj calc.
710    jquad = .false.
c    clear rwork for lsode
       do itt2=1,500
          rwork(itt2) = 0.0d0
       enddo
       do itt2=1,100
          iwork(itt2) = 0
       enddo
       if(irep.eq.0) then
c   svae old react.,gen. time and effective betas for second pass
         call rkldrp(y,resrep,ndg+1,1)
           rho2rp = rho2
           rho1rp = rho1
           rhonrp = rhonew
           rlam2rp = rlam2
           rlam1rp = rlam1
           rlamnrp = rlamnew
           bet2rp = zero
           bet1rp = zero
           betnrp = betn
           do i=1,ndg
           bete2rp(i) = betae2(i)
           bete1rp(i) = betae1(i)
           betenrp(i) = betnew(i)
           bet2rp     = bet2rp + bete2rp(i)
           bet1rp     = bet1rp + bete1rp(i)
           enddo
         elseif(irep.gt.0) then
c   put old data back
          call rkldrp(y,resrep,neq,0)
           rho2 = rho2rp
           rho1 = rho1rp
           rhonew = rhonrp
           rlam2 = rlam2rp
           rlam1 = rlam1rp
           rlamnew = rlamnrp
           bet2 = zero
           bet1 = zero
           betn = betnrp
           do i=1,ndg
           betae2(i) = bete2rp(i)
           betae1(i) = bete1rp(i)
           betnew(i) = betenrp(i)
           bet2     = bet2 + betae2(i)
           bet1     = bet1 + betae1(i)
           enddo
         endif
c      start dtj time loop
         lamp = 0
         dtj = zero
         dtj = dtn(itn)/idtj(itn)
c  open and close fort.45 and fort.46 to force buffer dump
         if(itn.eq.1.and.irep.eq.0) then
           open(45,status='unknown')
           open(46,status='unknown')
         else
           open(45,status='old',access='append')
           open(46,status='old',access='append')
         endif
         do 700 itj=1, idtj(itn)
c  transfer the times for irep = 0 case
         if(irep.eq.0.and.itj.eq.1.and.irepmx.gt.0) then
           ttrep1 = telaps
           ttrep2 = tout
           ttrep3 = t
         elseif(irep.gt.0.and.itj.eq.1.and.irepmx.gt.0) then
           telaps = ttrep1
           tout   = ttrep2
           t      = ttrep3
         endif
         hj1    = dtj
         hj2    = dtj
         ijamp = ijamp + 1
```

```
      if(itj.gt.2)    jquad = .true.
      if(itj.eq.1) istate = 1
c
      tnewj = itj*dtj
c
c  linear interp.
      if(ipkmod.eq.0) then
      call linter(ansh2,ansh1,ansh4,hi2,tnewj,indexm)
      call linter(shap2,shap1,shap4,hi2,tnewj,index)
      elseif(ipkmod.eq.1) then
      call linter(shap2,shap1,shap4,hi2,tnewj,index)
      endif
c
      idim  = ndg
c  calculate initial reac. at time 0. with pert. xsecs if
c  step perturbation and first pass
c
      if(itn.eq.1.and.itj.eq.1.and.irep.eq.irepmx) then
        open(35,file='pkparm',status='unknown')
        write(35,*) " elapsed time     rho        betaeft      rlamda"
        write(35,'(1p,2x,4(e10.3,2x))') telaps,rhonew,betaeft,rlamda
        close(35)
      endif
      if((itn.gt.1.and.itj.eq.1.and.irep.eq.irepmx).or.
     $ (iout.gt.0.and.irep.eq.irepmx)) then
        open(35,file='pkparm',status='old',access='append')
        write(35,'(1p,2x,4(e10.3,2x))') telaps,rhonew,betaeft,rlamda
        close(35)
      endif
c
      if(itj.eq.idtj(itn).and.irep.eq.irepmx) then
c  this is for recalculation of csubi in rkld, save the last values
      rlamold = rlamda
      ftermold = fterm
      endif
c  calc. pert xsecs for this itj
      tx = (telaps+dtj)/tltot
      if(itn.ge.ilpend) tx =1.
      if(tx.gt.1.) tx = 1.
      call dxsec5(xsts,xstp,xstd,xsfs,xsfp,xsfd,xsss,xssp,xssd
     $,xsf1,index,z1c,cp,igmw,index1,ihpw,ihtw,mtmw,izmw
     $,dum0,dum1,dumx,tx)
      call pkvar(rho,fterm,betaef,betaeft,rlamda,extsrc
     $ ,idim,shap4,ansh4,rhot3,rhot4,rhot5,anadj,xstd,scadj,xssd
     $ ,dumep1,xsfd,chit,dumep2,vr,dth,dz,dumep3,xsfp,dumep4
     $ ,index,index1,indexm,imw,jmw,kmw,igmw,mmw,ipkmod)
c
      call mvpk(rho2,rho1,rhonew,rho,bett,betae2,betae1,betnew
     $ ,betaef,bet2,bet1,betn,rlam2,rlam1,rlamnew,rlamda,q2,q1,qnew
     $ ,extsrc,n6pk,telaps+dtj,irep,irepmx,fterm,ndg,iout)
c
c    idtk = # of tk calc.
c
      dtk = zero
      dtk  = dtn(itn)/idtk(itn)
      time = zero
      do 600 itk=1,idtk(itn)/idtj(itn)
c      telaps = telaps + dtk
      tout = t + dtk
c
      call pkcal(rho,betaef,rlamda,extsrc,betaeft,ndg,time)
      rhop = rho
      do i=1,ndg
      betap(i) = betaef(i)
      enddo
      rlamp = rlamda
      betp = betaeft
      extp = extsrc
c
      call lsode(f,neq,y,t,tout,itol,rtol,atol,itask,istate,
     $    iopt,rwork,lrw,iwork,liw,jex,mf)
c
      if (istate .lt. 0) then
      write(n6pk,90)istate
90    format(///22h error halt.. istate =,i3)
c      stop ' solvem : lsode '
      write(6,*) ' stop .... solvem : lsode '
       goto 9999
      endif
c
       call rkunld(y,csubi,ndg,amplit0,n6pk,itn,itj,tout,irep,iout)
      if(irep.eq.irepmx) then
c       call rkunld(y,csubi,ndg,amplit0,n6pk,itn,itj,tout,irep,iout)
       write(41,*) telaps+(itk-1)*dtk,rhop,time
        sdump = y(1)
       write(52,*) sdump
      endif
c
      time = time + dtk
c
600   continue
c
      lamp = lamp + 1
      amp(lamp+1) = y(1)
c
      if(itn.eq.1) then
        tltot6 = zero
        tjtot6 = dtns(1)
        tjtot6 = tjtot6*itj/idtj(itn)
        tltot6 = tltot6 + tjtot6
```

```
      elseif(itn.gt.1) then
        tltot6 = dtns(itn-1)
        tjtot6 = dtns(itn) - dtns(itn-1)
        tjtot6=tjtot6*itj/idtj(itn)
        tltot6 = tltot6 + tjtot6
      endif
      telaps = tltot6
c
      write(45,*) telaps,amp(lamp+1)/amplit0,irep
c     end dtj time loop
700   continue
      close(45)
      close(46)
c  if last pass over this dtn, save the amplitute
      if(irep.eq.irepmx) then
      rewind(52)
      i1 = 1
740   read(52,*,err=750,end=750) dume3(i1)
      i1 = i1 + 1
      goto 740
750   i1 = i1 - 1
      if(itn.eq.1) then
       open(35,file='amplitute',status='unknown',form='unformatted')
      else
       open(35,file='amplitute',status='old',form='unformatted'
     $ ,access='append')
      endif
      write(35) (dume3(ii1),ii1=1,i1)
      close(35)
      endif
c   if this is the last dtn time interval then
c    there is no need to update shape and precursors, just exit.
      if(itn.eq.idtnm.and.irep.eq.irepmx) goto 890
c   test
      write(68,*) 'itn = ',itn,' lamp = ',lamp
      write(68,*) ' amp'
      do izgak=1,lamp+1
      write(68,*) amp(izgak)
      enddo
c end test
c
c  update precursors , calculate fixed source and update shape with
c  new tort calculation.
      dtnx = dtn(itn)/idtj(itn)
c
        tx1 = zero
        do i1=1,itn
        tx1 = tx1 + dtn(i1)
        enddo
        tx1 = tx1/tltot
        if(itn.ge.ilpend) tx1 =1.
        if(tx1.gt.1.) tx1 = 1.
        tx0 = zero
        do i1=1,itn-1
        tx0 = tx0 + dtn(i1)
        enddo
        tx0 = tx0/tltot
        if(tx0.lt.0.) tx0 = 0.
        if(tx0.gt.1.) tx0 = 1.
      call delsor2(cig1,cig2,alambda,beta,index,index1,ndg
     $,igmw,xsfp,xsfs,shap1,shap2,amp,ldamp,idtj,dume1,dume2,dtn,dtnx
     $,amp0,itn,irestx,0,dtj,xsf0,xsf1,ihtw,mtmw,izmw,z1c,cp,imw,jmw
     $,kmw,ihpw,tx0,tx1,ampold,indexn,dume,izart,n6pk)
c
c      calculate total power
c
      ampn = amp(lamp+1)
      call qaddj(ansh2,cig1,vinv,dtn(itn),alambda,mmw,lmw
     $ ,imw,jmw,kmw,igmw,ndg,index1,qmom,sweight,snpl,chid,iqs,ampn)
c
      call wdistr(igmw,imw,jmw,kmw,lmw,mmw,ntdsif,qmom,cx,cy,cz,denr)
c
      dampndt = (amp(lamp+1) - amp(lamp))/ dtj
      write(n6pk,*) ' amplitude derivative at ',telaps,' = ',dampndt
     $,' by delta (P) / delta (t)'
      write(n6pk,*) ' amplitude derivative at ',telaps,' = ',dampndtp
     $,' by LSODE'
      dampndt = dampndtp
c
      if(irep.eq.irepmx) goto 880
c
c   xsecs are calculated as needed
c
c   find total linear pert. elapse time
       if(itn.eq.1) then
        tltot = zero
        do i=1,ilpend-1
         tltot = tltot + dtn(i)
        enddo
       endif
c
      tx = telaps/tltot
      if(itn.ge.ilpend) tx =1.
      if(tx.gt.1.) tx = 1.
c  modified xsecs at the end of this time interval are on nts after this call
      call rdgip5(xsf0,xsf1,igmw,ihtw,ihpw,mtmw,betat
     $ ,ampn,dampndt,iqs,ncrxw,vinv,dtn(itn),ndg,nts,rxsec,tx)
c
c  also create zone perturbed xsecs
c
```

```
c   set input and output units for flux guess
c   note that they must be duifferent???? not sure ....
      if(itn.eq.1) then
      j3 = 0
      call rwinpw(j3,igmw,ndg,chi)
      if(j3.ne.0) then
      write(n6pk,*) 'error readw:jobw ....'
      goto 9999
      endif
      endif
c   set convergence parameters for fixed source if not done so before
c      if(ifix.ne.1.and.itn.eq.1) then
c      j3 = 0
c      call read66(j3)
c       if(j3.ne.0) then
c       write(n6pk,*) 'error read66:jobw ....'
c       goto 9999
c       endif
c      endif
c
        if(itn.gt.irstrt) then
c
      evold = evnew
      if(imany.le.99) then
        write(filnam,'(a,i2.2)') 'to',imany
        fils = ' <tortinw >'
        call system(xnam//fils//filnam)
      else
        write(filnam2,'(a,i3.3)') 'to',imany
        fils = ' <tortinw >'
        call system(xnam//fils//filnam2)
      endif
       open(35,file='evfil',status='old')
       read(35,*) evdum
       close(35)
        evnew = evdum
c
c     read the new shape in
      call fluxr(shap1,z1c,iaf1,1,cp,cx,cy,cz,enr,denr
     $ ,index,index1,izmw,imw,jmw,kmw,igmw)
      open(35,file='anflux',status='old',form='unformatted')
       read(35) (ansh1(iw),iw=1,indexm)
      close(35)
c     write to interface files
       if(irstrt.gt.0.and.itn.eq.irstrt+1) then
      open(35,file='shapet',status='old',form='unformatted')
      do iix=1,itn
       read(35)
      enddo
      write(35) (shap1(i),i=1,index)
      close(35)
      open(35,file='ashapet',status='old',form='unformatted')
      do iix=1,itn
       read(35)
      enddo
      write(35) (ansh1(i),i=1,indexm)
      close(35)
       else
      open(35,file='shapet',status='old',form='unformatted'
     $ ,access='append')
      write(35) (shap1(i),i=1,index)
      close(35)
      open(35,file='ashapet',status='old',form='unformatted'
     $ ,access='append')
      write(35) (ansh1(i),i=1,indexm)
      close(35)
       endif
c
         else
c   restart, read from interface files
c
      open(35,file='shapet',status='old',form='unformatted')
      do iix=1,itn
       read(35)    ! skip earlier records
      enddo
      read(35) (shap1(i),i=1,index)
      close(35)
      open(35,file='ashapet',status='old',form='unformatted')
      do iix=1,itn
       read(35)    ! skip earlier records
      enddo
      read(35) (ansh1(i),i=1,indexm)
      close(35)
c
         endif
c
        if(ipkmod.eq.0) then
      call ampil(ansh1,anadj,vinv,index,index1,igmw,iqs,1
     $ ,vr,dth,dz,imw,jmw,kmw,amplit,shap,indexm,mmw,sweight,1,n6pk,itn)
       call ampilh(shap1,scadj,vinv,index,index1,igmw,iqs,1
     $ ,vr,dth,dz,imw,jmw,kmw,amplit,shap,1,n6pk,itn)
        elseif(ipkmod.eq.1) then
      call ampilh(shap1,scadj,vinv,index,index1,igmw,iqs,1
     $ ,vr,dth,dz,imw,jmw,kmw,amplit,shap,1,n6pk,itn)
c
c   also normalize angular shape for qaddj
c
      do i=1,indexm
        ansh1(i) = ansh1(i)/ratio(itn+1)
      enddo
```

```
        endif
c
880     continue
c
      if(iout.eq.3) call outshp(n6pk,equal,irep,index,shap1,shap2,igmw)
c
        tx1 = zero
        do il=1,itn
        tx1 = tx1 + dtn(il)
        enddo
        tx1 = tx1/tltot
        if(itn.ge.ilpend) tx1 =1.
        if(tx1.gt.1.) tx1 = 1.
        tx0 = zero
        do il=1,itn-1
        tx0 = tx0 + dtn(il)
        enddo
        tx0 = tx0/tltot
        if(tx0.lt.0.) tx0 = 0.
        if(tx0.gt.1.) tx0 = 1.
       call delsor2(cig1,cig2,alambda,beta,index,index1,ndg
     $,igmw,xsfp,xsfs,shap1,shap2,amp,ldamp,idtj,dume1,dume2,dtn,dtnx
     $,amp0,itn,irestx,1,dtj,xsf0,xsf1,ihtw,mtmw,izmw,z1c,cp,imw,jmw
     $,kmw,ihpw,tx0,tx1,ampold,indexn,dume,izart,n6pk)
c
c      calculate total power
c
       call powcal(power,shap1,index,index1,igmw,xsfp
     $ ,vr,dth,dz,imw,jmw,kmw,rnu,pd)
c
c
          if(irstrt.eq.0.or.itn.gt.irstrt) then
c
c   print flux shape to a file
        if(imany.le.99) then
        call woutfl(shap1,index,0,48,filnam//"s"
     $ ,cx,cy,cz,imw,jmw,kmw,igmw,vr,dth,dz)
        call woutfl(pd,index1,1,49,filnam//"p"
     $ ,cx,cy,cz,imw,jmw,kmw,igmw,vr,dth,dz)
        else
        call woutfl(shap1,index,0,48,filnam2//"s"
     $ ,cx,cy,cz,imw,jmw,kmw,igmw,vr,dth,dz)
        call woutfl(pd,index1,1,49,filnam//"p"
     $ ,cx,cy,cz,imw,jmw,kmw,igmw,vr,dth,dz)
        endif
c
          endif
c
       write(n6pk,'(120(a1))') (equal,itest=1,115)
        write(n6pk,*)'power              = ',power,' fissions/sec'
        write(n6pk,*)'relative power     = ',power/pow0
       write(n6pk,*)'amplitude          = ',amplit
       write(n6pk,*)'relative amplitude = ',amplit/amplit0
       write(n6pk,*)'shape              = ',shap,' (volume and energy
     $ integrated)'
       write(n6pk,*)'eigenvalue         = ',evnew
c
c
        if(irep.eq.irepmx) goto 890
        irep = irep+1
        ijamp = ijamp - idtj(itn)
       close(52)    ! need to close so that in next loop it will be overriden
        goto 710
c       endif
890    continue
c   this is needed to keep initial values of y vector for next itn loop
c   the last arg. is : 1= y to resrep , otherwise resrep to y
       if(ipkmod.eq.0) then
       call flxgen(anfl1,ansh1,indexm,amp(lamp+1))
       call scalit(anfl1,indexm,flux1,index,mmw,sweight)
       elseif(ipkmod.eq.1) then
       call flxgen(flux1,shap1,index,amp(lamp+1))
       endif
c
c  linear pert. still going on
        tx = zero
        do il=1,itn
        tx = tx + dtn(il)
        enddo
        tx = tx/tltot
        if(itn.ge.ilpend) tx =1.
        if(tx.gt.1.) tx = 1.
        tx0 = zero
        do il=1,itn-1
        tx0 = tx0 + dtn(il)
        enddo
        tx0 = tx0/tltot
        if(tx0.lt.0.) tx0 = 0.
        if(tx0.gt.1.) tx0 = 1.
        call pownc5(pow0,shap1,index,igmw,xsf0,50,itn,vr,dth,dz
     $ ,imw,jmw,kmw,rnu,dtj,dtn(itn),amp(2),idtj(itn),telaps,51,dume3
     $ ,idtk(itn),52,dtk,dume1,dume1(index+1),shap2,xsf1,index1,ihpw
     $ ,ihtw,mtmw,izmw,z1c,cp,tx,tx0)
c
       ampold = amp(lamp+1)
       amp0   = amp(lamp+1)
c
       imany = imany +1
       close(52,status='delete')
c      end dtn time loop
```

```
800   continue
9999  return
      end
::::::::::::::
jobw6.f
::::::::::::::
c234567890123456789012345678901234567890123456789012345678901234567890123456789012
      subroutine jobw6(iot,cig0,scadj,vr,dth,dz,z1c,xsts,xstp,xstd,xsfs
     $ ,xsfp,xsfd,xsss,xssp,xssd,xsf0,xsf1,cp,dum0,dum1,dumx
     $ ,rxsec,chi,chit,qmom,cx,cy,cz,enr,denr,anadj,snp1,pd,amp,flux1
     $ ,flux2,shap1,shap2,shap3,shap4,anfl1,anfl2,ansh1,ansh2,ansh3
     $ ,ansh4,cig1,cig2,dume1,dume2,dume3,dumep1,dumep2,dumep3
     $ ,dumep4,dume,rhot3,rhot4,rhot5,xsf2)
c
      include 'comsd1.F'
      include 'comsd2.F'
      include 'comsd4.F'
      include 'comsd5.F'
      include 'comsd6.F'
      include 'comsd7.F'
      include 'comsd8.F'
      include 'comsd0.F'
      include 'comsdc.F'
c
c234567890123456789012345678901234567890123456789012345678901234567890123456789012
      real*4 cig0(indexn),scadj(index),vr(imw),dth(jmw),dz(kmw)
     $ ,z1c(index1),xsts(index),xstp(index),xstd(index),xsfs(index)
     $ ,xsfp(index),xsfd(index),xsss(index*igmw),xssp(index*igmw)
     $ ,xssd(index*igmw),xsf0(izart),xsf1(izart),cp(izmw),dum0(izart)
     $ ,dum1(izart),dumx(igmw*igmw*mtmw)
     $ ,rxsec(izart),chi(igmw),chit(igmw),qmom(index*lmw),cx(imw+1)
     $ ,cy(jmw+1),cz(kmw+1),enr(igmw+2),denr(igmw),anadj(indexm)
     $ ,snp1(idimsnp1),pd(index1),amp(ldamp+1),flux1(index)
     $ ,flux2(index),shap1(index),shap2(index),shap3(index),shap4(index)
     $ ,anfl1(indexm),anfl2(indexm),ansh1(indexm),ansh2(indexm)
     $ ,ansh3(indexm),ansh4(indexm),cig1(indexn),cig2(indexn)
     $ ,dume1(2*index),dume2(2*index*ndg),dume3(mdamp)
     $ ,dume(irestx),xsf2(izart)
c
c   for pkvar
      real*4 dumep1(index1),dumep2(index),dumep3(index1)
     $ ,dumep4(index),rhot3(index),rhot4(index),rhot5(index)
c
      real*4 telaps,rho,rlamda,extsrc,fterm,power,bett,tltot,tx,hhi1
     $ ,rho2rp,rho1rp,rhonrp,rlam2rp,rlam1rp,rlamnrp,bet2rp,bet1rp
     $ ,tx0,tx1,ampn,dampndt
      real*4 zero,one
      zero = 0.e0
      iot = n6pk
      one  = 1.e0
      xnam = 'tortnew '
c
c    iquad =  quadratic interpolation flag for flux
c    jquad =  quadratic interpolation flag for rho
c
c   note that z1c is for initial zone map
c    idtnm = # of tn calc.
c
       equal = '='
c   open output unit
      open(n6pk,file='pkout',status='unknown')
      call xsetun(n6pk)
c
c   rhonew is newest
c   rho1   is previous
c   rho2   is oldest
c
c    betaeft is sum(betaef(i))
c
c    initialize
c
      ijamp = 0
      neq = ndg+1
      itol = 1
      ilsrto = 0
      if(srtol.lt.0..or.satol.lt.0.) then
       ilsrto = 1
      endif
      rtol = 1.0d0*abs(srtol)
      atol = 1.0d0*abs(satol)
      itask = 1
      istate = 1
      iopt = 0
      lrw = 500
      liw = 100
      mf = 22
      t    = zero
      tout= zero
      telaps = zero
c
c     reset irepmx so that 1 extra iteration is performed by default
      irepmx = irepmx + 1
      evw = evnew
      outflg = .false.
      if(iout.gt.0) outflg = .true.
      rho1    = zero
      rho2    = zero
      rhonew = zero
      rho     = zero
      rlam1    = zero
```

```
      rlam2   = zero
      rlamnew = zero
      rlamda  = zero
      q1      = zero
      q2      = zero
      qnew    = zero
      extsrc = zero
      fterm  = zero
      power  = zero
      do i=1,ndg
       betae1(i) = zero
       betae2(i) = zero
       betnew(i) = zero
       betaef(i) = zero
       csubi(i) = zero
      enddo
      bet1  = zero
      bet2  = zero
      betn  = zero
      bett  = zero
c
      call rkld0(betaef0,alambda,rlamda0,n6pk,ndg,amplit0,y)
c   this is needed to keep initial values of y vector
c   the last arg. is : 1= y to resrep , otherwise resrep to y
      call rkldrp(y,resrep,neq,1)
c
      rhonew = rho0
      rlamnew = rlamda0
      betn = zero
      do ii=1,ndg
      betnew(ii) = betaef0(ii)
      betn = betn + betnew(ii)
      enddo
c
c   also normalize angular shape for qaddj
c
      do i=1,indexm
        anshl(i) = anshl(i)/ratio(1)
      enddo
c
c   find total linear pert. elapse time
        tltot = zero
        do i=1,ilpend-1
         tltot = tltot + dtn(i)
        enddo
c
      iquad = .false.
c     start dtn time loop
      do 800 itn = 1, idtnm
c   set lsode parameters
      if(ilsrto.eq.0) then
       rtol = dtn(itn)/idtk(itn)*srtol
       atol = dtn(itn)/idtk(itn)*satol
      endif
      if(ilsrto.eq.0.or.itn.eq.1) then
       write(n6pk,*) '**********************************'
       write(n6pk,*) ' rtol = ',rtol,' at time interval ',itn
       write(n6pk,*) ' atol = ',atol,' at time interval ',itn
       write(n6pk,*) '**********************************'
      endif
c
      irep = 0      ! this is to repeat pk calc. and update
      hi1 = dtn(itn)
      hi2 = dtn(itn)
c
c   shift fluxes
      call shftfl(shap1,shap2,shap3,anshl,ansh2,ansh3,index,indexm)
      call shftcg(cig1,cig2,indexn)
c   note : cig1 is latest (cig^(n+1)) & cig2 is prev. (cig^(n))
c
      if(itn.eq.1) then
        amp(1) = amplit0
        rlamda = rlamda0
        fterm  = fterm0
       hhi1=zero
c   save last amplitude as first for next time step
       ampold = amplit0
       amp0   = amplit0
      else
        rlamda = rlamold
        fterm  = ftermold
       hhi1 = dtn(itn-1)
      endif
c    generate precursor densities using last calculated delayed
c    neutron source and load onto y array
c
c   this next call causes discontinuities in y b/c csubi does not
c   match the value calculated by lsode
c   use with caution
      call rkld(y,rlamda,fterm,n6pk,ndg,cig2,scadj,imw,jmw,kmw,igmw
     $ ,vr,dth,dz,indexn,index,itn,index1,dume1,chid,ampold,amp0,amplit)
c
c   extrapolate to calculate first flux shape and normalize
c
c   extrapolate angular and scalar shape
c
        if(ipkmod.eq.0) then
      call extrap(anshl,ansh2,ansh3,hhi1,hi2,indexm,iextrp,n6pk)
      call extrap(shap1,shap2,shap3,hhi1,hi2,index,iextrp,n6pk)
      call ampil(anshl,anadj,vinv,index,index1,igmw,iqs,0
```

```
     $ ,vr,dth,dz,imw,jmw,kmw,amplit,shap,indexm,mmw,sweight,1,n6pk,itn)
      call ampilh(shap1,scadj,vinv,index,index1,igmw,iqs,0
     $ ,vr,dth,dz,imw,jmw,kmw,amplit,shap,1,n6pk,itn)
        elseif(ipkmod.eq.1) then
      call extrap(shap1,shap2,shap3,hhi1,hi2,index,iextrp,n6pk)
      call ampilh(shap1,scadj,vinv,index,index1,igmw,iqs,0
     $ ,vr,dth,dz,imw,jmw,kmw,amplit,shap,1,n6pk,itn)
        endif
c
c       calculate total power
c
      call powcal(power,shap1,index,index1,igmw,xsfp
     $ ,vr,dth,dz,imw,jmw,kmw,rnu,pd)
c
      call powcal(power,shap2,index,index1,igmw,xsfp
     $ ,vr,dth,dz,imw,jmw,kmw,rnu,pd)
c
c     idtj = # of tj calc.
710   jquad = .false.
c     clear rwork for lsode
      do itt2=1,500
         rwork(itt2) = 0.0d0
      enddo
      do itt2=1,100
         iwork(itt2) = 0
      enddo
      if(irep.eq.0) then
c    svae old react.,gen. time and effective betas for second pass
       call rkldrp(y,resrep,ndg+1,1)
         rho2rp = rho2
         rho1rp = rho1
         rhonrp = rhonew
         rlam2rp = rlam2
         rlam1rp = rlam1
         rlamnrp = rlamnew
         bet2rp = zero
         bet1rp = zero
         betnrp = betn
         do i=1,ndg
         bete2rp(i) = betae2(i)
         bete1rp(i) = betae1(i)
         betenrp(i) = betnew(i)
         bet2rp     = bet2rp + bete2rp(i)
         bet1rp     = bet1rp + bete1rp(i)
         enddo
       elseif(irep.gt.0) then
c    put old data back
        call rkldrp(y,resrep,neq,0)
         rho2 = rho2rp
         rho1 = rho1rp
         rhonew = rhonrp
         rlam2 = rlam2rp
         rlam1 = rlam1rp
         rlamnew = rlamnrp
         bet2 = zero
         bet1 = zero
         betn = betnrp
         do i=1,ndg
         betae2(i) = bete2rp(i)
         betae1(i) = bete1rp(i)
         betnew(i) = betenrp(i)
         bet2     = bet2 + betae2(i)
         bet1     = bet1 + betae1(i)
         enddo
       endif
c      start dtj time loop
       lamp = 0
       dtj = zero
       dtj = dtn(itn)/idtj(itn)
c   open and close fort.45 and fort.46 to force buffer dump
       if(itn.eq.1.and.irep.eq.0) then
         open(45,status='unknown')
         open(46,status='unknown')
       else
         open(45,status='old',access='append')
         open(46,status='old',access='append')
       endif
       do 700 itj=1, idtj(itn)
c   transfer the times for irep = 0 case
       if(irep.eq.0.and.itj.eq.1.and.irepmx.gt.0) then
         ttrep1 = telaps
         ttrep2 = tout
         ttrep3 = t
       elseif(irep.gt.0.and.itj.eq.1.and.irepmx.gt.0) then
         telaps = ttrep1
         tout   = ttrep2
         t      = ttrep3
       endif
       hj1    = dtj
       hj2    = dtj
       ijamp = ijamp + 1   ! not used
       if(itj.gt.2)    jquad = .true.
       if(itj.eq.1) istate = 1
c
       tnewj = itj*dtj
c
c   linear interp.
       if(ipkmod.eq.0) then
       call linter(ansh2,ansh1,ansh4,hi2,tnewj,indexm)
       call linter(shap2,shap1,shap4,hi2,tnewj,index)
```

```
      elseif(ipkmod.eq.1) then
      call linter(shap2,shap1,shap4,hi2,tnewj,index)
      endif
c
      idim  = ndg
c  calculate initial reac. at time 0. with pert. xsecs
c
      if(itn.eq.1.and.itj.eq.1) then
c
      call xsecr(xsf1,ihpw,mtmw,igmw,ntsigp)  ! reread for irep > 0
      tx = one
      call dxsec6(xsts,xstp,xstd,xsfs,xsfp,xsfd,xsss,xssp,xssd
     $,xsf1,xsf2,index,z1c,cp,igmw,index1,ihpw,ihtw,mtmw,izmw
     $,dum0,dum1,dumx,tx,mzm,mz,0)
c
      call pkvar(rho,fterm,betaef,betaeft,rlamda,extsrc
     $ ,idim,shap2,ansh2,rhot3,rhot4,rhot5,anadj,xstd,scadj,xssd
     $ ,dumep1,xsfd,chit,dumep2,vr,dth,dz,dumep3,xsfp,dumep4
     $ ,index,index1,indexm,imw,jmw,kmw,igmw,mmw,ipkmod)
c
      call mvpk(rho2,rho1,rhonew,rho,bett,betae2,betae1,betnew
     $ ,betaef,bet2,bet1,betn,rlam2,rlam1,rlamnew,rlamda,q2,q1,qnew
     $ ,extsrc,n6pk,telaps,irep,irepmx,fterm,ndg,iout)
      endif
c
      if(itn.eq.1.and.itj.eq.1.and.irep.eq.irepmx) then
        open(35,file='pkparm',status='unknown')
        write(35,*) " elapsed time     rho       betaeft     rlamda"
        write(35,'(1p,2x,4(e10.3,2x))') telaps,rho,betaeft,rlamda
        close(35)
      endif
      if((itn.gt.1.and.itj.eq.1.and.irep.eq.irepmx).or.
     $ (iout.gt.0.and.irep.eq.irepmx)) then
        open(35,file='pkparm',status='old',access='append')
        write(35,'(1p,2x,4(e10.3,2x))') telaps,rho,betaeft,rlamda
        close(35)
      endif
c
      if(itj.eq.idtj(itn).and.irep.eq.irepmx) then
c  this is for recalculation of csubi in rkld, save the last values
      rlamold = rlamda
      ftermold = fterm
      endif
c  calc. pert xsecs for this itj
          if(itn.eq.1.and.itj.eq.1) then
c   first read in new xsecs if not done so before
      call xsecr(xsf2,ihpw,mtmw,igmw,ntsigp)  !move initial step to xsf2
      call xsecr(xsf1,ihpw,mtmw,igmw,ntsigp2) !load new xsecs to xsf1
          endif
c
      tx = (telaps+dtj)/tltot
      if(itn.ge.ilpend) tx =1.
      if(tx.gt.1.) tx = 1.
      call dxsec6(xsts,xstp,xstd,xsfs,xsfp,xsfd,xsss,xssp,xssd
     $,xsf1,xsf2,index,z1c,cp,igmw,index1,ihpw,ihtw,mtmw,izmw
     $,dum0,dum1,dumx,tx,mzm,mz,1)
      call pkvar(rho,fterm,betaef,betaeft,rlamda,extsrc
     $ ,idim,shap4,ansh4,rhot3,rhot4,rhot5,anadj,xstd,scadj,xssd
     $ ,dumep1,xsfd,chit,dumep2,vr,dth,dz,dumep3,xsfp,dumep4
     $ ,index,index1,indexm,imw,jmw,kmw,igmw,mmw,ipkmod)
c
      call mvpk(rho2,rho1,rhonew,rho,bett,betae2,betae1,betnew
     $ ,betaef,bet2,bet1,betn,rlam2,rlam1,rlamnew,rlamda,q2,q1,qnew
     $ ,extsrc,n6pk,telaps+dtj,irep,irepmx,fterm,ndg,iout)
c
c    idtk = # of tk calc.
c
      dtk = zero
      dtk  = dtn(itn)/idtk(itn)
      time = zero
      do 600 itk=1,idtk(itn)/idtj(itn)
c      telaps = telaps + dtk
      tout = t + dtk
c
      call pkcal(rho,betaef,rlamda,extsrc,betaeft,ndg,time)
      rhop = rho
      do i=1,ndg
      betap(i) = betaef(i)
      enddo
      rlamp = rlamda
      betp = betaeft
      extp = extsrc
c
      call lsode(f,neq,y,t,tout,itol,rtol,atol,itask,istate,
     $    iopt,rwork,lrw,iwork,liw,jex,mf)
c
      if (istate .lt. 0) then
      write(n6pk,90)istate
90    format(///22h error halt.. istate =,i3)
c      stop ' solvem : lsode '
      write(6,*) ' stop .... solvem : lsode '
       goto 9999
      endif
c
       call rkunld(y,csubi,ndg,amplit0,n6pk,itn,itj,tout,irep,iout)
      if(irep.eq.irepmx) then
c       call rkunld(y,csubi,ndg,amplit0,n6pk,itn,itj,tout,irep,iout)
c       write(41,*) telaps+(itk-1)*dtk,rhop,time
        sdump = y(1)
       write(52,*) sdump
```

```
      endif
c
      time = time + dtk
c
600   continue
c
      lamp = lamp + 1
      amp(lamp+1) = y(1)
c
      if(itn.eq.1) then
        tltot6 = zero
        tjtot6 = dtns(1)
        tjtot6 = tjtot6*itj/idtj(itn)
        tltot6 = tltot6 + tjtot6
      elseif(itn.gt.1) then
        tltot6 = dtns(itn-1)
        tjtot6 = dtns(itn) - dtns(itn-1)
        tjtot6 = tjtot6*itj/idtj(itn)
        tltot6 = tltot6 + tjtot6
      endif
      telaps = tltot6
c
      write(45,*) telaps,amp(lamp+1)/amplit0,irep
c     end dtj time loop
700   continue
      close(45)
      close(46)
c  if last pass over this dtn, save the amplitute
      if(irep.eq.irepmx) then
      rewind(52)
      i1 = 1
740   read(52,*,err=750,end=750) dume3(i1)
      i1 = i1 + 1
      goto 740
750   i1 = i1 - 1
      if(itn.eq.1) then
       open(35,file='amplitute',status='unknown',form='unformatted')
      else
       open(35,file='amplitute',status='old',form='unformatted'
     $ ,access='append')
      endif
      write(35) (dume3(i1i),i1i=1,i1)
      close(35)
      endif
c   if this is the last dtn time interval then
c    there is no need to update shape and precursors, just exit.
      if(itn.eq.idtnm.and.irep.eq.irepmx) goto 890
c   test
      write(68,*) 'itn = ',itn,' lamp = ',lamp
      write(68,*) ' amp'
      do izgak=1,lamp+1
      write(68,*) amp(izgak)
      enddo
c end test
c
c  update precursors , calculate fixed source and update shape with
c  new tort calculation.
      dtnx = dtn(itn)/idtj(itn)
c
        tx1 = zero
        do i1=1,itn
        tx1 = tx1 + dtn(i1)
        enddo
        tx1 = tx1/tltot
        if(itn.ge.ilpend) tx1 =1.
        if(tx1.gt.1.) tx1 = 1.
        tx0 = zero
        do i1=1,itn-1
        tx0 = tx0 + dtn(i1)
        enddo
        tx0 = tx0/tltot
        if(tx0.lt.0.) tx0 = zero
        if(tx0.gt.1.) tx0 = 1.
c
      call delsor6(cig1,cig2,alambda,beta,index,index1,ndg
     $,igmw,xsfp,xsfs,shap1,shap2,amp,ldamp,idtj,dume1,dume2,dtn,dtnx
     $,amp0,itn,irestx,0,dtj,xsf0,xsf1,ihtw,mtmw,izmw,zlc,cp,imw,jmw
     $,kmw,ihpw,tx0,tx1,ampold,tltot,indexn,dume,izart,n6pk
     $,xsf2,mz,mzm)
c
      ampn = amp(lamp+1)
      call qaddj(ansh2,cig1,vinv,dtn(itn),alambda,mmw,lmw
     $ ,imw,jmw,kmw,igmw,ndg,index1,qmom,sweight,snp1,chid,iqs,ampn)
c
      call wdistr(igmw,imw,jmw,kmw,lmw,mmw,ntdsif,qmom,cx,cy,cz,denr)
c
      dampndt = (amp(lamp+1) - amp(lamp))/ dtj
      write(n6pk,*) ' amplitude derivative at ',telaps,' = ',dampndt
     $,' by delta (P) / delta (t)'
      write(n6pk,*) ' amplitude derivative at ',telaps,' = ',dampndtp
     $,' by LSODE'
      dampndt = dampndtp
c
      if(irep.eq.irepmx) goto 880
c
c   xsecs are calculated as needed
      tx = telaps/tltot
      if(itn.ge.ilpend) tx =1.
      if(tx.gt.1.) tx = 1.
c  xsf2 has old step perturbed xsecs at time zero
```

```
      call rdgip6(xsf0,xsf2,xsf1,igmw,ihtw,ihpw,mtmw,betat,mzm,mz
     $ ,ampn,dampndt,iqs,ncrxw,vinv,dtn(itn),ndg,nts,rxsec,tx)
c
c   set input and output units for flux guess
c   note that they must be duifferent???? not sure ....
      if(itn.eq.1) then
      j3 = 0
      call rwinpw(j3,igmw,ndg,chi)
      if(j3.ne.0) then
      write(n6pk,*) 'error readw:jobw ....'
      goto 9999
      endif
      endif
c   set convergence parameters for fixed source if not done so before
c      if(ifix.ne.1.and.itn.eq.1) then
c      j3 = 0
c      call read66(j3)
c       if(j3.ne.0) then
c       write(n6pk,*) 'error read66:jobw ....'
c       goto 9999
c       endif
c      endif
c
         if(itn.gt.irstrt) then
c
      evold = evnew
      if(imany.le.99) then
        write(filnam,'(a,i2.2)') 'to',imany
        fils = ' <tortinw >'
        call system(xnam//fils//filnam)
      else
        write(filnam2,'(a,i3.3)') 'to',imany
        fils = ' <tortinw >'
        call system(xnam//fils//filnam2)
      endif
       open(35,file='evfil',status='old')
       read(35,*) evdum
       close(35)
        evnew = evdum
c
c     read the new shape in
      call fluxr(shap1,z1c,iafl,1,cp,cx,cy,cz,enr,denr
     $ ,index,index1,izmw,imw,jmw,kmw,igmw)
      open(35,file='anflux',status='old',form='unformatted')
       read(35) (ansh1(iw),iw=1,indexm)
      close(35)
c     write to interface files
       if(irstrt.gt.0.and.itn.eq.irstrt+1) then
      open(35,file='shapet',status='old',form='unformatted')
      do iix=1,itn
       read(35)
      enddo
      write(35) (shap1(i),i=1,index)
      close(35)
      open(35,file='ashapet',status='old',form='unformatted')
      do iix=1,itn
       read(35)
      enddo
      write(35) (ansh1(i),i=1,indexm)
      close(35)
       else
      open(35,file='shapet',status='old',form='unformatted'
     $ ,access='append')
      write(35) (shap1(i),i=1,index)
      close(35)
      open(35,file='ashapet',status='old',form='unformatted'
     $ ,access='append')
      write(35) (ansh1(i),i=1,indexm)
      close(35)
       endif
c
         else
c   restart, read from interface files
c
      open(35,file='shapet',status='old',form='unformatted')
      do iix=1,itn
       read(35)   ! skip earlier records
      enddo
      read(35) (shap1(i),i=1,index)
      close(35)
      open(35,file='ashapet',status='old',form='unformatted')
      do iix=1,itn
       read(35)   ! skip earlier records
      enddo
      read(35) (ansh1(i),i=1,indexm)
      close(35)
c
         endif
c
        if(ipkmod.eq.0) then
      call ampil(ansh1,anadj,vinv,index,index1,igmw,iqs,1
     $ ,vr,dth,dz,imw,jmw,kmw,amplit,shap,indexm,mmw,sweight,1,n6pk,itn)
       call ampilh(shap1,scadj,vinv,index,index1,igmw,iqs,1
     $ ,vr,dth,dz,imw,jmw,kmw,amplit,shap,1,n6pk,itn)
         elseif(ipkmod.eq.1) then
      call ampilh(shap1,scadj,vinv,index,index1,igmw,iqs,1
     $ ,vr,dth,dz,imw,jmw,kmw,amplit,shap,1,n6pk,itn)
c
c   also normalize angular shape for qaddj
c
```

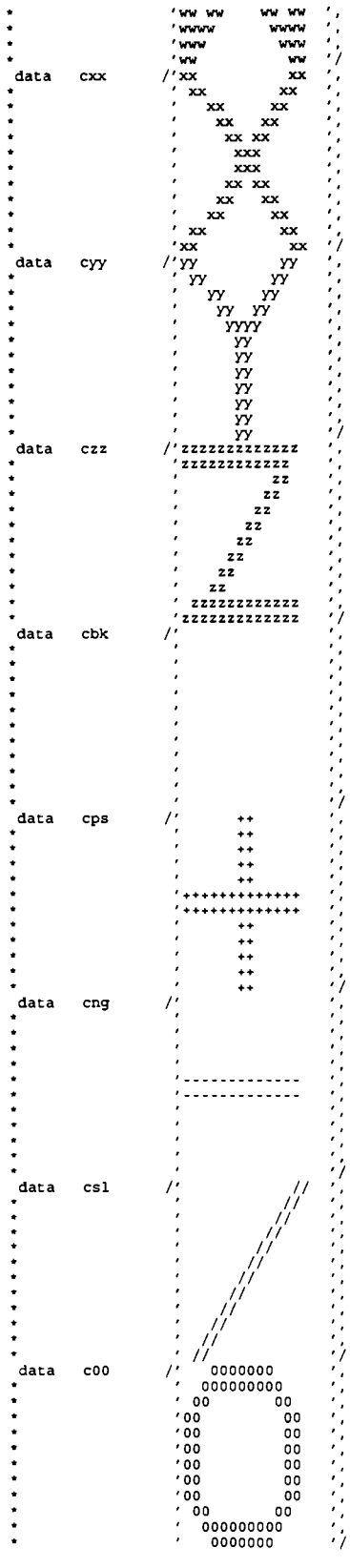
```
      do i=1,indexm
        anshl(i) = anshl(i)/ratio(itn+1)
      enddo
        endif
c
880    continue
c
      if(iout.eq.3) call outshp(n6pk,equal,irep,index,shap1,shap2,igmw)
c
        tx1 = zero
        do i1=1,itn
        tx1 = tx1 + dtn(i1)
        enddo
        tx1 = tx1/tltot
        if(itn.ge.ilpend) tx1 =1.
        if(tx1.gt.1.) tx1 = 1.
        tx0 = zero
        do i1=1,itn-1
        tx0 = tx0 + dtn(i1)
        enddo
        tx0 = tx0/tltot
        if(tx0.lt.0.) tx0 = zero
        if(tx0.gt.1.) tx0 = 1.
c
      call delsor6(cig1,cig2,alambda,beta,index,index1,ndg
     $,igmw,xsfp,xsfs,shap1,shap2,amp,ldamp,idtj,dume1,dume2,dtn,dtnx
     $,amp0,itn,irestx,1,dtj,xsf0,xsf1,ihtw,mtmw,izmw,z1c,cp,imw,jmw
     $,kmw,ihpw,tx0,tx1,ampold,tltot,indexn,dume,izart,n6pk
     $,xsf2,mz,mzm)
c
c      calculate total power
c
      call powcal(power,shap1,index,index1,igmw,xsfp
     $ ,vr,dth,dz,imw,jmw,kmw,rnu,pd)
c
      call powcal(power,shap2,index,index1,igmw,xsfp
     $ ,vr,dth,dz,imw,jmw,kmw,rnu,pd)
c
c
         if(irstrt.eq.0.or.itn.gt.irstrt) then
c
c   print flux shape to a file
      if(imany.le.99) then
      call woutfl(shap1,index,0,48,filnam//"s"
     $ ,cx,cy,cz,imw,jmw,kmw,igmw,vr,dth,dz)
      else
      call woutfl(shap1,index,0,48,filnam2//"s"
     $ ,cx,cy,cz,imw,jmw,kmw,igmw,vr,dth,dz)
      endif
c
         endif
c
      write(n6pk,'(120(a1))') (equal,itest=1,115)
       write(n6pk,*)'power                = ',power,' fissions/sec'
       write(n6pk,*)'relative power       = ',power/pow0
      write(n6pk,*)'amplitude            = ',amplit
      write(n6pk,*)'relative amplitude = ',amplit/amplit0
      write(n6pk,*)'shape                = ',shap,' (volume and energy
     $ integrated)'
      write(n6pk,*)'eigenvalue           = ',evnew
c
c
       if(irep.eq.irepmx) goto 890
       irep = irep+1
       ijamp = ijamp - idtj(itn)
      close(52)    ! need to close so that in next loop it will be overriden
       goto 710
c      endif
890   continue
c   this is needed to keep initial values of y vector for next itn loop
c   the last arg. is : 1= y to resrep , otherwise resrep to y
      if(ipkmod.eq.0) then
      call flxgen(anfl1,anshl,indexm,amp(lamp+1))
      call scalit(anfl1,indexm,flux1,index,mmw,sweight)
      elseif(ipkmod.eq.1) then
      call flxgen(flux1,shap1,index,amp(lamp+1))
      endif
c
c      calculate total power
c
      call powcal(power,shap1,index,index1,igmw,xsfp
     $ ,vr,dth,dz,imw,jmw,kmw,rnu,pd)
c
      call powcal(power,shap2,index,index1,igmw,xsfp
     $ ,vr,dth,dz,imw,jmw,kmw,rnu,pd)
c
         call pownc6(pow0,shap1,index,igmw,xsfp,50,itn,vr,dth,dz,imw
     $ ,jmw,kmw,rnu,dtj,dtn(itn),amp(2),idtj(itn),51,dume3,mdamp
     $ ,idtk(itn),52,dtk,dume1,dume1(index+1),xsf1,xsf2,ihpw,ihtw,mtmw
     $ ,izmw,z1c,cp,ilpend,tltot,telaps,shap2,dume)
c
      ampold = amp(lamp+1)
      amp0   = amp(lamp+1)
c
      imany = imany +1
      close(52,status='delete')
c     end dtn time loop
800   continue
9999  return
      end
```

```
::::::::::::::
lamcal.f
::::::::::::::
      subroutine lamcal(rlamda,fterm,ansh,anadj,dume2,vr,dth,dz
     $ ,index,index1,indexm,im,jm,km,igm,mm)
c     rlamda is lamda(t)
c     fterm is F(t)
c
      include 'comsd1.F'
      common sweight(320)
      real*4 sweight
c
      real*4 rlamda,fterm,ansh(indexm),anadj(indexm),dume2(index)
     $ ,vr(im),dth(jm),dz(km),sum,res
c
      call clearx( dume2,1,index)
c
c     multiply forward and adjoint fluxes and inverse speeds
c     and load onto dume2
      do 20 ig=1,igm
      do 30 i=1,index1
       sum = 0.
       do m=1,mm
       i1    = i +index1*((ig-1)+igm*(m-1))
        sum = sum + anadj(i1)*ansh(i1)*sweight(m)
       enddo
       i3    = i +index1*(ig-1)
       dume2(i3) = sum*vinv(ig)
30    continue
20    continue
c
c     perform integration over volume and energy
c
      call vgintg(dume2,vr,dth,dz,index,igm,im,jm,km,res)
      rlamda = res / fterm
c
      return
      end
::::::::::::::
lamcalh.f
::::::::::::::
      subroutine lamcalh(rlamda,fterm,shape,scadj,dume2,vr,dth,dz
     $ ,index,index1,im,jm,km,igm)
c     rlamda is lamda(t)
c     fterm is F(t)
c
      include 'comsd1.F'
c
      real*4 rlamda,fterm,shape(index),scadj(index),dume2(index)
     $ ,vr(im),dth(jm),dz(km),res
c
      call clearx( dume2,1,index)
c
c     multiply forward and adjoint fluxes and inverse speeds
c     and load onto dume2
      do 20 ig=1,igm
      do 30 i=1,index1
       i1 = i + index1*(ig-1)
       dume2(i1) = vinv(ig)*scadj(i1)*shape(i1)
30    continue
20    continue
c
c     perform integration over volume and energy
c
      call vgintg(dume2,vr,dth,dz,index,igm,im,jm,km,res)
      rlamda = res / fterm
c
      return
      end
::::::::::::::
linter.f
::::::::::::::
      subroutine linter(f1,f2,f3,h2,x,index)
      real*4 f1(index),f2(index),f3(index),h2,x
c
c     linear interpolator
c
c      note that x2=0, |x2|-|x1|=h1, x3-x2=h2
c       x1,y1     x2,y2
c         |         |
c        -x1        0
c       <---------->
c            h2
c
c      h2 = x2 - x1
c
c      note that interpolation is done for x2<x<x3
c
      do 220 i=1,index
      f3(i) = f1(i) + (x)/h2 * (f2(i)-f1(i))
220   continue
c
      return
      end
::::::::::::::
lttr.f
::::::::::::::
      block data lttr
c#####################################################################
c
```

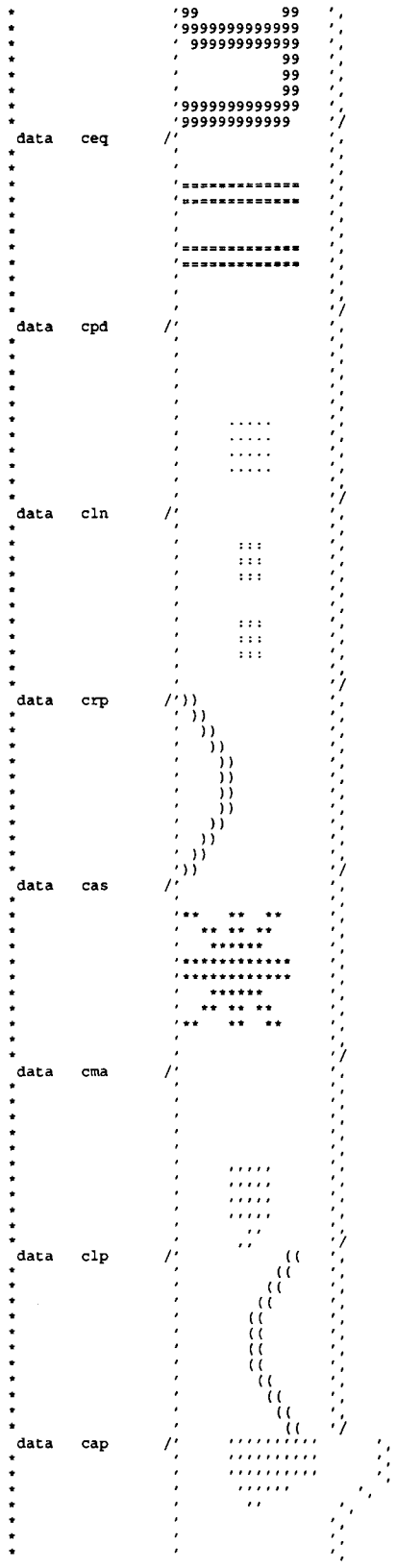
```
c               audit trail information
c
c     date the module was last permanently updated:     91/06/20
c     time the module was last permanently updated:     11:30:20
c     programmer name:                                  l.m.petrie
c     module name:                                      s7lttr
c     current archiving level number:                   00007
c     current number of permanent updates:              00007
c     date of last access by librarian:                 93/02/24
c     dataset name:  x4s.scale4.master
c
c#####################################################################
      implicit character*16 (c)
      common/letter/ caa,cbb,ccc,cdd,cee,cff,cgg,chh,cii,cjj,
     *               ckk,cll,cmm,cnn,coo,cpp,cqq,crr,css,ctt,
     *               cuu,cvv,cww,cxx,cyy,czz,cbk,cps,cng,csl,
     *               c00,c11,c22,c33,c44,c55,c66,c77,c88,c99,
     *               ceq,cpd,cln,crp,cas,cma,clp,cap,cam,cdr,cer
      dimension caa(12),cbb(12),ccc(12),cdd(12),cee(12),
     *          cff(12),cgg(12),chh(12),cii(12),cjj(12),
     *          ckk(12),cll(12),cmm(12),cnn(12),coo(12),
     *          cpp(12),cqq(12),crr(12),css(12),ctt(12),
     *          cuu(12),cvv(12),cww(12),cxx(12),cyy(12),
     *          czz(12),cbk(12),cps(12),cng(12),csl(12),
     *          c00(12),c11(12),c22(12),c33(12),c44(12),
     *          c55(12),c66(12),c77(12),c88(12),c99(12),
     *          ceq(12),cpd(12),cln(12),crp(12),cas(12),
     *          cma(12),clp(12),cap(12),cam(12),cdr(12),cer(12)
      data   caa      /'   aaaaaaaaa     ',
     *                 ' aaaaaaaaaaa    ',
     *                 'aa         aa   ',
     *                 'aa         aa   ',
     *                 'aa         aa   ',
     *                 'aaaaaaaaaaaaa   ',
     *                 'aaaaaaaaaaaaa   ',
     *                 'aa         aa   ',
     *                 'aa         aa   ',
     *                 'aa         aa   ',
     *                 'aa         aa   ',
     *                 'aa         aa   '/
      data   cbb      /'bbbbbbbbbbb    ',
     *                 'bbbbbbbbbbbbb   ',
     *                 'bb         bb   ',
     *                 'bb         bb   ',
     *                 'bb         bb   ',
     *                 'bbbbbbbbbbbb    ',
     *                 'bbbbbbbbbbbb    ',
     *                 'bb         bb   ',
     *                 'bb         bb   ',
     *                 'bb         bb   ',
     *                 'bbbbbbbbbbbbb   ',
     *                 'bbbbbbbbbbb     '/
      data   ccc      /' ccccccccccc    ',
     *                 'ccccccccccccc   ',
     *                 'cc         cc   ',
     *                 'cc              ',
     *                 'cc              ',
     *                 'cc              ',
     *                 'cc              ',
     *                 'cc              ',
     *                 'cc              ',
     *                 'cc         cc   ',
     *                 'ccccccccccccc   ',
     *                 ' ccccccccccc    '/
      data   cdd      /'ddddddddddd    ',
     *                 'ddddddddddddd   ',
     *                 'dd         dd   ',
     *                 'dd         dd   ',
     *                 'dd         dd   ',
     *                 'dd         dd   ',
     *                 'dd         dd   ',
     *                 'dd         dd   ',
     *                 'dd         dd   ',
     *                 'ddddddddddddd   ',
     *                 'ddddddddddd     '/
      data   cee      /'eeeeeeeeeeee   ',
     *                 'eeeeeeeeeeeee   ',
     *                 'ee              ',
     *                 'ee              ',
     *                 'ee              ',
     *                 'eeeeeeeee       ',
     *                 'eeeeeeeee       ',
     *                 'ee              ',
     *                 'ee              ',
     *                 'ee              ',
     *                 'eeeeeeeeeeeee   ',
     *                 'eeeeeeeeeeeee   '/
      data   cff      /'fffffffffffff  ',
     *                 'fffffffffffff  ',
     *                 'ff              ',
     *                 'ff              ',
     *                 'ff              ',
     *                 'fffffffff       ',
     *                 'fffffffff       ',
     *                 'ff              ',
     *                 'ff              ',
     *                 'ff              ',
     *                 'ff              ',
     *                 'ff              '/
```

```
 data   cgg      /' gggggggggg    ',
*                 'gggggggggggg   ',
*                 'gg          gg ',
*                 'gg             ',
*                 'gg             ',
*                 'gg    ggggggg  ',
*                 'gg    ggggggg  ',
*                 'gg          gg ',
*                 'gg          gg ',
*                 'gg          gg ',
*                 'gggggggggggg   ',
*                 ' gggggggggg    '/
 data   chh      /'hh          hh  ',
*                 'hh          hh  ',
*                 'hh          hh  ',
*                 'hh          hh  ',
*                 'hh          hh  ',
*                 'hhhhhhhhhhhh    ',
*                 'hhhhhhhhhhhh    ',
*                 'hh          hh  ',
*                 'hh          hh  ',
*                 'hh          hh  ',
*                 'hh          hh  ',
*                 'hh          hh  '/
 data   cii      /'iiiiiiiiiii    ',
*                 'iiiiiiiiiii    ',
*                 '     ii        ',
*                 '     ii        ',
*                 '     ii        ',
*                 '     ii        ',
*                 '     ii        ',
*                 '     ii        ',
*                 '     ii        ',
*                 'iiiiiiiiiii    ',
*                 'iiiiiiiiiii    '/
 data   cjj      /'            jj  ',
*                 '            jj  ',
*                 '            jj  ',
*                 '            jj  ',
*                 '            jj  ',
*                 '            jj  ',
*                 '            jj  ',
*                 '            jj  ',
*                 '            jj  ',
*                 'jj          jj  ',
*                 'jjjjjjjjjjjj    ',
*                 ' jjjjjjjjjj     '/
 data   ckk      /'kk          kk  ',
*                 'kk         kk   ',
*                 'kk        kk    ',
*                 'kk       kk     ',
*                 'kk      kk      ',
*                 'kkkkkkkk        ',
*                 'kkkkkkkk        ',
*                 'kk       kk     ',
*                 'kk        kk    ',
*                 'kk         kk   ',
*                 'kk          kk  ',
*                 'kk           kk '/
 data   cll      /'ll             ',
*                 'll             ',
*                 'll             ',
*                 'll             ',
*                 'll             ',
*                 'll             ',
*                 'll             ',
*                 'll             ',
*                 'll             ',
*                 'll             ',
*                 'llllllllllll   ',
*                 'llllllllllll   '/
 data   cmm      /'mm           mm  ',
*                 'mmmm         mmmm ',
*                 'mmmmm       mmmmm ',
*                 'mm mm     mm mm  ',
*                 'mm   mm mm   mm  ',
*                 'mm    mmm    mm  ',
*                 'mm     m     mm  ',
*                 'mm           mm  ',
*                 'mm           mm  ',
*                 'mm           mm  ',
*                 'mm           mm  ',
*                 'mm           mm  '/
 data   cnn      /'nn           nn  ',
*                 'nnn          nn  ',
*                 'nnnn         nn  ',
*                 'nn nn        nn  ',
*                 'nn   nn      nn  ',
*                 'nn    nn     nn  ',
*                 'nn     nn    nn  ',
*                 'nn      nn   nn  ',
*                 'nn       nn nn  ',
*                 'nn        nnnn  ',
*                 'nn         nnn  ',
*                 'nn          nn  '/
 data   coo      /' oooooooooo    ',
*                 'oooooooooooo   ',
*                 'oo          oo ',
*                 'oo          oo ',
```

```
*                'oo          oo   ',
*                'oo          oo   ',
*                'oo          oo   ',
*                'oo          oo   ',
*                'oo          oo   ',
*                'oo          oo   ',
*                'oooooooooooo     ',
*                ' oooooooooo      '/
 data   cpp      /'pppppppppppp     ',
*                'pppppppppppppp   ',
*                'pp          pp   ',
*                'pp          pp   ',
*                'pp          pp   ',
*                'pppppppppppppp   ',
*                'pppppppppppp     ',
*                'pp               ',
*                'pp               ',
*                'pp               ',
*                'pp               ',
*                'pp               '/
 data   cqq      /'    qqqqqqq      ',
*                '   qqqqqqqqq     ',
*                ' qq       qq     ',
*                'qq         qq    ',
*                'qq         qq    ',
*                'qq         qq    ',
*                'qq         qq    ',
*                'qq         qq    ',
*                'qq      qq qq    ',
*                ' qq      qqq     ',
*                '  qqqqqqqqqq     ',
*                '   qqqqqqq qq    '/
 data   crr      /'rrrrrrrrrrr     ',
*                'rrrrrrrrrrrr     ',
*                'rr          rr   ',
*                'rr          rr   ',
*                'rr          rr   ',
*                'rrrrrrrrrrrr     ',
*                'rrrrrrrrrrr      ',
*                'rr        rr     ',
*                'rr        rr     ',
*                'rr         rr    ',
*                'rr          rr   ',
*                'rr          rr   '/
 data   css      /'  sssssssssss   ',
*                'ssssssssssss     ',
*                'ss          ss   ',
*                'ss               ',
*                'ss               ',
*                'ssssssssssss     ',
*                ' sssssssssss     ',
*                '           ss    ',
*                '           ss    ',
*                'ss          ss   ',
*                'ssssssssssss     ',
*                ' sssssssssss     '/
 data   ctt      /'ttttttttttt     ',
*                'ttttttttttt      ',
*                '    tt           ',
*                '    tt           ',
*                '    tt           ',
*                '    tt           ',
*                '    tt           ',
*                '    tt           ',
*                '    tt           ',
*                '    tt           ',
*                '    tt           '/
 data   cuu      /'uu          uu    ',
*                'uu          uu    ',
*                'uu          uu    ',
*                'uu          uu    ',
*                'uu          uu    ',
*                'uu          uu    ',
*                'uu          uu    ',
*                'uu          uu    ',
*                'uu          uu    ',
*                'uuuuuuuuuuuu      ',
*                ' uuuuuuuuuu       '/
 data   cvv      /'vv           vv   ',
*                'vv           vv   ',
*                'vv           vv   ',
*                'vv           vv   ',
*                'vv           vv   ',
*                'vv           vv   ',
*                ' vv         vv    ',
*                '   vv      vv     ',
*                '    vv    vv      ',
*                '     vv vv        ',
*                '      vvv         ',
*                '       v          '/
 data   cww      /'ww           ww   ',
*                'ww           ww   ',
*                'ww           ww   ',
*                'ww           ww   ',
*                'ww           ww   ',
*                'ww     w     ww   ',
*                'ww    www    ww   ',
*                'ww   ww ww   ww   ',
```


data cxx
data cyy
data czz
data cbk
data cps
data cng
data csl
data c00

```
 data   c11      /'      11        ',
+                 '     111        ',
+                 '    1111        ',
+                 '      11        ',
+                 '      11        ',
+                 '      11        ',
+                 '      11        ',
+                 '      11        ',
+                 '      11        ',
+                 '      11        ',
+                 '   11111111     ',
+                 '   11111111     '/
 data   c22      /' 22222222222    ',
+                 '2222222222222   ',
+                 '22          22   ',
+                 '            22   ',
+                 '            22   ',
+                 '          22     ',
+                 '        22       ',
+                 '      22         ',
+                 '    22           ',
+                 ' 22              ',
+                 '2222222222222   ',
+                 '2222222222222   '/
 data   c33      /' 33333333333    ',
+                 '3333333333333   ',
+                 '33          33   ',
+                 '            33   ',
+                 '            33   ',
+                 '          333    ',
+                 '          333    ',
+                 '            33   ',
+                 '            33   ',
+                 '33          33   ',
+                 '3333333333333   ',
+                 ' 33333333333    '/
 data   c44      /'          44    ',
+                 '         444    ',
+                 '        4444    ',
+                 '       44 44    ',
+                 '     44  44     ',
+                 '    44   44     ',
+                 '   44    44     ',
+                 ' 444444444444   ',
+                 '4444444444444   ',
+                 '            44   ',
+                 '            44   ',
+                 '            44   '/
 data   c55      /'5555555555555   ',
+                 '5555555555555   ',
+                 '55              ',
+                 '55              ',
+                 '55              ',
+                 '555555555555    ',
+                 '5555555555555   ',
+                 '            55   ',
+                 '            55   ',
+                 '55          55   ',
+                 '5555555555555   ',
+                 ' 55555555555    '/
 data   c66      /' 666666666666   ',
+                 '6666666666666   ',
+                 '66              ',
+                 '66              ',
+                 '66              ',
+                 '666666666666    ',
+                 '6666666666666   ',
+                 '66          66   ',
+                 '66          66   ',
+                 '66          66   ',
+                 '6666666666666   ',
+                 ' 66666666666    '/
 data   c77      /'7777777777777   ',
+                 '777777777777    ',
+                 '77          77   ',
+                 '           77    ',
+                 '          77     ',
+                 '         77      ',
+                 '         77      ',
+                 '        77       ',
+                 '        77       ',
+                 '       77        ',
+                 '       77        ',
+                 '       77        '/
 data   c88      /' 88888888888    ',
+                 '8888888888888   ',
+                 '88          88   ',
+                 '88          88   ',
+                 '88          88   ',
+                 ' 88888888888    ',
+                 ' 88888888888    ',
+                 '88          88   ',
+                 '88          88   ',
+                 '88          88   ',
+                 '8888888888888   ',
+                 ' 88888888888    '/
 data   c99      /' 99999999999    ',
+                 '9999999999999   ',
+                 '99          99   ',
+                 '99          99   ',
```


data ceq /
data cpd /
data cln /
data crp /
data cas /
data cma /
data clp /
data cap /

```
*                ',                ',
*                ',                ',
*                ',                ',
*                '                 '/
 data   cam     /' &&&&&&&&&&      ',
*                '&&&&&&&&&&&&     ',
*                '&&          &&   ',
*                '&&               ',
*                ' &&&&&&&&        ',
*                ' &&&&&&&&        ',
*                '&&               ',
*                '&&     &&&&&     ',
*                '&&     &&&&&&    ',
*                '&&         &&    ',
*                '&&&&&&&&&&&&     ',
*                ' &&&&&&&&&&      '/
 data   cdr     /'       $$        ',
*                ' $$$$$$$$$$      ',
*                '$$$$$$$$$$$$     ',
*                '$$    $$         ',
*                '$$    $$         ',
*                '$$$$$$$$$$$$     ',
*                ' $$$$$$$$$$$$    ',
*                '      $$   $$    ',
*                '      $$   $$    ',
*                '$$$$$$$$$$$$$    ',
*                ' $$$$$$$$$$      ',
*                '      $$         '/
 data   cer     /'**********       ',
*                '*          *     ',
*                '*          *     ',
*                '*          *     ',
*                '*          *     ',
*                '*  invalid *     ',
*                '* character *     ',
*                '*          *     ',
*                '*          *     ',
*                '*          *     ',
*                '*          *     ',
*                '**********       '/
 end
::::::::::::::
mvpk.f
::::::::::::::
      subroutine mvpk(rho2,rho1,rhonew,rho,bett,betae2,betae1,betnew
     $ ,betaef,bet2,bet1,betn,rlam2,rlam1,rlamnew,rlamda,q2,q1,qnew
     $ ,extsrc,n6pk,telaps,irep,irepmx,fterm,ndg,iout)
      real*4  rho2,rho1,rhonew,rho,bett,betae2(10),betae1(10),betnew(10)
     $ ,betaef(10),bet2,bet1,betn,rlam2,rlam1,rlamnew,rlamda,q2,q1,qnew
     $ ,extsrc,telaps,fterm
      character*12 dumpr(8)
      data dumpr/12h    time    ,12h amplitude  ,12h  csubi(1)
     $ ,12h  csubi(2)  ,12h  csubi(3)  ,12h  csubi(4)  ,12h  csubi(5)
     $ ,12h  csubi(6)  /
      character*80 equal
      equal = "========================================================="
c
      rho2   = rho1
      rho1   = rhonew
      rhonew = rho
c
      bett = zero
      do i = 1, ndg
        betae2(i) = betae1(i)
        betae1(i) = betnew(i)
        betnew(i) = betaef(i)
        bett      = bett + betnew(i)
      enddo
      bet2 = bet1
      bet1 = betn
      betn = bett
c
      rlam2   = rlam1
      rlam1   = rlamnew
      rlamnew = rlamda
c
      q2     = q1
      q1     = qnew
      qnew   = extsrc
c
      if(irep.eq.irepmx.or.iout.eq.2) then
      call outpk(rho,betaef,bett,rlamda,telaps,irep,ndg,n6pk)
c      write(n6pk,'(a)') equal
c      write(n6pk,*) ' rho           = ',rho,' at ',telaps,' for irep = '
c     $ ,irep
c      write(n6pk,*) ' LAMBDA        = ',rlamda
c      write(n6pk,*) ' Q             = ',extsrc
c      write(n6pk,*) ' betaef(1:ndg) = ',(betaef(i),i=1,ndg)
c      write(n6pk,*) ' betaeft       = ',bett
c      write(n6pk,*) ' fterm         = ',fterm
      endif
      if(iout.ne.0) write(n6pk,90) (dumpr(i),i=1,ndg+2)
90    format(1x,3hitn,2x,5h itj ,2x,8(a12))
c      write(42,*) rlamda
c      write(43,*) (betaef(i),i=1,ndg)
c
      return
      end
::::::::::::::
na.f
```

```
::::::::::::::
      function na(len)
      common /nacom/ n,ndoub
      if(ndoub.eq.1) then
      na = n
      n  = n + len
      elseif(ndoub.eq.2) then
      n = (n/2)*2 + 1
      na = n
      n = n + len*2
      endif
      return
      end
::::::::::::::
normaf.f
::::::::::::::
      subroutine normaf(flux,index,aflux,idim,pow0,powini)
      real*4 flux(index),aflux(idim),pow0,powini
      do ii=1,index
       flux(ii)  = flux(ii) *powini/pow0
      enddo
      do ii=1,idim
       aflux(ii) = aflux(ii)*powini/pow0
      enddo
      return
      end
::::::::::::::
normap.f
::::::::::::::
      subroutine normap(shap1,ansh1,xsfs,im,jm,km,igm,indexm
     $ ,index,rnu,econfr,vr,dth,dz)
c
      real*4 shap1(index),ansh1(indexm)
     $ ,xsfs(index),rnu,econfr,vr(im),dth(jm),dz(km),vol,voltot,power
     $ ,pownrm
c
      pownrm = 0.25e-6     ! 1/4 of core is considered
      voltot = 0.0
      power = 0.0
      do  ig=1,igm
      do  k =1,km
      do  j =1,jm
      do  i =1,im
       i1 = i + im*((j-1) + jm*((k-1) + km*(ig-1)))
       if(xsfs(i1).gt.0.0) then
       vol = vr(i)*dth(j)*dz(k)
        if(ig.eq.1) voltot = voltot + vol
        power = power + xsfs(i1)*shap1(i1)*vol/rnu*econfr
       endif
      enddo
      enddo
      enddo
      enddo
c
      avpowdi = power/voltot
      do i=1,index
       shap1(i) = shap1(i)/avpowdi*pownrm
      enddo
c
      do i=1,indexm
       ansh1(i) = ansh1(i)/avpowdi*pownrm
      enddo
c
      voltot = 0.0
      power = 0.0
      do  ig=1,igm
      do  k =1,km
      do  j =1,jm
      do  i =1,im
       i1 = i + im*((j-1) + jm*((k-1) + km*(ig-1)))
       if(xsfs(i1).gt.0.0) then
       vol = vr(i)*dth(j)*dz(k)
        if(ig.eq.1) voltot = voltot + vol
        power = power + xsfs(i1)*shap1(i1)*vol/rnu*econfr
       endif
      enddo
      enddo
      enddo
      enddo
      power = power/voltot
c
      return
      end
::::::::::::::
out0.f
::::::::::::::
      subroutine out0(powxt,tempold,index1,xsfp,index,itn,itj,telaps
     $ ,n6pk)
c
      real*4 powxt(index1),tempold(index1),xsfp(index)
c
c     find max./min. power and temp change
c
      powmax = zero
      tempmax= zero
      do iix=1,index1
       if(powxt(iix).gt.powmax) then
         powmax = powxt(iix)
         ipow   = iix
       endif
```

```
c
       if(tempold(iix).gt.tempmax) then
         tempmax = tempold(iix)
         itemp   = iix
       endif
      enddo
c
      powmin = powmax
      tempmin= tempmax
      do iix=1,index1
       if(xsfp(iix).gt.0.0) then
        if(powxt(iix).lt.powmin) then
          powmin = powxt(iix)
          ipowm  = iix
        endif
c
        if(tempold(iix).lt.tempmin) then
          tempmin = tempold(iix)
          itempm  = iix
        endif
       endif
      enddo
c
c      if(irep.eq.irepmx) then
       if(itn.eq.1.and.itj.eq.1) then
         open(35,file='maxpow',status='unknown')
       else
         open(35,file='maxpow',status='old',access='append')
       endif
       write(35,*) telaps,powmax,ipow
       close(35)
       if(itn.eq.1.and.itj.eq.1) then
         open(35,file='minpow',status='unknown')
       else
         open(35,file='minpow',status='old',access='append')
       endif
       write(35,*) telaps,powmin,ipowm
       close(35)
       if(itn.eq.1.and.itj.eq.1) then
         open(35,file='maxtemp',status='unknown')
       else
         open(35,file='maxtemp',status='old',access='append')
       endif
       write(35,*) telaps,tempmax,itemp
       close(35)
       if(itn.eq.1.and.itj.eq.1) then
         open(35,file='mintemp',status='unknown')
       else
         open(35,file='mintemp',status='old',access='append')
       endif
       write(35,*) telaps,tempmin,itempm
       close(35)
       write(n6pk,*) ' Max Power = ',powmax,' at ',ipow
     $ , ' ;  Max Temperature = ',tempmax,' at ',itemp
       write(n6pk,*) ' Min Power = ',powmin,' at ',ipowm
     $ , ' ;  Min Temperature = ',tempmin,' at ',itempm
c      endif
c
      return
      end
:::::::::::::::
outpk.f
:::::::::::::::
      subroutine outpk(rho,beta,betat,rlamda,time,irep,ndg,n6)
      real*4 rho,beta(ndg),betat,rlamda
      character*1 equal
      equal = '='
c
      write(n6,'(120(a1))') (equal,itest=1,115)
      write(n6,890) time,irep
      write(n6,900) rho,rho/betat
      write(n6,910) (beta(i),i=1,ndg)
      write(n6,920) betat
c      write(n6,930) (csubi(i),i=1,ndg)
      write(n6,940) rlamda
c      write(n6,*) ' fterm         = ',fterm0
      write(n6,'(120(a1))') (equal,itest=1,115)
890   format(1x,8h Time = ,1pe13.6,5x,20h Iteration Number = ,i2)
900   format(1x,44h Reactivity                                = ,1pe13.6
     $ ,2x,1h(,e13.6,2h$))
910   format(1x,44h Effective Delayed Neutron Fractions       =
     $ ,1p,6(2x,e10.3))
920   format(1x,44h Effective Total Delayed Neutron Fraction = ,1pe10.3)
c930   format(1x,44h Weighted Precursor Densities             = ,1p
c     $ ,6(2x,e11.4))
940   format(1x,44h Generation Time                           = ,1pe10.3)
c
      return
      end
:::::::::::::::
outshp.f
:::::::::::::::
      subroutine outshp(n6pk,equal,irep,index,shap1,shap2,igm)
      real*4 shap1(index),shap2(index)
      character*1 equal
      write(n6pk,'(120(a1))') (equal,itest=1,115)
      write(n6pk,'(8h irep = ,i2)')irep
      write(n6pk,'(44h          Previous  Current          Percent  )')
      write(n6pk,'(44h index     Shape     Shape        Difference )')
      if(igm.gt.2) then
```

```
      do itt1=1,index
       gak = shap1(itt1)-shap2(itt1)
       guk = gak / shap2(itt1) * 100.
       write(n6pk,'(1x,i5,1x,1p,(1x,3(e12.5,1x)))') itt1
     $ ,shap2(itt1),shap1(itt1),guk
      enddo
      else
      do itt1=1,index/igm
       gak = (shap1(itt1)-shap2(itt1)) / shap2(itt1) * 100.
       itt2 = itt1 + index/igm
       guk = (shap1(itt2)-shap2(itt2)) / shap2(itt2) * 100.
       write(n6pk,'(1x,i5,1x,1p,1x,3(e12.5,1x),1x,i5,1x,1p,1x
     $ ,3(e12.5,1x))')
     $  itt1,shap2(itt1),shap1(itt1),gak
     $ ,itt2,shap2(itt2),shap1(itt2),guk
      enddo
      endif
      write(n6pk,'(120(a1))') (equal,itest=1,115)
      return
      end
::::::::::::::
pdcal.f
::::::::::::::
      subroutine pdcal(csubi,rlamda,fterm,idim,cig,scadj,dume2
     $ ,vr,dth,dz,im,jm,km,igm,index,indexn,index1,chid)
c     rlamda is lamda(t)
c     fterm is F(t)
c
      real*4 cig(indexn),dume2(index),scadj(index),chid(100)
     $ ,vr(im),dth(jm),dz(km),csubi(10),rlamda,fterm,res
c
c     multiply adjoint flux and prec. densities
      do 5 id =1,idim
      call clearx(dume2,1,index)
      do 10 ig=1,igm
      do 15 i=1,index1
       i2 = i + index1*(id-1)
       i3 = i + index1*(ig-1)
       dume2(i3) = scadj(i3)*cig(i2)*chid(ig)
15    continue
10    continue
c
      call vgintg(dume2,vr,dth,dz,index,igm,im,jm,km,res)
      csubi(id) = res/(fterm*rlamda)
5     continue
c
      return
      end
::::::::::::::
pertur.f
::::::::::::::
      subroutine pertur(rijzn,index1,itn,itj,imw,jmw)
c
      include 'comsd1.F'
c
c   rijzn must be the last zone zone map
c   on return rijzn is the new zone
c
      real*4 rijzn(index1)
c
c   linear rod movement
c
c       find where we end up
      iend = 0
      do i=1,itn-1
       iend = iend + idtj(i)     ! sum all previous ones
      enddo
      iend = iend + itj    !add the section in this dtn
c
       idostp = 1
       if(klo.gt.kup) idostp = -1
       klo = mvkl
       kup = iend*mvks + mvkl - 1
       if(kup.gt.mvku) kup = mvku
       if(kup.lt.mvkl) kup = mvkl
       jlo = mvjl
       jup = iend*mvjs + mvjl - 1
       if(jup.gt.mvju) jup = mvju
       if(jup.lt.mvjl) jup = mvjl
       ilo = mvil
       iup = iend*mvis + mvil - 1
       if(iup.gt.mviu) iup = mviu
       if(iup.lt.mvil) iup = mvil
c   modify the zone data for the section
c      passed to this routine in this call
      do 20 k = klo, kup, idostp
c   skip previous planes
      i1 = (k-1)*imw*jmw
      do 20 j=jlo, jup
c   skip previous columns
      i2 = (j-1)*imw
      do 20 i=ilo, iup
      i3 = i1+i2+i
      rijzn(i3) = real(mvmatr)
20    continue
c
      return
      end
::::::::::::::
pint.f
```

```
::::::::::::::
      subroutine pint(f1,f2,f3,f4,h1,h2,x)
      real*4 f1,f2,f3,f4
     $ ,t1,t2,t3,t4,h1,h2,x
c    linear or quadratic interpolator
c     lct1=0 and h1=0 signals for linear interpolation
c
c     linear/quadratic interpolator
c
c
c      note that x2=0, |x2|-|x1|=h1, x3-x2=h2
c       x1,y1      x2,y2       x3,y3
c         |          |           |
c        -x1         0           x3
c        <----------><----------->
c             h1           h2
c
c      h1 = x2 - x1
c      h2 = x3 - x2
c
c    tnew is the time from t2
c
c      note that interpolation is done for x2<x<x3
c
c    check for linear interpolation:
c
      if(h1.eq.0) goto 200
      if(h1.ne.h2) goto 100
      t1 = (f3 - f1)/(2.*h1)
      t2 = (f3 - 2.*f2 + f1)/(2.*(h1**2.))
      f4 = f2 + t1*(x) + t2*((x)**2.)
      goto 900
100   t1 = (f3 - f1)/(h1+h2)
      t2 = (f3 - f2)*(h2-h1)/(h2*h1)
      t3 = (f3 - 2.*f2 + f1)/(2.*h1*h2)
      t4 = (f3 - f1)*(h2-h1)/((2.*h1*h2)*(h1+h2))
      f4 = f2 + (x)*(t1+t2) + ((x)**2.)*(t3+t4)
      goto 900
200   f4 = f2 + (x)/h2 * (f3-f2)
c
900   return
      end
::::::::::::::
pinter.f
::::::::::::::
      subroutine pinter(f1,f2,f3,f4,h1,h2,x,index)
      real*4 f1(index),f2(index),f3(index),f4(index)
     $ ,t1,t2,t3,t4,h1,h2,x
c    linear or quadratic interpolator
c     lct1=0 and h1=0 signals for linear interpolation
c
c     linear/quadratic interpolator
c
c
c      note that x2=0, |x2|-|x1|=h1, x3-x2=h2
c       x1,y1      x2,y2       x3,y3
c         |          |           |
c        -x1         0           x3
c        <----------><----------->
c             h1           h2
c
c      h1 = x2 - x1
c      h2 = x3 - x2
c
c    tnew is the time from t2
c
c      note that interpolation is done for x2<x<x3
c
c    check for linear interpolation:
c
c      if(h1.eq.0.and.lct1.eq.0) goto 200
      if(h1.eq.0) goto 200
      if(h1.ne.h2) goto 100
      do 20 i=1,index
      t1 = (f3(i) - f1(i))/(2.*h1)
      t2 = (f3(i) - 2.*f2(i) + f1(i))/(2.*(h1**2.))
      f4(i) = f2(i) + t1*(x) + t2*((x)**2.)
20    continue
      goto 900
100   do 120 i=1,index
      t1 = (f3(i) - f1(i))/(h1+h2)
      t2 = (f3(i) - f2(i))*(h2-h1)/(h2*h1)
      t3 = (f3(i) - 2.*f2(i) + f1(i))/(2.*h1*h2)
      t4 = (f3(i) - f1(i))*(h2-h1)/((2.*h1*h2)*(h1+h2))
      f4(i) = f2(i) + (x)*(t1+t2) + ((x)**2.)*(t3+t4)
120   continue
      goto 900
200   do 220 i=1,index
      f4(i) = f2(i) + (x)/h2 * (f3(i)-f2(i))
220   continue
c
900   return
      end
::::::::::::::
pintx.f
::::::::::::::
      subroutine pintx(sig0,sig1,sigf,x,tltot,index)
c
      real*4 sig0(index),sig1(index),sigf(index),x,tltot
c
```

```
       do i=1,index
        sigf(i) = (sig1(i)-sig0(i))*x/tltot + sig0(i)
       enddo
c
       return
       end
::::::::::::::
pkcal.f
::::::::::::::
      subroutine pkcal(rho,betaef,rlamda,extsrc,bet,ndg,time)
      include 'comsd7.F'
      real*4 rho,betaef(10),rlamda,extsrc,bet,dt,t1,t2
      real*4 zero
      zero = 0.e0
c
c
      dt = time
      if(.not.jquad) then
      rho    = rho1 + dt/hj1 * (rhonew-rho1)
      rlamda = rlam1 + dt/hj1 * (rlamnew-rlam1)
      extsrc = q1 + dt/hj1 * (qnew-q1)
       bet = zero
       do id =1, ndg
      betaef(id) = betae1(id) + dt/hj1*(betnew(id)-betae1(id))
      bet        = bet + betaef(id)
       enddo
      else
c
      t1  = (rhonew - rho2)/(2.*hj1)
      t2  = (rhonew - 2*rho1 + rho2)/(2.*(hj1**2.))
      rho = rho1 + t1*dt + t2*(dt**2.)
c
      t1  = (rlamnew - rlam2)/(2.*hj1)
      t2  = (rlamnew - 2*rlam1 + rlam2)/(2.*(hj1**2.))
      rlamda = rlam1 + t1*dt + t2*(dt**2.)
c
      extsrc = zero
c
       bet = zero
       do id =1, ndg
      t1  = (betnew(id) - betae2(id))/(2.*hj1)
      t2  = (betnew(id) - 2*betae1(id) + betae2(id))/(2.*(hj1**2.))
      betaef(id) = betae1(id) + t1*dt + t2*(dt**2.)
      bet        = bet + betaef(id)
       enddo
      endif
c
      return
c
      rho  = rhonew
      rlamda= rlamnew
      extsrc  = qnew
      bet = zero
      do i=1,ndg
       betaef(i) = betnew(i)
       bet       = bet + betaef(i)
      enddo
      return
c
      end
::::::::::::::
pkvar.f
::::::::::::::
      subroutine pkvar(rho,fterm,betaef,betaeft,rlamda,extsrc
     $ ,idim,shape,ansh,rhot3,rhot4,rhot5,anadj,xstd,scadj,xssd
     $ ,dume1,xsfd,chit,dume2,vr,dth,dz,dume3,xsfp,dume4
     $ ,index,index1,indexm,im,jm,km,igm,mm,ipkmod)
c2345678901234567890123456789012345678901234567890123456789012345678901234567890123456789012
c
      real*4 shape(index),ansh(indexm),rhot3(index),rhot4(index)
     $ ,rhot5(index),anadj(indexm),xstd(index),scadj(index)
     $ ,xssd(index*igm),dume1(index1),xsfd(index),chit(igm)
     $ ,dume2(index),vr(im),dth(jm),dz(km),dume3(index1)
     $ ,xsfp(index),dume4(index)
c
      real*4 rho,fterm,betaef(idim),betaeft,rlamda,extsrc
      real*4 zero
      zero = 0.e0
c
      rho = zero
      fterm = zero
      call clearx(betaef,1,idim)
      call rhocal(rho,fterm,shape,ansh,rhot3,rhot4,rhot5,anadj
     $,xstd,scadj,xssd,dume1,xsfd,chit,dume2,vr,dth,dz,dume3,xsfp,dume4
     $ ,index,index1,indexm,im,jm,km,igm,mm,ipkmod)
c
      call betcal(betaef,fterm,idim,dume1,shape,xsfp,dume2
     $ ,scadj,vr,dth,dz,index,index1,im,jm,km,igm)
      betaeft = zero
      do i = 1, idim
        betaeft   = betaeft + betaef(i)
      enddo
c
      if(ipkmod.eq.0) then
      call lamcal(rlamda,fterm,ansh,anadj,dume2,vr,dth,dz
     $ ,index,index1,indexm,im,jm,km,igm,mm)
      elseif(ipkmod.eq.1) then
      call lamcalh(rlamda,fterm,shape,scadj,dume2,vr,dth,dz
     $ ,index,index1,im,jm,km,igm)
      endif
```

```
c
      extsrc = zero
c
      return
      end
::::::::::::::
powcal.f
::::::::::::::
      subroutine powcal(power,flux,index,index1,igm,xsecf
     $ ,vr,dth,dz,im,jm,km,rnu,powden)
      real*4 power,flux(index),xsecf(index),vr(im),dth(jm),dz(km)
     $ ,rnu,powden(index1),pow
      real*4 zero
      zero = 0.e0
c
c
      call clearx(powden,1,index1)
      power = zero
      do 10 ig=1,igm
      do 20 k =1,km
      do 30 j =1,jm
      do 40 i =1,im
       vol = vr(i)*dth(j)*dz(k)
       i1 = i + im*((j-1) + jm*((k-1) + km*(ig-1)))
       i2 = i + im*((j-1) + jm*(k-1))
       pow   = xsecf(i1)*flux(i1)/rnu
       power = power + pow*vol
       powden(i2) = powden(i2) + pow
40    continue
30    continue
20    continue
10    continue
c
      return
      end
::::::::::::::
pownc0.f
::::::::::::::
      subroutine pownc0(pow0,shap1,index,igm,sigf,itn,vr,dth,dz,im,jm
     $ ,km,rnu,h,noup,amptk,itkm,x2,shap2,shape,index1,powxt,mtm
     $ ,itj,dtj,irep,irepmx,dttot,mdamp,volz,izm,zorg,zmat,powzt)
c
c    power is total power,
c    powxt is power for each cell
c
      real*4 power1,timep2,vol,econfr
      real*4 pow0,shap1(index),sigf(index),vr(im),dth(jm),dz(km),rnu,h
     $ ,amptk(mdamp),x2,shap2(index),shape(index),powxt(index1),dtj
     $ ,dttot,volz(mtm),zorg(index1),zmat(izm),powzt(mtm)
      real*4 zero
      zero = 0.e0
      econfr = 3.204e-11
      call clearx(powxt,1,index1)
c  open and close the file to force writing of buffer
      if(itn.eq.1.and.itj.eq.1.and.irep.eq.0) then
        open(noup,status='unknown')
      else
        open(noup,status='old',access='append')
      endif
c
c    calculate zone volumes on first pass
c
      if(itn.eq.1) then
        call clearx(volz,1,mtm)
        do k=1,km
        do j=1,jm
        do i=1,im
         i1 = i + im*((j-1) + jm*(k-1))
         morg = int(zorg(i1))
         mat  = int(zmat(morg))
         volz(mat) = volz(mat) + vr(i)*dth(j)*dz(k)  ! mat # = zone #
        enddo
        enddo
        enddo
c
        voltot = zero
        do iz=1,mtm
         voltot = voltot + volz(iz)
        enddo
c
      endif
c
c ------------------------------------------
c
      open(35,file='bpower',status='unknown',form='unformatted')
      irecs = itkm
      write(35) irecs
c
      open(36,file='bpowerz',status='unknown',form='unformatted')
      write(36) irecs
c
      xtim   = dtj*(itj-1)
      timep2 = dttot + xtim
      do ij=1,itkm
      call clearx(powxt,1,index1)
      call linter(shap2,shap1,shape,h,xtim+x2*ij,index)
      power1 = zero
      do 410 ig=1,igm
      do 420 k =1,km
      do 430 j =1,jm
```

```
      do 440 i =1,im
       vol = vr(i)*dth(j)*dz(k)
       i1 = i + im*((j-1) + jm*((k-1) + km*(ig-1)))
       i2 = i + im*((j-1) + jm*(k-1))
       power1 = power1 + sigf(i1)*shape(i1)*vol/rnu
       powxt(i2) = powxt(i2) +
     $                 (sigf(i1)*shape(i1)/rnu)*amptk(ij)*econfr
440   continue
430   continue
420   continue
410   continue
      power1 = power1*econfr
c   save to calc. temp dist. and clear for next ij
      write(35) (powxt(i),i=1,index1)
      power  = power1*amptk(ij)
      timep2 = timep2 + x2
      stimep = timep2
      spow   = power/pow0
      write(noup,*) stimep,spow
c
c    calculate zone powers from mesh powers
c
      call clearx(powzt,1,mtm)
      do k =1,km
      do j =1,jm
      do i =1,im
        vol = vr(i)*dth(j)*dz(k)
        i2 = i + im*((j-1) + jm*(k-1))
        morg = int(zorg(i2))
        mat  = int(zmat(morg))
c   can only sum powers , but sum as a weighted power
        powzt(mat) = powzt(mat) + powxt(i2)*vol/volz(mat)
      enddo
      enddo
      enddo
      write(36) (powzt(i),i=1,mtm)
c
      enddo
c
      close(noup)
      close(35)
      close(36)
c
      if(irep.eq.irepmx) then
        write(50,*) stimep,spow*pow0
      endif
c
      return
      end
::::::::::::::
powncl.f
::::::::::::::
      subroutine powncl(pow0,shape1,index,igm,sigf,itn,vr,dth,dz,im,jm
     $ ,km,rnu,h,noup,amptk,itkm,x2,shape0,shape,itj,dtj,irep,irepmx
     $ ,dttot)
c
c    power is total power,
c
      real*4 power1,timep2,vol,power
      real*4 pow0,shape1(index),sigf(index),vr(im),dth(jm),dz(km),rnu,h
     $ ,amptk(itkm),x2,shape0(index),shape(index),dtj,dttot
      real*4 zero
      zero = 0.e0
c  open and close the file to force writing of buffer
      if(itn.eq.1.and.itj.eq.1.and.irep.eq.0) then
        open(noup,status='unknown')
      else
        open(noup,status='old',access='append')
      endif
c
c -----------------------------------------
c
      xtim   = dtj*(itj-1)
      timep2 = dttot + xtim
      do ij=1,itkm
      call linter(shape0,shape1,shape,h,xtim+x2*ij,index)
      power1 = zero
      do 410 ig=1,igm
      do 420 k =1,km
      do 430 j =1,jm
      do 440 i =1,im
       vol = vr(i)*dth(j)*dz(k)
       i1 = i + im*((j-1) + jm*((k-1) + km*(ig-1)))
       power1 = power1 + sigf(i1)*shape(i1)*vol/rnu
440   continue
430   continue
420   continue
410   continue
      power  = power1*amptk(ij)
      timep2 = timep2 + x2
      stimep = timep2
      spow   = power/pow0
      write(noup,*) stimep,spow
      enddo
c
      close(noup)
c
      if(irep.eq.irepmx) then
        write(50,*) stimep,spow
      endif
```

```
c
      return
      end
::::::::::::::
pownc2.f
::::::::::::::
      subroutine pownc2(pow0,shape1,index,igm,sigf,nout,itn,vr,dth,dz,im
     $ ,jm,km,rnu,x,h,amp1,idtj,noup,amp2,idtk,ninamp,x2,shape0
     $ ,shape)
c
      common /pwr0/ time,time2
      real*4 time,time2,power1,timep,timep2,vol
      real*4 pow0,shape1(index),sigf(index),vr(im),dth(jm),dz(km),rnu
     $ ,x,h,amp1(idtj),amp2(idtk),x2,shape0(index),shape(index)
      real*4 zero
      zero = 0.e0
c      econfr = 3.204e-11
c  open and close the file to force writing of buffer
      if(itn.eq.1) then
        open(nout,status='unknown')
        open(noup,status='unknown')
        open(35,file='pshort',status='unknown')
      else
        open(nout,status='old',access='append')
        open(noup,status='old',access='append')
        open(35,file='pshort',status='old',access='append')
      endif
c
      if(idtj.le.10) then
       ishort = 0
      else
       ishort = 1
      iskip = idtj/10
      endif
c
      if(itn.eq.1) then
       time  = zero
       time2 = zero
      endif
      power1 = zero
c
c ----------------------------------------
c
      timep = time
      do ij=1,idtj
      call linter(shape0,shape1,shape,h,x*ij,index)
      power1 = zero
      do 310 ig=1,igm
      do 320 k =1,km
      do 330 j =1,jm
      do 340 i =1,im
       vol = vr(i)*dth(j)*dz(k)
       i1 = i + im*((j-1) + jm*((k-1) + km*(ig-1)))
       power1 = power1 + sigf(i1)*shape(i1)*vol/rnu
340   continue
330   continue
320   continue
310   continue
      power = power1*amp1(ij)
c
c ----------------------------------------
c
      timep = timep + x
       stimep = timep
       spow   = power/pow0
      write(nout,*) stimep,spow
      if(ishort.eq.1) then
       i2 = mod(ij,iskip)
       if(i2.eq.(iskip-1).or.ij.eq.1.or.ij.eq.idtj) then
        write(35,*) stimep,spow
       endif
      else
        write(35,*) stimep,spow
      endif
      enddo
c
c  this is for longer output
      timep2 = time2
      rewind(ninamp)
      do ij=1,idtk
       read(ninamp,*) amp2(ij)
      enddo
      do ij=1,idtk
      call linter(shape0,shape1,shape,h,x2*ij,index)
      power1 = zero
      do 410 ig=1,igm
      do 420 k =1,km
      do 430 j =1,jm
      do 440 i =1,im
       vol = vr(i)*dth(j)*dz(k)
       i1 = i + im*((j-1) + jm*((k-1) + km*(ig-1)))
       power1 = power1 + sigf(i1)*shape(i1)*vol/rnu
440   continue
430   continue
420   continue
410   continue
      power = power1*amp2(ij)
      timep2 = timep2 + x2
       stimep = timep2
       spow   = power/pow0
```

```
      write(noup,*) stimep,spow
      enddo
c
c
      time   = timep
      time2  = timep2
c
      close(nout)
      close(noup)
      close(35)
      return
      end
::::::::::::::
pownc5.f
::::::::::::::
      subroutine pownc5(pow0,shap1,index,igm,sigma0,nout,itn,vr,dth,dz
     $ ,im,jm,km,rnu,x,h,amp1,idtj,telaps,noup,amp2
     $ ,idtk,ninamp,x2,sig0,sig1,shap0,sigma1,index1,ihp
     $ ,iht,mtm,izm,zorg,zmat,tx1,tx0)
      real*4 term1,term2,term3,term4,timep,timep2,vol
      real*4 pow0,shap1(index),sigma0(ihp,mtm,igm),vr(im),dth(jm),dz(km)
     $ ,rnu,x,h,amp1(idtj),telaps,amp2(idtk),x2,shap0(index)
     $ ,sig0(index),sig1(index),sigma1(ihp,mtm,igm),zorg(index1)
     $ ,zmat(izm),tx1,tx0
      common /pwr0/ time
      real*4 time
      real*4 zero
      zero = 0.e0
c  x=dtj, h= dtn, x2=dtk
c      econfr = 3.204e-11
c  open and close the file to force writing of buffer
      if(itn.eq.1) then
        open(nout,status='unknown')
        open(noup,status='unknown')
       open(35,file='pshort',status='unknown')
      else
        open(nout,status='old',access='append')
        open(noup,status='old',access='append')
       open(35,file='pshort',status='old',access='append')
      endif
c
      if(idtj.le.10) then
       ishort = 0
      else
       ishort = 1
      iskip = idtj/10
      endif
c
c
      if(itn.eq.1) then
       time  = zero
       timep  = zero
      else
       timep = time
      endif
c
c   create initial and final zone cross sections
c
      ih = iht-1
c
      do ig=1,igm
      do k =1,km
      do j =1,jm
      do i =1,im
       i1 = i + im*((j-1) + jm*((k-1) + km*(ig-1)))
       i2 = i + im*((j-1) + jm*(k-1))
       morg    = int(zorg(i2))
       mat = int(zmat(morg))
       sig0(i1) = sigma0(ih,mat,ig)*(1.-tx0) + sigma1(ih,mat,ig)*tx0
       sig1(i1) = sigma0(ih,mat,ig)*(1.-tx1) + sigma1(ih,mat,ig)*tx1
      enddo
      enddo
      enddo
      enddo
c
c
c -----------------------------------------
c
c  calculate intermediate powers
      term1 = zero
      do 10 ig=1,igm
      do 20 k =1,km
      do 30 j =1,jm
      do 40 i =1,im
       vol = vr(i)*dth(j)*dz(k)
       i1 = i + im*((j-1) + jm*((k-1) + km*(ig-1)))
       term1 = term1 + sig0(i1)*shap0(i1)*vol/rnu
40    continue
30    continue
20    continue
10    continue
c
      term2 = zero
      do 110 ig=1,igm
      do 120 k =1,km
      do 130 j =1,jm
      do 140 i =1,im
       vol = vr(i)*dth(j)*dz(k)
       i1 = i + im*((j-1) + jm*((k-1) + km*(ig-1)))
       term2 = term2 + sig1(i1)*shap1(i1)*vol/rnu
```

```
140   continue
130   continue
120   continue
110   continue
c
      term3 = zero
      do 210 ig=1,igm
      do 220 k =1,km
      do 230 j =1,jm
      do 240 i =1,im
       vol = vr(i)*dth(j)*dz(k)
       i1 = i + im*((j-1) + jm*((k-1) + km*(ig-1)))
       term3 = term3 + sig0(i1)*shap1(i1)*vol/rnu
240   continue
230   continue
220   continue
210   continue
c
      term4 = zero
      do 310 ig=1,igm
      do 320 k =1,km
      do 330 j =1,jm
      do 340 i =1,im
       vol = vr(i)*dth(j)*dz(k)
       i1 = i + im*((j-1) + jm*((k-1) + km*(ig-1)))
       term4 = term4 + sig1(i1)*shap0(i1)*vol/rnu
340   continue
330   continue
320   continue
310   continue
c -----------------------------------------
c
      do i=1,idtj
      power  = amp1(i)*(term1*(1.- (x*i)/h)**2.
     $              +   term2*((x*i)/h)**2.
     $              +   term3*(x*i)/h*(1. - (x*i)/h)
     $              +   term4*(x*i)/h*(1. - (x*i)/h))
      timep = timep + x
       stimep = timep
       spow   = power/pow0
      write(nout,*) stimep,spow
c
      if(ishort.eq.1) then
       i2 = mod(ij,iskip)
       if(i2.eq.(iskip-1).or.ij.eq.1.or.ij.eq.idtj) then
        write(35,*) stimep,spow
       endif
      else
        write(35,*) stimep,spow
      endif
c
      enddo
c  this is for longer output
      rewind(ninamp)
      do i=1,idtk
       read(ninamp,*) amp2(i)
      enddo
      do i=1,idtk
      power  = amp2(i)*(term1*(1.- (x2*i)/h)**2.
     $              +   term2*((x2*i)/h)**2.
     $              +   term3*(x2*i)/h*(1. - (x2*i)/h)
     $              +   term4*(x2*i)/h*(1. - (x2*i)/h))
      timep2 = timep2 + x2
       stimep = timep2
       spow   = power/pow0
      write(noup,*) stimep,spow
      enddo
c
      time = timep
c
      close(nout)
      close(noup)
      close(35)
      return
      end
::::::::::::::
pownc6.f
::::::::::::::
      subroutine pownc6(pow0,shape1,index,igm,sigf,nout,itn,vr,dth,dz,im
     $ ,jm,km,rnu,x,h,amp1,idtj,noup,amp2,mdamp
     $ ,idtk,ninamp,x2,sig0,sig1,xsf1,xsf2,ihp,iht,mtm
     $ ,izm,zorg,zmat,ilpend,tltot,telaps,shape0,shape)
c
      common /pwr0/ time,time2
      real*4 time,time2,power1,timep,timep2,vol,tltot,telaps
      real*4 pow0,shape1(index),sigf(index),vr(im),dth(jm),dz(km),rnu
     $ ,x,h,amp1(idtj),amp2(mdamp),x2,shape0(index),shape(index)
     $ ,sig0(index),sig1(index),xsf1(ihp,mtm,igm),xsf2(ihp,mtm,igm)
     $ ,zorg(index/igm),zmat(izm)
      real*4 zero
      zero = 0.e0
c      econfr = 3.204e-11
c  open and close the file to force writing of buffer
      if(itn.eq.1) then
        open(nout,status='unknown')
        open(noup,status='unknown')
        open(35,file='pshort',status='unknown')
      else
        open(nout,status='old',access='append')
        open(noup,status='old',access='append')
```

```
        open(35,file='pshort',status='old',access='append')
      endif
c
      if(idtj.le.10) then
       ishort = 0
      else
       ishort = 1
      iskip = idtj/10
      endif
c
      if(itn.eq.1) then
       time  = zero
       time2 = zero
      endif
      power1 = zero
c
c ----------------------------------------
c
c   create zone xsecs if pert. is still going on
c    otherwise sigf has all you need
c    xsf1 is final xsf2 is initial step perturbed
      if(itn.lt.ilpend) then
c
      ih = iht-1
      do ig=1,igm
      do k =1,km
      do j =1,jm
      do i =1,im
       i1 = i + im*((j-1) + jm*((k-1) + km*(ig-1)))
       i2 = i + im*((j-1) + jm*(k-1))
       morg   = int(zorg(i2))
       mat = int(zmat(morg))
       sig0(i1) = xsf2(ih,mat,ig)    ! initial
       sig1(i1) = xsf1(ih,mat,ig)    ! final
      enddo
      enddo
      enddo
      enddo
c
      endif
c
      timep = time
      do ij=1,idtj
      call linter(shape0,shape1,shape,h,x*ij,index)
      if(itn.lt.ilpend) then
       x1 = telaps - h + x*ij
       call pintx(sig0,sig1,sigf,x1,tltot,index)
c
      endif
      power1 = zero
      do 310 ig=1,igm
      do 320 k =1,km
      do 330 j =1,jm
      do 340 i =1,im
       vol = vr(i)*dth(j)*dz(k)
       i1 = i + im*((j-1) + jm*((k-1) + km*(ig-1)))
       power1 = power1 + sigf(i1)*shape(i1)*vol/rnu
340   continue
330   continue
320   continue
310   continue
      power = power1*amp1(ij)
c
c ----------------------------------------
c
      timep = timep + x
       stimep = timep
       spow   = power/pow0
      write(nout,*) stimep,spow
      if(ishort.eq.1) then
       i2 = mod(ij,iskip)
       if(i2.eq.(iskip-1).or.ij.eq.1.or.ij.eq.idtj) then
        write(35,*) stimep,spow
       endif
      else
        write(35,*) stimep,spow
      endif
      enddo
c
c  this is for longer output
      timep2 = time2
      rewind(ninamp)
      do ij=1,idtk
       read(ninamp,*) amp2(ij)
      enddo
      do ij=1,idtk
      call linter(shape0,shape1,shape,h,x2*ij,index)
      if(itn.lt.ilpend) then
       x1 = telaps - h + x2*ij
       call pintx(sig0,sig1,sigf,x1,tltot,index)
      endif
      power1 = zero
      do 410 ig=1,igm
      do 420 k =1,km
      do 430 j =1,jm
      do 440 i =1,im
       vol = vr(i)*dth(j)*dz(k)
       i1 = i + im*((j-1) + jm*((k-1) + km*(ig-1)))
       power1 = power1 + sigf(i1)*shape(i1)*vol/rnu
440   continue
```

```
430   continue
420   continue
410   continue
      power = power1*amp2(ij)
      timep2 = timep2 + x2
       stimep = timep2
       spow   = power/pow0
      write(noup,*) stimep,spow
      enddo
c
c
      time   = timep
      time2  = timep2
c
      close(nout)
      close(noup)
      close(35)
      return
      end
::::::::::::::
powx.f
::::::::::::::
      subroutine powx(power,flux,index,igm,sigf
     $ ,vr,dth,dz,im,jm,km,rnu)
      real*4 power,flux(index),sigf(index),vr(im),dth(jm),dz(km),rnu,vol
      real*4 zero
      zero = 0.e0
c
      power = zero
      do 10 ig=1,igm
      do 20 k =1,km
      do 30 j =1,jm
      do 40 i =1,im
       vol = vr(i)*dth(j)*dz(k)
       i1 = i + im*((j-1) + jm*((k-1) + km*(ig-1)))
       power = power + sigf(i1)*flux(i1)*vol/rnu
40    continue
30    continue
20    continue
10    continue
c
      return
      end
::::::::::::::
qadd.f
::::::::::::::
c234567890123456789012345678901234567890123456789012345678901234567890123456789012
      subroutine qadd(cigx,alambda,mm,lm,im,jm,km,igm,ndg
     $ ,index1,qmom,weight,chid,iqs,ampn)
c     generate distributed source and dist. source moments
c
c     qmom  is moments
c
      real*4 cigx(index1,ndg),alambda(10),chid(100),sum1,sum
     $ ,qmom(im,jm,km,lm,igm),weight(320),ampn
      real*4 zero
      zero = 0.e0
c
      call clearx(qmom,1,index1*lm*igm)
c      write(59,*) '======================'
      do  5 ig=1,igm
      do 15  k=1,km
      do 20  j=1,jm
      do 25  i=1,im
       sum1 = zero
       in = i + im*((j-1) + jm*(k-1))
       do jd=1,ndg
        sum1 = sum1 + alambda(jd)*cigx(in,jd)*chid(ig)
       enddo
       sum1 = sum1 / ampn
c      write(59,*) 'sum1 =',sum1
c   sum1 is a function of position due to cigx
      do 30  l=1,lm
      sum = zero
      do 10  m=1,mm
      if(l.eq.1) sum = sum + sum1*weight(m)
10    continue
c         write(59,*) sum1,sum
      qmom(i,j,k,l,ig) = sum
30    continue
25    continue
20    continue
15    continue
5     continue
c
      return
      end
::::::::::::::
qaddj.f
::::::::::::::
c234567890123456789012345678901234567890123456789012345678901234567890123456789012
      subroutine qaddj(aflux,cigx,vinv,dtnx,alambda,mm,lm
     $ ,im,jm,km,igm,ndg,index1,qmom,weight,coef,chid,iqs,ampn)
c     generate distributed source and dist. source moments
c
c     qmom  is moments
c
      real*4 cigx(index1,ndg),alambda(10),vinv(igm),chid(100),dtnx
     $ ,aflux(im,jm,km,igm,mm),qmom(im,jm,km,lm,igm),coef(lm,mm)
     $ ,weight(320),sum,sum1,vindtn,ampn
```

```
c
c   first lm-1 terms of coef is used
c
      real*4 zero
      zero = 0.e0
c
      call clearx(qmom,1,index1*lm*igm)
c
c      write(59,*) '======================'
      do  5 ig=1,igm
      vindtn = vinv(ig)/dtnx
      do 15  k=1,km
      do 20  j=1,jm
      do 25  i=1,im
       sum1 = zero
       in = i + im*((j-1) + jm*(k-1))
       do jd=1,ndg
        sum1 = sum1 + alambda(jd)*cigx(in,jd)*chid(ig)
       enddo
       sum1 = sum1 / ampn
c      write(59,*) 'sum1 =',sum1
c   sum1 is a function of position due to cigx
      do 30  l=1,lm
      sum = zero
      do 10  m=1,mm
      if(l.gt.1) sum = sum + (      vindtn*aflux(i,j,k,ig,m))
     $                          *weight(m)*coef(l-1,m)
      if(l.eq.1) sum = sum + (sum1 +vindtn*aflux(i,j,k,ig,m))
     $                          *weight(m)
10    continue
      qmom(i,j,k,l,ig) = sum
30    continue
25    continue
20    continue
15    continue
5     continue
c
      return
      end
::::::::::::::
qtrap.f
::::::::::::::
      subroutine qtrap(p,sa,sb,sigfa,sigfb,beta,alambda,a,b
     $ ,res,itm,jlast)
c      calculates the source for one cell and one energy group
c      the result is in res
c      assumes xsecs and shape changes linearly
c
      parameter(eps=1.e-2,jmax=200,jmaxp=jmax+1,k=5,km=k-1)
      real*4 p(itm+1),sa,sb,sigfa,sigfb,beta,alambda
     $ ,a,b,res
      real*4 olds,eps,dtn,resa,resb,funca,funcb,s,tnm,del,x
     $ ,sum,resx,funcx,px,sx,sigfx,dss,h(jmaxp),s(jmaxp)
c
      olds = -1.e30
      h(1) = 1.
      dtn  = b - a
      do j=1,jmax
       if(j.eq.1) then
         resa = beta * exp(-alambda*(dtn-0.))
         resb = beta * exp(-alambda*(0.))
      funca = sigfa*p(1)*sa*resa
      funcb = sigfb*p(1)*sb*resb
      s(j) = .5*(b-a)*(funca+funcb)
      else
      it = 2**(j-2)
      tnm = it
      del = (b-a)/tnm
      x = a + .5*del
      sum = 0.
      do jint=1,it
       resx = beta * exp(-alambda*(dtn-x))
      sigfx = (sigfb-sigfa)*x/(b-a) + sigfa
      call intp(p,px,a,b,x,itm)
      sx = (sb-sa)*x/(b-a) + sa
      funcx = sigfx*px*sx*resx
      sum = sum + funcx
      x = x + del
      enddo
      s(j) = .5*(s(j)+(b-a)*sum/tnm)
      endif
       jlast = j
      if(j.ge.k) then
         call polint(h(j-km),s(j-km),k,0.,ss,dss)
         res = ss
         if(abs(dss).le.eps*abs(ss)) return
      endif
      s(j+1) = s(j)
      h(j+1) = .25*h(j)
      enddo
      stop 'too many steps in qtrap '
      end
c
      subroutine intp(p,px,a,b,x,itm)
      real*4 p(itm+1),px,a,b,x
c
c      find where x falls
       if(x.eq.a) then
        px = p(1)
        return
```

```
        elseif(x.eq.b) then
         px = p(itm+1)
         return
        endif
c
      tm = itm
      dt = (b-a)/tm
        xn = a
      do i=2,itm+1
      xn = xn + dt
      if(x.lt.xn) then
        px = (p(i)-p(i-1))/dt*(x-(xn-dt)) + p(i-1)
        return
      endif
      enddo
c  if it comes here x may be slightly larger than upper limit
c  due to round off
c  set it equal to max
      px = p(itm+1)
      end
c
      subroutine polint(xa,ya,n,x,y,dy)
      real*4 dy,x,y,xa(n),ya(n)
      parameter(nmax=10)
      real*4 den,dif,dift,ho,hp,w,c(nmax),d(nmax)
      ns = 1
      dif = abs(x-xa(1))
      do i=1,n
        dift = abs(x-xa(i))
        if(dift.lt.dif) then
          ns = i
          dif = dift
        endif
        c(i) = ya(i)
        d(i) = ya(i)
      enddo
      y = ya(ns)
      ns = ns-1
      do m=1,n-1
        do i=1,n-m
         ho = xa(i) - x
         hp = xa(i+m) - x
         w = c(i+1) - d(i)
         den = ho - hp
         if(den.eq.0.) stop 'polint failed'
         den = w/den
         d(i) = hp*den
         c(i) = ho*den
        enddo
        if(2*ns.lt.n-m) then
           dy = c(ns+1)
        else
           dy = d(ns)
           ns = ns - 1
        endif
        y = y + dy
      enddo
      return
      end
::::::::::::::
qtstrp0.f
::::::::::::::
c234567890123456789012345678901234567890123456789012345678901234567890123456789012
      subroutine qtstrp0(itm,index,alambda,amptk,ndg,shapn,shapo,index1
     $ ,igm,yx,yen,beta,yold,sigfn,dtmax,jlast,itn,dtnx,itj,idatk)
      real*4 alambda(10),amptk(idatk),shapn(index),shapo(index)
     $ ,yx(index1),yen(index1*ndg),beta(10),yold(index1*ndg)
     $ ,sigfn(index),dtmax
c
      include 'comsdc.F'
c
      real*4 res,a,b,sigfa,sigfb,sa,sb
      real*4 zero
      zero = 0.e0
c   calls qtrap to calcultae delyaed neutron source
c   at the end of time step
c
      call clearx(yen,1,index1*ndg)
c
      a = zero
      b = dtmax
      do id=1,ndg  ! loop over prec. groups
      call clearx(yx,1,index1)
      do ig=1,igm
      do i=1,index1
      i1 = i + index1*(ig-1)
c  calculate shape and cross sections for this cell and group
c       sa = shapo(i1)*ratio(itn)      ! use unnormalized shape
c       sb = shapn(i1)*ratio(itn+1)    ! use unnormalized shape
       sax = shapo(i1)                ! use normalized shape
       sbx = shapn(i1)                ! use normalized shape
       call pint(0.,sax,sbx,sa,0.,dtnx,dtmax*(itj-1))
       call pint(0.,sax,sbx,sb,0.,dtnx,dtmax*itj)
       sigfa = sigfn(i1)    ! fission does not change
       sigfb = sigfn(i1)    ! fission does not change
       res = 0.
       call qtrap(amptk,sa,sb,sigfa,sigfb,beta(id),alambda(id),a,b
     $ ,res,itm,jlast)
       yx(i) = yx(i) + res
       enddo
```

```
      enddo
c
      do i=1,index1
       i2 = i + index1*(id-1)
       yen(i2) = yx(i) + yold(i2)
      enddo
c
      enddo
c
      return
      end
::::::::::::::
qtstrp2.f
::::::::::::::
      subroutine qtstrp2(itm,dt,index,alambda,p,ndg,sigma0,sigma1
     $ ,shapn,shapo,index1,igm,y,yen,beta,yold
     $ ,sigfn,sigfo,ihf,mtm,izm,zorg,zmat,im,jm,km,ihp
     $ ,tx0,tx1,amp0,amp,dtmax,jlast)
      real*4 dt,alambda(10),p(itm)
     $ ,sigma0(ihp,mtm,igm),sigma1(ihp,mtm,igm),shapn(index)
     $ ,shapo(index),y(index1),yen(index1*ndg),beta(10)
     $ ,yold(index1*ndg),sigfn(index),sigfo(index),zorg(index1)
     $ ,zmat(izm),tx0,tx1,amp0,amp(itm+1),dtmax
      real*4 res,a,b,sigfa,sigfb,sa,sb
      real*4 zero
      zero = 0.e0
c   calculates the cross sections at the beginning and end of this
c   time step and calls qtrap to calcultae delyaed neutron source
c   at the end of time step
c
      call clearx(y,1,index1)
      call clearx(yen,1,index1*ndg)
      call clearx(sigfn,1,index)
      call clearx(sigfo,1,index)
      call clearx(amp,1,itm+1)
c
      amp(1) = amp0
      do i=2,itm+1
       amp(i) = p(i-1)
      enddo
c
      do ig=1,igm
      do k =1,km
      do j =1,jm
      do i =1,im
       i1 = i + im*((j-1) + jm*((k-1) + km*(ig-1)))
       i2 = i + im*((j-1) + jm*(k-1))
       morg   = int(zorg(i2))
       mat = int(zmat(morg))
       sigfo(i1) = sigma0(ihf,mat,ig)*(1.-tx0) + sigma1(ihf,mat,ig)*tx0
       sigfn(i1) = sigma0(ihf,mat,ig)*(1.-tx1) + sigma1(ihf,mat,ig)*tx1
      enddo
      enddo
      enddo
      enddo
c
      a = zero
      b = dtmax
      do id=1,ndg  ! loop over prec. groups
      call clearx(y,1,index1)
      do ig=1,igm
      do i=1,index1
      i1 = i + index1*(ig-1)
c  calculate shape and cross sections for this cell and group
      sa = shapo(i1)
      sb = shapn(i1)
      sigfa = sigfo(i1)
      sigfb = sigfn(i1)
      res = 0.
c    integrate over time
       call qtrap(amp,sa,sb,sigfa,sigfb,beta(id),alambda(id),a,b
     $ ,res,itm,jlast)
       y(i) = y(i) + res
       enddo
       enddo
c
       do i=1,index1
        i2 = i + index1*(id-1)
        yen(i2) = y(i) + yold(i2)
       enddo
c
       enddo
c
       return
       end
::::::::::::::
qtstrp6.f
::::::::::::::
       subroutine qtstrp6(itm,dt,index,alambda,p,ndg,sigma0,sigma1
      $ ,shapn,shapo,index1,igm,y,yen,beta,yold,sigma2,mzm,mz
      $ ,sigfn,sigfo,ihf,mtm,izm,zorg,zmat,im,jm,km,ihp
      $ ,tx0,tx1,amp0,amp,dtmax,jlast)
       real*4 dt,alambda(10),p(itm)
      $ ,sigma0(ihp,mtm,igm),sigma1(ihp,mtm,igm),shapn(index)
      $ ,shapo(index),y(index1),yen(index1*ndg),beta(10)
      $ ,yold(index1*ndg),sigfn(index),sigfo(index),zorg(index1)
      $ ,zmat(izm),tx0,tx1,amp0,amp(itm+1),dtmax,sigma2(ihp,mtm,igm)
       dimension mz(10)
       real*4 res,a,b,sigfa,sigfb,sa,sb
       real*4 zero
```

```
      zero = 0.e0
c   calculates the cross sections at the beginning and end of this
c   time step and calls qtrap to calcultae delyaed neutron source
c   at the end of time step
c
      call clearx(y,1,index1)
      call clearx(yen,1,index1*ndg)
      call clearx(sigfn,1,index)
      call clearx(sigfo,1,index)
      call clearx(amp,1,itm+1)
c
      amp(1) = amp0
      do i=2,itm+1
       amp(i) = p(i-1)
      enddo
c
      do ig=1,igm
      do k =1,km
      do j =1,jm
      do i =1,im
       i1 = i + im*((j-1) + jm*((k-1) + km*(ig-1)))
       i2 = i + im*((j-1) + jm*(k-1))
       morg   = int(zorg(i2))
       mat = int(zmat(morg))
       sigfo(i1) = sigma0(ihf,mat,ig)*(1.-tx0) + sigma1(ihf,mat,ig)*tx0
       sigfn(i1) = sigma0(ihf,mat,ig)*(1.-tx1) + sigma1(ihf,mat,ig)*tx1
          do iz=1,mzm
            if(morg.eq.mz(iz)) then
              sigfo(i1) = sigma2(ihf,mat,ig)
              sigfn(i1) = sigma2(ihf,mat,ig)
            endif
          enddo
      enddo
      enddo
      enddo
      enddo
c
      a = zero
      b = dtmax
      do id=1,ndg  ! loop over prec. groups
      call clearx(y,1,index1)
      do ig=1,igm
      do i=1,index1
      i1 = i + index1*(ig-1)
c  calculate shape and cross sections for this cell and group
      sa = shapo(i1)
      sb = shapn(i1)
      sigfa = sigfo(i1)
      sigfb = sigfn(i1)
      res = 0.
c     integrate over time
      call qtrap(amp,sa,sb,sigfa,sigfb,beta(id),alambda(id),a,b
     $ ,res,itm,jlast)
      y(i) = y(i) + res
      enddo
      enddo
c
      do i=1,index1
       i2 = i + index1*(id-1)
       yen(i2) = y(i) + yold(i2)
      enddo
c
      enddo
c
      return
      end
::::::::::::::
rdgip0.f
::::::::::::::
      subroutine rdgip0(igm,iht,ihp,mtm,betat,ampn,dampndt,iqs,izm
     $ ,vinv,dtnx,nts,sigma,imany,ntsigf,index1,zlast,zmat,uxsfp)
      real*4 betat,dtnx,sigma(ihp,mtm,igm),vinv(100)
     $ ,vindtn,ampn,dampndt,term1
     $ ,zlast(index1),zmat(izm),uxsfp(ihp,mtm,igm)
      real*4 zero
      zero = 0.e0
c
c   this routine
c   modifies the total cross section to include the extra terms
c   note that it assumes delayed neutron spectrum is same for all
c   precursor groups
c   and it modifies the final xsecs
c   note that this deletes the old cross section file
      open(nts,form='unformatted',status='unknown')
c
c   first load ss xsecs
c
         call xsecr(sigma,ihp,mtm,igm,ntsigf)
c
c   overwrite absorbtion and total
c
      do 15 ig=1,igm
      do 17 mt=1,mtm
        sigma(1,mt,ig)   = uxsfp(1,mt,ig)
        sigma(iht,mt,ig) = uxsfp(iht,mt,ig)
17    continue
15    continue
c
c   modify total xsecs  and fission xsecs
c
```

```
      term1 = zero
      do 20 ig=1,igm
      if(imany.lt.0) then
       vindtn = zero
      else
       vindtn = vinv(ig)/dtnx
      endif
       term1 = dampndt / ampn * vinv(ig)
      do 25 mt=1,mtm
      sigma(iht,mt,ig)      = sigma(iht,mt,ig) + vindtn + term1
      sigma(iht-1,mt,ig)    = sigma(iht-1,mt,ig)*(1-betat)
25    continue
c
      write(nts) ((sigma(ih,mt,ig),ih=1,ihp),mt=1,mtm)
c
20    continue
      close(nts)
c
      return
      end
:::::::::::::::
rdgip1.f
:::::::::::::::
      subroutine rdgip1(igm,iht,ihp,mtm,betat,ampn,dampndt,iqs,izm
     $ ,vinv,dtnx,nts,sigma,imany,ntsigf,index1,uxsfp,zlast,zmat)
      real*4 betat,dtnx,sigma(ihp,mtm,igm),vinv(100)
     $ ,uxsfp(ihp,mtm,igm),vindtn,ampn,dampndt,term1
     $ ,zlast(index1),zmat(izm)
      real*4 zero
      zero = 0.e0
c
c   this routine
c   modifies the total cross section to include the extra terms
c   note that it assumes delayed neutron spectrum is same for all
c   precursor groups
c   and it modifies the final xsecs
c   note that this deletes the old cross section file
      open(nts,form='unformatted',status='unknown')
c
c   first load ss xsecs
c
        call xsecr(sigma,ihp,mtm,igm,ntsigf)
c
c   overwrite all
c
      do 15 ig=1,igm
      do 17 mt=1,mtm
      do 19 ih=1,ihp
        sigma(ih,mt,ig) = uxsfp(ih,mt,ig)
19    continue
17    continue
15    continue
c
c   modify total xsecs  and fission xsecs
c
      term1 = zero
      do 20 ig=1,igm
      if(imany.lt.0) then
       vindtn = zero
      else
       vindtn = vinv(ig)/dtnx
      endif
       term1 = dampndt / ampn * vinv(ig)
      do 25 mt=1,mtm
      sigma(iht,mt,ig)      = sigma(iht,mt,ig) + vindtn + term1
      sigma(iht-1,mt,ig)    = sigma(iht-1,mt,ig)*(1-betat)
25    continue
c
      write(nts) ((sigma(ih,mt,ig),ih=1,ihp),mt=1,mtm)
c
20    continue
      close(nts)
c
      return
      end
:::::::::::::::
rdgip2.f
:::::::::::::::
      subroutine rdgip2(xsecs,igm,iht,ihp,mtm,betat,ampn,dampndt,iqs
     $ ,vinv,dtnx,nts,sigma,imany)
      real*4 xsecs(ihp,mtm,igm),betat,dtnx,sigma(ihp,mtm,igm),vinv(100)
     $ ,vindtn,ampn,dampndt,term1
      real*4 zero
      zero = 0.e0
c
c   this routine
c   modifies the total cross section to include the extra terms
c   note that it assumes delayed neutron spectrum is same for all
c   precursor groups
c   and it only modifies the original xsecs
c   note that this deletes the old cross section file
      open(nts,form='unformatted',status='unknown')
c   clear single array
c    copy original xsecs onto sigma
      do 15 ig=1,igm
      do 16 mt=1,mtm
      do 17 ih=1,ihp
      sigma(ih,mt,ig) = xsecs(ih,mt,ig)
17    continue
16    continue
```

```
15    continue
c   modify total xsecs  and fission xsecs
      term1 = zero
      do 20 ig=1,igm
      if(imany.lt.0) then
       vindtn = zero
      else
       vindtn = vinv(ig)/dtnx
      endif
       term1 = dampndt / ampn * vinv(ig)
      do 25 mt=1,mtm
      sigma(iht,mt,ig)      = sigma(iht,mt,ig) + vindtn + term1
      sigma(iht-1,mt,ig)    = sigma(iht-1,mt,ig)*(1-betat)
25    continue
c
      write(nts) ((sigma(ih,mt,ig),ih=1,ihp),mt=1,mtm)
c
20    continue
      close(nts)
c
      return
      end
::::::::::::::
rdgip5.f
::::::::::::::
      subroutine rdgip5(xsecs,xsecp,igm,iht,ihp,mtm,betat
     $,ampn,dampndt,iqs,ncrx,vinv,dtnx,ndg,nts,sigma,tx)
      real*4 xsecs(ihp,mtm,igm),sigma(ihp,mtm,igm),xsecp(ihp,mtm,igm)
     $ ,vinv(100),betat,dtnx,tx,vindtn,ampn,dampndt,term1
      real*4 zero
      zero = 0.e0
c
c   this routine calculates the extra terms in BT and
c   modifies the total cross section to include the extra terms
c   note that it assumes delayed neutron spectrum is same for all
c   precursor groups
c   and it only modifies the original xsecs
c
c   note that this deletes the old cross section file
      open(nts,form='unformatted',status='unknown')
c clear single array
c    copy original xsecs onto sigma
      do 15 ig=1,igm
      do 16 mt=1,mtm
      do 17 ih=1,ihp
      sigma(ih,mt,ig) = (1.-tx)*xsecs(ih,mt,ig)+xsecp(ih,mt,ig)*tx
17    continue
16    continue
15    continue
c   modify total xsecs  and fission xsecs
      term1 = zero
      do 20 ig=1,igm
      vindtn = vinv(ig)/dtnx
       term1 = dampndt / ampn * vinv(ig)
      do 25 mt=1,mtm
      sigma(iht,mt,ig)    = sigma(iht,mt,ig) + vindtn + term1
      sigma(iht-1,mt,ig)    = sigma(iht-1,mt,ig)*(1-betat)
25    continue
c
      write(nts) ((sigma(ih,mt,ig),ih=1,ihp),mt=1,mtm)
20    continue
      close(nts)
c
      return
      end
::::::::::::::
rdgip6.f
::::::::::::::
      subroutine rdgip6(xsecs,xsecpo,xsecp,igm,iht,ihp,mtm,betat
     $,mzm,mz,ampn,dampndt,iqs,ncrx,vinv,dtnx,ndg,nts,sigma,tx)
      real*4 xsecs(ihp,mtm,igm),sigma(ihp,mtm,igm),xsecp(ihp,mtm,igm)
     $ ,vinv(100),betat,dtnx,tx,vindtn,ampn,dampndt,term1
     $ ,xsecpo(ihp,mtm,igm)
      dimension mz(10)
      real*4 zero
      zero = 0.e0
c
c   this routine
c   modifies the total cross section to include the extra terms
c   note that it assumes delayed neutron spectrum is same for all
c   precursor groups
c   and it only modifies the original xsecs
c
c   note that this deletes the old cross section file
      open(nts,form='unformatted',status='unknown')
c   clear single array
c    copy original xsecs onto sigma
      do 15 ig=1,igm
      do 16 mt=1,mtm
      do 17 ih=1,ihp
      sigma(ih,mt,ig) = (1.-tx)*xsecs(ih,mt,ig)+xsecp(ih,mt,ig)*tx
        do iz=1,mzm
         if(mt.eq.mz(iz)) then
           sigma(ih,mt,ig) = xsecpo(ih,mt,ig)
         endif
        enddo
17    continue
16    continue
15    continue
c   modify total xsecs  and fission xsecs
```

```
      term1 = zero
      do 20 ig=1,igm
      vindtn = vinv(ig)/dtnx
       term1 = dampndt / ampn * vinv(ig)
      do 25 mt=1,mtm
      sigma(iht,mt,ig)    = sigma(iht,mt,ig) + vindtn + term1
      sigma(iht-1,mt,ig)    = sigma(iht-1,mt,ig)*(1-betat)
25    continue
c
      write(nts) ((sigma(ih,mt,ig),ih=1,ihp),mt=1,mtm)
20    continue
      close(nts)
c
      return
      end
::::::::::::::
read60.f
::::::::::::::
      subroutine read60(j3,l160,n6sc,n6fi)
      dimension l160(*), l(99)
      rewind (n6sc)
      l(61)=1
      l(62)=11
      l(63)=47
      j3=0
      call fidas(l160,l160,l,1,j3,n6sc,n6fi)
      return
      end
::::::::::::::
read64.f
::::::::::::::
      subroutine read64(j3,l164,n6sc,n6fi)
      dimension l164(*), l(99)
      rewind (n6sc)
      l(63)=1
      l(64)=19
      l(65)=26
      j3=0
      call fidas(l164,l164,l,1,j3,n6sc,n6fi)
      return
      end
::::::::::::::
read66.f
::::::::::::::
      subroutine read66(j3)
c
      include 'comsd5.F'
c
      character*80 cny,gak
      dimension  dx(300),l(2)
c
c      to read and change 66**
c
      open(n5in,file='tortinw',status='old')
      open(n6tor,file='dltme1',status='unknown')
      open(n6sc,file='dltme',status='unknown')
c
1     read(n5in,'(a80)',end=100) cny
      if(cny(1:4).eq.'66**'.or.cny(2:5).eq.'66**') then
       write(n6sc,'(a80)') cny
2      read(n5in,'(a80)',end=100) cny
       if((cny(1:4).eq.'61$$'.or.cny(2:5).eq.'61$$').or.
     $    (cny(1:4).eq.'62$$'.or.cny(2:5).eq.'62$$').or.
     $    (cny(1:4).eq.'63$$'.or.cny(2:5).eq.'63$$').or.
     $    (cny(1:4).eq.'64$$'.or.cny(2:5).eq.'64$$').or.
     $    (cny(1:1).eq.'t'.or.cny(2:2).eq.'t').or.
     $    (cny(3:3).eq.'t'.or.cny(4:4).eq.'t')) then
         goto 5
       else
         write(n6sc,'(a80)') cny
         goto 2
       endif
      else
       write(n6tor,'(a80)') cny
       goto 1
      endif
5     write(n6sc,*) 't'
      l(66) = 1
      l(67) = 26
      rewind (n6sc)
      do ii=1,300
       dx(i) = 0.
      enddo
      j3=0
      call fidas(dx,dx,l,1,j3,n6sc,n6fi)
      if(j3.ne.0) goto 101
c     set csolin, csolcn
      dx(4) = 0.5
      dx(8) = 3.
      dx(9) = -3.
      close(n6sc,status='delete')
      open(n6sc,file='dltme',status='unknown')
c     this is for 66**
c     temporaryly write to n6sc (exponential format causes problems b/c of capital E)
      do j=1, 5
       k=5*(j-1)+1
       kmax= k+4
       if(kmax.gt.25) kmax=25
       if(j.eq.1)
     $ write(n6sc,72) (dx(i),i=k,kmax)
```

```
       if(j.ne.1)
     $ write(n6sc,73) (dx(i),i=k,kmax)
       end do
       rewind (n6sc)
c  copy onto input file
       do j=1,5
       read(n6sc,'(a72)') gak
        do ij=1,72
         if(gak(ij:ij).eq.'E') gak(ij:ij) = 'e'
        enddo
       write(n6tor,'(a72)') gak
       enddo
c
c    this is to write the last line read but not written (62$$)
c
       write(n6tor,'(a80)') cny
c
72     format('66** ',1p,5(e12.4,1x))
73     format('     ',1p,5(e12.4,1x))

10     read(n5in,'(a80)',end=15) cny
       write(n6tor,'(a80)') cny
       goto 10
100    write(6,*) 'error: reached the end of file'
       j3=98
       goto 101
15     write(6,*) 'finished modifying input file tortinw'
101    close(n5in)
       close(n6tor)
       close(n6sc,status='delete')
c
c      read the moddified file and write it onto tortinw
c
       open(n6tor,file='tortinw',status='unknown')
       open(n5in,file='dltme1',status='old')
c
80     read(n5in,'(a80)',end=25) cny
       write(n6tor,'(a80)') cny
       goto 80
c
25     close(n6tor)
       close(n5in,status='delete')
       return
       end
:::::::::::::::
read6w.f
:::::::::::::::
       subroutine read6w(j3,l160,n6sc,n6fi)
       dimension l160(*), l(99)
       rewind (n6sc)
       l(61)=1
       l(62)=11
       l(63)=47
       l(64)=65
       j3=0
       call fidas(l160,l160,1,1,j3,n6sc,n6fi)
       return
       end
:::::::::::::::
reader.f
:::::::::::::::
c

       subroutine reader(j3)
c
c      reads the input file that has transient and static information
c      in the format
c
c    =transient
c --------
c --------
c --------
c --------
c    end
c    =tort
c --------
c --------
c --------
c --------
c    end
c
c       and copies first part (transient to end)
c       to file inputs
c       and second part (tort to end)
c       to file tortinf
c       main program will read the transient information from
c       inputs
c       tort will read the static information from
c       tortinf
c
c
c       variables
c       n6sed = output file for transient
c       n6tor = output file for tort
c       n5in  = input file
c
       include 'comsd5.F'
c
       character*1 cizgi(80)
       character*1 cny(80)
```

```
      character*1 cn1,cn2,cn3,cn4,cn5,cn6
      character*1 cm1,cm2,cm3,cm4,cm5,cm6
      character*1 cnsa,cnsq,cneq
      character*1 cnse,cnsn,cnsd
      data cn1, cn2, cn3, cn4, cn5, cn6
     $  /  1ht, 1hr, 1ha, 1hn, 1hs, 1hi  /
      data cm1, cm2, cm3, cm4, cm5, cm6
     $  /  1ht, 1ho, 1hr, 1ht, 1h , 1h   /
c      data cizgi /1h-/
      data  cnsa, cnsq, cneq
     $   /   1h', 1h", 1h=/
      data  cnse, cnsn, cnsd
     $   /   1he, 1hn, 1hd/

      do 1 iic=1,72
       cizgi(iic)="-"
1     continue

       open(n5in,file='inputa',status='old')
       open(n6sed,file='inputs',status='unknown')
       open(n6tor,file='tortinf',status='unknown')
       n6out=n6sed
c
  112 read (n5in,10) cny
   10 format(80a1)
c
      if(cny(1).eq.cnsa) go to 110
c
      if(cny(1).eq.cnsq) go to 110
c
c     first data block name is checked here
c
      if(cny(1).ne.cneq ) go to 114
       if(cny(2).eq.cn1.and.
     $cny(3).eq.cn2.and.
     $cny(4).eq.cn3.and.
     $cny(5).eq.cn4.and.
     $cny(6).eq.cn5.and.
     $cny(7).eq.cn6)  then
       n6out=n6sed
       goto 111
       endif
c
      if(cny(1).ne.cneq ) go to 114
       if(cny(2).eq.cm1.and.
     $ cny(3).eq.cm2.and.
     $ cny(4).eq.cm3.and.
     $ cny(5).eq.cm4)  then
       n6out=n6tor
       goto 121
      endif



110   write(n6out,10) cny
      go to 112
114   write(6,*) "the first column must be = sign"
      write(6,10) cizgi
      write(6,10) cny
      write(6,10) cizgi
      j3=98
      goto 140
c
c - - - - -
111   read(n5in,10) cny
      if(cny(1).eq.cnse.and.cny(2).eq.cnsn.and.cny(3).eq.cnsd) then
       write(6,*) 'finished reading transient input'
       go to 112
      endif
c
      write(n6out,10) cny
      goto 111
c - - - - -
121   read(n5in,10) cny
      if(cny(1).eq.cnse.and.cny(2).eq.cnsn.and.cny(3).eq.cnsd) goto 132
c
      write(n6out,10) cny
      goto 121

132   write(6,*) 'finished reading tort input'
      close (n6sed)
      close (n6tor)
      close (n5in)
140   return
      end
::::::::::::::
reades.f
::::::::::::::
      subroutine reades(j3,ierrfl)
c
      include 'comsd1.F'
      include 'comsdx.F'
      include 'comsd4.F'
      include 'comsd5.F'
      common /irho16/ il6a
c
c      rdnp is the common block name for real delayed neutron parameters
c      idnp is the common block name for integer delayed neutron parameters
c
      common /nacom/ n,ndoub
```

```
      double precision doub
      equivalence ( d(1),doub(1) )
      dimension l(2000) , doub(2000)
      dimension d(2000)
      dimension ldtk(2000)
      character*1 star
      data star/"*"/
c
c      variables
c      alambda = delayed neutron precursor decay constant, 4** array
c      beta    = is beta  , 5** array
c      vinv    = inverse speeds,  6** array
c      chid    = chi for delayed neutrons , 7** array
c      rnu     = nu                    1st entry on 8** array
c      powini  = powini                2nd entry on 8** array
c      srtol   = rtol                  3nd entry on 8** array
c      satol   = atol                  4nd entry on 8** array
c    1$$ array
c       ndg     = number of delayed neutron precursor groups. It should be
c                 in position 1 of 1$$ array.
c       irepmx  = number of shape iterations ; pos 2 of 1$$
c       ixtr    = method of leakage calc. flag; pos 3 of 1$$
c       ixtr    =  '{source-removal} / L{fbar} ---- (0/1) '
c       n6fi    = output unit for fidas messages (def=6); pos 4 of 1$$
c       iout    = output flag ; pos 5 of 1$$
c       ifix    = flag to perform steady-state fixed source ; pos 6 of 1$$
c       idtnm   = number of tn calculations   ; pos 7 of 1$$
c       idtjm   = total number of tj calculations   ; pos 8 of 1$$
c       ioutsh  = output flag for shape ; pos 9 of 1$$
c       iafl    = adjoint flux calc. flag (default = 0 , calculate
c                 and put adjoint flux on default unit 26,
c                 1 = read the flux from unit 26 in varscl format
c                 2 = use flat adjoint flux)
c       iqsf    = method flag (0 adiabatic, 1 IQS) default = 0
c       ipint   = normalization flag for shape (1 calc. int.d shape and apply normalization, 0 don't)
c       iagrev   = adjoint ordering reversal flag
c       iextrp   = extrapolation flag (0 don't,1 extrapolate but if negatives are encountered
c                  don't extrapolate, 2 extrapolate but if negatives are encountered set them
c                  equal to 0.
c       irstrt   = restart flag
c                  It will read the shapes from previously calculated
c                  and stored file shapet, and anshapet and calculate
c                  the amplitude until it reaches the irstrt'th shape
c                  time step
c        end this array with an e
c
c     2$$ array
c       movem   = (0/1/2/3/5/6/9/93/96) linear/linear/step w xsec file/2 steps/
c                         linear w xsec file rod movement/step followed by linear/
c                    do one forward and quit(to determine k $ critical xsecs) for 0,1,2 and 5
c                    do one forward and quit(to determine k $ critical xsecs) for 3
c                    do one forward and quit(to determine k $ critical xsecs) for 6
c       ilpend  = shape calc. number at which lin. pert. ends (movem=5 or 6 only)
c                 shape calc. number at which 2nd step pert. occurs (movem=3 only)
c       ipkmod  = point kinetics mode flag (0 B&G method, 1 Henry method)
c       iqdmod  = delayed neutron source calc. method (0 approximation,
c                                                      1 direct integration)
c       iqs     = method flag (1 = traditional iqs shape calculation
c                              0 = Henry's method flux calculation)
c       ifeed   = doppler feedback (0 no, 1 = yes, )
c       mzm     = number of zones that were step perturbed (movem=6) max 10
c       mz(i),i=1,mzm zone numbers that were step perturbed (movem=6)
c
c        end this array with an e
c
c     9** array
c        dtnx  = tn time intervals (idtnm values)
c    10$$ array
c        idtj  = number of tj calc. per tn calc. (idtnm values)
c    11$$ array
c        idtk  = total number of tk calc. per tn calc. (idtnm values)
c
c    70$ array (movem=0&1)
c        mvmatr  = unit number that has replacing material
c        mvkl    = starting plane   for movement
c        mvku    = ending   plane   for movement
c        mvjl    = starting column  for movement
c        mvju    = ending   column  for movement
c        mvil    = starting row     for movement
c        mviu    = ending   row     for movement
c         end this array with an e
c        reads the inputs file that has transient information
c        and writes the info to file transin
c
c     71* array (movem=0&1)
c        trmst   = time at which rod starts moving (default = 0.0)
c        trmnd   = time at which rod stops moving
c        trmsp   = speed at which rod moves  (if 0.0, calculated from data)
c        ieditn = number of edits of power output (14a2 only)(max=5)
c        tedit(ieditn) = times of power edit
c        end this array with an e
c
c
       limit =    2000
c     clear arrays
       do ic=1,limit
        d(i) =0.
        ldtk(i) = 0
        l(i) = 0
       enddo
```

```
c
c     the foolowing statements are to set the 1$ array size to 1 (l(2)-l(1))
c     and it is stored in d and ldtk array at position 1 (l(1)=1)
c
      l(1) = 1
      l(2) = 21
      l(3) = 50
c
c     l(2) is here just because of the discussion above
c     ll2 is the starting position in the array. Therefore always set to 1.
c
      ll2=1
      open(n5in,file='inputs',status='old')
      open(n6sed,file='transin',status='old',access='append')
c
c     this call is for 1$ and 2$ arrays
c
c - - - - - linkage to fidas input processor
c     ll2    = position in loco of locor for 1$/1* array
c     j3     = error flag,  fidas sets to number of input errors
c     n5     = input  logical unit
c     n6     = output logical unit
c     d      = container array to receive real    data
c     ldtk   = container array to receive integer data
c     loco   = locator array
c     l=loco
c
c    this call is for 1$ and 2$ arrays
c    after this call, d(1) and ldtk(1) will have the first number in
c    1$ array. it is the number of delayed groups for now.
c
      call fidas (d,ldtk,l,ll2,j3,n5in,n6)
c
      if(j3.ne.0) then
        ierrfl = 1
        goto 140
      endif
c
      ndg   = ldtk(1)
      irepmx = ldtk(2)
      ixtr  = ldtk(3)
      n6fi  = ldtk(4)
      iout  = ldtk(5)
      ifix  = ldtk(6)
      idtnm = ldtk(7)
      idtjm = ldtk(8)
      il6a  = 4
      if(idtjm.lt.1000) il6a = 4
      if(idtjm.eq.1000) il6a = 1
      if(idtjm.eq.2000) il6a = 2
      if(idtjm.eq.3000) il6a = 3
      if(idtjm.eq.4000) il6a = 4
      ioutsh= ldtk(9)
      iafl  = ldtk(10)
      iffl  = ldtk(11)
      ipint = ldtk(12)
      iagrev= ldtk(13)
      iextrp= ldtk(14)
      irstrt= ldtk(15)
c
      movem = ldtk(21)
      ilpend= ldtk(22)
      ipkmod= ldtk(23)
      iqdmod= ldtk(24)
      iqs   = ldtk(25)
      ifeed = ldtk(26)
      mzm   = ldtk(27)
      if(mzm.gt.10) stop 'error : reades .. mzm >10'
      do i=1,mzm
        mz(i) = ldtk(27+i)
      enddo
c234567890123456789012345678901234567890123456789012345678901234567890123456789012
c
      i = 1
      write(n6sed,*)
      write(n6sed,900) (star,j=1,103)
      write(n6sed,910) (star,j=1,45),i,(star,j=1,45)
      write(n6sed,900) (star,j=1,103)
      write(n6sed,*)
      write(n6   ,*)
      write(n6   ,900) (star,j=1,103)
      write(n6   ,910) (star,j=1,45),i,(star,j=1,45)
      write(n6   ,900) (star,j=1,103)
      write(n6   ,*)
c
      write(n6sed,*) 'number of precursor groups            = ',ndg
      write(n6   ,*) 'number of precursor groups            = ',ndg
      write(n6sed,*) 'number of updates over shape interval = ',irepmx
      write(n6   ,*) 'number of updates over shape interval = ',irepmx
      write(n6sed,*) 'not used (enter 0)                    = ',ixtr
      write(n6   ,*) 'not used (enter 0)                    = ',ixtr
      write(n6sed,*) 'output unit for fidas messages        = ',n6fi
      write(n6   ,*) 'output unit for fidas messages        = ',n6fi
      write(n6sed,*) 'output flag                           = ',iout
     $ ,' (0=short,1=long,2=longer,3=all)'
      write(n6   ,*) 'output flag                           = ',iout
     $ ,' (0=short,1=long,2=longer,3=all)'
      write(n6sed,*) 'steady state fixed source flag        = ',ifix
      write(n6   ,*) 'steady state fixed source flag        = ',ifix
      write(n6sed,*) 'number of shape calculations          = ',idtnm
```

```
      write(n6   ,*) 'number of shape calculations          = ',idtnm
      write(n6sed,*) 'not used (enter 0)                    = ',idtjm
      write(n6   ,*) 'not used (enter 0)                    = ',idtjm
      write(n6sed,*) 'output flag for shape                 = ',ioutsh
     $ ,' (0= no,1= write shape to a file after each shape calculation)'
      write(n6   ,*) 'output flag for shape                 = ',ioutsh
     $ ,' (0= no,1= write shape to a file after each shape calculation)'
      write(n6sed,*) 'adjoint calculation flag              = ',iafl
     + ,'  (0 calculate,'
      write(n6sed,*) '                                        1 adjoint
     +flux is at unit 26(tort output varscl format),'
      write(n6sed,*) '                                        2 adjoint
     +flux is at unit 39(i,f format),'
      write(n6sed,*) '                                        3 use flat
     + adjoint flux ))'
      write(n6   ,*) 'adjoint calculation flag              = ',iafl
     + ,'  (0 calculate,'
      write(n6   ,*) '                                        1 adjoint
     +flux is at unit 26(tort output varscl format),'
      write(n6   ,*) '                                        2 adjoint
     +flux is at unit 39(i,f format),'
      write(n6   ,*) '                                        3 use flat
     + adjoint flux ))'
      write(n6sed,*) 'forward calculation flag              = ',iffl
     + ,'  (0 calculate,'
      write(n6sed,*) '                                        1 forward
     +flux is at unit 27(tort output varscl format))'
      write(n6   ,*) 'forward calculation flag              = ',iffl
     + ,'  (0 calculate,'
      write(n6   ,*) '                                        1 forward
     +flux is at unit 27(tort output varscl format))'
      write(n6sed,*) "shape flag                            = ",ipint
     + ,"  (0 don't renormalize the shape,"
      write(n6sed,*) "                                        1 do)"
      write(n6   ,*) 'shape flag                            = ',ipint
     + ,"  (0 don't renormalize the shape,"
      write(n6   ,*) "                                        1 do)"
      write(n6sed,*) "adjoint energy group flag             = ",iagrev
     + ,"  (0 don't do anything,"
      write(n6sed,*) "                                        1 reverse
     +group ordering)"
      write(n6   ,*) "adjoint energy group flag             = ",iagrev
     + ,"  (0 don't do anything,"
      write(n6   ,*) "                                        1 reverse
     +group ordering)"
      write(n6sed,*) "Extrapolation flag                    = ",iextrp
     + ," (0 don't,"
      write(n6sed,*) "                                         1 extrapola
     +te but if negatives are encountered don't extrapolate,"
      write(n6sed,*) "                                         2 extrapola
     +te but if negatives are encountered set them equal to 0)"
      write(n6   ,*) "Extrapolation flag                    = ",iextrp
     + ," (0 don't,"
      write(n6   ,*) "                                         1 extrapola
     +te but if negatives are encountered don't extrapolate,"
      write(n6   ,*) "                                         2 extrapola
     +te but if negatives are encountered set them equal to 0)"
      write(n6sed,*) "Restart flag                          = ",irstrt
      write(n6   ,*) "Restart flag                          = ",irstrt
c
      write(n6sed,*) "perturbation type                     = ",movem
     + ," (0  rod movement with feedback,"
      write(n6sed,'(46x,28h1  rod movement w/o feedback)')
      write(n6sed,'(46x,31h2  step with cross section file)')
      write(n6sed,'(46x,10h3  2 steps)')
      write(n6sed,'(46x,33h5  linear with cross section file)')
      write(n6sed,'(46x,26h6  step followed by linear)')
      write(n6sed,'(46x,41h92 do one forward to determine k_eff and
     +,23hcritical cross sections,/,49x,26hand quit (for 0,1,2 and 5))')
      write(n6sed,'(46x,41h93 do one forward to determine k_eff and
     +,23hcritical cross sections,/,49x,16hand quit (for 3))')
      write(n6sed,'(46x,41h96 do one forward to determine k_eff and
     +,23hcritical cross sections,/,49x,16hand quit (for 6))')
      write(n6   ,*) "perturbation type                     = ",movem
     + ," (0  rod movement with feedback,"
      write(n6   ,'(46x,28h1  rod movement w/o feedback)')
      write(n6   ,'(46x,31h2  step with cross section file)')
      write(n6   ,'(46x,10h3  2 steps)')
      write(n6   ,'(46x,33h5  linear with cross section file)')
      write(n6   ,'(46x,26h6  step followed by linear)')
      write(n6   ,'(46x,41h92 do one forward to determine k_eff and
     +,23hcritical cross sections,/,49x,26hand quit (for 0,1,2 and 5))')
      write(n6   ,'(46x,41h93 do one forward to determine k_eff and
     +,23hcritical cross sections,/,49x,16hand quit (for 3))')
      write(n6   ,'(46x,41h96 do one forward to determine k_eff and
     +,23hcritical cross sections,/,49x,16hand quit (for 6))')
      if(movem.eq.3) then
      write(n6sed,*)"Second step perturbation occurs at time step "
     $ ,ilpend
      write(n6   ,*)"Second step perturbation occurs at time step "
     $ ,ilpend
      elseif(movem.eq.5.or.movem.eq.6) then
      write(n6sed,*)"Ramp perturbation ends at time step ",ilpend
      write(n6   ,*)"Ramp perturbation ends at time step ",ilpend
      else
      write(n6sed,*) 'not used (enter 0)                    = ',ilpend
      write(n6   ,*) 'not used (enter 0)                    = ',ilpend
      endif
      write(n6sed,*) 'Point kinetics method                 = ',ipkmod
     + ,' (0 B&G, 1 Henry)'
```

```
      write(n6   ,*) 'Point kinetics method                 = ',ipkmod
     + ,' (0 B&G, 1 Henry)'
      write(n6sed,*) 'Delayed neutron source calc. method   = ',iqdmod,
     $' (0 sedat, 1 direct integration)'
      write(n6   ,*) 'Delayed neutron source calc. method   = ',iqdmod,
     $' (0 sedat, 1 direct integration)'
      write(n6sed,*) 'Method flag                           = ',iqs
     $ ,' (1 = shape calculation'
      write(n6sed,*)'                                             0 = fl
     +ux calculation)'
      write(n6   ,*) 'Method flag                           = ',iqs
     $ ,' (1 = shape calculation'
      write(n6   ,*)'                                             0 = fl
     +ux calculation)'
      write(n6sed,*) 'Doppler feedback flag                 = ',ifeed
     + ,' (0=no, 1=yes)'
      write(n6   ,*) 'Doppler feedback flag                 = ',ifeed
     + ,' (0=no, 1=yes)'
      if(movem.eq.6) then
      write(n6sed,*) 'Number of zones perturbed with step pert. = ',mzm
      write(n6   ,*) 'Number of zones perturbed with step pert. = ',mzm
      write(n6sed,*) 'Zone numbers that are perturbed during step pert.'
      write(n6   ,*) 'Zone numbers that are perturbed during step pert.'
       do i=1,mzm
        write(n6sed,*) mz(i)
        write(n6   ,*) mz(i)
       enddo
      endif
c
c     the following statements put lambdas in d array from positions 1 (n=1)
c     to ldtk(1), betas in d array from positions ldtk(1)+1 to
c     ldtk(1) + ldtk(1), vinv'd are in d array from 2(ldtk(1))+1
c     to 2(ldtk(1)) + igmw
c     length of lambda array is ndg=ldtk(1)
c     length of beta array is ndg=ldtk(1)
c     length of vinv array is igmw
c     length of chid array is igmw
c
      ndoub = 1
      n     = 1
      llambda     =na( ndg )
      lbeta       =na( ndg )
      lvinv       =na( igmw )
      lchid       =na( igmw )
      lgak        =na( 4 )
      ltimen      =na( idtnm )
      lend = n
      if(lend.gt.limit) stop16
      l(4) = llambda
      l(5) = lbeta
      l(6) = lvinv
      l(7) = lchid
      l(8) = lgak
      l(9) = ltimen
      l(10) = lend
c     this reads the second block
c     clear arrays
      do ic=1,limit
       d(ic) =0.
      enddo
c
      call fidas(d,d,1,1,j3,n5in,n6fi)
c
      if(j3.ne.0) then
        ierrfl = 2
        goto 140
      endif
c
c
      i = 2
      write(n6sed,*)
      write(n6sed,900) (star,j=1,103)
      write(n6sed,910) (star,j=1,45),i,(star,j=1,45)
      write(n6sed,900) (star,j=1,103)
      write(n6sed,*)
      write(n6   ,*)
      write(n6   ,900) (star,j=1,103)
      write(n6   ,910) (star,j=1,45),i,(star,j=1,45)
      write(n6   ,900) (star,j=1,103)
      write(n6   ,*)
c
c
      do  i=1, igmw
       vinv(i)= d(2*ndg+i)
      end do
c
      do  i=1, igmw
       chid(i)= d(2*ndg+igmw+i)
      end do
c
      lpoint = 2*(ndg+igmw)+1
      rnu    = d(lpoint)
      powini = d(lpoint+1)
      srtol  = d(lpoint+2)
      satol  = d(lpoint+3)
c
c     save times in dtns
c
      do iin=1,idtnm
        dtns(iin) = d(lpoint+3+iin)
```

```
      enddo
c
      dtn(1) = d(lpoint+3+1)
      istop = 0
      do iin=2,idtnm
      dtn(iin)  = d(lpoint+3+iin)-d(lpoint+3+iin-1)
       if(dtn(iin).lt.0.) then
       write(n6sed,*) '************************************************'
       write(n6sed,*) 'ERROR (Reades) : time intervals are not positive'
       write(n6sed,*) '                 for ',iin
       write(n6sed,*) '************************************************'
       write(n6,*) '************************************************'
       write(n6,*) 'ERROR (Reades) : time intervals are not positive'
       write(n6,*) '                 for ',iin
       write(n6,*) '************************************************'
        istop = 1
       endif
      enddo
      write(n6sed,*)
      write(n6   ,*)
      write(n6sed,*) '    lambda         beta'
      write(n6sed,*) '---------------------------'
      write(n6   ,*) '    lambda         beta'
      write(n6   ,*) '---------------------------'
c
      betat = 0.0
      do i = 1, ndg
       alambda(i) = d(i)
       beta(i)    = d(ndg+i)
       betat      = betat + beta(i)
       write(n6sed,'(1p,2(1x,e12.6,2x))') alambda(i), beta(i)
       write(n6   ,'(1p,2(1x,e12.6,2x))') alambda(i), beta(i)
      end do
       write(n6sed,'((1x,13htotal beta = ,1pe12.6,2x))') betat
       write(n6   ,'((1x,13htotal beta = ,1pe12.6,2x))') betat
      write(n6sed,*)
      write(n6   ,*)
c
      write(n6sed,*) ' inverse speeds for ',igmw,' groups follows'
      jmax=igmw/11
      if(jmax*11.ne.igmw) jmax=jmax+1
      do j=1, jmax
       k=11*(j-1)+1
       kmax= k+10
       if(kmax.gt.igmw) kmax=igmw
       write(n6sed,'(1x,11(1pe12.5,1x))') (vinv(i),i=k,kmax)
      end do
      write(n6,*) ' inverse speeds for ',igmw,' groups follows'
      do j=1, jmax
       k=11*(j-1)+1
       kmax= k+10
       if(kmax.gt.igmw) kmax=igmw
       write(n6,'(1x,11(1pe12.5,1x))') (vinv(i),i=k,kmax)
      end do
c
      write(n6sed,*)
      write(n6   ,*)
      write(n6sed,*) ' nu      = ',rnu
      write(n6   ,*) ' nu      = ',rnu
      write(n6sed,*) ' powini  = ',powini,' (Reactor power is normalized
     + to number if not zero)'
       write(n6   ,*) ' powini  = ',powini,' (Reactor power is normalized
     + to number if not zero)'
       write(n6sed,*) ' rtol    = ',srtol
       write(n6   ,*) ' rtol    = ',srtol
       write(n6sed,*) ' atol    = ',satol
       write(n6   ,*) ' atol    = ',satol
       write(n6sed,*)
       write(n6   ,*)
c      write(n6sed,*)'The shape calculations will be performed at times:'
c      write(n6   ,*)'The shape calculations will be performed at times:'
c      write(n6sed,920) (dtns(i),i=1,idtnm)
c      write(n6   ,920) (dtns(i),i=1,idtnm)
c920   format(1p,6(1x,e13.5,1x))
c      write(n6sed,*)
c      write(n6   ,*)
c
      if(istop.eq.1) stop
c
      n = 1
      ndoub = 1
      l12   = 1
      do iiz=1,20
      ldtk(iiz) = 0
      enddo
      l(10) = 1
      l(11) = 1+idtnm
      l(12) = 1+2*idtnm
c   this reads the third block
      call fidas (d,ldtk,l,l12,j3,n5in,n6fi)
c
      if(j3.ne.0) then
        ierrfl = 3
        goto 140
      endif
c
      do iin=1,idtnm
       idtj(iin)    = ldtk(iin)
       idtk(iin)    = ldtk(iin+idtnm)
      enddo
```

```
c
      i = 3
      write(n6sed,*)
      write(n6sed,900) (star,j=1,103)
      write(n6sed,910) (star,j=1,45),i,(star,j=1,45)
      write(n6sed,900) (star,j=1,103)
      write(n6sed,*)
      write(n6   ,*)
      write(n6   ,900) (star,j=1,103)
      write(n6   ,910) (star,j=1,45),i,(star,j=1,45)
      write(n6   ,900) (star,j=1,103)
      write(n6   ,*)
c
      write(n6sed,*)'          time       tn time     min. time    '
     $,'total tj    total tk '
      write(n6sed,*)'                    interval      interval      '
     $,'calc.       calc.'
      write(n6sed,*)('-',ix=1,70)
      write(n6   ,*)'          time       tn time     min. time    '
     $,'total tj    total tk '
      write(n6   ,*)'                    interval      interval      '
     $,'calc.       calc.'
      write(n6   ,*)('-',ix=1,70)
      do iin=1,idtnm
      write(n6sed,'(1x,i3,1x,1pe13.5,2x,e11.5,2x,e11.5,4x,i5,6x,i5)')
     $ iin,dtns(iin),dtn(iin),dtn(iin)/idtk(iin),idtj(iin),idtk(iin)
      write(n6   ,'(1x,i3,1x,1pe13.5,2x,e11.5,2x,e11.5,4x,i5,6x,i5)')
     $ iin,dtns(iin),dtn(iin),dtn(iin)/idtk(iin),idtj(iin),idtk(iin)
      enddo
c     this reads the fourth block if required
c
      if(movem.eq.0.or.movem.eq.1) then
      n = 1
      ndoub = 1
      l12  = 1
      do iiz=1,20
      ldtk(iiz) = 0
      enddo
      do iiz=1,40
      d(iiz) = 0.0
      enddo
      l(70) = 1
      l(71) = 21
      l(72) = 31
c   this reads the fourth block
      call fidas (d,ldtk,1,l12,j3,n5in,n6fi)
c
      if(j3.ne.0) then
        ierrfl = 4
        goto 140
      endif
c
c
      i = 4
      write(n6sed,*)
      write(n6sed,900) (star,j=1,103)
      write(n6sed,910) (star,j=1,45),i,(star,j=1,45)
      write(n6sed,900) (star,j=1,103)
      write(n6sed,*)
      write(n6   ,*)
      write(n6   ,900) (star,j=1,103)
      write(n6   ,910) (star,j=1,45),i,(star,j=1,45)
      write(n6   ,900) (star,j=1,103)
      write(n6   ,*)
c
      mvmatr= ldtk(1)
      mvkl  = ldtk(2)
      mvku  = ldtk(3)
      mvjl  = ldtk(4)
      mvju  = ldtk(5)
      mvil  = ldtk(6)
      mviu  = ldtk(7)
c
      trmst = d(21)
      trmnd = d(22)
      trmsp = d(23)
      iediten = int(d(24))
      do ix=1,iediten
       tedit(ix) = d(24+ix)
      enddo
c
      if(mvku.eq.mvkl) then
      write(n6sed,*) 'no rod movement has been specified!'
      write(n6   ,*) 'no rod movement has been specified!'
      else
      write(n6sed,*) 'rod will be moved from k= ',mvkl,'  to k= ',mvku
      write(n6sed,*) '                        from j= ',mvjl,'  to j= ',mvju
      write(n6sed,*) '                        from i= ',mvil,'  to i= ',mviu
      write(n6sed,*) 'the replacing material is read from unit ', mvmatr
      write(n6   ,*) 'rod will be moved from k= ',mvkl,'  to k= ',mvku
      write(n6   ,*) '                        from j= ',mvjl,'  to j= ',mvju
      write(n6   ,*) '                        from i= ',mvil,'  to i= ',mviu
      write(n6   ,*) 'the replacing material is read from unit ', mvmatr
      endif
      write(n6sed,*) 'time at which rod starts moving = ',trmst
     $ ,' (default = 0.0)'
      write(n6sed,*) 'time at which rod stops moving  = ',trmnd
      write(n6sed,*) 'speed at which rod moves        = ',trmsp
     $ ,' (if 0.0, calculated from data)'
      write(n6   ,*) 'time at which rod starts moving = ',trmst
```

```
     $ ,' (default = 0.0)'
      write(n6   ,*) 'time at which rod stops moving  = ',trmnd
      write(n6   ,*) 'speed at which rod moves        = ',trmsp
     $ ,' (if 0.0, calculated from data)'
      write(n6sed,*) 'Number of power edits           = ',ieditn
      write(n6   ,*) 'Number of power edits           = ',ieditn
      if(ieditn.gt.0) then
        write(n6sed,*) 'Power edit times are            = '
        write(n6   ,*) 'Power edit times are            = '
        write(n6sed,'(1x,1p,5e13.5)') (tedit(ix),ix=1,ieditn)
        write(n6   ,'(1x,1p,5e13.5)') (tedit(ix),ix=1,ieditn)
      endif
c
      endif
140   close(n5in)
      close(n6sed)
900   format(1x,103(a1))
910   format(1x,45(a1),1x,10hB L O C K ,i1,1x,45(a1))
      return
      end
::::::::::::::
readet.f
::::::::::::::
c
      subroutine readet(j3,ifora)
c
      include 'comsd4.F'
      include 'comsd5.F'
c
      character*80 cny
      dimension l160(99)
c
c     to read and change 61$$ and 62$$ arrays
c
c     variables
c     l160(1:10)    = 61$$ array (length >10 but only 10 used )
c     l160(11:46)   = 62$$ array (length >36 but only 36 used )
c
c     if ifora=0
c     reads the tortinf file that has tort information
c     calls routines to modify the input file for adjoint calculation
c     also calls routines to extract fission spectrum from
c     tort input file
c     and writes the info to file transin, and tortina.
c
c     if ifora=1
c     reads the tortinf file that has tort information
c     calls routines to modify the input file same as adjoint calculation
c
c     reset the l160 array
      do i=1,99
       l160(i)=0
      end do
      if(ifora.eq.0) then
        open(n5in,file='tortinf',status='old')
c        open(n6sed,file='transin',status='unknown')
        open(n6tor,file='tortina',status='unknown')
        open(n6sc,file='dltme',status='unknown')
      elseif(ifora.eq.1) then
        open(n5in,file='tortinf',status='old')
        open(n6tor,file='dltme1',status='unknown')
        open(n6sc,file='dltme',status='unknown')
      endif
c
1     read(n5in,'(a80)',end=100) cny
      if(cny(1:4).eq.'61$$'.or.cny(2:5).eq.'61$$') then
       write(n6sc,'(a80)') cny
2      read(n5in,'(a80)',end=100) cny
       if(cny(1:4).eq.'63$$'.or.cny(2:5).eq.'63$$') then
         goto 5
         else
         write(n6sc,'(a80)') cny
         goto 2
       endif
      else
       write(n6tor,'(a80)') cny
       goto 1
      endif
5     write(n6sc,*) 't'
      call read60(j3,l160,n6sc,n6fi)
      if(j3.ne.0) goto 101
      close (n6sc, status='delete')
c
      if(l160(21).ne.0) then
       write(n6,*) 'a forward calculation is not specified in input'
       write(n6,*) ' terminating execution '
       j3=99
       goto 101
      endif
c
      nscttw   = l160(20)
c
c      set output units and other control integers
c
      l160(2)  = ntfogf
      l160(14) = ifxprtw
      l160(15) = modepw
      if(ifora.eq.0) then
        l160(2)  = ntfoga
        l160(3)  = ntsiga
```

```
c        l160(5)  = 0
         l160(8)  = ntscla
         l160(10) = ntdira
         l160(21) = 1
       else
         l160(2)  = ntfogf
         l160(3)  = ntsigf
         l160(8)  = ntsclf
         l160(10) = ntdirf
       endif
c
c
       write(n6tor,30) (l160(i),i=1,10)
c      the following is needed to end the card
       write(n6tor,*) ' e'
c
       write(n6tor,20) (l160(i),i=11,46)
c      the following is needed to end the card
       write(n6tor,*) ' e'
c
30     format('61$$',10(1x,i4))
20     format('62$$',10(1x,i4)/
      1           4x,10(1x,i4)/
      2           4x,10(1x,i4)/
      3           4x,10(1x,i4))
c
c      this is to write the last line read but not written (63$$)
c
       write(n6tor,'(a80)') cny
10     read(n5in,'(a80)',end=15) cny
c       if(ifora.eq.0) then
c       if(cny(1:4).eq.'91**'.or.cny(2:5).eq.'91**'.
c      $or.cny(1:4).eq.'92**'.or.cny(2:5).eq.'92**'.
c      $or.cny(1:4).eq.'93**'.or.cny(2:5).eq.'93**'.
c      $or.cny(1:4).eq.'94**'.or.cny(2:5).eq.'94**'.
c      $or.cny(1:4).eq.'95**'.or.cny(2:5).eq.'95**'.
c      $or.cny(1:4).eq.'96**'.or.cny(2:5).eq.'96**') goto 10
c       endif
       write(n6tor,'(a80)') cny
       goto 10
100    write(n6,*) 'error: reached the end of file'
       j3=98
       goto 101
15     write(n6,*) 'finished '
101    if(ifora.eq.0) then
        close(n5in)
        close(n6sed)
        close(n6tor)
        close(n6sc)
       elseif(ifora.eq.1) then
        close(n5in)
        close(n6tor)
        close(n6sc)
         open(n6tor,file='tortinf',status='unknown')
         open(n5in,file='dltme1',status='old')
24     read(n5in,'(a80)',end=25) cny
       write(n6tor,'(a80)') cny
       goto 24
25     close(n6tor)
       close(n5in,status='delete')
       endif
       return
       end
::::::::::::::
readev.f
::::::::::::::
c
       subroutine readev(j3)
c
       include 'comsd1.F'
       include 'comsd4.F'
       include 'comsd5.F'
c
       character*80 cny
       dimension l164(25)
c
c      variables
c      l164(1:18)    = 63$$ array
c      l164(19:25)   = 64$$ array
c      im, jm, km are same as in tort
c
c      reads the tortinf file (63$$ and 64$$) that has tort information
c      calls routines to find number of neutron energy groups from
c      tort input file
c      and writes the info to file transin.
c
       open(n5in,file='tortinf',status='old')
       open(n6sed,file='transin',status='old',access='append')
       open(n6sc,file='dltme',status='unknown')
c
1      read(n5in,'(a80)',end=100) cny
       if(cny(1:4).eq.'63$$'.or.cny(2:5).eq.'63$$') then
        write(n6sc,'(a80)') cny
2       read(n5in,'(a80)',end=100) cny
        if(cny(1:4).eq.'66**'.or.cny(2:5).eq.'66**') then
          goto 5
        else
          write(n6sc,'(a80)') cny
          goto 2
        endif
```

```
      else
c        write(n6tor,'(a80)') cny
         goto 1
      endif
5     write(n6sc,*) 't'
      call read64(j3,ll64,n6sc,n6fi)
      if(j3.ne.0) goto 101
      close (n6sc, status='delete')
c
c
      write(n6sed,*) 'number of neutron energy groups = ',ll64(19)
      write(n6,*) 'number of neutron energy groups = ',ll64(19)
c
c     this is to set number of energy groups (igmw) variable,
c     and also other variables that are used to calculate x-sec array length
      igmw   = ll64(19)
      ncrxw  = ll64(20)
      nsctmw = ll64(21)
       lmw   = (nsctmw+1)*(nsctmw+1)
       if(nscttw.ge.0) lmw = (nscttw+1)*(nscttw+1)
      ihtw   = ll64(22)
      ihmw   = ll64(23)
      iupsw  = ll64(24)
      ihsw   = iupsw + ihtw + 1
c
      imw  = ll64(1)
      jmw  = ll64(2)
      kmw  = ll64(3)
      izmw = ll64(11)
      mmw  = ll64(13)
c
      write(n6sed,*) 'number of x intervals          = ',ll64(1)
      write(n6,*) 'number of x intervals          = ',ll64(1)
      write(n6sed,*) 'number of y intervals          = ',ll64(2)
      write(n6,*) 'number of y intervals          = ',ll64(2)
      write(n6sed,*) 'number of z intervals          = ',ll64(3)
      write(n6,*) 'number of z intervals          = ',ll64(3)
      write(n6sed,*) 'number of material zones       = ',ll64(11)
      write(n6,*) 'number of material zones       = ',ll64(11)
c
      goto 15
c
c
100   write(n6,*) 'error: reached the end of file'
      j3=98
      goto 101
15    write(n6,*) 'finished '
101   close(n5in)
      close(n6sed)
      return
      end
::::::::::::::
readw.f
::::::::::::::
c
c
      subroutine readw(j3,imany)
c
      include 'comsd5.F'
c
      character*80 cny
      dimension llw(99)
c
c      to read and change 61$$ and 63$$ arrays
c
c      variables
c      llw( 1:10)    = 61$$ array (length >10 but only 10 used )
c      llw(11:46)    = 62$$ array (length >36 but only 36 used )
c      llw(47:64)    = 63$$ array (length >18 but only 18 used )
c
c      reads the tortinf file that has tort information
c      calls routines to modify the input file for zone map reading
c      from a file
c
c      reset the llw array
      do i=1,99
       llw(i)=0
      end do
      open(n5in,file='tortinf',status='old')
      open(n6tor,file='dltme1',status='unknown')
      open(n6sc,file='dltme',status='unknown')
c
1     read(n5in,'(a80)',end=100) cny
      if(cny(1:4).eq.'61$$'.or.cny(2:5).eq.'61$$') then
       write(n6sc,'(a80)') cny
2      read(n5in,'(a80)',end=100) cny
       if(cny(1:4).eq.'64$$'.or.cny(2:5).eq.'64$$') then
         goto 5
       else
         write(n6sc,'(a80)') cny
         goto 2
       endif
      else
       write(n6tor,'(a80)') cny
       goto 1
      endif
5     write(n6sc,*) 't'
      call read6w(j3,llw,n6sc,n6fi)
      if(j3.ne.0) goto 101
      close (n6sc, status='delete')
```

```
c
      llw(1)  = 0
      llw(2)  = ntfogf
c     set ktype for fixed source
      llw(19) = 0
c
      write(n6tor,30) (llw(i),i=1,10)
c     the following is needed to end the card
      write(n6tor,*) ' e'
c
30    format('61$$',10(1x,i4))
c
      write(n6tor,31) (llw(i),i=11,46)
c     the following is needed to end the card
      write(n6tor,*) ' e'
c
31    format('62$$',10(1x,i4)/
     1           4x,10(1x,i4)/
     2           4x,10(1x,i4)/
     3           4x, 6(1x,i4))
c
      write(n6tor,32) (llw(i),i=47,64)
c     the following is needed to end the card
      write(n6tor,*) ' e'
c
32    format('63$$',10(1x,i4)/
     1           4x, 8(1x,i4))
c
c
c     this is to write the last line read but not written (64$$)
c
      write(n6tor,'(a80)') cny
10    read(n5in,'(a80)',end=15) cny
      write(n6tor,'(a80)') cny
      goto 10
100   write(n6,*) 'error: reached the end of file'
      j3=98
      goto 101
15    write(n6,*) 'finished preparing input files!'
101   close(n5in)
      close(n6tor)
      close(n6sc,status='delete')
      open(n6tor,file='tortinw',status='unknown')
      open(n5in,file='dltme1',status='old')
24    read(n5in,'(a80)',end=25) cny
      write(n6tor,'(a80)') cny
      goto 24
25    close(n6tor)
      close(n5in,status='delete')
      return
      end
::::::::::::::
rhocal.f
::::::::::::::
      subroutine rhocal(rho,fterm,shape,ansh,rhot3,rhot4,rhot5,anadj
     $,xstd,scadj,xssd,dume1,xsfd,chit,dume2,vr,dth,dz,dume3,xsfp,dume4
     $ ,index,index1,indexm,im,jm,km,igm,mm,ipkmod)
c
c     rho is rho(t)
c     fterm is F(t)
c
c23456789012345678901234567890123456789012345678901234567890123456789012
c
      real*4 rho,fterm,shape(index),ansh(indexm),rhot3(index)
     $ ,rhot4(index),rhot5(index),anadj(indexm),xstd(index),scadj(index)
     $ ,xssd(index*igm),dume1(index1),xsfd(index),chit(igm)
     $ ,dume2(index),vr(im),dth(jm),dz(km),dume3(index1),xsfp(index)
     $ ,dume4(index),res,result
c
      common sweight(320)
      real*4 sweight
c
      call clearx( rhot3,1,index)
      call clearx( rhot4,1,index)
      call clearx( rhot5,1,index)
      call clearx( dume1,1,index1)
      call clearx( dume2,1,index)
      call clearx( dume3,1,index1)
      call clearx( dume4,1,index)
c
c     calculate third term in rho
      if(ipkmod.eq.0) then
       call rhot3r(rhot3,anadj,ansh,index,xstd,mm,sweight)
      else
       call rhot3h(rhot3,scadj,shape,index,xstd)
      endif
c
c     calculate fourth term in rho
      call rhot4r(rhot4,scadj,shape,index,index1,igm,xssd)
c
c     calculate fifth term in rho
      call rhot5r(rhot5,dume1,scadj,shape,index,index1,igm,xsfd,chit)
c
c     load the function to be integrated onto dume2
      do 90 i =1, index
       dume2(i) = - rhot3(i) + rhot4(i) + rhot5(i)
90    continue
c
c     perform integration over volume and energy
c
```

```
      call vgintg(dume2,vr,dth,dz,index,igm,im,jm,km,result)
c
c     start calculating F(t)
c
c    perform the integral over all other energies first
c    and load the result onto dume3
      call gtermr(dume3,shape,xsfp,index,index1,igm)
c     note gofr does not depend on energy
c
c     multiply by spectrum and adjoint flux
c     and load after loce+index1 (has gofr)
      do 20 ig=1,igm
      do 30 i=1,index1
       i1    = i + index1*(ig-1)
       dume4(i1) = dume3(i)*chit(ig)*scadj(i1)
30    continue
20    continue
c
c    perform integration over volume and energy
c
      call vgintg(dume4,vr,dth,dz,index,igm,im,jm,km,res)
      fterm = res
c
c     end calculating F(t)
c
c     calculate rho
c
      rho = result/fterm
c
      return
      end
::::::::::::::
rhot3h.f
::::::::::::::
      subroutine rhot3h(rhot3, phiadj, phi, index, dsigt)
c     calculates the third term in rho
c     i spans all groups and cells
      real*4 rhot3(index),phiadj(index),phi(index),dsigt(index)
      do 10 i=1, index
       rhot3(i) = dsigt(i)*phiadj(i)*phi(i)
10    continue
      return
      end
::::::::::::::
rhot3r.f
::::::::::::::
      subroutine rhot3r(rhot3, phiadj, phi, index, dsigt,mm,weight)
c     calculates the third term in rho
c     i spans all groups and cells
      real*4 rhot3(index),phiadj(index*mm),phi(index*mm),dsigt(index)
     $ ,weight(320),sum
      real*4 zero
      zero = 0.e0
      do 10 i=1, index
      sum = zero
      do 20 m=1,mm
       i1 = i + index*(m-1)
       sum = sum + phiadj(i1)*phi(i1)*weight(m)
20    continue
       rhot3(i) = dsigt(i)*sum
10    continue
      return
      end
::::::::::::::
rhot4r.f
::::::::::::::
      subroutine rhot4r(rhot4,phiadj,phi,index,index1,igm,dsigs)
c     calculates the fourth term in rho
      common /irho16/ i16a
      real*4 rhot4(index),phiadj(index),phi(index),dsigs(index1,igm,igm)
      real*4 zero
      zero = 0.e0
c
c   scattering is from second dim. to third dim.
c e.g., dsigs(i,ig,igp) is scattering from ig to igp for cell i.
c
      do 10 ig=1,igm
      do 20 ir=1,index1
       j = ir+ index1*(ig-1)
c    sum over all other energies
c
      rhot4(j) = zero
      do 30 igp=1,igm
        k   = ir+index1*(igp-1)
        if(i16a.le.1) rhot4(j) = rhot4(j) + dsigs(ir,ig,igp)*phi(j)
        if(i16a.eq.2) rhot4(j) = rhot4(j) + dsigs(ir,igp,ig)*phi(j)
        if(i16a.eq.3) rhot4(j) = rhot4(j) + dsigs(ir,ig,igp)*phi(k)
        if(i16a.eq.4) rhot4(j) = rhot4(j) + dsigs(ir,igp,ig)*phi(k)
30    continue
c
c     mult. by adjoint flux
         rhot4(j)  =  rhot4(j)*phiadj(j)
20    continue
10    continue
c
c
      return
      end
::::::::::::::
rhot5r.f
```

```
::::::::::::::
      subroutine rhot5r(rhot5,rhot5a,phiadj,phi,index,index1,igmw
     $                 ,dsigf,chit)
c     calculates the fifth term in rho
c     rhot5a is scratch space
      real*4 rhot5(index),phiadj(index),phi(index),dsigf(index)
     $ ,chit(igmw),rhot5a(index1)
c
c    perform the integral over all other energies first
c    and load the result at rhot5a
      call gtermr(rhot5a,phi,dsigf,index,index1,igmw)
c     note gofr=rhot5a(1:index1) does not depend on energy
c
c     multiply by spectrum and adjoint flux
c
      do 20 ig=1,igmw
      do 30 i=1,index1
       i1    = i + index1*(ig-1)
       rhot5(i1) = rhot5a(i)*chit(ig)*phiadj(i1)
30    continue
20    continue
c
      return
      end
::::::::::::::
rkld.f
::::::::::::::
      subroutine rkld(y,rlamda,fterm,n6pk,ndg,cig2,scadj,im,jm,km,igm
     $ ,vr,dth,dz,indexn,index,itn,index1,dume2,chid,ampold,amp0,amplit)
      real*4 rlamda,fterm,cig2(indexn),scadj(index),vr(im),dth(jm)
     $ ,dz(km),dume2(index),csubi(10),chid(100),ampold,amp0,amplit
      double precision y
      dimension y(*)
c
      return
c
      call clearx(csubi,1,10)
c
      call pdcal(csubi,rlamda,fterm,ndg,cig2,scadj,dume2
     $ ,vr,dth,dz,im,jm,km,igm,index,indexn,index1,chid)
c
      write(n6pk,*) ' itn   old     new '
      write(n6pk,910) itn,y(1),amplit,(y(1+i),csubi(i),i=1,ndg)
910   format(1x,i3,1x,1p20(e12.5,2x))
      if(amplit.ne.0.) then  ! this is for first time this is called
c      ampold = amplit
c      amp0   = amplit
c      y(1)   = amplit
      endif
      do ii=1,ndg
       y(ii+1) = csubi(ii)
      enddo
c
      write(n6pk,*) ' itn amplitude    weighted precursors '
      write(n6pk,*) itn,(y(i),i=1,ndg+1)
c
      return
      end
::::::::::::::
rkld0.f
::::::::::::::
      subroutine rkld0(bet0,alambda,rlam0,n6pk,ndg,amplit0,y)
      real*4 bet0(10),alambda(10),rlam0,amplit0
      double precision y
      dimension y(*)
c
      write(n6pk,*)'initial amplitude              = ',amplit0
      write(n6pk,*)'initial normalized amplitude   = ',amplit0/amplit0
      do ii=1,ndg
c
      y(ii+1)  =  bet0(ii)*amplit0/(alambda(ii)*rlam0)
       write(n6pk,*)'initial weighted precursors  2....  c(',ii,') = '
     $ ,y(ii+1)
      enddo
c
      y(1) = amplit0
      write(46,*) 0.,y(1)/amplit0
c
      return
      end
::::::::::::::
rkldrp.f
::::::::::::::
      subroutine rkldrp(y,resrep,id,ir)
      double precision y(10),resrep(10)
c
      if(ir.eq.0) then
      do i=1,id
       y(i) = resrep(i)
      enddo
      else
      do i=1,id
       resrep(i) = y(i)
      enddo
      endif
c
      return
      end
::::::::::::::
rkunld.f
```

```
::::::::::::::
      subroutine rkunld(y,csubi,ndg,amplit0,n6pk,itn,itj,t,irep,iout)
      double precision y,t
      real*4 csubi(10),amplit0
      dimension y(*)
c
      times=t
      p  = y(1)
      do ii=1,ndg
       csubi(ii) = y(ii+1)
      enddo
      if(iout.ne.0) write(n6pk,100) itn,itj,times,p,(csubi(i),i=1,ndg)
      write(46,*) times,p/amplit0,irep
100   format(1x,i3,2x,i5,2x,1p,8(e11.4,1x))
c
      return
      end
::::::::::::::
rldem.f
::::::::::::::
      subroutine rldem(snpl,mm,vr,dth,dz,im,jm,km,chi,chs,igm,nsctt,idm)
      common sweight(320)
      real*4 sweight
      real*4 snpl(idm),vr(im),dth(jm),dz(km),chi(igm),chs(igm)
      include 'comsd9.F'
c
      if(nsctt.gt.0) then
       open(35,file='snpls',status='old',form='unformatted')
       read(35) i1
      call clearx(snpl,1,i1)
       read(35) (snpl(i),i=1,i1)
       close(35)
      endif
c
      open(35,file='weights',status='old',form='unformatted')
      call clearx(sweight,1,mm)
      read(35) (sweight(i),i=1,mm)
      close(35)
c
      open(35,file='vols',status='old',form='unformatted')
      call clearx(sone,1,im+jm+km)
      read(35) (sone(i),i=1,im+jm+km)
      do i=1,im
       vr(i) = sone(i)
      enddo
      do j=1,jm
       i1 = j + im
       dth(j) = sone(i1)
      enddo
      do k=1,km
       i1 = k + im+jm
       dz(k) = sone(i1)
      enddo
      close(35)
c
      open(35,file='chis',status='old',form='unformatted')
      call clearx(sone,1,2*igm)
      read(35) (sone(i),i=1,igm),(sone(j),j=igm+1,igm+igm)
      do i=1,igm
       chi(i) = sone(i)
       chs(i) = sone(i+igm)
      enddo
      close(35)
c
      return
      end
::::::::::::::
rwinpw.f
::::::::::::::
      subroutine rwinpw(j3,igm,ndg,chi)
c
      include 'comsd5.F'
c
      character*80 cny
      dimension llw(99), dx(300),l(2)
      real*4 chi(igm)
c
c      to read and change 61$$ and 63$$ arrays
c
c      variables
c      llw( 1:10)    = 61$$ array (length >10 but only 10 used )
c
c      reset the llw array
      do i=1,99
       llw(i)=0
      end do
      open(n5in,file='tortinw',status='old')
      open(n6tor,file='dltmel',status='unknown')
      open(n6sc,file='dltme',status='unknown')
c
1     read(n5in,'(a80)',end=100) cny
      if(cny(1:4).eq.'61$$'.or.cny(2:5).eq.'61$$') then
       write(n6sc,'(a80)') cny
2      read(n5in,'(a80)',end=100) cny
       if(cny(1:4).eq.'62$$'.or.cny(2:5).eq.'62$$') then
         goto 5
       else
         write(n6sc,'(a80)') cny
         goto 2
       endif
```

```
      else
       write(n6tor,'(a80)') cny
       goto 1
      endif
5     write(n6sc,*) 't'
      call read6w(j3,llw,n6sc,n6fi)
      if(j3.ne.0) goto 101
c
      llw(1)  = 0
      llw(2)  = ntfogf
c   set xsec unit
      llw(3)  = nts
c   set distributed source input unit
      llw(5)  = ntdsif
c
      write(n6tor,30) (llw(i),i=1,10)
c     the following is needed to end the card
      write(n6tor,*) ' e'
c
30    format('61$$',10(1x,i4))
c
c    this is to write the last line read but not written (62$$)
c
      write(n6tor,'(a80)') cny
      do i=1,99
       llw(i) = 0
      enddo
c     start search for 1** if this is first pass
      close(n6sc,status='delete')
      open(n6sc,file='dltme',status='unknown')
50    read(n5in,'(a80)') cny
      if(cny(1:3).eq.'1**'.or.cny(1:4).eq.' 1**') then
       write(n6sc,'(a80)') cny
55     read(n5in,'(a80)',end=100) cny
      if((cny(1:3).eq.'2**'.or.cny(1:4).eq.' 2**').or.
     $   (cny(1:3).eq.'3**'.or.cny(1:4).eq.' 3**').or.
     $   (cny(1:3).eq.'4**'.or.cny(1:4).eq.' 4**').or.
     $   (cny(1:3).eq.'5**'.or.cny(1:4).eq.' 5**').or.
c    $    (cny(1:3).eq.'6**'.or.cny(1:4).eq.' 6**').or.
     $   (cny(1:3).eq.'7**'.or.cny(1:4).eq.' 7**').or.
     $   (cny(1:3).eq.'9$$'.or.cny(1:4).eq.' 9$$').or.
     $   (cny(1:4).eq.'20**'.or.cny(1:5).eq.' 20**').or.
     $   (cny(1:4).eq.'21$$'.or.cny(1:5).eq.' 21$$').or.
     $   (cny(1:4).eq.'22**'.or.cny(1:5).eq.' 22**').or.
     $   (cny(1:4).eq.'23**'.or.cny(1:5).eq.' 23**').or.
     $   (cny(1:4).eq.'24**'.or.cny(1:5).eq.' 24**').or.
     $   (cny(1:4).eq.'26**'.or.cny(1:5).eq.' 26**').or.
     $   (cny(1:4).eq.'27**'.or.cny(1:5).eq.' 27**').or.
     $   (cny(1:4).eq.'30$$'.or.cny(1:5).eq.' 30$$').or.
     $   (cny(1:4).eq.'34**'.or.cny(1:5).eq.' 34**').or.
     $   (cny(1:4).eq.'91**'.or.cny(1:5).eq.' 91**').or.
     $   (cny(1:4).eq.'92**'.or.cny(1:5).eq.' 92**').or.
     $   (cny(1:4).eq.'93**'.or.cny(1:5).eq.' 93**').or.
     $   (cny(1:4).eq.'94**'.or.cny(1:5).eq.' 94**').or.
     $   (cny(1:4).eq.'95**'.or.cny(1:5).eq.' 95**').or.
     $   (cny(1:4).eq.'96**'.or.cny(1:5).eq.' 96**')) then
       goto 60
      else
       write(n6sc,'(a80)') cny
       goto 55
      endif
      else
       write(n6tor,'(a80)') cny
       goto 50
      endif
60    write(n6sc,*) 't'
      l(1)=1
      l(6)=igm+1
      l(7)=igm+igm+1
      rewind (n6sc)
c      call fidas(dx,dx,1,1,j3,n6sc,n6fi)
c      if(j3.ne.0) goto 101
c
c    use modified spectrum
      do ig=1, igm
       dx(ig) = chi(ig)
      enddo
c     adjust the spectrum for source
      do ig=igm+1,igm+igm
       dx(ig)= 1.
      enddo
      jmax=igm/6
      if(jmax*6.ne.igm) jmax=jmax+1
c    this is for 1**
      do j=1, jmax
       k=6*(j-1)+1
       kmax= k+5
       if(kmax.gt.igm) kmax=igm
       if(j.eq.1)
     $ write(n6tor,70) (dx(i),i=k,kmax)
       if(j.ne.1)
     $ write(n6tor,71) (dx(i),i=k,kmax)
      end do
c    this is for 6**
      do j=1, jmax
       k=6*(j-1)+1
       kmax= k+5
       if(kmax.gt.igm) kmax=igm
       if(j.eq.1)
     $ write(n6tor,72) (dx(i+igm),i=k,kmax)
```

```
        if(j.ne.1)
     $ write(n6tor,73) (dx(i+igm),i=k,kmax)
        end do
c    this is to write the last line read but not written
c
        write(n6tor,'(a80)') cny
c
70      format(' 1** ',6(f10.8,1x))
71      format('     ',6(f10.8,1x))
72      format(' 6** ',6(f10.8,1x))
73      format('     ',6(f10.8,1x))

10      read(n5in,'(a80)',end=15) cny
        write(n6tor,'(a80)') cny
        goto 10
100     write(6,*) 'error: reached the end of file'
        j3=98
        goto 101
15      write(6,*) 'finished modifying input file tortinw'
101     close(n5in)
        close(n6tor)
        close(n6sc,status='delete')
c
c       read the moddified file and write it onto tortinw
c
        open(n6tor,file='tortinw',status='unknown')
        open(n5in,file='dltme1',status='old')
c
80      read(n5in,'(a80)',end=25) cny
        if(cny(1:4).eq.'96**'.or.cny(1:5).eq.' 96**') goto 85
        write(n6tor,'(a80)') cny
        goto 80
85      read(n5in,'(a80)',end=25) cny
        do i1=1,72
        if(cny(i1:i1).eq.'t') then
87      read(n5in,'(a80)',end=25) cny
        write(n6tor,'(a80)') cny
        goto 87
        else
        goto 85
        endif
        enddo
c
25      close(n6tor)
        close(n5in,status='delete')
        return
        end
:::::::::::::::
rxtemp.f
:::::::::::::::
      subroutine rxtemp(tempold,index1,itn,itkm,powxt,dtk,powxb,tempiz
     $ ,mtm,powzt,powzb,volz)
c
c   note that temperatures at tk time points are used for dopler
c   feedback.  Only the last temperature is saved
c   powxt must be total power for each cell
c
        parameter(alfa=3.83e-11, econfr=3.204e-11)
        real*4 powxt(index1),tempold(index1),dtk,powxb(index1),tempiz(mtm)
     $ ,powzt(mtm),powzb(mtm),volz(mtm)
c
        open(35,file='bpower',status='old',form='unformatted')
        open(36,file='bpowerz',status='old',form='unformatted')
c
c   skip first record
          read(35) itkx
          read(36) itkx
c
        do itim=1,itkm
        read(35) (powxt(i),i=1,index1)
        read(36) (powzt(i),i=1,mtm)
        if(itn.eq.1.and.itim.eq.1) then
          do i=1,index1
            powxb(i) = powxt(i)
          enddo
          do i=1,mtm
            powzb(i) = powzt(i)
          enddo
        endif
c
c    powxt has energy conversion factor in it
c
        do i=1,index1
          tempold(i) = tempold(i) + alfa*dtk*(powxt(i)+powxb(i))/2./econfr
        enddo
c
        do i=1,index1
          powxb(i) = powxt(i)
        enddo
c
        do i=1,mtm
          tempiz(i) = tempiz(i) + alfa*dtk*(powzt(i)+powzb(i))/2./econfr
        enddo
c
        do i=1,mtm
          powzb(i) = powzt(i)
        enddo
c
        enddo
c
```

```
      close(35)
      close(36)
c
      return
      end
::::::::::::::
scalit.f
::::::::::::::
       subroutine scalit(afl,indexm,sfl,index,mm,weight)
       real*4 afl(indexm),sfl(index),weight(320),sum
      real*4 zero
      zero = 0.e0
c   creates scalar flux (sfl) from angular flux (afl)
       do i=1,index
       sum = zero
        do m=1,mm
        il = i + index*(m-1)
        sum = sum + afl(il)*weight(m)
        enddo
       sfl(i) = sum
       enddo
c
       return
       end
::::::::::::::
shftcg.f
::::::::::::::
      subroutine shftcg(cig1,cig2,indexn)
      real*4 cig1(indexn),cig2(indexn)
c
      do i=1,indexn
       cig2(i) = cig1(i)
      enddo
c
      return
      end
::::::::::::::
shftfl.f
::::::::::::::
      subroutine shftfl(shap1,shap2,shap3
     $    ,ansh1,ansh2,ansh3,index,indexm)
      real*4 shap1(index),shap2(index),shap3(index)
     $ ,ansh1(indexm),ansh2(indexm),ansh3(indexm)
c
      do i=1,index
       shap3(i) = shap2(i)
       shap2(i) = shap1(i)
      enddo
c
      do i=1,indexm
       ansh3(i) = ansh2(i)
       ansh2(i) = ansh1(i)
      enddo
c
      return
      end
::::::::::::::
steady.f
::::::::::::::
       subroutine steady(xsf0,xsf1,anadj,scadj,cx,cy,cz,snp1
     $ ,vr,dth,dz,chi,chs,denr,dume,z1c,cp,enr,xs,ansh1,shap1
     $ ,rxsec,cig0,cig1,xsfs,qmom,pd)
c
c    this routine is to perform initial adjoint and forward calculations
c     it saves the fluxes in s array
c
c   steady is called to perform initial adjoint and forward calculations
c     for other calls to tort, some other routine will be used
c
c     index  is (xinterval*yinterval*zinterval*numberofgroups)
c     index1 is (index/numberofgroups)
c
c     ifscon is convergence flag (=ifconv)
c
c23456789012345678901234567890123456789012345678901234567890123456789012
       include 'comsd1.F'
       include 'comsd2.F'
       include 'comsd4.F'
       include 'comsd5.F'
       include 'comsd6.F'
c
      real*4 xsf0(izart),xsf1(izart),anadj(indexm),scadj(index)
     $ ,cx(imw+1),cy(jmw+1),cz(kmw+1),snp1(idimsnp1)
     $ ,vr(imw),dth(jmw),dz(kmw),chi(igmw),chs(igmw)
     $ ,denr(igmw),dume(irest),z1c(index1),cp(izmw),enr(igmw+2)
     $ ,xs(izart),ansh1(indexm),shap1(index)
     $ ,rxsec(izart),cig0(indexn),cig1(indexn),xsfs(index)
     $ ,qmom(index*lmw),pd(index1),ampn,dampndt,dtnx
c
      common sweight(320)
      real*4 sweight
c
      character*1 equal
c
       integer status,system
       character*10 xnam
       character*9 filnam
       character*20 fils
c
       equal = '='
```

```
      xnam = 'tortnew '
c
      imany  = 0
       open(n6pk,file='pkout',status='unknown')
c
c  first read in forward X-secs
      call xsecr(xs,ihpw,mtmw,igmw,ntsiga)
      call xsecr(xsf0,ihpw,mtmw,igmw,ntsigf)
      call xsecr(xsf1,ihpw,mtmw,igmw,ntsigp)
c      open(35,file='xsec0',status='unknown')
c       do mt=1,mtmw
c       do ih=1,ihpw
c       i1 = ih + ihpw*((mt-1) + mtmw*(1-1))
c       i2 = ih + ihpw*((mt-1) + mtmw*(2-1))
c       write(35,*) xsf0(i1),xsf0(i2)
c      enddo
c      enddo
c      close(35)
c      open(35,file='xsec1',status='unknown')
c       do mt=1,mtmw
c       do ih=1,ihpw
c       i1 = ih + ihpw*((mt-1) + mtmw*(1-1))
c       i2 = ih + ihpw*((mt-1) + mtmw*(2-1))
c       write(35,*) xsf1(i1),xsf1(i2)
c      enddo
c      enddo
c      close(35)
c
      if(movem.ge.9) goto 100
c
       if(iafl.eq.0) then
       fils = ' <tortina >tortoua '
       status = system(xnam//fils)
       if(status.ne.0) stop "error : steady"
       open(35,file='evfil',status='old')
       read(35,*) evnew
       close(35)
      call fluxr(scadj,z1c,iafl,0,cp,cx,cy,cz,enr,denr
     $ ,index,index1,izmw,imw,jmw,kmw,igmw)
      open(35,file='anflux',status='old',form='unformatted')
       read(35)  (anadj(i),i=1,indexm)
      close(35)
      open(35,file='adjoint',status='unknown',form='unformatted')
       write(35) (anadj(i),i=1,indexm)
      close(35)
       elseif(iafl.eq.1) then
      call fluxr(scadj,z1c,iafl,0,cp,cx,cy,cz,enr,denr
     $ ,index,index1,izmw,imw,jmw,kmw,igmw)
      open(35,file='adjoint',status='old',form='unformatted',err=80)
       read(35)  (anadj(i),i=1,indexm)
      close(35)
      elseif(iafl.eq.2) then
       do i1=1,index
       read(39,*) in1,flx
       scadj(i1) = flx
       enddo
      close(39)
      if(ipkmod.eq.0) then
      open(35,file='adjoint',status='old',form='unformatted')
       read(35) (anadj(iw),iw=1,indexm)
      close(35)
      endif
      elseif(iafl.gt.2) then
       do i=1,indexm
        anadj(i) = 1.
       enddo
c   create scalar flux after reading in the weights
      endif
c
c     this call is for forward initial flux at t=0
c
100    if(iffl.eq.0.or.movem.ge.9) then
       fils = ' <tortinf >tortouf '
       status = system(xnam//fils)
       if(status.ne.0) stop "error : steady"
       open(35,file='evfil',status='old')
       read(35,*) evnew
       close(35)
       imany = imany +1
c   load and edit forward flux and/or shape
      call fluxr(shap1,z1c,iafl,1,cp,cx,cy,cz,enr,denr
     $ ,index,index1,izmw,imw,jmw,kmw,igmw)
c   copy forward flux file onto forwards
      status = system("cp fort.27 forwards")
       if(status.ne.0) stop "error : steady"
      call woutfl(shap1,index,0,48,"shap0",cx,cy,cz,imw,jmw,kmw,igmw
     $ ,vr,dth,dz)
      if(movem.ge.9) then
      call giper(xs,xsf0,xsf1,ihpw,mtmw,igmw,evnew)
       if(movem.eq.93.or.movem.eq.96) then
c   create second pert. xsecs
      call xsecr(xsf1,ihpw,mtmw,igmw,ntsigp2)
      call giper2(xsf1,ihpw,mtmw,igmw,evnew)
       endif
      stop 'critical case cross sections are generated and
     $put on units 56,57,58 for adjoint,forward,and perturbed
     $second perturbed is on unit 55 if requested'
      endif
c
c    set delta(E_n)
```

```
      do ig=1,igmw
       enr(ig) = 1.0
      end do
c
       idm = idimsnpl
      call rldem(snpl,mmw,vr,dth,dz,imw,jmw,kmw,chi,chs,igmw,nscttw,idm)
      open(35,file='anflux',status='old',form='unformatted')
       read(35) (anshl(iw),iw=1,indexm)
      close(35)
c   write forward angular flux to a file to read in later
      open(35,file='forward',status='unknown',form='unformatted')
       write(35) (anshl(iw),iw=1,indexm)
      close(35)
      elseif(iffl.gt.0) then
c     read in scalar forward flux
       status = system("cp forwards fort.27")
       if(status.ne.0) goto 120
       call fluxr(shap1,z1c,iafl,1,cp,cx,cy,cz,enr,denr
     $ ,index,index1,izmw,imw,jmw,kmw,igmw)
       call woutfl(shap1,index,0,48,"shap0",cx,cy,cz,imw,jmw,kmw,igmw
     $ ,vr,dth,dz)
c
c     set delta(E_n)
       do ig=1,igmw
        enr(ig) = 1.0
       end do
c
       idm = idimsnpl
      call rldem(snpl,mmw,vr,dth,dz,imw,jmw,kmw,chi,chs,igmw,nscttw,idm)
c    read in angular forward flux
      open(35,file='forward',status='old',form='unformatted',err=120)
       read(35) (anshl(iw),iw=1,indexm)
      close(35)
      imany = imany + 1
      endif
c
c    create adjoint scalar flux now that read sweight in.
      if(iagrev.gt.0.and.igmw.gt.1.and.iafl.lt.3) then
c     change group ordering
c
      call clearx(dume,1,indexm)
      do ii=1,indexm
       dume(ii) = anadj(ii)
      enddo
       do m=1,mmw
       do ig=1,igmw
       do ir=1,index1
        i1 = ir + index1*((ig-1) + igmw*(m-1))
        igx = igmw-ig+1
        i2 = ir + index1*((igx-1) + igmw*(m-1))
       anadj(i2) = dume(i1)
       enddo
       enddo
       enddo
c
c    end of change of group ordering
      endif
c    create adjoint scalar flux now that read sweight in.
      if(iafl.ne.2) then
      call scalit(anadj,indexm,scadj,index,mmw,sweight)
      endif
c
c   run staedy state fixed source calculation
c       if(ifix.eq.1) then
c first call generate ss cross sections
      do ig=1,igmw
      do i =1,index1
      j = i + index1*(ig-1)
      morg    = int(z1c(i))
      mt = int(cp(morg))
       i1 = ihtw-1 + ihpw*((mt-1) + mtmw*(ig-1))
      xsfs(j) = xsf0(i1)
      enddo
      enddo
      call cigint(shap1,beta,alambda,ndg,igmw,index1
     $ , cig0, cig1,index, xsfs)
c       call wout(cig0,indexn,6,0)
      ampn = 1.
      dampndt = 0.
      dtnx = dtn(1)
      call rdgip2(xsf0,igmw,ihtw,ihpw,mtmw,betat,ampn,dampndt,iqs
     $ ,vinv,dtnx,nts,rxsec,-1)
      call qadd(cig0,alambda,mmw,lmw,imw,jmw,kmw,igmw,ndg
     $ ,index1,qmom,sweight,chid,iqs,ampn)
c
c    set input and output units for flux guess
c    note that they must be duifferent???? not sure ....
      j3 = 0
      call rwinpw(j3,igmw,ndg,chi)
      if(j3.ne.0) then
      write(n6pk,*) 'error readw:jobw ....'
      stop
      endif
c    set convergence parameters for fixed source
c       j3 = 0
c       call read66(j3)
c       if(j3.ne.0) then
c       write(n6pk,*) 'error read66:jobw ....'
c       stop
c       endif
```

```
c
      call wdistr (igmw,imw,jmw,kmw,lmw,mmw,ntdsif,qmom
     $ ,cx,cy,cz,enr)
c
c     run a new tort case
c
      if(iffl.eq.0) then
      evold = evnew
      filnam = 'tortouc'
       fils = ' <tortinw > '
       status = system("tortnew <tortinw >tortouc")
       if(status.ne.0) stop "error : steady"
       open(35,file='evfil',status='old')
       read(35,*) evnew
       close(35)
c
      open(35,file='anflux',status='old',form='unformatted')
       read(35) (anshl(iw),iw=1,indexm)
      close(35)
c
      call fluxr(shap1,z1c,iafl,1,cp,cx,cy,cz,enr,denr
     $ ,index,index1,izmw,imw,jmw,kmw,igmw)
c
c    save fluxes for later
       status = system("cp fort.27 fixfors")
       if(status.ne.0) stop "error : steady"
      open(35,file='fixfor',status='unknown',form='unformatted')
       write(35) (anshl(iw),iw=1,indexm)
      close(35)
      elseif(iffl.gt.0) then
       status = system("cp fixfors fort.27")
       if(status.ne.0) goto 120
      call fluxr(shap1,z1c,iafl,1,cp,cx,cy,cz,enr,denr
     $ ,index,index1,izmw,imw,jmw,kmw,igmw)
      open(35,file='fixfor',status='old',form='unformatted',err=120)
       read(35) (anshl(iw),iw=1,indexm)
      close(35)
      endif
c
c      calculate initial total power with fixed source flux
c
      call powcal(pow0,shap1,index,index1,igmw,xsfs
     $ ,vr,dth,dz,imw,jmw,kmw,rnu,pd)
c
c
c    print flux shape to a file
      call woutfl(shap1,index,0,48,"fs0.1",cx,cy,cz,imw,jmw,kmw,igmw
     $ ,vr,dth,dz)
      call woutfl(pd,index1,1,49,"pd0.1",cx,cy,cz,imw,jmw,kmw,igmw
     $ ,vr,dth,dz)
c
      write(n6pk,'(120(a1))') (equal,itest=1,115)
      write(n6pk,*)'Initial power        = ',pow0,' fissions/sec'
      write(n6pk,*)'Initial norm.d power = ',pown,' fissions/sec'
      write(n6pk,*)'Initial amplitude    = ',amplit0
      write(n6pk,*)'Initial shape        = ',shap0
     $ , ' (volume and energy integrated)'
      write(n6pk,*)'Initial eigenvalue   = ',evnew
      write(n6pk,'(120(a1))') (equal,itest=1,115)
c      endif
c    end steady state fixed source calculation
c
c     write ss shape to interface file if not restart
c
      if(irstrt.eq.0) then
      open(35,file='shapet',status='unknown',form='unformatted')
      write(35) (shap1(i),i=1,index)
      close(35)
      open(35,file='ashapet',status='unknown',form='unformatted')
      write(35) (anshl(i),i=1,indexm)
      close(35)
      endif
      return
80    write(6,*)'+++++++++++++++++++++++++++++++++++++++++++++'
      write(6,*)'++      old adjoint file does not exist     ++'
      write(6,*)'++ you must first run a case with iafl = 0 ++'
      write(6,*)'+++++++++++++++++++++++++++++++++++++++++++++'
      stop
120   write(6,*)'+++++++++++++++++++++++++++++++++++++++++++++'
      write(6,*)'++      old forward file does not exist     ++'
      write(6,*)'++ you must first run a case with iffl = 0 ++'
      write(6,*)'+++++++++++++++++++++++++++++++++++++++++++++'
      stop
      end
:::::::::::::::
tdtort.f
:::::::::::::::
      program tdtort
c  this is the driver program
       real timediff, timearray(2),dtime
       integer wallclock, elapsed
c  start clock
         call startclock
       timediff = dtime( timearray )
c    read inputs and set pointers
      call fmgra
c
      call work(iot,n6sed)
       timediff = dtime( timearray )
       elapsed = wallclock()
```

```
       imin = elapsed/60
       isec = elapsed - imin*60
       open(n6sed,file='transin',status='old',access='append')
       write(iot, 20 ) imin,isec
       write(n6sed, 20 ) imin,isec
20       format(/,'Problem elapsed time ',i5,' min. ',i2,' seconds'/)
        if(timediff.gt.60.) then
         idmin = timediff/60.
         idsec = timediff - idmin*60
         iamin = timearray(1)/60.
         iasec = timearray(1) - iamin*60
         ibmin = timearray(2)/60.
         ibsec = timearray(2) - ibmin*60
       write(iot,101) idmin,idsec,iamin,iasec,ibmin,ibsec
       write(n6sed,101) idmin,idsec,iamin,iasec,ibmin,ibsec
101    format(" CPU time = ",i5,"m",i2.2,"s"," ; user time = ",i5,"m"
     + ,i2"s"," ; system time = ",i5,"m",i2,"s"/)
        else
       write(iot,102) timediff,timearray(1),timearray(2)
       write(n6sed,102) timediff,timearray(1),timearray(2)
102    format(" CPU time (sec.) ",f10.3," ; user time (sec.) ",f10.3
     +  , " ; system time (sec.) ",f9.3/)
        endif
      end

      subroutine work(iot,n6out)
c
      include 'comsd1.F'
      include 'comsd2.F'
      include 'comsd3.F'
      include 'comsd4.F'
      include 'comsd5.F'
      include 'comsd6.F'
      include 'comsd7.F'
c
c   determine remaining of the container array
      irest = isdim - loce
      l8 = loce + 4*index + 2*indexn + 2*index1 + mdamp+1
      llend = l8 + index1 + 4*index + idtjm + indexm
      open(n6sed,file='transin',status='old',access='append')
      write(n6sed,*)"-------------------------------------------------"
      write(n6   ,*)"-------------------------------------------------"
      write(n6sed,*)"  End of container array        = ",loce
      write(n6   ,*)"  End of container array        = ",loce
      write(n6sed,*)"  Scratch space available      = ",irest
      write(n6   ,*)"  Scratch space available      = ",irest
      write(n6sed,*)"  End of Total Space Used      = ",llend
      write(n6   ,*)"  End of Total Space Used      = ",llend
      write(n6sed,*)"  End of Total Space Available = ",isdim
      write(n6   ,*)"  End of Total Space Available = ",isdim
      write(n6sed,*)"-------------------------------------------------"
      write(n6   ,*)"-------------------------------------------------"
c
      if(llend.ge.isdim) then
       write(n6sed,*)"Error : (tdtort) not enough scratch space"
       write(n6   ,*)"Error : (tdtort) not enough scratch space"
       stop 16
      endif
c
c     initialize the container array
c
      call clearx(s,1,llend)
c
c234567890123456789012345678901234567890123456789012345678901234567890123456789012
      call steady(s(lxsf0),s(lxsf1),s(locaa),s(1),s(loccx),s(loccy)
     $ ,s(loccz),s(lsnpl),s(locvr),s(locdth),s(locdz),s(locchi)
     $ ,s(locchs),s(ldenr),s(loce),s(loc1c),s(loccp),s(locenr),s(locxs)
     $ ,s(locsa1),s(locsh1),s(lrxsec),s(lcig0),s(lcig1),s(lxsfs)
     $ ,s(locqm),s(locpd))
c
c     calculate eq 3.a
      call chitr(s(lchit), igmw, betat, chid, beta, ndg, s(locchi))
c
c     transfer original spectrum to lchi0
      do ig=1,igmw
       s(lchi0+ig-1) = s(locchi+ig-1)
      enddo
c
c  create zone xsecs for ss
      call dxsec(s(lxsts),s(lxstp),s(lxstd),s(lxsfs),s(lxsfp),s(lxsfd)
     $,s(lxsss),s(lxssp),s(lxssd),s(lxsf0),s(lxsf1),index
     $,s(loc1c),s(loccp),igmw,index1,ihpw,ihtw,mtmw,izmw
     $,0,s(ldum0),s(ldum1),s(ldumx))
c  perform initial calculations
      l1 = loce
      l2 = l1 + index1
      l3 = l2 + index
      l4 = l3 + index1
      l5 = l4 + index
c234567890123456789012345678901234567890123456789012345678901234567890123456789012
      call inical(s(lxsfs),s(locvr),s(locdth),s(locdz),s(locpd),s(lcig0)
     $ ,s(lcig1),s(locsa1),s(1),s(locaa),s(locsh1),s(loccx),s(loccy)
     $ ,s(loccz),s(lxsf0),s(lrxsec),s(locqm),s(l5),s(locchi),s(locenr)
     $ ,s(loc1c),s(loccp),s(ldenr),s(lxstd),s(lxssd),s(lxsfd),s(lchit)
     $ ,s(lxsfp),s(l1),s(l2),s(l3),s(l4),s(lrhot3),s(lrhot4),s(lrhot5))
      write(45,*) 0.,amplit0/amplit0
c
      l1 = loce
      l2 = l1 + 2*index
      l3 = l2 + 2*indexn
```

```
      lp1 = l3 + mdamp + 1    ! +1 is for jobw0
      lp2 = lp1 + index1
      lp3 = lp2 + index
      lp4 = lp3 + index1
      l8  = lp4 + index
      irestx = isdim - l8
      close(n6sed)
c234567890123456789012345678901234567890123456789012345678901234567890123456789012
      if(movem.eq.2) then
      call jobw2(iot,s(lcig0),s(1),s(locvr),s(locdth),s(locdz),s(loc1c)
     $ ,s(lxsts),s(lxstp),s(lxstd),s(lxsfs),s(lxsfp),s(lxsfd),s(lxsss)
     $ ,s(lxssp),s(lxssd),s(lxsf0),s(lxsf1),s(loccp),s(ldum0),s(ldum1)
     $ ,s(ldumx),s(lrxsec),s(locchi),s(lchit)
     $ ,s(locqm),s(loccx),s(loccy),s(loccz),s(locenr),s(ldenr),s(locaa)
     $ ,s(lsnpl),s(locpd),s(locamp),s(loc1),s(loc2),s(locsh1),s(locsh2)
     $ ,s(locsh3),s(locsh4),s(loc1a),s(loc2a),s(locsa1),s(locsa2)
     $ ,s(locsa3),s(locsa4),s(lcig1),s(lcig2),s(l1),s(l2),s(l3)
     $ ,s(lp1),s(lp2),s(lp3),s(lp4),s(l8),s(lrhot3),s(lrhot4),s(lrhot5)
     $ ,s(lxsf2))
      elseif(movem.eq.3) then
      call jobw3(iot,s(lcig0),s(1),s(locvr),s(locdth),s(locdz),s(loc1c)
     $ ,s(lxsts),s(lxstp),s(lxstd),s(lxsfs),s(lxsfp),s(lxsfd),s(lxsss)
     $ ,s(lxssp),s(lxssd),s(lxsf0),s(lxsf1),s(loccp),s(ldum0),s(ldum1)
     $ ,s(ldumx),s(lrxsec),s(locchi),s(lchit)
     $ ,s(locqm),s(loccx),s(loccy),s(loccz),s(locenr),s(ldenr),s(locaa)
     $ ,s(lsnpl),s(locpd),s(locamp),s(loc1),s(loc2),s(locsh1),s(locsh2)
     $ ,s(locsh3),s(locsh4),s(loc1a),s(loc2a),s(locsa1),s(locsa2)
     $ ,s(locsa3),s(locsa4),s(lcig1),s(lcig2),s(l1),s(l2),s(l3)
     $ ,s(lp1),s(lp2),s(lp3),s(lp4),s(l8),s(lrhot3),s(lrhot4),s(lrhot5)
     $ ,s(lxsf2))
      elseif(movem.eq.5) then
      call jobw5(iot,s(lcig0),s(1),s(locvr),s(locdth),s(locdz),s(loc1c)
     $ ,s(lxsts),s(lxstp),s(lxstd),s(lxsfs),s(lxsfp),s(lxsfd),s(lxsss)
     $ ,s(lxssp),s(lxssd),s(lxsf0),s(lxsf1),s(loccp),s(ldum0),s(ldum1)
     $ ,s(ldumx),s(lrxsec),s(locchi),s(lchit)
     $ ,s(locqm),s(loccx),s(loccy),s(loccz),s(locenr),s(ldenr),s(locaa)
     $ ,s(lsnpl),s(locpd),s(locamp),s(loc1),s(loc2),s(locsh1),s(locsh2)
     $ ,s(locsh3),s(locsh4),s(loc1a),s(loc2a),s(locsa1),s(locsa2)
     $ ,s(locsa3),s(locsa4),s(lcig1),s(lcig2),s(l1),s(l2),s(l3)
     $ ,s(lp1),s(lp2),s(lp3),s(lp4),s(l8),s(lrhot3),s(lrhot4),s(lrhot5)
     $ ,s(lxsf2))
      elseif(movem.eq.6) then
      call jobw6(iot,s(lcig0),s(1),s(locvr),s(locdth),s(locdz),s(loc1c)
     $ ,s(lxsts),s(lxstp),s(lxstd),s(lxsfs),s(lxsfp),s(lxsfd),s(lxsss)
     $ ,s(lxssp),s(lxssd),s(lxsf0),s(lxsf1),s(loccp),s(ldum0),s(ldum1)
     $ ,s(ldumx),s(lrxsec),s(locchi),s(lchit)
     $ ,s(locqm),s(loccx),s(loccy),s(loccz),s(locenr),s(ldenr),s(locaa)
     $ ,s(lsnpl),s(locpd),s(locamp),s(loc1),s(loc2),s(locsh1),s(locsh2)
     $ ,s(locsh3),s(locsh4),s(loc1a),s(loc2a),s(locsa1),s(locsa2)
     $ ,s(locsa3),s(locsa4),s(lcig1),s(lcig2),s(l1),s(l2),s(l3)
     $ ,s(lp1),s(lp2),s(lp3),s(lp4),s(l8),s(lrhot3),s(lrhot4),s(lrhot5)
     $ ,s(lxsf2))
      elseif(movem.eq.0) then
      call jobw0(iot,s(lcig0),s(1),s(locvr),s(locdth),s(locdz),s(loc1c)
     $ ,s(lxsts),s(lxstp),s(lxstd),s(lxsfs),s(lxsfp),s(lxsfd),s(lxsss)
     $ ,s(lxssp),s(lxssd),s(lxsf0),s(lxsf1),s(loccp),s(ldum0),s(ldum1)
     $ ,s(ldumx),s(lrxsec),s(locchi),s(lchit)
     $ ,s(locqm),s(loccx),s(loccy),s(loccz),s(locenr),s(ldenr),s(locaa)
     $ ,s(lsnpl),s(locpd),s(locamp),s(locsh1),s(locsh2)
     $ ,s(locsh3),s(locsh4),s(locsa1),s(locsa2)
     $ ,s(locsa3),s(locsa4),s(lcig1),s(lcig2),s(l1),s(l2),s(l3)
     $ ,s(lp1),s(lp2),s(lp3),s(lp4),s(l8),s(lrhot3),s(lrhot4),s(lrhot5)
     $ ,s(lxsf2),s(ltempold),s(lxsas),s(lxsap),s(lpowxt),s(lpowxb)
     $ ,s(luxsfp),s(ltemprp),s(ltmpz),s(ltmpzrp),s(lpowzt),s(lpowzb)
     $ ,s(lvolz))
      elseif(movem.eq.1) then
      call jobw1(iot,s(lcig0),s(1),s(locvr),s(locdth),s(locdz),s(loc1c)
     $ ,s(lxsts),s(lxstp),s(lxstd),s(lxsfs),s(lxsfp),s(lxsfd),s(lxsss)
     $ ,s(lxssp),s(lxssd),s(luxsfp),s(lxsf0),s(lxsf1),s(loccp)
     $ ,s(ldum0),s(ldum1),s(ldumx),s(lrxsec),s(locchi),s(lchit)
     $ ,s(locqm),s(loccx),s(loccy),s(loccz),s(locenr),s(ldenr),s(locaa)
     $ ,s(lsnpl),s(locpd),s(locamp),s(locsh1),s(locsh2)
     $ ,s(locsh3),s(locsh4),s(locsa1),s(locsa2)
     $ ,s(locsa3),s(locsa4),s(lcig1),s(lcig2),s(l1),s(l2),s(l3)
     $ ,s(lp1),s(lp2),s(lp3),s(lp4),s(l8),s(lrhot3),s(lrhot4),s(lrhot5)
     $ ,s(lxsf2),s(lxsfnew),s(luxsfn))
      endif
c
      n6out = n6sed
      return
      end


       subroutine startclock
       common / myclock / mytime
       integer mytime
       integer time
       mytime = time()
       return
       end

       function wallclock
       integer wallclock
       common / myclock / mytime
       integer mytime
       integer time
       integer newtime
       newtime = time()
       wallclock = newtime - mytime
       mytime = newtime
```

```
      return
      end

::::::::::::::
vgintg.f
::::::::::::::
      subroutine vgintg(func,vr,dth,dz,index,igm,im,jm,km,result)
c     performs integration of func over volume and energy
c     and puts the result on result
c     since the fluxes are cell averaged, the integral is carried out
c     by MVT
      real*4 func(index),vr(im),dth(jm),dz(km),vol,result
      real*4 zero
      zero = 0.e0
      result = zero
      do 100 ig=1, igm
      do 200  k=1, km
      do 300  j=1, jm
      do 400  i=1, im
      vol    = vr(i)*dth(j)*dz(k)
      il= i + im*((j-1) + jm*((k-1) + km*((ig-1))))
c     volume integral
      result  = result  + func(il)*vol
400   continue
300   continue
200   continue
100   continue
      return
      end
::::::::::::::
wdistr.f
::::::::::::::
      subroutine wdistr(igm,im,jm,km,lm,mm,ntdsif,qmom,x,y,z,denr)
c   this routine is to write distributed source moments to a file in flxmom format
      character hname*6,huse(2)*6,ivers*6
      character date*6,user*6,charge*6,case*6,times*6,titl(12)*6
      real*4 qmom(im,jm,km,lm,igm),x(im+1),y(jm+1),z(km+1),denr(igm)
c
      open(ntdsif,status='unknown',form='unformatted')
      rewind (ntdsif)
      izero = 0
      ifmom = 1
      ifbnd = 0
      idim  = 3
      zero  = 0.
      hname ='varscl'
      huse(1) ='ahblah'
      huse(2) ='blahbl'
      ivers = 'ahblah'
      date ='blahbl'
      user ='ahblah'
      charge ='blahbl'
      case ='ahblah'
      times ='blahbl'
      do i=1,12
       titl(i) = 'sedat1'
      enddo
      write(ntdsif) hname,(huse(i),i=1,2),ivers
      write(ntdsif) date,user,charge,case, times,(titl(i),i=1,12)
      write(ntdsif) ifmom,ifbnd,idim,igm,igm,im,jm,km,lm,mm,
     $(izero,i=1,15)
      write(ntdsif) (izero,i=1,10)
c
      write(ntdsif) (x(i),i=1,im+1),(y(j),j=1,jm+1)
     $ ,(z(k),k=1,km+1)
     $ ,(denr(ig),ig=1,igm),zero,zero,(zero,ig=1,2*igm)
c
      do 10 ig=1,igm
      do 30 kk=1,km
      do 40 jk=1,jm
      write(ntdsif) ((qmom(i,jk,kk,l,ig),i=1,im),l=1,lm)
40    continue
30    continue
10    continue
c
      close(ntdsif)
      return
      end
::::::::::::::
wout.f
::::::::::::::
      subroutine wout(a,in,it,n6)
      real*4 a(in)
      if(it.gt.7) return
      if(n6.eq.0) open(35,file='delsor',status='unknown')
      if(n6.eq.1) open(35,file='delsor',status='old',access='append')
        mmtest = in/it +1
       do mtest =1, mmtest
         m2 = (mtest-1)*it+1
         m3 = m2 +it-1
         if(m3.gt.in) m3=in
         write(35,'(i5,1x,7(1pe12.5,1x))') m2,(a(m1),m1=m2,m3)
       enddo
      close(35)
      return
      end
::::::::::::::
woutfl.f
::::::::::::::
      subroutine woutfl(a,in,it,n6,filnam,x,y,z,im,jm,km,igm,vr,dth,dz)
```

```
c
c    it = 0 signals flux output for all planes, all columns, all rows
c                              and all groups
c    it = 1 signals power density output for all planes, all columns
c                              and all rows
      include 'comsd1.F'
c
      real*4 a(in),x(im+1),y(jm+1),z(km+1),vr(im),dth(jm),dz(km)
      character*6 filnam
c
      if(ioutsh.eq.0) return
      open(n6,file=filnam,status='unknown')
      if(jm.eq.1.and.km.eq.1) goto 200
      if(it.eq.0) then
      write(n6,*) '      x             y             z             flux '
      do 10 ig=1,igm
      write(n6,*) 'flux for group ',ig
      do 20 k =1,km
      do 30 j =1,jm
      do 40 i =1,im
       xx = (x(i+1)+x(i)) / 2.
       yy = (y(j+1)+y(j)) / 2.
       zz = (z(k+1)+z(k)) / 2.
       i1 = i + im*((j-1) + jm*((k-1) + km*(ig-1)))
       write(n6,100) xx,yy,zz,a(i1)
40    continue
30    continue
20    continue
10    continue
      elseif(it.eq.1) then
      write(n6,*) 'power density '
      write(n6,*) '      x             y             z             pd '
      do 120 k =1,km
      do 130 j =1,jm
      do 140 i =1,im
       vol = vr(i)*dth(j)*dz(k)
       xx = (x(i+1)+x(i)) / 2.
       yy = (y(j+1)+y(j)) / 2.
       zz = (z(k+1)+z(k)) / 2.
       i1 = i + im*((j-1) + jm*(k-1))
       write(n6,100) xx,yy,zz,a(i1),vol
140   continue
130   continue
120   continue
      endif
      return
200   write(n6,*) '      x     flux '
      do 210 ig=1,igm
      write(n6,*) 'flux for group ',ig
      do 240 i =1,im
       xx = (x(i+1)+x(i)) / 2.
       i1 = i + im*(ig-1)
       write(n6,100) xx,a(i1)
240    continue
210    continue
      close(n6)
100   format(1p,5(1x,e12.5,1x))
c
      return
      end
::::::::::::::
xsecr.f
::::::::::::::
      subroutine xsecr(xsec,ihp,mtm,igm,ntsig)
c     loads the cross sections for all groups and all materials
c     into array s
c
      real*4 xsec(ihp,mtm,igm)
c
      open(ntsig,form='unformatted',status='old')
      do 10 ig=1,igm
      read(ntsig) ((xsec(ih,mt,ig),ih=1,ihp),mt=1,mtm)
10    continue
      close(ntsig)
      return
      end
::::::::::::::
zoneinfo.f
::::::::::::::
      subroutine zoneinfo(xsfs,z1c,cp,im,jm,km,index,index1,igm,izm)
      real*4 xsfs(index),z1c(index1),cp(izm)
c
      dimension list(273),is(27000),liste(1694),listz(121)
          data listz/
     $  2,1,1,1,1,2,2,3,3,5,5,
     $  1,1,1,1,1,1,1,3,3,5,5,
     $  1,1,1,1,1,1,1,3,3,5,5,
     $  1,1,1,1,1,1,1,3,3,5,5,
     $  1,1,1,1,1,1,1,3,3,5,5,
     $  2,1,1,1,1,2,2,3,3,5,5,
     $  2,1,1,1,1,2,2,3,3,5,5,
     $  3,3,3,3,3,3,3,4,5,5,5,
     $  3,3,3,3,3,3,3,5,5,5,5,
     $  5,5,5,5,5,5,5,5,5,5,5,
     $  5,5,5,5,5,5,5,5,5,5,5/
      data is/27000*0/
      data list/5
     $         ,5,3,3
     $         ,5,4,3,3,3
     $         ,2,2,1,1,2
```

```
     $         ,2,2,1,1,2
     $         ,3,1,1,1
     $         ,3,1,1
     $         ,3,1,1
     $         ,3,2,1,2
     $         ,160*0     ! to be filled later
     $         ,80*3/     ! control rod region
      if(izm.gt.273) goto 10
      do i=1,32
        list(1+32+i) = list(1+i)
        list(1+64+i) = list(1+i)
        list(1+96+i) = list(1+i)
        list(1+128+i) = list(1+i)
        list(1+160+i) = list(1+i)
      enddo
      goto 20
10    continue
          do i=1,242
          liste(i) = 5
          enddo
c
          do i=1,121
          liste(242+i) = listz(i)
          liste(242+i+121) = listz(i)
          liste(242+i+2*121) = listz(i)
          liste(242+i+3*121) = listz(i)
          liste(242+i+4*121) = listz(i)
          liste(242+i+5*121) = listz(i)
          liste(242+i+6*121) = listz(i)
          liste(242+i+7*121) = listz(i)
          liste(242+i+8*121) = listz(i)
          liste(242+i+9*121) = listz(i)
          enddo
c
          do i=1453,1694
          liste(i) = 5
          enddo
c
20      open(35,file='zoneinfo',status='unknown')
      do k=km,1,-1
        write(35,*) '--------- k = ',k,' -------------------'
       do j=jm,1,-1
        i1 = 1 + im*((j-1) + jm*(k-1))
        i2 = i1 + im - 1
       write(35,'(40(1x,i4))') (int(cp(int(z1c(i)))),i=i1,i2)
       enddo
      enddo
c
       close(35)
c
       open(35,file='matinfo',status='unknown')
       do k=km,1,-1
        write(35,*) '--------- k = ',k,' -------------------'
        do j=jm,1,-1
         i1 = 1 + im*((j-1) + jm*(k-1))
         i2 = i1 + im - 1
         if(izm.eq.273) then
         write(35,'(40(1x,i2))') (list(int(cp(int(z1c(i))))),i=i1,i2)
         elseif(izm.eq.1694) then
         write(35,'(40(1x,i2))') (liste(int(cp(int(z1c(i))))),i=i1,i2)
         else
         write(35,'(40(1x,i3))') (list(int(cp(int(z1c(i))))),i=i1,i2)
         endif
        enddo
       enddo
c
       close(35)
c
       open(35,file='indexinfo',status='unknown')
       do k=km,1,-1
        write(35,*) '--------- k = ',k,' -------------------'
        do j=jm,1,-1
         i1 = 1 + im*((j-1) + jm*(k-1))
         i2 = i1 + im - 1
         write(35,'(40(1x,i5))') (i,i=i1,i2)
        enddo
       enddo
c
       close(35)
c
       open(35,file='xsecinfo',status='unknown')
       do k=km,1,-1
         write(35,*) '--------- k = ',k,' -------------------'
         do j=jm,1,-1
           i1 = 1 + im*((j-1) + jm*(k-1))
           i2 = i1 + im - 1
           do ix=i1,i2
             if(xsfs(ix).ne.0.) is(ix) = 1
           enddo
           write(35,'(40(1x,i1))') (is(i),i=i1,i2)
         enddo
       enddo
c
       close(35)
c
       return
       end
c
::::::::::::::
comsd0.F
```

```
::::::::::::::
      external f
      common sweight(320)
      real*4 sweight
c
      double precision resrep(10),ttrep2,ttrep3
      real*4 betelrp(10),bete2rp(10),betenrp(10)
c
      double precision y, atol, rwork, rtol, t,  tout
      dimension y(10), rwork(500), iwork(100)
      real*4 csubi(10),betaef(10)
c
      character*4 filnam
      character*5 filnam2
      character*1 equal
      logical iquad,outflg
c
      character*10 xnam
      character*11 fils
::::::::::::::
comsd1.F
::::::::::::::
c
      common /rdnpd/ dtn,dtns,dtj,dtk
      real*4  dtn(2000),dtns(2000),dtj,dtk
      common /rdnp/ alambda,beta,vinv,betat,chid,rnu,powini,srtol,satol
      real*4 alambda(10),beta(10),vinv(100),betat,chid(100),rnu,powini
     $ ,srtol,satol
      common /rdnpin/ betaef0,csubi0,pow0,pown,amplit0,shap0,rho0
     $ ,fterm0,rlamda0,extsrc0,betaeft
      real*4 betaef0(10),csubi0(10),pow0,pown,amplit0,shap0,rho0
     $ ,fterm0,rlamda0,extsrc0,betaeft
      common /idnp/ ndg,movem,idtnm,idtjm,ioutsh,idtj(2000),idtk(2000)
     $ ,iout,ifix,iafl,ipint,iagrev,ilpend,iextrp,irstrt
     $ ,ipkmod,iqdmod,ifeed,iqs,iffl,mzm,mz(10)
c
::::::::::::::
comsd2.F
::::::::::::::
      common /imany/ imany,locaa,loccp,loccx,loccy,loccz
     $ ,lxsf0,lxsf1,lxsf2,locxs,lxsas,lxsap
     $ ,lxsts,lxstp,lxstd,lxsfs,lxsfp,lxsfd,lxsss,lxssp,lxssd
     $ ,ldenr,ldum0,ldum1,ldumx,idimsnpl,lpowxb,lxsfnew,luxsfn,luxsfp
     $ ,locvo,locah,locav,locvr,locdth,locdz,locenr,lsnpl,lpowxtl
     $ ,locpd,locpp,locap,ltempa,ltempb,ltempold,ltempmt,lpowxt,lpowxt0
     $ ,locsh1,locsh2,locsh3,locsh4,locki1,locki2,locki3,locki4
     $ ,loc1a,loc2a,loc3a,loc4a,locsa1,locsa2,locsa3,locsa4
     $ ,loc1,loc11,loc1c ,loc2,loc21,loc2c
     $ ,loc3,loc31,loc3c ,loc4,loc41,loc4c,  indexm,indexn
     $ ,local,loce,index,izart,index1,locchi,locchs,lchi0,irest,irestx
     $ ,ltemprp,ltmpz,ltmpzrp,lpowzt,lpowzb,lvolz
::::::::::::::
comsd3.F
::::::::::::::
      parameter (isdim = 8 000 000)
      common /swork/ s
      real*4 s(isdim)
::::::::::::::
comsd4.F
::::::::::::::
      common /ijkmgw/ imw,jmw,kmw,igmw,izmw,mmw,lmw,mtmw
     $ ,ncrxw,nsctmw,nscttw,ihmw,ihpw,ihtw,iupsw,ihsw
      common /ifscon/ ifscon,ixtr,iroute,irepmx
      common /wfscon/ evw,evnew,evold
c      real*4 evw,evnew,evold
::::::::::::::
comsd5.F
::::::::::::::
      common /iunits/ n6,n5in,n6sed,n6tor,n6sc,n6pk,n6fi
     $ ,ifxprtw,ntfogf,ntsigf,ntsigp,ntsclf,ntdirf,ntflxf
     $ ,modepw,ntfoga,ntsiga,ntscla,ntdira,ntzonf,ntdsif
     $ ,nts,idjlek,ntsigp2
::::::::::::::
comsd6.F
::::::::::::::
      common /idisp/ lchit
      common /irhol/ lrhot3,lrhot4,lrhot5,locq,locqa,locqs
     $ ,locqm,lrp1,lrp2,lrp3,locamp,ldamp,mdamp
     $ ,lrxsec,lcig0,lcig1,lcig2,lcig4
::::::::::::::
comsd7.F
::::::::::::::
      common /pkdat/ rho1,rho2,rhonew,rlam1,rlam2,rlamnew
     $ ,betae1,betae2,betnew,bet1,bet2,betn
     $ ,q1,q2,qnew,hi1,hi2,hj1,hj2
      real*4  rho1,rho2,rhonew,rlam1,rlam2,rlamnew
     $ ,betae1(10),betae2(10),betnew(10),bet1,bet2,betn
     $ ,q1,q2,qnew,hi1,hi2,hj1,hj2
      logical jquad
      common /swlog/ jquad
::::::::::::::
comsd8.F
::::::::::::::
      common /pkprt/ rhop,rlamp,extp,betap,betp,dampndtp
      real*4  rhop,rlamp,extp,betap(10),betp,dampndtp
::::::::::::::
comsd9.F
::::::::::::::
       common /snglr/ sone(5 000)
::::::::::::::
```

```
comsdc.F
::::::::::::::
      common /con/ const,constnew,ratio(2000)
      real*4 const,constnew,ratio
::::::::::::::
comsdx.F
::::::::::::::
c
      common /idmv/ mvkl,mvku,mvks,mvjl,mvju,mvjs,mvil,mviu,mvis,mvmatr
     $ ,ieditn,nedit(5)
      common /idmvx/ trmst,trmnd,trmsp,tedit(5)
c
```

## Vita

Sedat Goluoglu was born in Ankara, Turkey on December 11, 1966. He graduated from Ankara High School in June 1984. The following August he entered the Hacettepe University, Ankara, where he received a Bachelor of Science degree in Nuclear Engineering in July 1989.

In August 1989 he accepted a Graduate Research Assistantship from Hacettepe University Nuclear Engineering Department and began study toward a Master of Science degree in Nuclear Engineering. This degree was awarded in March 1993.

In January 1992 he accepted a Graduate Research Assistantship from the University of Tennessee Nuclear Engineering Department and began study toward a Master of Science degree in Nuclear Engineering. This degree was awarded in August 1994. He is pursuing a Doctor of Philosophy degree in Nuclear Engineering at the University of Tennessee which he expects to receive in August 1997. Currently, he has accepted a research position at the University of Tennessee in order to contibute to the development of a time-dependent Monte Carlo neutron transport code with thermal-hydraulic feedback.