AN ABSTRACT OF THE THESIS OF

Bahram Nassersharif for the degree of Doctor of Philosophy

in Nuclear Engineering presented on December 13, 1982

Title: DESCRIBING FUNCTION THEORY AS APPLIED TO THERMAL AND NEUTRONIC PROBLEMS

Abstract approved: Redacted for Privacy

K. L. Peddicord

Describing functions have traditionally been used to obtain the solutions of systems of ordinary differential equations. In this work the describing function concept has been extended to include nonlinear, distributed parameter partial differential equations. A three-stage solution algorithm is presented which can be applied to any nonlinear partial differential equation. Two generalized integral transforms were developed as the T-transform for the time domain and the B-transform for the spatial domain. As specific applications of the solution technique two cases are considered: the heat conduction equation and the neutron diffusion equation.

The thermal diffusion describing function (TDDF) is developed for conduction of heat in solids and a general iterative solution along with convergence criteria is presented. The proposed solution method is used to solve the problem of heat transfer in nuclear fuel rods with annular fuel pellets. As a special instance the solid cylindrical fuel pellet is examined. A computer program is written which uses the describing function concept for computing fuel pin temperatures in the radial direction during reactor transients. It was found that the quasi-linear method used in the describing function method is as accurate as nonlinear treatments and as fast as true linearization methods.

The second problem investigated was the neutron diffusion equation which is intrinsically different from the first case. Although, for most situations, it can be treated as a linear differential equation, the describing function method is still applicable. A describing function solution is derived for two possible cases: constant diffusion coefficient and variable diffusion coefficient. Two classes of describing functions are defined for each case which portray the leakage and absorption phenomena. A study of the convergence criteria is also included. For the specific case of a slab reactor criticality problem the comparison between analytical and describing function solutions revealed an excellent agreement.

DESCRIBING FUNCTION THEORY AS APPLIED TO THERMAL AND NEUTRONIC PROBLEMS

by

Bahram Nassersharif

A THESIS

submitted to

Oregon State University

in partial fulfillment of
the requirements for the
degree of
Doctor of Philosophy

Completed December 1982

Commencement June 1983

APPROVED:

Redacted for Privacy

Professor of Nuclear Engineering in charge of major

Redacted for Privacy

Head of Department of Nuclear Engineering

Redacted for Privacy

Dean of Graduate School

Date thesis is presented December 13, 1982

Typed by Bahram Nassersharif for Bahram Nassersharif

ACKNOWLEDGEMENTS

I would like to thank my major professor, Dr. K. L. Peddicord, for his perpetual encouragement, guidance, and advice throughout the course of this work.

Special tributes go to my committee members, Dr. B. I. Spinrad, Dr. A. H. Robinson, Dr. M. N. L. Narasimhan, and Dr. J. A. Gardner, for assisting me to plan a proper program of study.

Computer funding provided by the Oregon State University Computer Center is gratefully appreciated.

This thesis is dedicated to my parents and my wife Ronda, for their understanding, patience, and love, without which, the completion of this work would have been impossible.

TABLE OF CONTENTS

Chapter Page

LIST OF FIGURES

Figure Page

Figure | Page

LIST OF TABLES

DESCRIBING FUNCTION THEORY AS APPLIED TO THERMAL AND NEUTRONIC PROBLEMS

I. INTRODUCTION

I.1 General Background

One of the most developed areas of computational mathematics and numerical methods is the subject of linear systems of equations. The complicated linear problem can be broken into a number of smaller problems which are simpler to solve. The set of solutions to the sub-problems can then be assembled together by using the superposition principle to obtain a solution to the overall problem. This method of breaking up the complicated problem into smaller sub-problems can also be applied to integro-differential systems. Once applied in this manner an approximation is necessary. The domain of the problem can therefore be broken into a number of smaller subdomains over which the differential terms are approximated. The resultant linear algebraic equations are then easily solved for the solution set.

The case of the nonlinear system is much more cumbersome. The most powerful method is iteration. But iterative methods have the problem of convergence. Therefore, along with an iterative method the convergence limits must also be derived. Except for a few useful theorems of differential calculus there are no standard methods in solving nonlinear integro-differential equations. Most approaches use a linearization technique and do not constitute general methods.

Finite element and finite difference methods are powerful techniques for computer adaptation of the differential equations and are based on simplification of the system over finite subdomains. We emphasize that for nonlinear problems an iteration sequence over each subdomain is required. Therefore, the subdomains must satisfy certain conditions depending on the nature of the nonlinearity.

Perturbation and variational techniques are elegant analytical methods which have been used for systems which can be linearized over some domain of interest. Collocation is another semi-analytical approach which is effective and uses a semi-linear (polynomial approximation) solution technique.

Thermal and neutronic characteristics of a nuclear reactor are of utmost importance in their design and safe operation. Large system codes like RETRAN and RELAP have been developed that use conventional numerical techniques to model thermal, mechanical and neutronic aspects of a nuclear power plant. Faster and more accurate technique are required to prevent unnecessarily long computation times for modeling reactor transients. The linearization techniques must also be replaced with nonlinear treatment of equations for accuracy of solutions.

I.2. Research Objectives and Activities

The purpose of this work is to extend the describing function method for determining the solution of nonlinear, distributed parameter partial differential equations. The theory developed is then used to obtain solution algorithms for three dimensional transient thermal and neutron diffusion problems. The analytical results can then be used for code development by utilizing numerical techniques.

Chapter II contains a brief history of the use of describing functions. The transfer and describing function concepts are introduced and similarities and differences between them are discussed. The new approach is then introduced and a general solution scheme for partial differential equations is outlined.

The methodology developed in Chapter II is then applied to the heat conduction equation in Chapter III. An analytical development of the iterative method and conditions of convergence are presented. The analytical theory is then demonstrated by a nonlinear one dimensional transient problem. A computer program was written which uses the describing function method to calculate the temperature field within a nuclear fuel element.

The describing function solution for the neutron diffusion equation is developed in Chapter IV. This section further illustrates the application of describing functions to a wider class of partial differential equations. The example provided, a slab reactor problem, deviates slightly from the general theory developed within the chapter and is an extension to the theory presented.

Chapter V contains a summary of the work and the conclusions. Also provided are suggestions for future research on applications of the describing function theory.

The appendices summarize the basic laws of continuum mechanics and mathematical methods used in this work.

II. THE DESCRIBING FUNCTION CONCEPT

II.1. Introduction

The theory of describing functions of a nonlinear system is based on the transfer function concept for linear systems. We therefore start our discussion of the describing function concept by briefly summarizing transfer functions.

The most straightforward case is that of linear systems with constant coefficients. When we use Laplace (or Fourier) transforms in such chains of systems: the transform of the overall system is simply the generalized product of the transforms of the individual elements. This specialized form of the super position principle is known as the convolution theorem. More specifically consider the time dependent solution to an impulse source g(t). This solution is a Green's function. The response to any source can be obtained by integrating all the elementary solutions in the form

$$X(t) = \int_0^t g(t-\lambda)\ S(\lambda)\ d\lambda \qquad (t >= 0), \qquad \text{II.1}$$

where the integral upper limit t, is the latest time that a source can affect the response. It is convenient to extend this upper limit to ∞ which we do artificially by introducing the Heaviside function (a unit step function), h(t), which is zero before t = 0 and unity for t > 0 (for further information see Braun, 1978, Kreyszig, 1979, Wylie, 1975). Thus

$$L\,[X(t)] = \int_0^\infty e^{-st}\,[\int_0^\infty h(t-\lambda)\ g(t-\lambda)\ S(\lambda)\ d\lambda]\ dt$$

$$= \overline{g}(s) \int_0^\infty e^{-st}\ S(t)\ dt = G(s)\ \overline{S}(s) = \overline{X}(s) \qquad \text{II.2}$$

We observe that for a Dirac delta distribution the response will be equal to the transfer function itself as

$$\overline{X}(s) = G(s), \qquad \text{for } S(t) = \delta(t). \qquad \text{II.3}$$

The more general case will involve arbitrary initial conditions and driving functions. Just as G(s) is the system response to an impulse source, we may see that G(iw) is the system response to a sinusoidal excitation of unit amplitude at frequency w, an observation which has made it possible for the experimentalists to determine G(iw). Therefore, if a sinusoidal input of frequency w is applied to a linear feedback system, the time-dependent component of the output will have the same frequency as the input.

II.2. Traditional Methods

Describing functions have traditionally been used for solution of ordinary differential equation systems such as those given by point reactor kinetics theory (for example see Smets, 1959, Wasserman, 1962, Akcasu, 1967, Schmidt, 1970, and Akcasu, 1972) and control theory (Gelb, 1968). Linear, constant property assumptions have been made in most cases studied in the past. For example, Gyftopoulos and Smets (1959) derived transfer functions for a flat plate reactor and a countercurrent flow heat exchanger. Iriarte (1960), and Zaalouk (1974) derived approximate transfer function expressions for cylindrical fuel elements. Iriarte (1960) had assumed a spatially uniform heat source, whereas, Zaalouk (1974) used a modified Bessel function of the first kind for the heat source given by neutron diffusion theory. The results of both papers were given for fixed values of thermal properties, fuel dimensions and heat transfer coefficient. Kendall (1975) used approximate transfer functions to calculate transient temperatures in simulated fuel rods. He used convolution of linearly varying inputs with first order logs to obtain the time response. Green and Kornfilt (1978) applied z-transforms to a set of approximate transfer functions to obtain the time response.

Guidotti (1982) used a quasi-linear method to derive a describing function for calculating one dimensional transient temperature profiles within a solid cylindrical fuel pin and the ANS 5.1 standard decay heat curve (June 1978). His exponential describing function produced excellent results when compared to a finite element code (COYOTE) for a reactor scram transient. Nielsen (1980) used the describing function code developed by Guidotti (1980) for transient fuel pin temperature calculation in a finite difference sense. He interfaced the describing function code with a finite difference fuel rod analysis code (FREY). He discovered that when the finite difference temperature subroutine of FREY was replaced with the describing function subroutine the running time remained practically constant. Nielsen's findings proved that for the same number of intervals the finite difference and describing function methods have the same efficiency. However, same results could have been obtained by the describing function method over the entire domain in the amount of time necessary for a finite difference method to complete its calculation for only one interval. Based on the aforementioned studies we shall start our discussion by considering linear and nonlinear systems and a simple example to follow.

The nonlinear system differs from the linear system in two respects. First, if the feedback system includes a nonlinear element, higher harmonics of the input frequency will be present in the output. Second, the amplitude and phase of the output terms depend nonlinearly on the amplitude of the input signal.

We can illustrate this by considering a nonlinear switch as shown in Figure II.1(a) which is designed to move a control rod in or out of the reactor in response to a demand signal, only when the demand exceeds some deadband D.

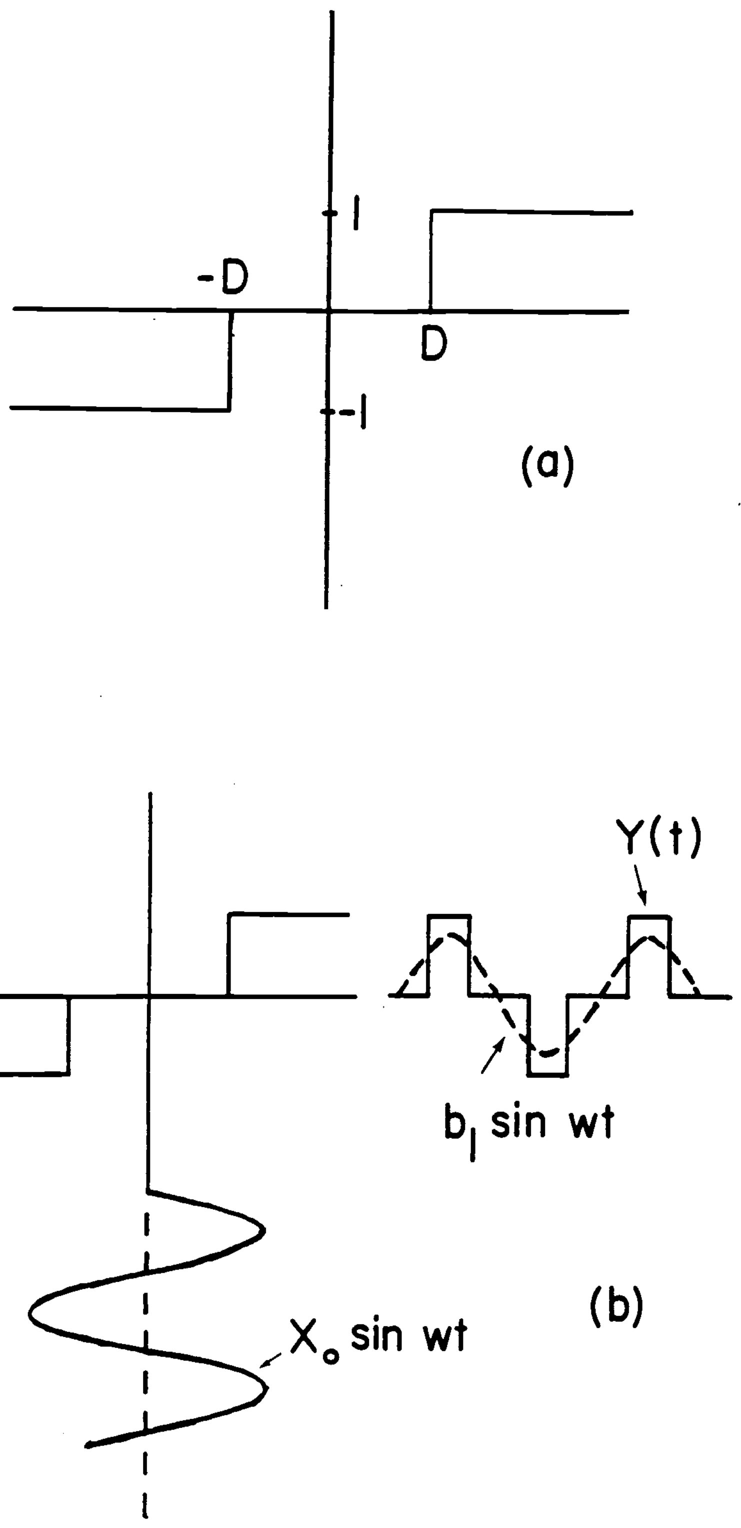


Figure II.1

Two-way switch with deadband for describing function

The output Y(t) for an oscillatory input $X(t)=X_o \sin wt$ is readily determined as a function of the ratio X_o/D and is illustrated in Figure II.1(b).

The Fourier series representation of Y(t) is evidently a sine series with no cosine component, so

$$Y(t) = \sum_{\ell} b_{\ell} \sin(\ell wt) \qquad \text{II.4}$$

and the fundamental mode has an amplitude b_1 obtained from the orthogonality condition

$$b_1 \int_0^{\pi} \sin^2(wt)\, d(wt) = \int_0^{\pi} Y \sin(wt)\, d(wt)$$

$$= \int_{\theta}^{\pi} \sin(wt)\, d(wt), \qquad \text{II.5}$$

where

$$\theta = \sin^{-1}(D/X_o).$$

Hence

$$b_1 = \frac{4}{\pi}\sqrt{1 - [D/X_o]^2} \qquad \text{for } X >= D$$

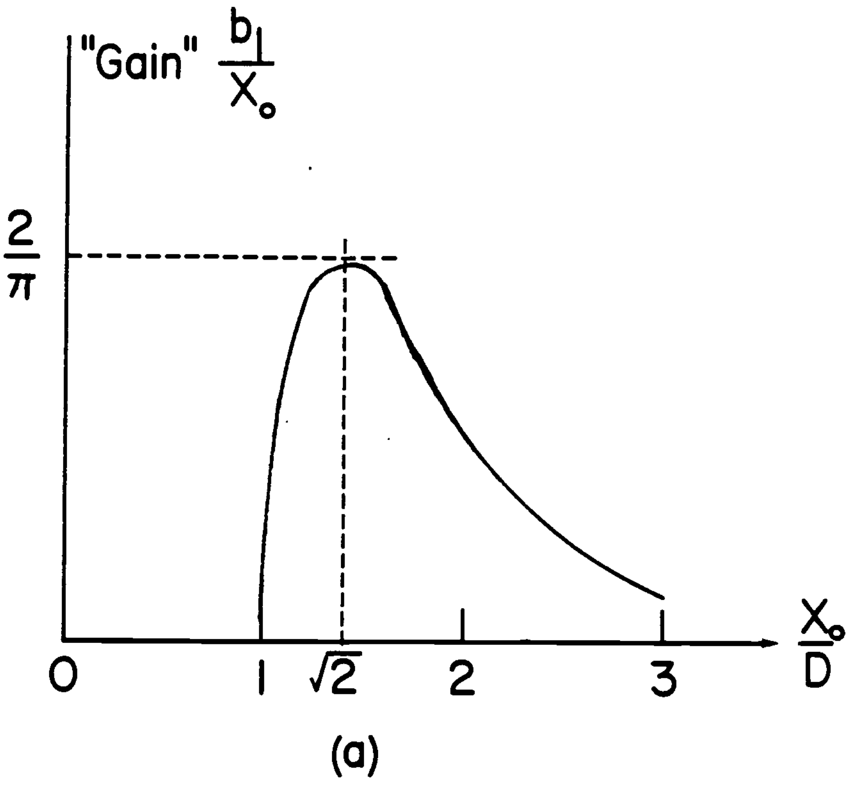
$$= 0 \qquad \text{for } X < D \qquad \text{II.6}$$

Figures II.2(a) and II.2(b) illustrate the phase and amplitude of the describing function. We note that the phase is always zero and the amplitude, although dependent on X_o/D, is independent of frequency w. The gain is zero until such time as the input signal exceeds the deadband; it rises quickly to a peak then falls off with increasing input signal since the output signal is limited to amplitude one.

The describing function is therefore an extension of the transfer function concept for nonlinear systems.

II.3. The Case of Nonlinear Partial Differential Equations

We define the describing function D(w,I) of a nonlinear element as the ratio of the fundamental component of the output to the fundamental component of the input, In general, for a nonlinear element, D is a function of the frequency w and the input amplitude I. For the limit of small I, the equations describing the behavior of the system may be linearized, and D becomes independent of I. In this case D(w) is equal to G(w), the transfer function of the linearized system.



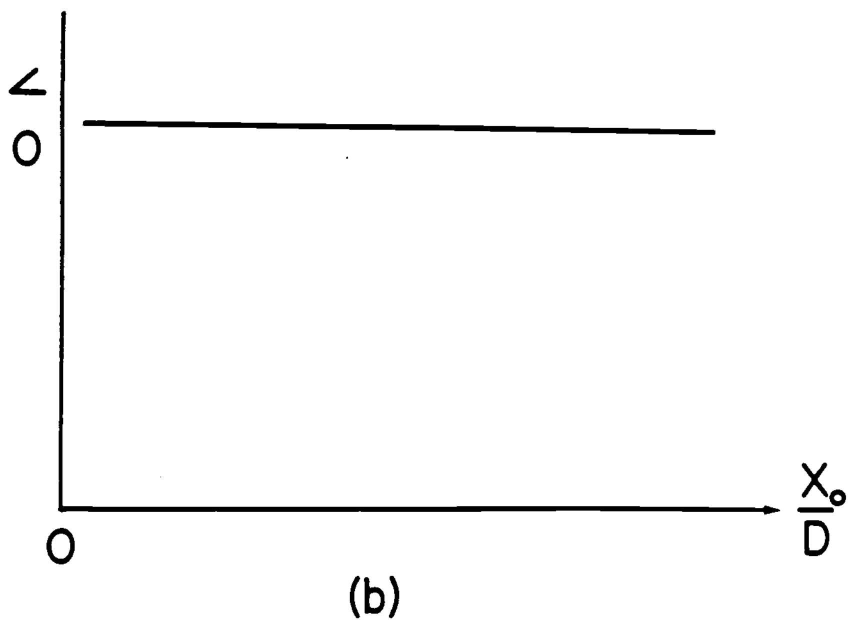


Figure II.2

Describing function for
two-way switch with deadband D

Based on the discussions of the last section we shall obtain a useful approach for derivation of describing functions of nonlinear systems represented by partial differential equations. The following method applies to nonlinear feedback systems whose characteristics do not change with time. It is necessary to know explicitly the integro-differential equations governing the behavior of each nonlinear element in the system. However, with this method more than one nonlinear element may be included in the system, any periodic input may be applied, and no restriction is placed on the harmonic content of the feedback signal.

The procedure consists of first combining the integro-differential equations for all the elements of the system into a single nonlinear integro-differential equation. The input and output elements are then expanded in appropriate form and substituted back into the composite system equation. This leads to a set of coupled algebraic equations in the expansion coefficients of the output signal either by an iterative or by a perturbation procedure. The describing function is finally determined as the ratio of the fundamental component of the output to the fundamental component of the input. This definition is directly used in deriving an explicit form for the describing function.

In the case of nonlinear systems which are represented by a single partial differential equation we need to be more specific about our definition. Therefore, we define the describing function as the ratio of integral transform of the output to the integral transform of the input.

The describing function method in a sense is used to replace a nonlinear operator which depends upon certain properties of the input. The method is therefore based upon quasi-linearization and must be distinguished from true linearization which yields a linear operator that is independent of the input. Consequently, the describing function is valid over the entire domain of the problem.

Figure II.3 illustrates the output of true linearization (or transfer function) as compared to the output of the actual nonlinear system to be approximated. The curve labeled "true gain" represents the actual nonlinear operator between input and output. The true system output is obtained by mapping the system input through the true gain curve and is plotted versus time. We note that the time domain of input and output profiles is identical.

The concept of true linearization involves the selection of a nominal value of input about which the true input is assumed to vary linearly. For time values in the domain and infinitesimally close to the nominal value, the true output and the linearized output are close. However, because the slope of the linearized gain is constant over the full range of input, the method produces a poor result for points outside the small linearization subinterval for which it is valid. The approximation is good only in a close neighborhood of the linearization nominal.

One method of overcoming the inherent limitations in the linearization process (transfer function method) is by using quasi-linearization (describing functions). This differs from true linearization in that instead of using a constant linearization

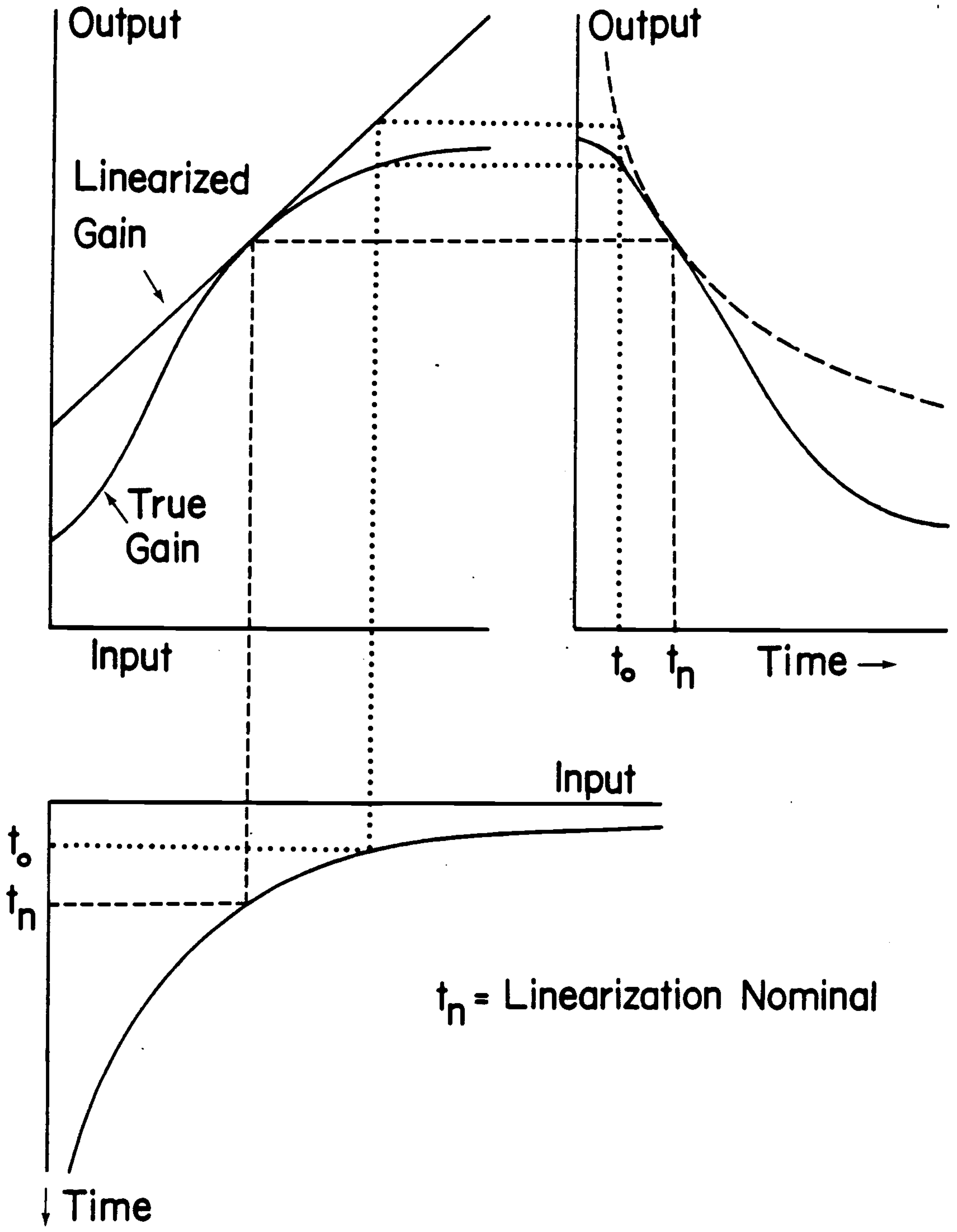


Figure II.3

Linearization of a feedback system

nominal, we determine an input dependent nominal which is valid over the entire domain of input (see Figure II.4). The variable nominal does not have to be identical to the input but it should only approximate it in both shape and magnitude. Consequently, the quasi-linear gain varies with the quasi-linear input resulting in good approximation of the true gain over the full range of input. Therefore, the true output and the quasi-linearized output are in good agreement over the entire domain.

Next we will propose a method of converting the partial differential equation into a feedback system such that the above definitions can be used.

II.4. General Solution Algorithm

Most nonlinear partial differential equation can be converted into a linear form by introducing proper variable substitutions. The resultant linear differential equation will then contain more than one unknown variable. The equations for variable transformations will serve as the additional constraints necessary to close the system. In this manner the nonlinearities are extracted from the system.

Integral transforms can then be used to transform the linear differential equation into a linear algebraic form. The nonlinear variable definitions together with the linear algebraic equation will form the feedback system desired. Figure II.5 illustrates this concept. The describing function of the feedback system can be

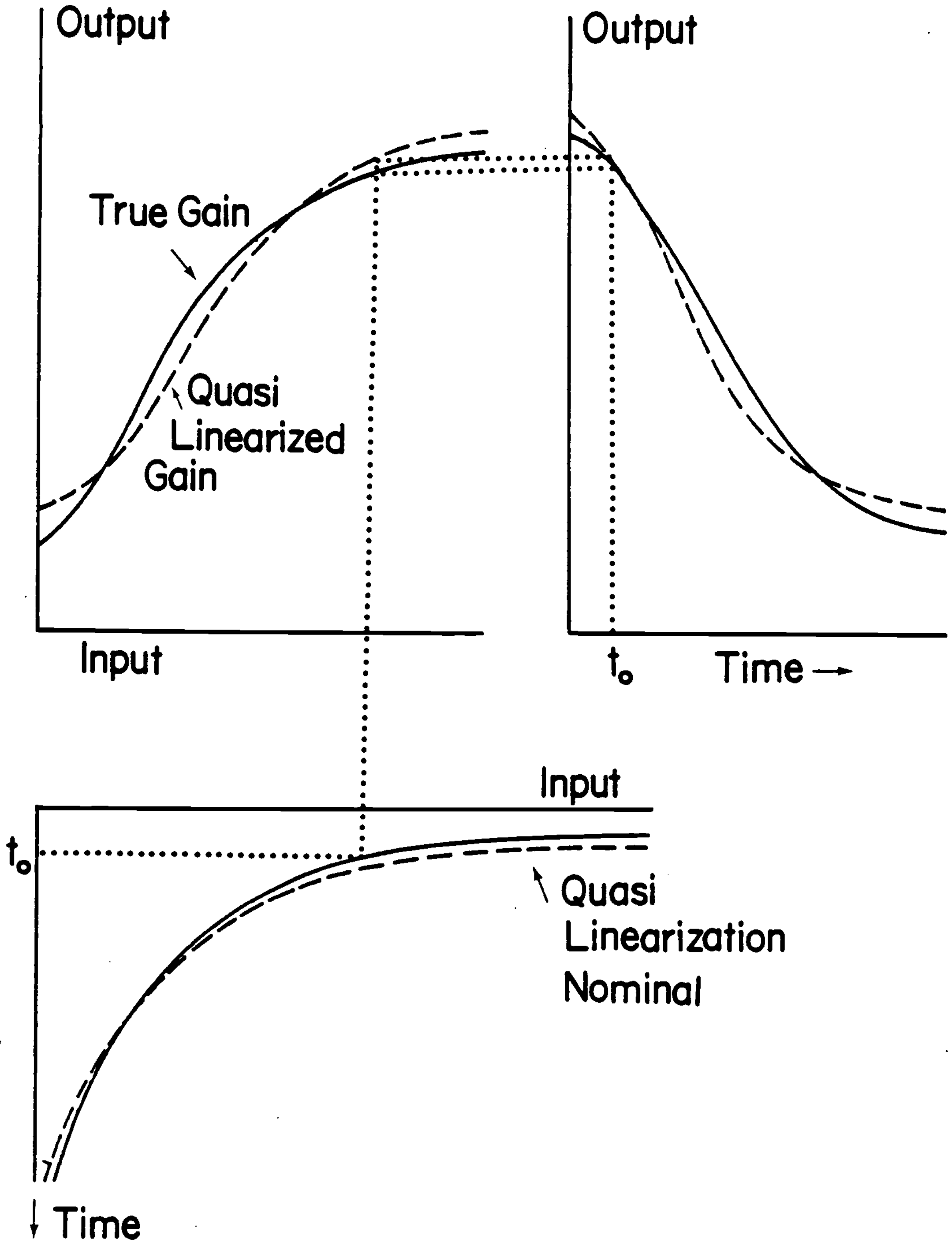


Figure II.4

Quasi linearization of a feedback system

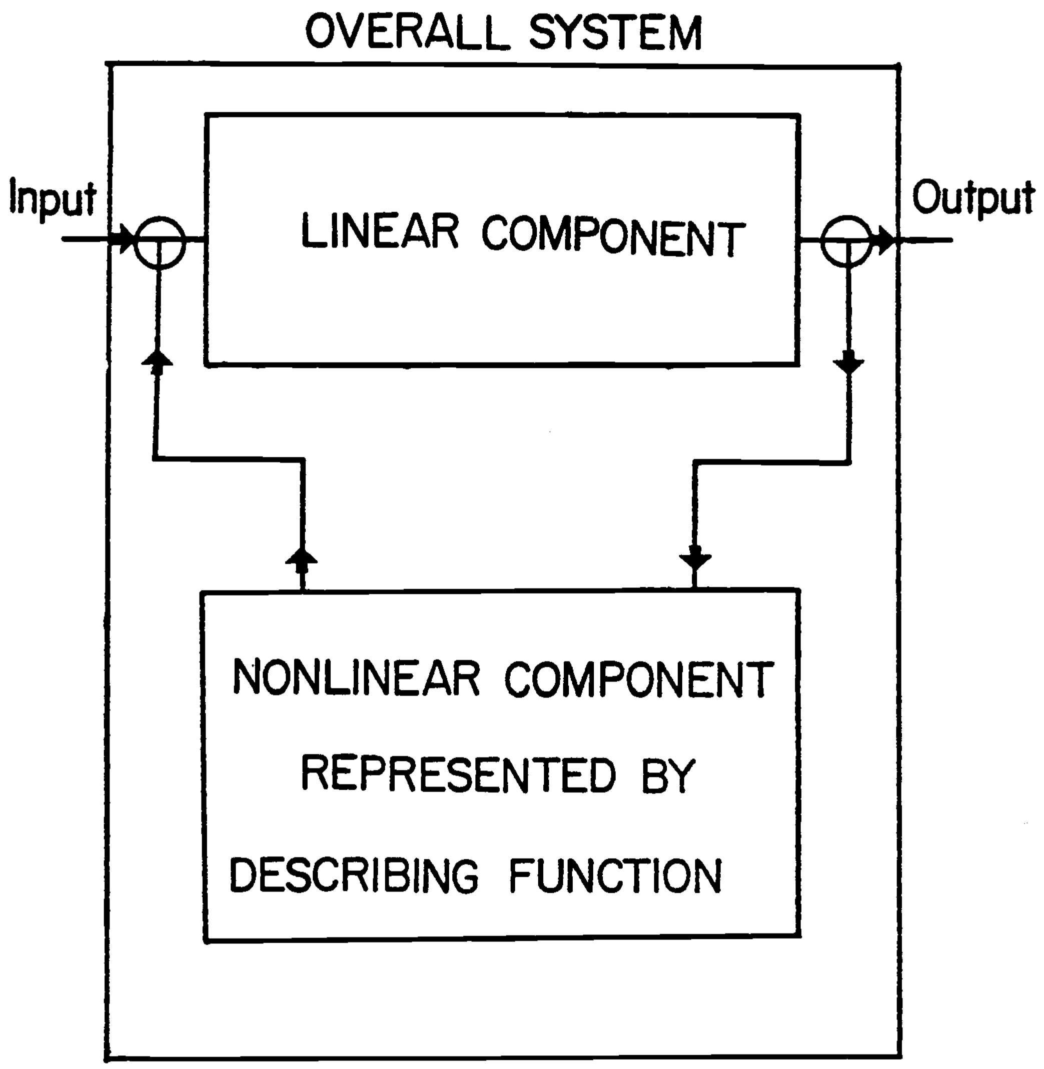


Figure II.5

The feedback system

determined from the aforementioned definitions and an assumed form for the input.

Once the describing function and the linear component are explicitly determined an iterative method can be used to obtain the solution in the transformed space. The final step is an inversion of the transformed solution back into the real space.

In the chapters that follow we will illustrate the use of this approach for two important partial differential equations in nuclear engineering: the heat conduction equation and the neutron diffusion equation.

III. DESCRIBING FUNCTION METHOD AS APPLIED TO TO THE HEAT CONDUCTION EQUATION

III.1 Introduction

The natural laws of conduction, like any other law of physics, are established inductively on the basis of evidence collected from experimental observations. The heat conduction equation is a special case of the more general law of conservation of energy (the first law of classical thermodynamics). A continuum mechanical derivation of this law along with other fundamental laws of continuum mechanics are presented in Appendix D. The conservation of thermomechanical energy equation in its raw form is not solvable. There are too many unknowns for the number of equations. A description which provides a good approximation of the experimental observations is Fourier's law of conduction which relates the heat flux vector to the temperature gradient through a constant of proportionality called the conductivity. In this section we will consider the conduction of heat in solids.

The study of conduction of heat in solids is based on the general form of the heat conduction equation. We ignore all mechanical forms of transfer of energy and consider only those that are purely thermal and due to internal agitation of the molecules and atoms. In this sense we apply a traditional continuum mechanical approach to the topic of heat conduction in solids.

The fundamental equation of concern is the form of heat transport equation into which the Fourier's law of conduction has been incorporated, namely,

$$\mathrm{div}(k \,\mathrm{grad}\, T(\mathbf{r},t)) + q'''(\mathbf{r},t) = \rho C \frac{\partial T(\mathbf{r},t)}{\partial t}, \qquad \text{III.1}$$

where

k	= thermal conductivity,
T	= temperature,
ρ	= density,
C	= heat capacity,
q'''	= heat source.

This equation is, in general, a nonlinear partial differential equation. It is observed that the solution of the heat conduction equation in four dimensions (3 spatial dimensions and one time dimension) is not a trivial task. Various methods of solution for this equation exist, most of which assume a linearized form of the equation and are essentially extensions of solution techniques for linear partial differential equations. The describing function technique, however, utilizes a quasi-linearization approach for a fast converging iterative solution.

III.2. Application of the describing function method

We recall from Chapter II the basic solution steps in the describing function method.

III.2.1. **Variable transformations**

The heat conduction equation may be reduced to a linear differential equation by introducing Kirchhoff transformations (Arpaci, 1966),

$$I(T) = \int_{T_{ref}}^{T} k(\lambda)\, d\lambda, \qquad \text{III.2}$$

and

$$U(T) = \int_{T_{ref}}^{T} \rho\, C(\lambda)\, d\lambda, \qquad \text{III.3}$$

where T_{ref} denotes a convenient reference temperature. We use the terminology integral conductivity and integral heat capacity for I and U respectively. We note that I and U are both functions of temperature.

The two new variables incorporate the nonlinear terms of the original equation. Therefore, we have the equation

$$\frac{\partial U(\mathbf{r},t)}{\partial t} = \text{Lap } I(\mathbf{r},t) + q'''(\mathbf{r},t). \qquad \text{III.4}$$

This is a linear partial differential equation with two unknown functions I and U. We, therefore, have simplified the problem only in appearance.

III.2.2. **Integral transformations**

From the methods developed in Appendix B for transformation of the time differential (the T-transform) and the Laplace operator (the B-transform of the first kind), we have

$$\bar{f}(s) = T[f(t)] = \int_0^\infty e^{-st} f(t)\, dt, \qquad \text{III.5}$$

and

$$\bar{g}(\boldsymbol{\rho}) = B_2[g(\mathbf{r})] = \int_{\text{Vol}} g(\mathbf{r})\, K(\mathbf{r},\boldsymbol{\rho})\, dV, \qquad \text{III.6}$$

where we have chosen Laplace transform for simplicity. The time differential is transformed as

$$T\left\{\frac{\partial U(\mathbf{r},t)}{\partial t}\right\} = s\overline{U}(\mathbf{r},s) - U(\mathbf{r},0), \qquad \text{III.7}$$

and the Laplacian is transformed as

$$B\,[\text{Lap } I(\mathbf{r},t)] = -\,\Omega(\boldsymbol{\rho})\,\overline{I}(\boldsymbol{\rho},t). \qquad \text{III.8}$$

The double transformed linear algebraic equation becomes

$$s\overline{U}(\boldsymbol{\rho},s) - \overline{U}(\boldsymbol{\rho},0) = -\,\Omega(\boldsymbol{\rho})\,\overline{I}(\boldsymbol{\rho},s) + \overline{q}'''(\boldsymbol{\rho},s). \qquad \text{III.9}$$

There are two unknowns in this algebraic equation $\overline{I}(\boldsymbol{\rho},s)$, the double transformed integral conductivity and $\overline{U}(\boldsymbol{\rho},s)$ the double transformed integral heat capacity. We note that the B-transformed initial integral heat capacity $\overline{U}(\boldsymbol{\rho},0)$ is known from initial conditions of the problem. The double transformed heat source term is also known from the problem statement. A relationship between the two unknowns of the problem U and I is therefore necessary in order to hope for a solution.

III.2.3. The Thermal Diffusion Describing Function

From Chapter II we recall the definition of a describing function as "the ratio of the integral transform of the output to the same integral transform of the input." Therefore, we define

$$D_{th} = \frac{\overline{U}(\rho,s)}{\overline{I}(\rho,s)}, \qquad \text{III.10}$$

as the thermal diffusion describing function and we note that for a linear problem by using the fundamental theorem of integral calculus we get

$$D_{th} = \frac{1}{\alpha}, \qquad \text{III.11}$$

where α is the thermal diffusivity constant given by

$$\alpha = \frac{k}{\rho C}. \qquad \text{III.12}$$

All mathematical operations performed on the heat conduction equation have been exact and no approximations are introduced. It is observed that the thermal diffusion describing function (TDDF) is a function of input parameters and it "describes" the nature of the nonlinearity over the entire domain of the problem. In a linear problem the thermal diffusivity is the input-to-output gradient descriptor or gain.

Existence and uniqueness of solutions of partial differential equations are well studied fields and a variety of references on these subjects may be found in standard text books (for example see Braun, 1978, Fucik, 1980, Schwabik, 1979). The overall system can be represented by two equations

$$\overline{I}(\boldsymbol{\rho},s) = \frac{\overline{U}(\boldsymbol{\rho},0) + \overline{q}'''(\boldsymbol{\rho},s)}{\Omega(\boldsymbol{\rho}) + sD_{th}}, \qquad \text{III.13}$$

and

$$D_{th} = \frac{\overline{U}(\boldsymbol{\rho},s)}{\overline{I}(\boldsymbol{\rho},s)}. \qquad \text{III.14}$$

The solution derived by the describing function method would have been an exact solution if we had an exact expression for the TDDF. However, like most other nonlinear solution techniques the describing function method relies upon an iterative method. The iteration is started by assuming an input function $I_o(\mathbf{r},t)$. The describing function is then determined and the linear algebraic equation is used to find a new function $I_1(\mathbf{r},t)$. If all goes well as the number of iterations increases the iterative input functions will converge to a common function which is the solution. We note that a unique solution exists because we have incorporated the boundary and initial conditions into the problem.

The entire iteration cycle is performed in the transformed domain; however, the convergence test must be satisfied in both the real and transformed spaces of the problem. Figure III.1

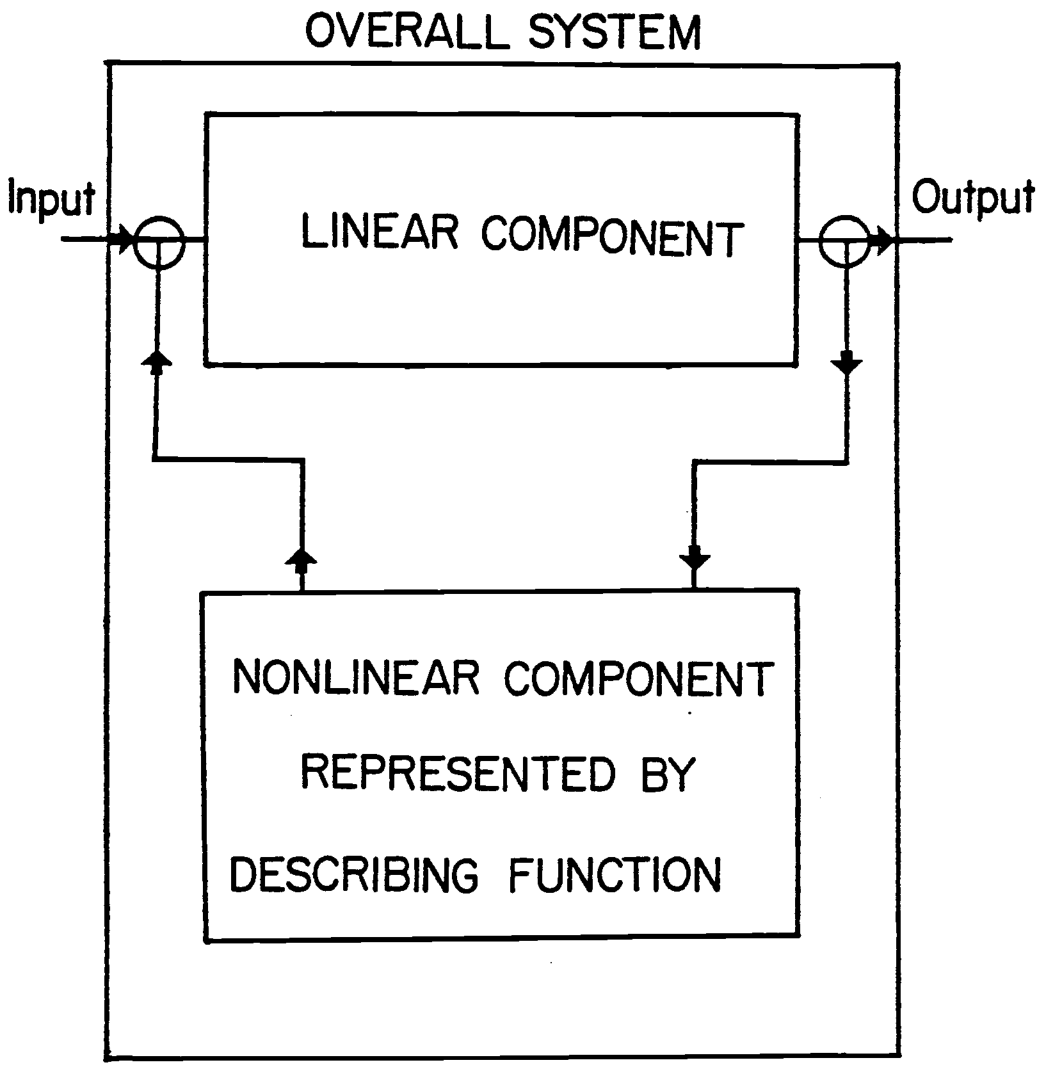


Figure III.1

Separation of the nonlinear heat conduction equation into linear and nonlinear components in the form of a feedback system illustrates the separation of the overall system into linear and nonlinear components. This is the nonlinear feedback model upon which the describing function solution is based.

III.3. Convergence in General

As mentioned earlier the iterative method is started by an initial estimate of the solution $I_o(r,t)$. This will result in a describing function in the form

$$D_{th_o} = \frac{\overline{U}(\overline{I}_o(\rho,s))}{\overline{I}_o(\rho,s)}. \qquad \text{III.15}$$

The "new" solution is therefore

$$\overline{I}_1(\rho,s) = \frac{\overline{U}(\rho,0) + \overline{q}'''(\rho,s)}{\Omega(\rho) + sD_{th_o}}. \qquad \text{III.16}$$

For the nth iteration we can write

$$\overline{I}_n(\rho,s) = \frac{\overline{U}(\rho,0) + \overline{q}'''(\rho,s)}{\Omega(\rho) + sD_{th_{n-1}}}. \qquad \text{III.17}$$

So in terms of the previous iterations we have

$$\overline{I}_n = \overline{I}_o + (\overline{I}_1 - \overline{I}_o) + (\overline{I}_2 - \overline{I}_1) + \ldots + (\overline{I}_n - \overline{I}_{n-1}),$$

III.18

or

$$\overline{I}_n = \overline{I}_o + \sum_{\ell=1}^{n} (\overline{I}_\ell - \overline{I}_{\ell-1}) \qquad \text{III.19}$$

A necessary condition for convergence is that

$$\lim_{n\to\infty} \overline{I}_n(\rho,s) \quad \text{exists.} \qquad \text{III.20}$$

This requires the convergence of the series in equation III.19. A necessary and sufficient condition for convergence of the series is the convergence of partial sums, i.e.,

$$\lim_{\ell\to\infty} |\overline{I}_\ell - \overline{I}_{\ell-1}| \longrightarrow 0. \qquad \text{III.21}$$

From equation III.17 we can write

$$|\overline{I}_\ell(\rho,s) - \overline{I}_{\ell-1}(\rho,s)| =$$

$$|\overline{U}(\rho,0) + \overline{q}'''(\rho,s)| \left| \frac{s(D_{th_{\ell-1}} - D_{th_{\ell-2}})}{(\Omega(\rho)+sD_{th_{\ell-1}})(\Omega(\rho)+sD_{th_{\ell-2}})} \right|. \qquad \text{III.22}$$

From differential calculus we can write

$$|D_{th_{\ell-1}} - D_{th_{\ell-2}}| <= |D'_{th}||\overline{I}_{\ell-2} - \overline{I}_{\ell-1}|, \qquad \text{III.23}$$

where

$$D'_{th} = \frac{dD_{th}}{d\overline{I}}. \qquad \text{III.24}$$

Therefore

$$|\overline{I}_{\ell} - \overline{I}_{\ell-1}| <= \alpha_c|\overline{I}_{\ell-1} - \overline{I}_{\ell-2}|, \qquad \text{III.25}$$

where

$$\alpha_c = \left|\frac{s(\overline{U}(\rho,0) + \overline{q}'''(\rho,s))D'_{th}}{(\Omega(\rho)+sD_{th_{\ell-1}})(\Omega(\rho)+sD_{th_{\ell-2}})}\right|, \qquad \text{III.26}$$

then it is obvious that

$$|\overline{I}_{\ell} - \overline{I}_{\ell-1}| <= \alpha_c^{\ell}\ |\overline{I}_1 - \overline{I}_o|. \qquad \text{III.27}$$

For convergence we need to show $\alpha < 1$, for all ρ and s in the domain. Convergence is strongly dependent on material properties, initial condition, the heat source and the input parameters. The convergence has a semilocal form (because of the heat source characteristics) and is dominated by the describing function and the input. Consequently, one would expect the convergence to be dependent on the form of the input. A global convergence does exist for a certain class of problems where any input guess will yield a converged solution (for example the case of no heat source or a constant heat source).

III.4. Nonlinear One Dimensional Transient Problem

A fundamental concern in nuclear reactor design and safety is effective heat removal from the fuel pins. There are close to 40,000 fuel rods in a typical pressurized water reactor. Analysis of temperature profile within fuel rods is an important part of reactor safety. We shall consider this problem as an example of application of describing function theory to the heat conduction equation.

III.4.1. **Problem Statement**

Figure III.2 shows the fuel pin cross section. For generality we have included a center hole in the fuel pellet. In our analysis we will make the following assumptions:

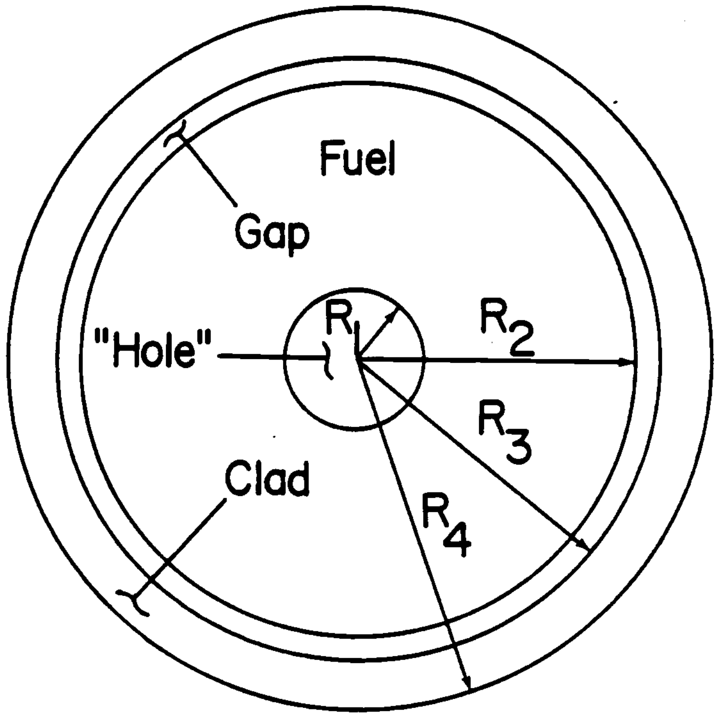


Figure III.2

Annular fuel pin geometry

1. Radial power generation in the fuel is only a function of radius and time.
2. No heat generation in the clad.
3. The heat capacity of the fuel-clad gap is neglected.
4. Thermal conductivity, heat capacity and density of fuel are known functions of temperature.
5. Thermal conductivity, heat capacity and density of clad are constant.
6. Coolant temperature is a function of time.
7. Gap conductance is a function of time.
8. The "hole" is simply a vacuum, i.e., no heat generation in the center hole.

Because of the above assumptions the clad and fuel region conduction problems are solved differently. The fuel region equation is a nonlinear partial differential equation whereas the clad region equation is linear. We will concentrate our efforts on the solution of fuel region problem and use results produced by Guidotti (1980) for the clad region. The fuel region equation can be written as:

$$\frac{1}{r}\frac{\partial}{\partial r}\left(r\, k_f(T_f)\frac{\partial T_f(r,t)}{\partial r}\right) + q'''(r,t) = \rho C(T_f)\frac{\partial T_f(r,t)}{\partial r}, \qquad \text{III.28}$$

here T_f is the fuel temperature and k_f is the fuel conductivity. The clad region governing equation is linear and can be expressed as:

$$\frac{1}{r}\frac{\partial}{\partial r}\left(r\frac{\partial T_c}{\partial r}\right) = \frac{1}{\alpha_c}\frac{\partial T_c}{\partial t} \qquad \text{III.29}$$

where T_c is the clad temperature and α_c is the clad thermal diffusivity. Equations III.28 and III.29 are both second order partial differential equations, therefore, four boundary conditions are necessary. These four boundary conditions are:

1. The heat flux is zero at the fuel inner radius. From Fourier's law of conduction this implies a zero temperature gradient.

$$\frac{\partial T_f}{\partial r}(R_1,t) = 0 \qquad \text{III.30}$$

2. Continuity of heat flux at the fuel-clad interface.

$$- R_2 K_f(T_f)\frac{\partial T_f}{\partial r}(R_2,t) = - R_3 k_c \frac{\partial T_c}{\partial r}(R_3,t) \qquad \text{III.31}$$

3. Temperature jump condition (gap conductance, or contact resistance) at the fuel-clad interface.

$$- k_f(T_f)\frac{\partial T_f}{\partial r}(R_2,t) = h_g[T_f(R_2,t) - T_c(R_3,t)] \qquad \text{III.32}$$

4. Convective boundary condition at the clad outer surface

$$- k_c \frac{\partial T_c}{\partial r}(R_4,t) = h\,[T_c(R_4,t) - T_\infty(t)] \qquad \text{III.33}$$

where T_∞ is the coolant temperature. This concludes our problem statement and we have all the necessary information, except for material property functions, to obtain a solution.

III.4.2 **Material properties**

Thermal properties of UO_2 fuel and zircaloy cladding have been measured and tabulated by a number of experimentalists. The following empirical relations are taken from MATPRO (1979). UO_2 thermal conductivity is shown in Figure III.3 and is given by

$$k_f(T) = \frac{4040.}{464+T} + 0.01216\ e^{0.001867T}, \qquad \text{III.34}$$

for 0°C < T < 1650°C, and

$$k_f(T) = 1.91 + 0.01216\ e^{0.001867T}, \qquad \text{III.35}$$

for 1650°C <= T < 2840°C, where

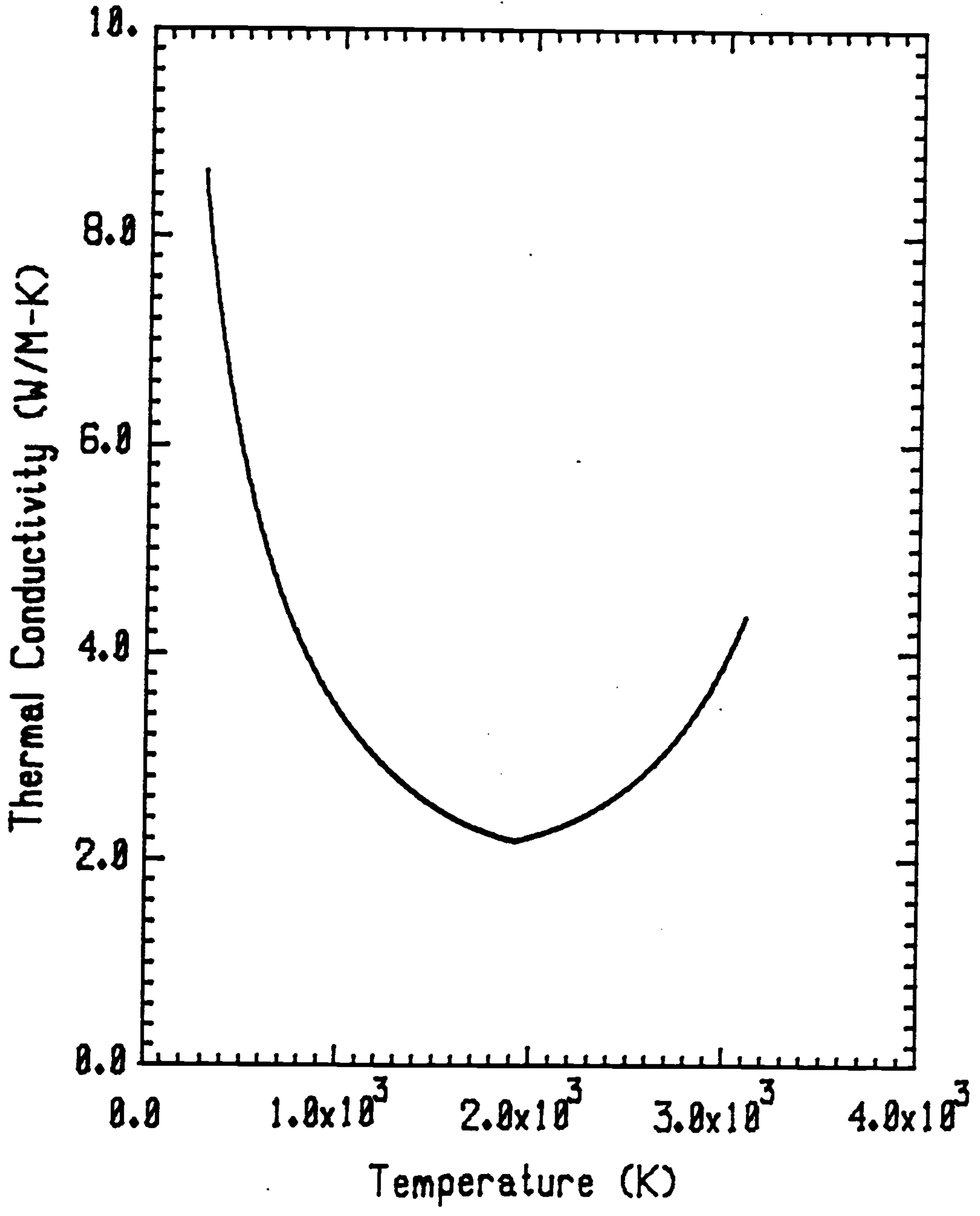


Figure III.3

Uranium dioxide thermal conductivity
as a function of temperature

k_f = UO_2 thermal conductivity $\frac{W}{m\ K}$,

T = temperature (°C).

Uranium dioxide specific heat is plotted in Figure III.4 for 95 percent theoretical density (ρ = 10.4 g/cm^3) and is given by

$$C(T) = 15.496 \left[\frac{K_1 \theta^2 e^{\theta/T}}{T^2 (e^{\theta/T} - 1)^2} + 2K_2 T + \frac{K_3 E_D e^{-E_D/RT}}{RT^2} \right], \qquad \text{III.36}$$

where

C = UO_2 specific heat $\frac{KJ}{Kg\ K}$,

T = temperature (K),

K_1 = 19.145 $\frac{J}{mol\ K^2}$,

K_2 = 7.8473 x 10^{-4} $\frac{J}{mol\ K^2}$,

K_3 = 5.6437 x 10^6 $\frac{J}{mol}$,

θ = 535.285 K,

E_D = 1.577 x 10^5 $\frac{J}{mol}$,

R = 8.314 $\frac{J}{mol\ K}$.

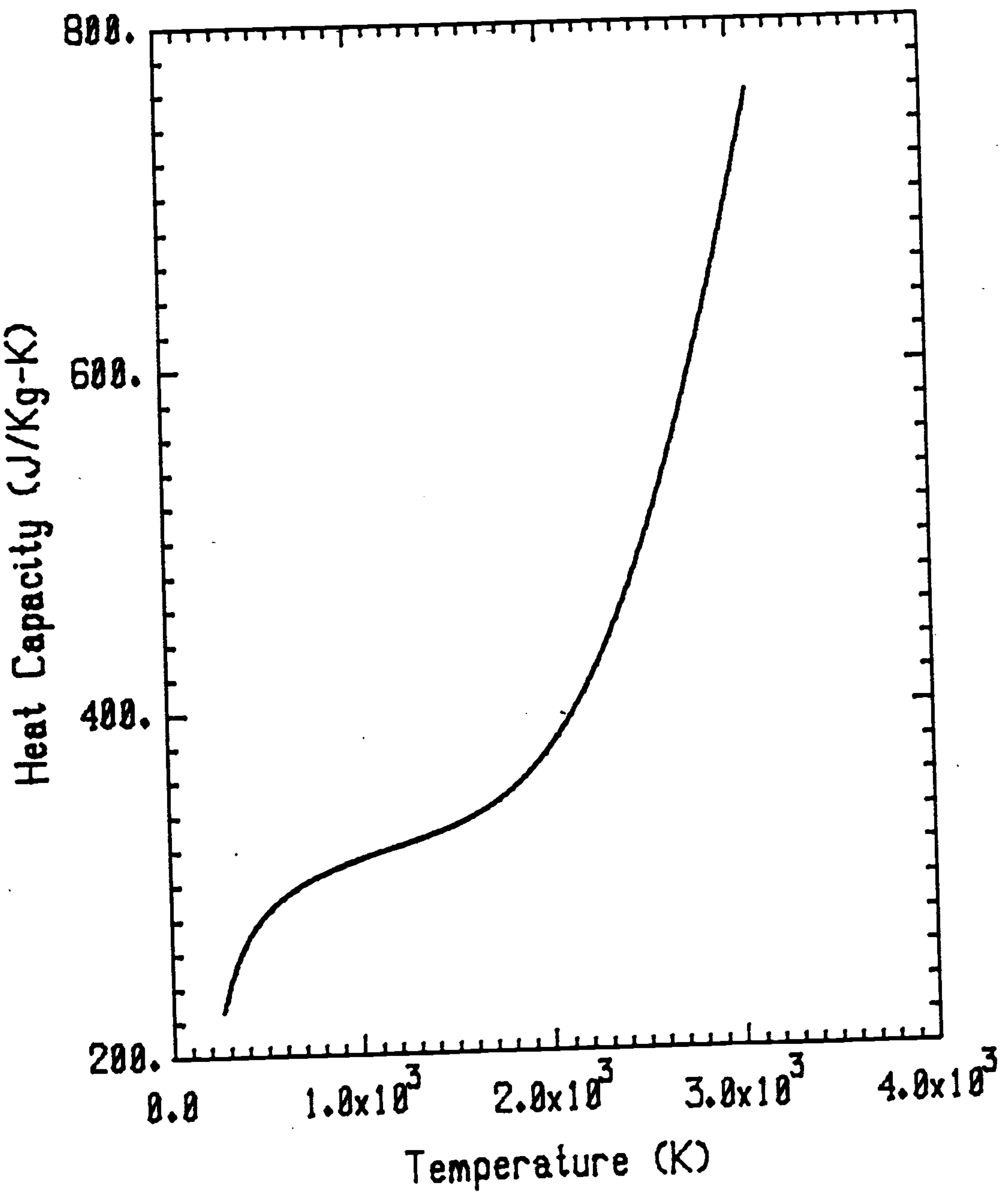


Figure III.4

Uranium dioxide heat capacity
as a function of temperature

The thermal properties of zircaloy clad are assumed constant for the following reasons:

1. The clad thickness is much smaller than the dimension of the fuel.
2. The clad temperature range is much smaller than the temperature range in the fuel.

The thermal property transformations, integral conductivity and integral heat capacity can be derived by direct integration of conductivity and heat capacity relations respectively. The integral conductivity for UO_2 is illustrated as a function of temperature in Figure III.5 and is given by

$$I(T) = 4040 \ln(464+T) + 6.513\, e^{0.001867T} - 24812 \qquad \text{III.37}$$

for $0°C < T < 1650°C$, and

$$I(T) = 1.91T + 6.513\, e^{0.001867T} + 1968.4 \qquad \text{III.38}$$

for $1650°C <= T < 2840°C$, where

I = UO_2 integral conductivity $\frac{W}{M}$,

T = temperature (°C).

The inverse integral conductivity expressions are also given by MATPRO:

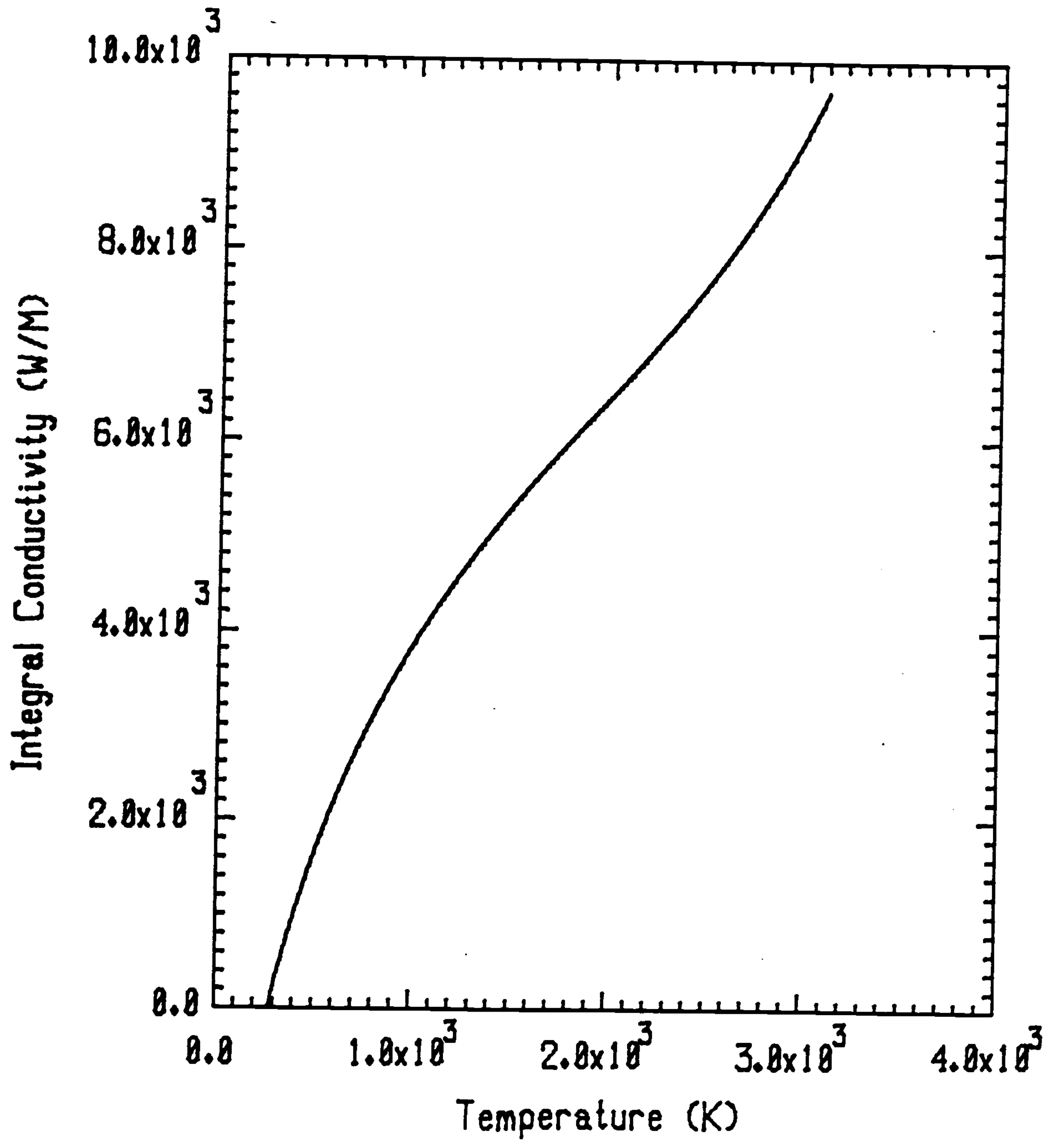


Figure III.5

Uranium dioxide integral conductivity as a function of temperature

$$T(I) = 464.8\, e^{I/4040} - 485.761 + 0.0134I - 3.51\times10^{-6}I^2 \qquad \text{III.39}$$

for 0°C < T < 1650°C, and

$$T(I) = -\, 2669.82 + 0.9122I + 3.55\times10^{-5}I^2 \qquad \text{III.40}$$

for 1650°C <= T < 2840°C. A more accurate inverse may be obtained by performing a polynomial regression on data generated by I(T). We have used the HIPLOT plotting and regression routine (Appendix F). The polynomial form can be expressed for 0°C < T < 2840°C as:

$$I(T) = \sum_{\ell=0}^{6} a_\ell\, T^\ell, \qquad \text{III.41}$$

and

$$T(I) = \sum_{\ell=0}^{7} b_\ell\, T^\ell. \qquad \text{III.42}$$

where the coefficients are given in Table III-1. Integral heat capacity expression is obtained by analytical integration of equation III.36 and is illustrated in Figure III.6. It is given by

TABLE III-1

Integral Conductivity Polynomial Regression Coefficients

6th order coefficients for integral conductivity
(correlation coefficient = 0.9991)

Order	Coefficient
1	$7.97465\text{x}10^{-2}$
2	$- 4.45429\text{x}10^{-5}$
3	$8.80830\text{x}10^{-9}$
4	$7.63618\text{x}10^{-12}$
5	$- 4.85220\text{x}10^{-15}$
6	$8.04943\text{x}10^{-19}$

7th order coefficients for inverse
(correlation coefficient = 0.9995)

Order	Coefficient
0	- 37.5779
1	23.7335
2	- 0.985593
3	$4.52817\text{x}10^{-2}$
4	$- 8.22123\text{X}10^{-3}$
5	$6.24382\text{x}10^{-5}$
6	0.648410
7	$- 2.24516\text{x}10^{-9}$

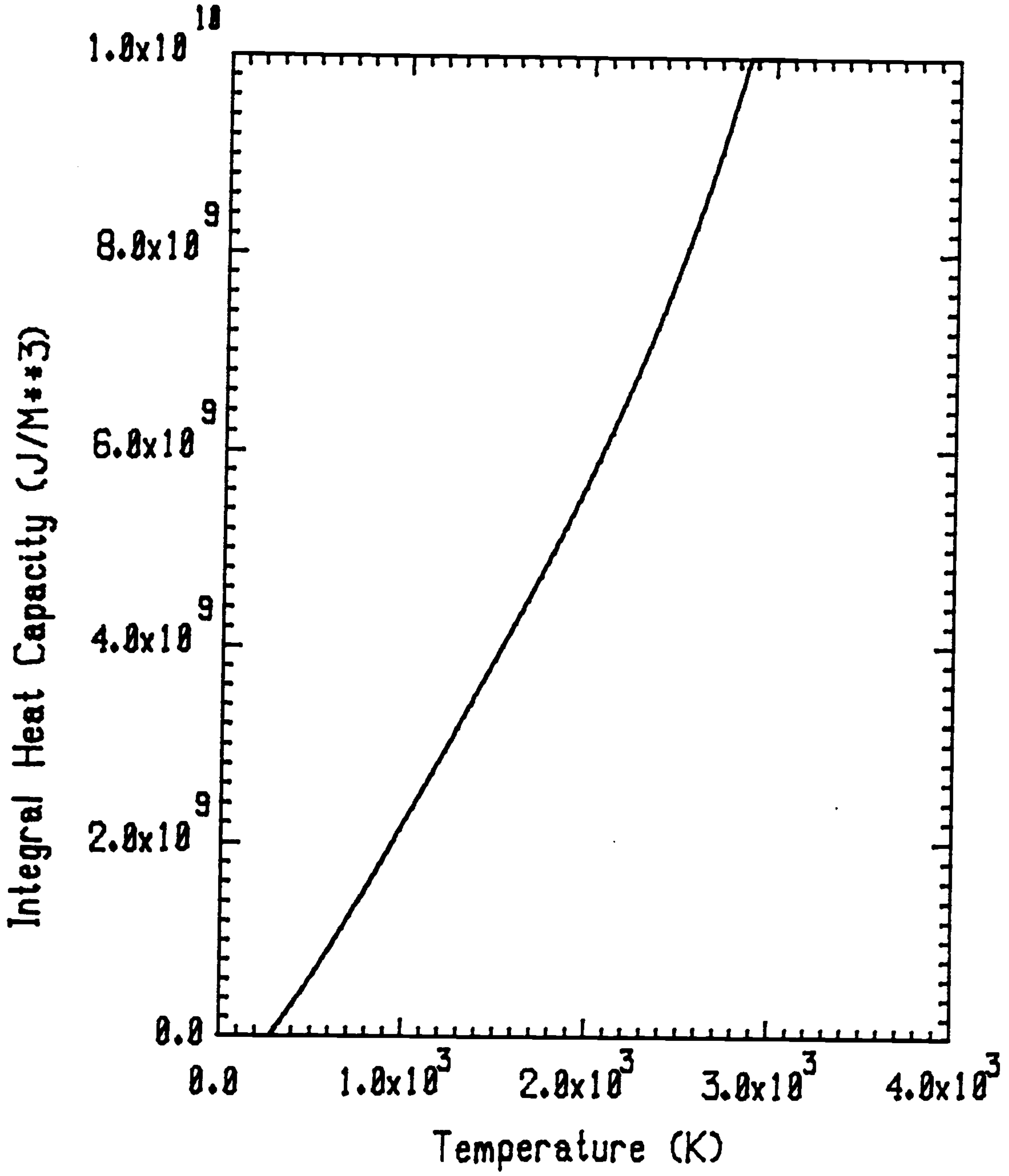


Figure III.6

Uranium dioxide integral heat capacity as a function of temperature

$$U(T) = 15.496\rho \left[\frac{K_1\Theta}{(e\Theta/T - 1)} + K_2T^2 + K_3e^{-(E_D/RT)} \right]_{T_{ref}}^{T} . \qquad \text{III.43}$$

Since both integral conductivity and integral heat capacity are functions of temperature they can be related. It is therefore possible to express integral heat capacity as a function of integral conductivity. This can be done by performing a polynomial regression on the numerical data obtained from equations III.37-III.43 as

$$U = U(I) = c_1I + c_2I^2 \qquad \text{III.44}$$

where

U = UO_2 integral heat capacity (J/m^3) relative to a fixed reference temperature of 0°C,

I = UO_2 integral conductivity (W/m) relative to a fixed reference temperature of 0°C,

c_1 = 143895.0 s/m,

c_2 = 114.284 s/W m.

The correlation coefficient for this regression was 0.999651. The correlation coefficient is a measure of the nature of the regression. The closer this coefficient is to one the better the regression fit. The quadratic polynomial does not contain a constant term, because both integral conductivity and integral heat capacity must vanish at the reference temperature, i.e.,

$$U(T_{ref}) = I(T_{ref}) = 0.$$

The integral heat capacity plot versus integral conductivity and the polynomial fit are illustrated in Figure III.7.

III.4.3. **Steady State Solution**

The steady state case is used as the initial condition for the transient problem, therefore, it is considered first. The governing equations in this case reduce to

$$\frac{1}{r}\frac{d}{dr}\left(r\frac{dI}{dr}\right) + q'''(r,0) = 0 \quad \text{in the fuel region,} \qquad \text{III.45}$$

and

$$\frac{1}{r}\frac{d}{dr}\left(r\frac{dT_c}{dr}\right) = 0 \quad \text{in the clad region,} \qquad \text{III.46}$$

and the boundary conditions are:

$$1. \quad \frac{dI}{dr}(R_1) = 0 \qquad \text{III.47}$$

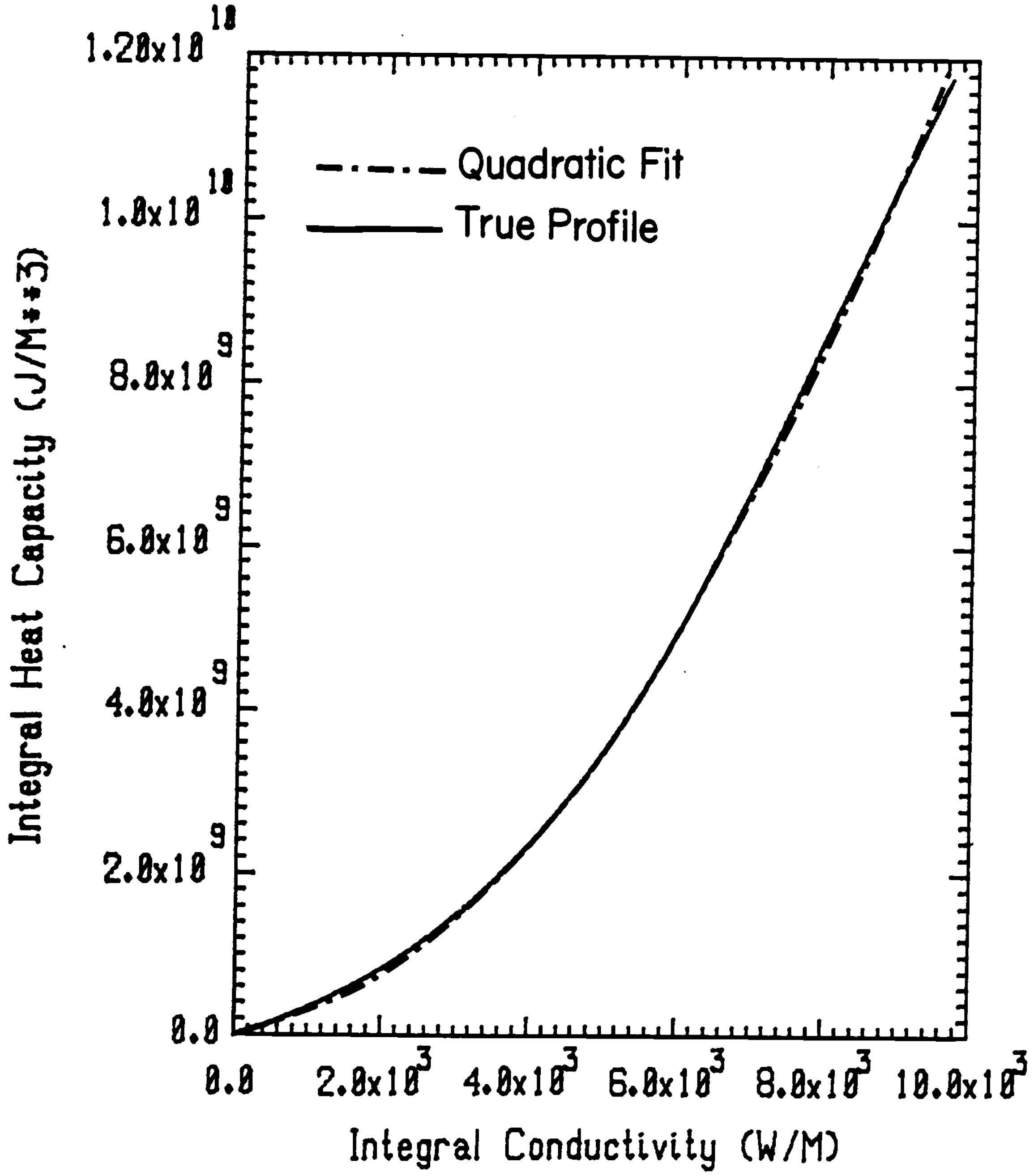


Figure III.7

Uranium dioxide integral heat capacity
as a function of integral conductivity

2. $$-\frac{dI}{dr}(R_2) = h_g[T_f(R_2) - T_c(R_3)] \qquad \text{III.48}$$

3. $$R_2[\frac{dI}{dr}(R_2)] = R_3k_c[\frac{dT_c}{dr}(R_3)] \qquad \text{III.49}$$

4. $$-k_c\frac{dT_c}{dr}(R_4) = h[T_c(R_4) - T_\infty] \qquad \text{III.50}$$

After some tedious algebra we can write the steady state temperature profiles in the fuel and clad as:

$$T_f(r,0) = T(I) =$$

$$T\left\{I\left\{T_\infty + q_f''(R_2)\left[\frac{1}{h_g} + \frac{R_2}{hR_4} + \frac{R_2}{k_c}\ln\frac{R_4}{R_3}\right]\right\}\right.$$

$$\left. + \int_r^{R_2} \frac{1}{y} \int_{R_1}^{y} x\, q'''(x,0)\, dx dy\right\} \qquad \text{III.51}$$

and

$$T_c(r,0) = T_\infty + R_2q_f''(R_2)\left[\frac{1}{hR_4} + \frac{1}{k_c}\ln\frac{R_4}{r}\right] \qquad \text{III.52}$$

where $q_f''(R_2)$ is the fuel outer surface value of the heat flux.

III.4.4. Integral Transforms

As mentioned earlier the integral transforms used in this problem are the B-transform of the first kind and the Laplace transform (a special case of the T-transform). We recall from Appendix B that the B-transform kernel of the first kind is the solution to the Helmholtz equation for the kernel in the given geometry. This will yield a cross product of Bessel functions in an annular cylindrical geometry. The resultant integral transformation is more commonly known as the finite Hankel transform and can be represented as

$$\bar{f}_H(\rho) = \int_{R_1}^{R_2} r\, f(r)\, B_n(\rho r)\, dr, \qquad \text{III.53}$$

where

$$B_n(\rho r) = J_n(\rho r)\, Y_n(\rho R_1) - Y_n(\rho r)\, J_n(\rho R_1). \qquad \text{III.54}$$

For the special case of $R_1 = 0$ the transformation kernel simplifies to

$$B_n(\rho r) = J_n(\rho r) \quad \text{if } R_1 = 0. \qquad \text{III.55}$$

The inversion formula can explicitly be derived and is given by

$$f(r) = \frac{\pi^2}{2} \sum_{\rho} B_n(\rho r) \frac{\rho^2 J_n^2(\rho R_2)}{J_n^2(\rho R_1) - J_n^2(\rho R_2)} \bar{f}_H(\rho) \qquad \text{III.56}$$

We note that the boundary conditions require that

$$B_n(\rho R_2) = J_n(\rho R_2)\, Y_n(\rho R_1) - J_n(\rho R_1)\, Y_n(\rho R_2) = 0, \qquad \text{III.57}$$

which determines the discrete values of ρ. We also set n equal to zero and ignore the higher harmonics of the Bessel function set. The generalized numerical integration algorithm developed in Appendix E can be used to perform the numerical finite Hankel transform. Equation III.57 requires the dependent variables to be zero at the fuel pellet surface ($r = R_2$). Therefore, we define

$$I_s(r,t) = I(r,t) - I(R_2,t), \qquad \text{III.58}$$

$$U_s(r,t) = U(r,t) - U(R_2,t), \qquad \text{III.59}$$

and

$$q'''_s(r,t) = q'''(r,t) - q'''(R_2,t). \qquad \text{III.60}$$

After substituting the above definitions into the governing equation a new form of the fuel region equation is obtained

$$\frac{1}{r}\frac{\partial}{\partial r}\left(r\frac{\partial I_s(r,t)}{\partial r}\right) + q''{}'_s(r,t) + q''{}'(R_2,t) =$$

$$\frac{\partial U_s(r,t)}{\partial t} + \frac{\partial U(R_2,t)}{\partial t}$$

III.61

Applying Hankel and Laplace transforms in spatial and time domain respectively, yields

$$- \rho_\ell^2\, \bar{I}_s(\rho_\ell,s) + \bar{q}''{}'_s(\rho_\ell,s) - s\bar{U}_s(\rho_\ell,s) + \bar{U}(\rho_\ell,0) =$$

$$-\frac{J_o(\rho_\ell R_2)}{J_o(\rho_\ell R_1)}\, I_s(R_1,t) + H_o[1][s\bar{U}(R_2,s) - U(R_2,0) - \bar{q}''{}'(R_2,s)]$$

III.62

For the case where R_1=0 (see Figure III.8) the algebraic form of the governing equation becomes

$$- \rho_\ell^2\, \bar{I}_s(\rho_\ell,s) + \bar{q}''{}'_s(\rho_\ell,s) - s\bar{U}_s(\rho_\ell,s) + \bar{U}(\rho_\ell,0) =$$

$$-\frac{R_2 J_1(\rho_\ell R_2)}{\rho_\ell}\,[s\bar{U}(R_2,s) - \bar{U}(R_2,0) - \bar{q}''{}'(R_2,s)], \qquad \text{III.63}$$

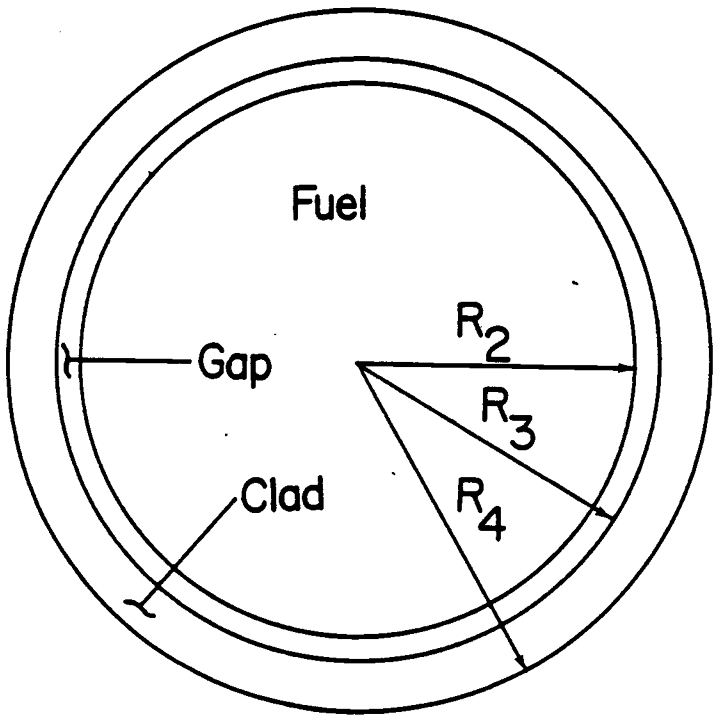


Figure III.8

Solid cylinder fuel pin geometry

Equations III.62 and III.63 contain two important unknowns. The first, $\overline{I}_s(\rho_\ell,s)$ is the double transformed integral conductivity relative to its fuel surface value. The second unknown, $\overline{U}_s(\rho_\ell,s)$, is the double transformed integral heat capacity relative to its fuel surface value. All other terms are determined from boundary and initial conditions or are given.

Since there are two unknowns and one equation, one more equation is necessary to relate the unknown variables. Both integral conductivity and integral heat capacity are functions of temperature, therefore, they can be related to one another. This quadratic relationship has been discussed in the section on material properties and is used to derive the explicit form of the thermal diffusion describing function.

III.4.5. **Describing Function Solution**

The nonlinear elements of the heat conduction equation have been identified and isolated. The nonlinearities are contained in the two variables introduced as integral conductivity and integral heat capacity. By definition of describing functions from Chapter II the TDDF becomes

$$D_{th} = \frac{\overline{U}_s(\rho,s)}{\overline{I}_s(\rho,s)}. \qquad \text{III.64}$$

The algebraic solution to the heat conduction problem is therefore expressed as:

$$\overline{I}_s(\rho_\ell,s) =$$

$$\frac{\overline{q}'''_s(\rho_\ell,s)+\overline{U}_s(\rho_\ell,s)\ -\frac{R_2}{\rho_\ell}J_1(\rho_\ell R_2)[s\overline{U}(R_2,s)-U(R_2,0)]}{sD_{th}(\rho_\ell,s,a_\ell,b_\ell,\ldots)+\rho_\ell^2} \qquad \text{III.65}$$

The separation of the governing equation into linear and nonlinear components is illustrated in Figure III.9. It is observed the nonlinear component (TDDF) behaves as a feedback to the linear component of the equation.

The material describing function relates U_s and I_s which are defined relative to their fuel pellet surface value. If the relationship between U and I was linear then we could write

$$U(r,t) = U[I(r,t)], \qquad \text{III.66}$$

and

$$U_s(r,t) = U[I_s(r,t)]. \qquad \text{III.67}$$

However, because the relationship is nonlinear we must perform some algebra and we get

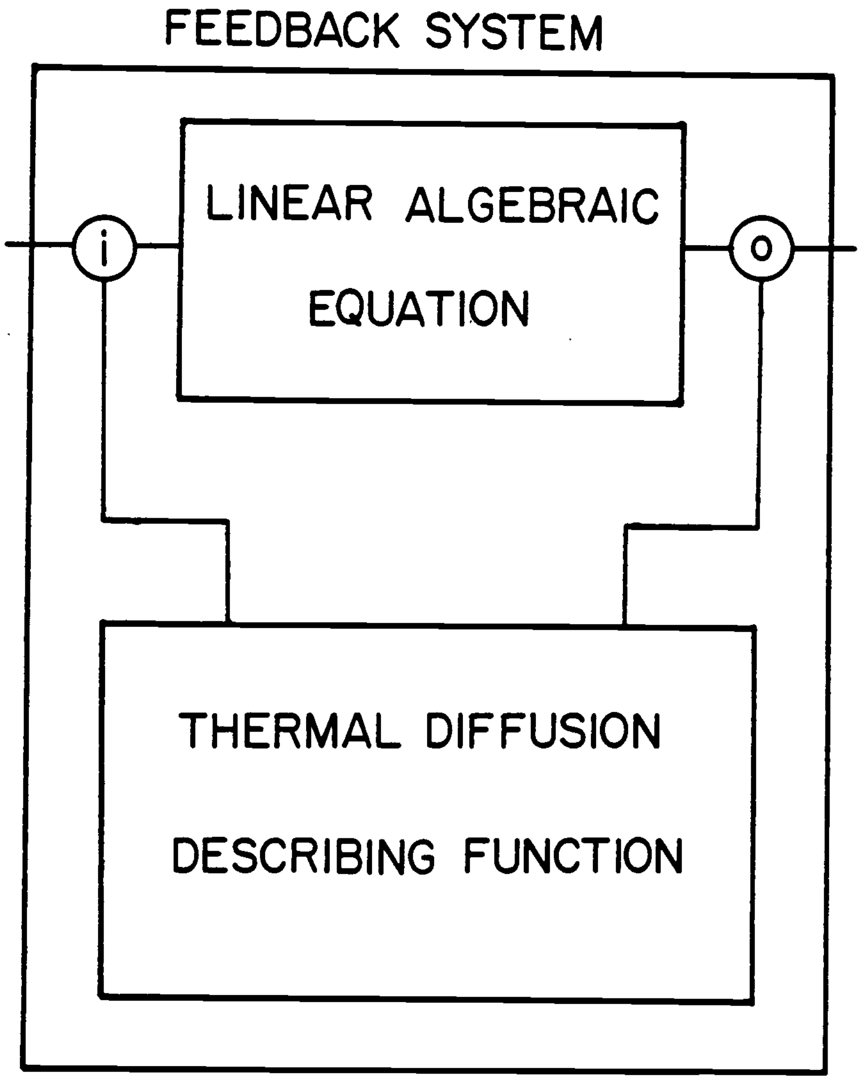

FEEDBACK SYSTEM
LINEAR ALGEBRAIC EQUATION
i
o
THERMAL DIFFUSION DESCRIBING FUNCTION

Figure III.9

Separation of the nonlinear transient one dimensional system into linear algebraic component and nonlinear thermal diffusion describing function component

$$U_s(r,t) = U[I_s(r,t) + I(R_2,t)] - U[I(R_2,t)] \qquad \text{III.68}$$

Because of the nonlinear nature of the problem an iterative method is necessary. The iteration is started by assuming an initial functional form for the input. We use here one of the most important aspects of the describing function method, namely, one must know the behavior of the solution. For example, if the heat source is an exponential function in time (e.g. ANS 5.1 standard decay heat curve) then the input function would contain exponential terms

$$I_s(r,t) = \sum_{\ell} a_{\ell}\, B_o(\rho_{\ell} r)\, e^{-b_{\ell} t}. \qquad \text{III.69}$$

For a sinusoidal heat source the input function takes the form

$$I_s(r,t) = \sum_{\ell} a_{\ell}\, B_o(\rho_{\ell} r)\, \sin(b_{\ell} t), \qquad \text{III.70}$$

where a_{ℓ} and b_{ℓ} are known constants. In other words the heat source behaves as the driving force for the final temperature profile. A suitable input to the nonlinear problem would be the linear solution of the same problem. After the form of the input is determined then the describing function can be derived explicitly (in terms of input parameters). The explicit form of the describing function is then used in the iterative method.

III.4.6. **Convergence**

We recall the definition of radius of convergence from section III.3 and rewrite it for the fuel pin problem as:

$$\alpha_c = \left| \frac{[\overline{U}(\rho,0) + \overline{q}'''_s(\rho,s)]\, D'_{th}}{(-\rho^2 + sD_{th_{\ell-1}})(-\rho^2 + sD_{th_{\ell-2}})} \right| . \qquad \text{III.71}$$

For convergence we need to show that

$$\max_{\ell} |\alpha_c| < 1. \qquad \text{III.72}$$

The expression for the radius of convergence can be evaluated for a known input and describing function. The double transformed heat source can be derived analytically because the heat source function is known. The Hankel transformed initial integral heat capacity can be calculated numerically from the steady state solution. Therefore all the terms are known for a specific problem and a maximum value for the radius of convergence can be calculated. The set of radii calculated in this manner must lie within a unit hypersphere in the phase space for the iterative method to converge. Typical values of the radius of convergence for the fuel pin problem are in the range of zero to 0.3929 with a global mean of 0.1983. These values are considerably less than the critical value of one. This range of values is the reason for fast convergence of the describing function procedure.

III.4.7. Numerical Results

DFUNC is a FORTRAN V computer program (Appendix H) which uses the describing function method to solve the nonlinear transient one dimensional heat conduction equation for an exponential describing function.

III.4.7.1. ANS 5.1 Standard Decay Heat Source

As a numerical example the case of a reactor scram to power zero is considered here. After reactor shutdown the heat energy is produced by the radioactive decay of fission products contained in the fuel. The heat source function for this case is given by the ANS 5.1 standard decay heat curve.

The decay heat source is illustrated in Figure III.10 as a function of time. This function consists of the summation of 27 exponential terms. An approximation of this function is therefore useful. The HIPLOT plotting and regression routine (Appendix F) is used to perform a polynomial regression on the heat source profile and a polynomial expression is obtained.

$$\ln[P(t)] = \sum_{\ell} \beta_{\ell} \, [\ln(t)]^{\ell}. \qquad \text{III.73}$$

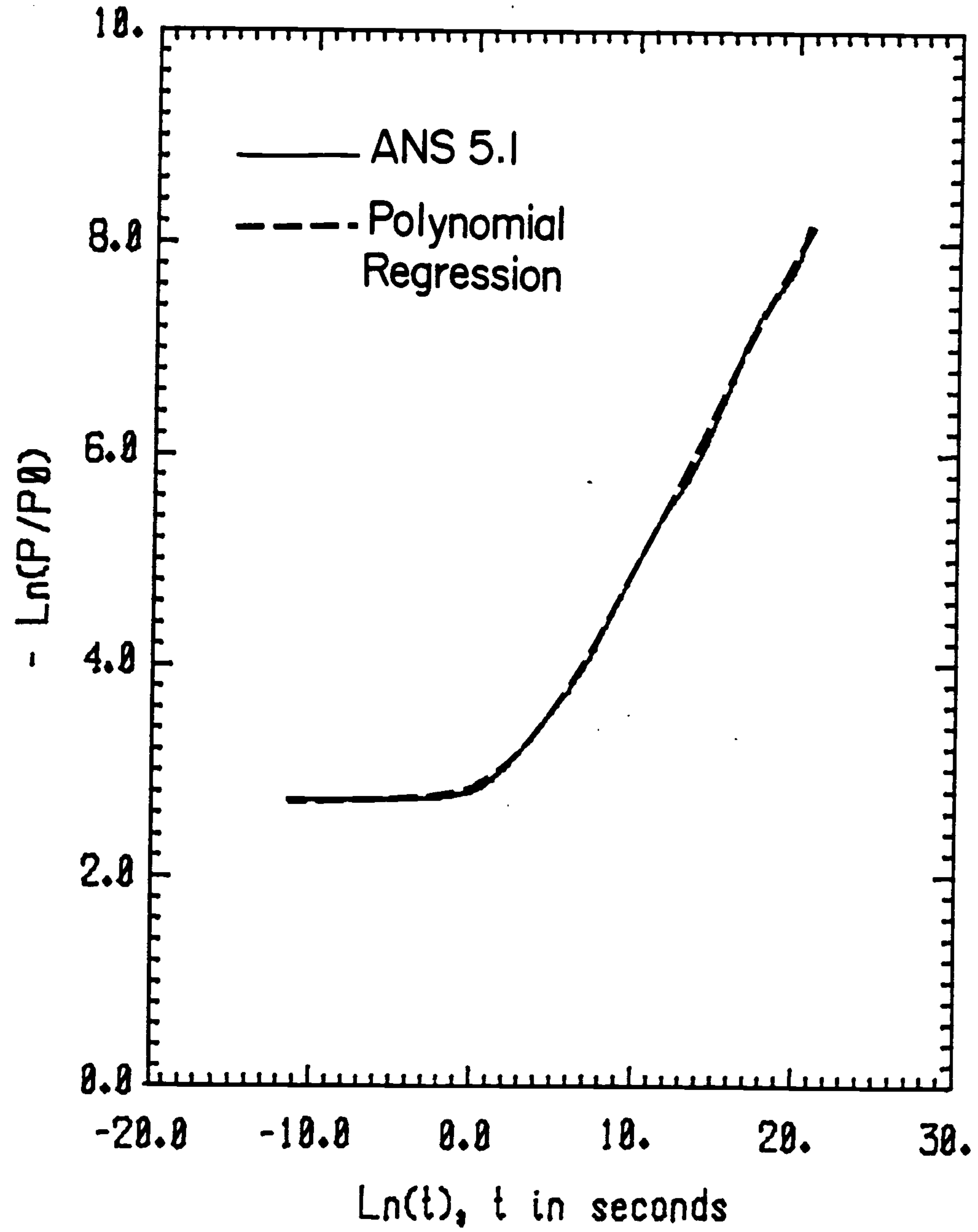


Figure III.10

ANS 5.1 standard decay heat curve

The coefficients β_ℓ are given by Table III-2. Because of the exponential nature of the heat source the input is given by equation III.69. This input is then used for derivation of an explicit form for the describing function. The final form is:

$$D_{th}(\rho_\ell, s, a_\ell, b_\ell, I) =$$

$$c_1 + 2c_2(s+b_\ell) \int_0^\infty e^{-(s+b_\ell)t} I(R_2,t)\, dt$$

$$+ \frac{2c_2 a_\ell}{R_2 J_1^2(\rho_\ell R_2)} \frac{s+b_\ell}{(s+2b_\ell)\rho_\ell^2} d_\ell, \qquad \text{III.74}$$

where

$$d_\ell = \rho_\ell^2 \int_0^R r J_o^3(\rho_\ell r)\, dr. \qquad \text{III.75}$$

The input data for a sample run of DFUNC are presented in Table III-3. This data is typical data for a pressurized water reactor. Six iterations were necessary for completion of the solution at 2.6 CPU seconds on a CDC CYBER 170/720 computer. The describing function profile in the transformed space is illustrated in a three dimensional perspective in Figure III.11a for the initial profile and Figure III.11b for the converged profile (iteration number 6).

TABLE III-2

ANS 5.1 Standard Decay Heat Source (June 1978)

8th order Polynomial Regression Coefficients

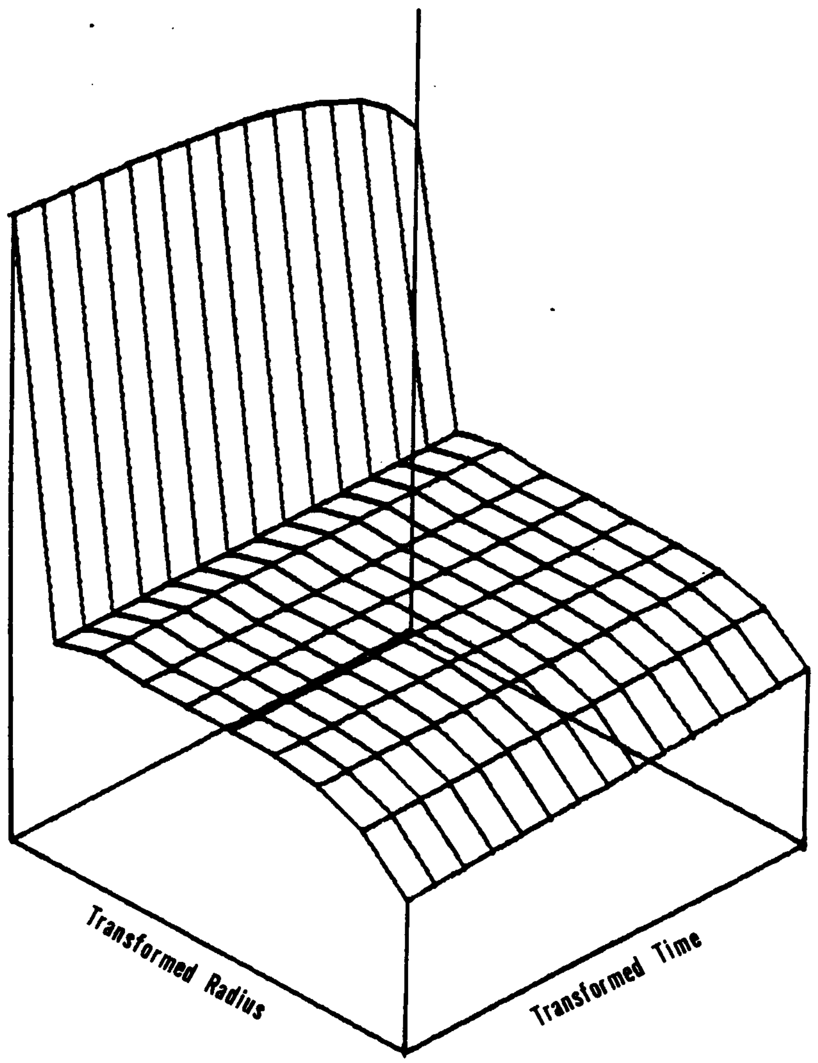
(correlation coefficient = 0.9994)

Order	Coefficient
0	-2.79106
1	-7.07709×10^{-2}
2	-1.61443×10^{-2}
3	-4.29611×10^{-4}
4	9.65087×10^{-5}
5	-6.65904×10^{-7}
6	-3.18022×10^{-7}
7	1.21536×10^{-8}
8	-8.99423×10^{-11}

TABLE III-3

Fuel Pin Data Used in DFUNC and COYOTE

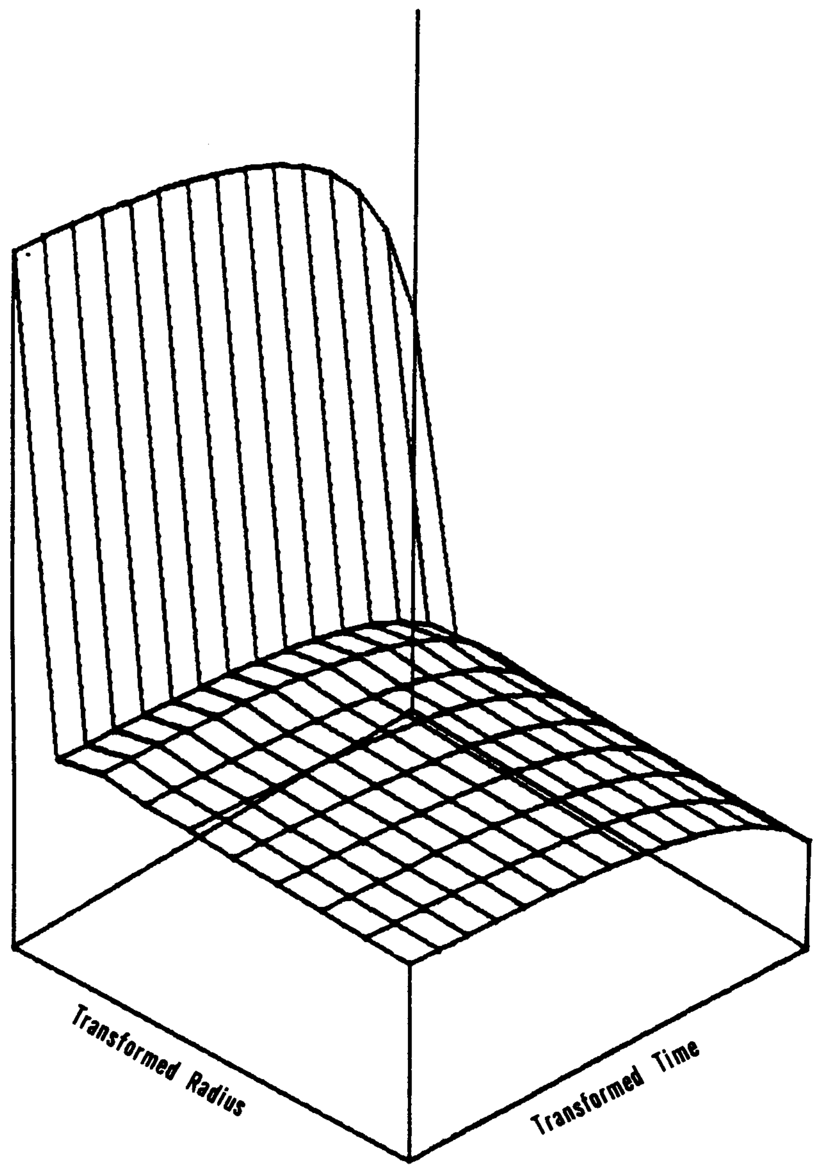
Fuel radius	0.00601	m
Clad inner surface radius	0.00621	m
Clad outer surface radius	0.00715	m
Clad thermal diffusivity	7.678×10^{-6}	m^2/s
Clad thermal conductivity	16.9	W/mK
Initial gap conductance	7800	W/m^2K
Initial coolant temperature	509	K
Coolant heat transfer coefficient	3276	W/m^2K
Linear power	31.5	KW/m
Peak-to-average ratio	1.23	



(a)

Figure III.11

Perspective describing function surface
(a) initial



(b)

Figure III.11

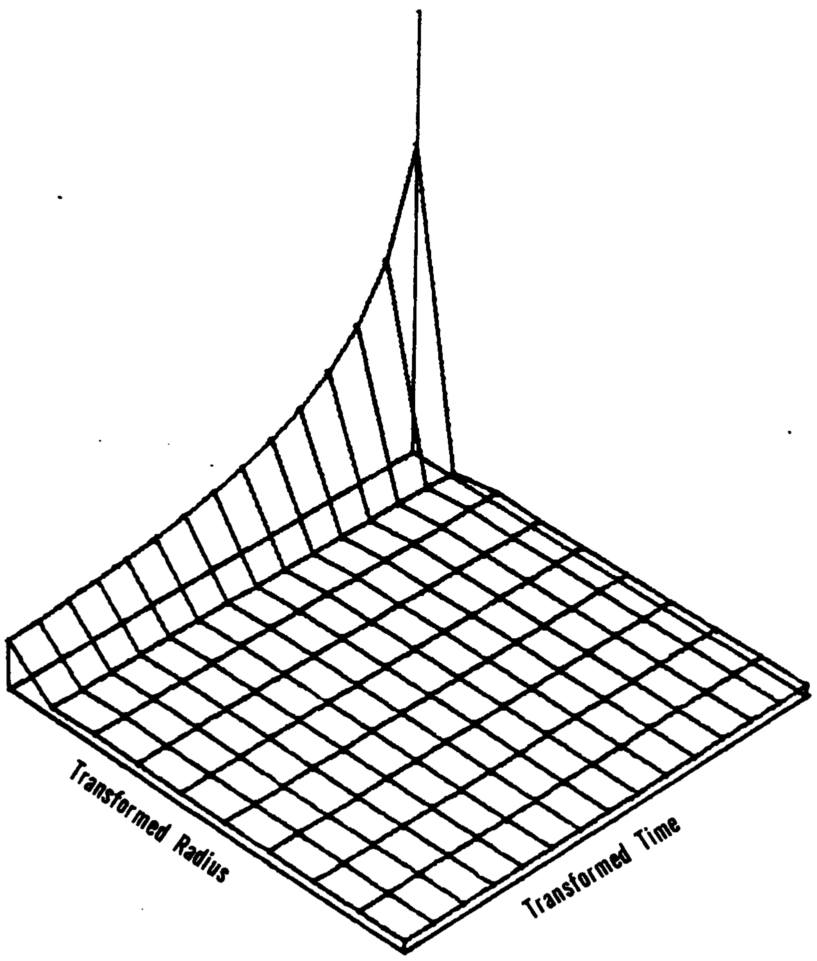
Perspective describing function surface
(b) iteration number 6 (converged)

The nonlinear nature of the describing function is observed over entire domain of the problem. The describing function maintains the form of the initial profile throughout iterations (see Figure III.11b).

The input profile in the transformed domain is illustrated in Figures III.12a for the initial profile and Figure III.12b for the converged profile (iteration number 6). The input maintains the form of the equation III.69 but the coefficients a_ℓ and b_ℓ change (see Table III-4).

The overall temperature solution is presented in 3-D perspective in Figure III.13. Both the fuel and clad regions are illustrated. The fuel temperature has a Bessel function behavior in radial direction and a decaying exponential behavior in time. The clad solution is almost linear in radial direction and exponential in time.

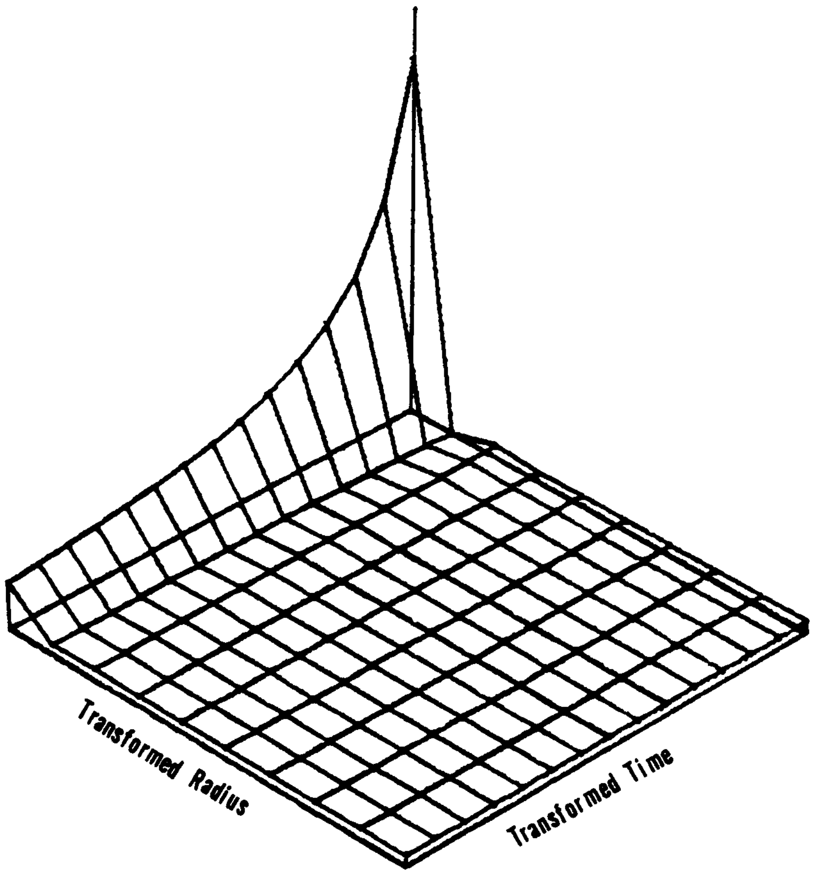
To verify the describing function results a finite element code COYOTE is used to solve the same problem. COYOTE is a nonlinear two dimensional transient finite element code which solves the heat conduction equation. The comparison of DFUNC results to COYOTE is presented in Figure III.14. There is an excellent agreement between the results (a maximum temperature difference of about 10 K). It should be noted that the exponential input is hardwired in the DFUNC code. Therefore, DFUNC can only be used to model exponentially decaying heat sources. Generality is therefore sacrificed for speed of computation.



(a)

Figure III.12

Perspective integral conductivity surface
(a) initial



(b)

Figure III.12

Perspective integral conductivity surface
(b) iteration number 6 (converged)

TABLE III-4

Input Parameters (a_ℓ and b_ℓ coefficients) as a Function of Iteration Number

	iteration number					
	1	2	3	4	5	6
a_1	2595.36	2746.43	2636.35	2666.85	2654.44	2658.55
a_2	-303.319	-364.880	-331.032	-338.052	-334.664	-335.677
a_3	70.3200	110.789	94.5666	98.1268	96.4986	96.9968
a_4	-26.2809	-51.6843	-42.6852	-44.6838	-43.7819	-44.0594
a_5	12.8471	29.1590	23.6355	24.8645	24.3128	24.4829
a_6	-7.38622	-18.3554	-14.7157	-15.5252	-15.1625	-15.2744
a_7	4.71189	12.4253	9.89212	10.4550	10.2031	10.2809
a_8	-3.22896	-8.86443	-7.02458	-7.43311	-7.25035	-7.30679
a_9	2.33112	6.58148	5.19861	5.50547	5.36823	5.41062
a_{10}	-1.75111	-5.04259	-3.97377	-4.21082	-4.10482	-4.13756
b_1	.119915	.0774291	.0639727	.0649562	.0640134	.0642414
b_2	.517101	.270127	.205731	.209478	.205201	.206155
b_3	.588419	.317340	.214029	.223829	.216455	.218305
b_4	.550989	.349002	.222492	.236175	.226914	.229320
b_5	.512871	.370121	.229307	.245354	.234950	.237694
b_6	.486740	.384252	.234401	.251902	.240785	.243738
b_7	.469207	.393818	.237993	.256441	.244856	.247946
b_8	.457289	.400502	.240583	.259669	.247767	.250950
b_9	.449084	.405301	.242469	.262006	.249879	.253128
b_{10}	.443485	.408848	.243881	.263747	.251455	.254751

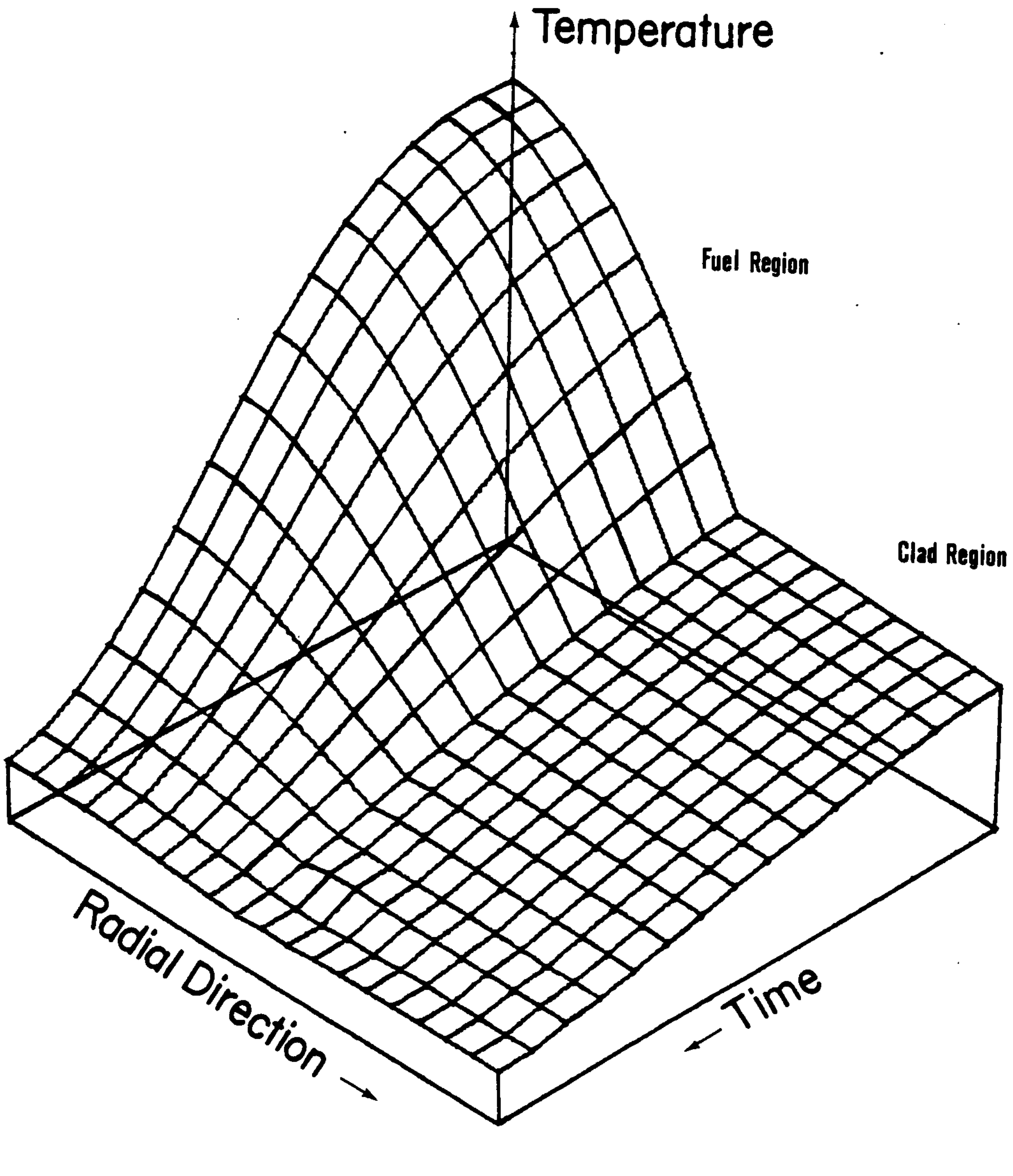


Figure III.13

Perspective temperature field in time and space
(Reactor scram from full power)

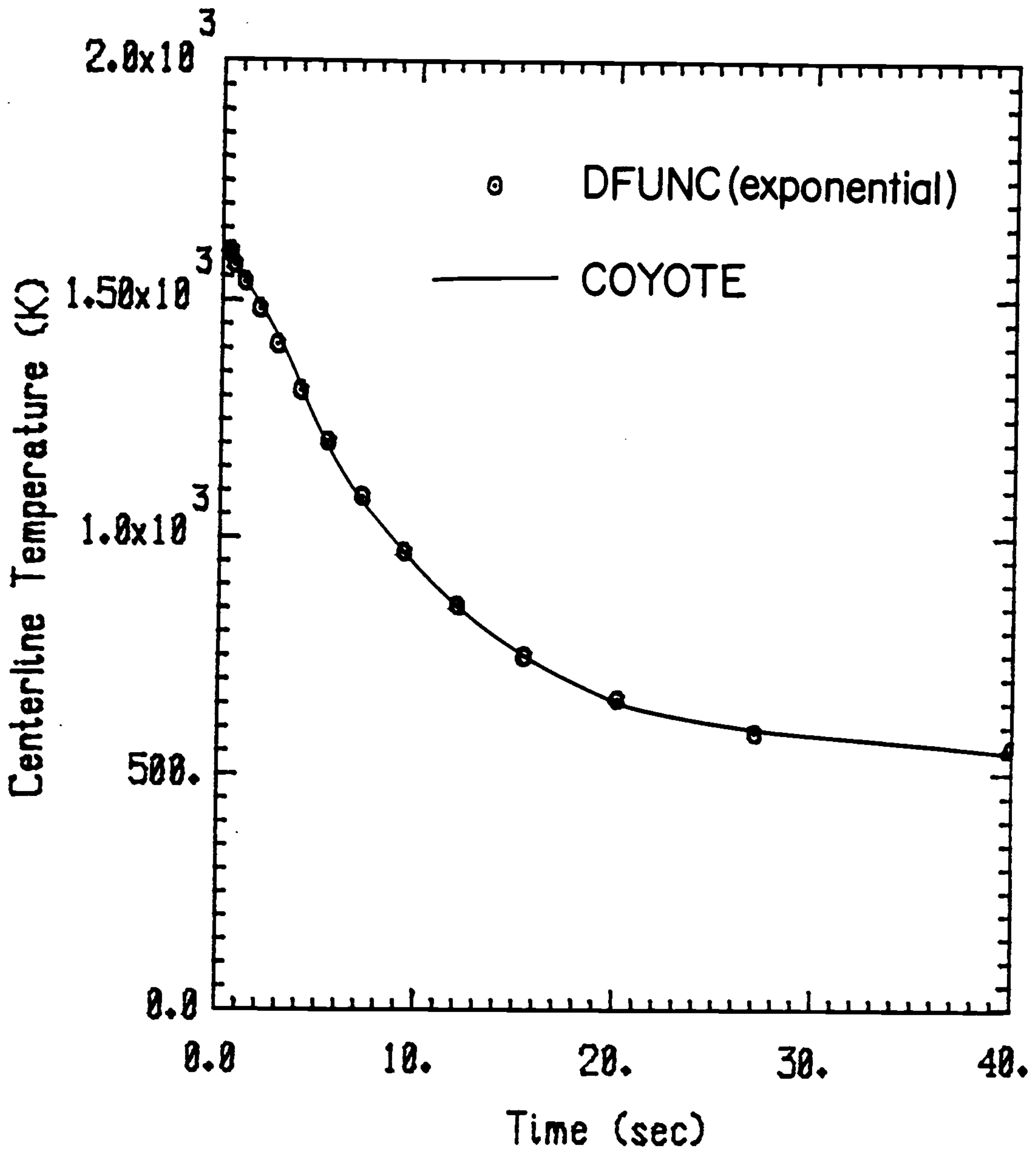


Figure III.14

Centerline temperature as a function of time for an exponential describing function (ANS 5.1 standard decay heat source)

III.4.7.2. **Sinusoidal Heat Source**

As a test of the DFUNC exponential describing function code a sinusoidal heat source is used to replace the exponential heat source (see Figure III.15). The describing function and the input form are kept the same. All other input parameters are fixed (Table III.3). The resulting temperature field is compared to a COYOTE solution in Figure III.16. A poor comparison is observed. This is expected because the heat source is the driving force in determining the temperature profile in heat transfer by conduction. By changing the heat source function we have changed the system response curve. Without improving the nominal about which the response is approximated a well behaved output cannot be obtained. In regards to this we refer back to Figures II.3 and II.4 which clarify the concept. Therefore, the thermal diffusion describing function can only be derived for problems with a predetermined form of the heat source function. If the heat source changes the describing function and the input form must also change accordingly. This can be illustrated by deriving a sinusoidal input describing function and replacing the old describing function.

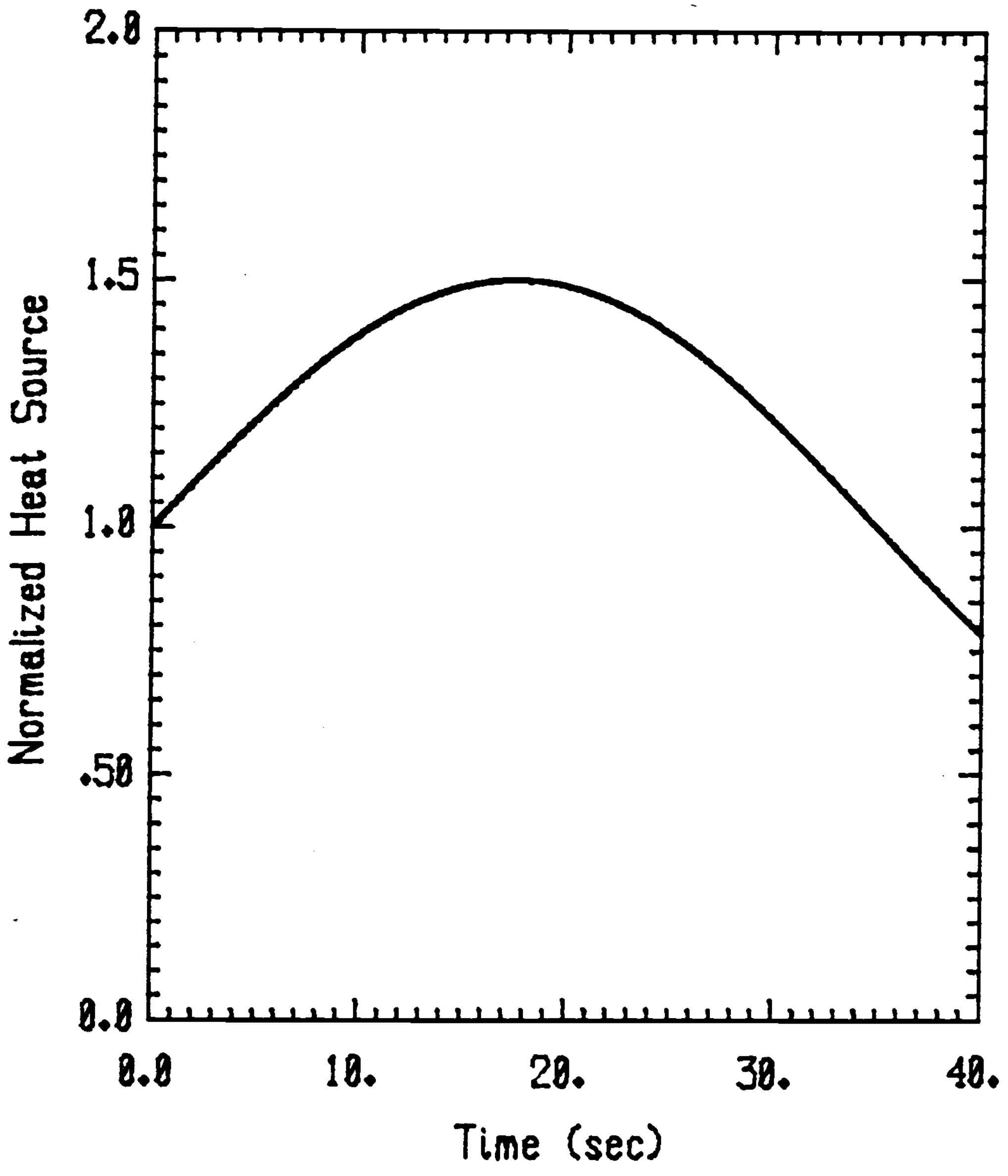


Figure III.15

Sinusoidal heat source

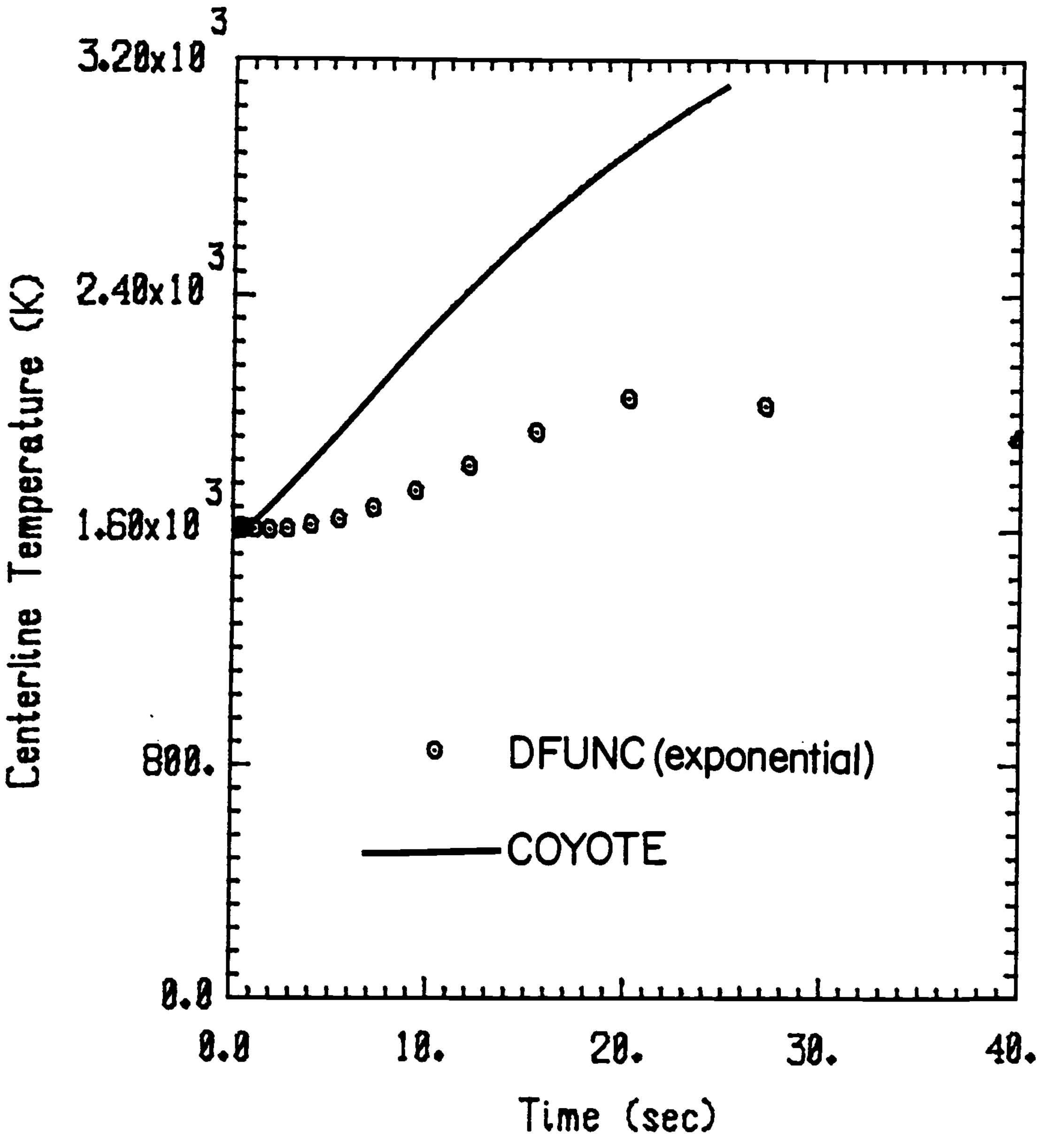


Figure III.16

Centerline temperature as a function of time for an exponential describing function (sinusoidal heat source)

$$D_{th}(\rho_\ell, s, a_\ell, b_\ell, I) =$$

$$c_1 + \frac{2c_2(s^2 + b_\ell^2)}{b\ell} \int_0^\infty e^{-st} \sin(b_\ell t)\ I(R_2,t)\ dt$$

$$+ \frac{c_2 a_\ell b_\ell d_\ell}{sR_2^2 J_1^2(\rho_\ell R_2)}, \qquad \text{III.76}$$

where d_ℓ is defined as before (equation III.75). Figure III.17 illustrates the results. A much better agreement between the describing function solution (DFUNC) and finite element solution is observed.

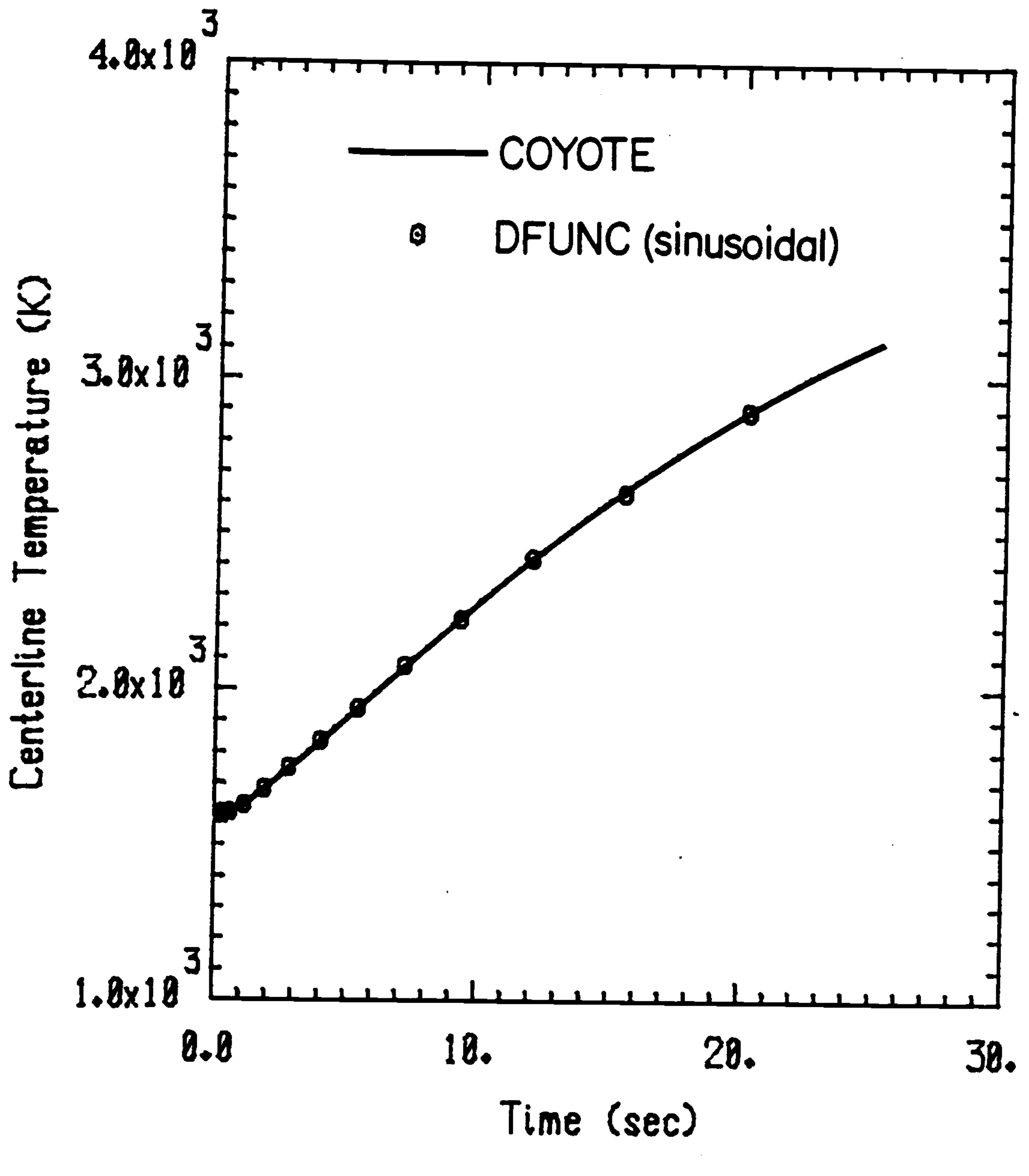


Figure III.17

Centerline temperature as a function of time
for a sinusoidal describing function
(sinusoidal heat source)

IV. A DESCRIBING FUNCTION THEORY SOLUTION OF THE NEUTRON DIFFUSION EQUATION

IV.1. Introduction

Neutrons in a reactor move in complicated, random walk paths due to multiple collisions with nuclei. As a result of this type of behavior, neutrons are transported from one region of the phase space to another.

An exact equation governing the neutron transport phenomenon can easily be derived. This is called the Boltzmann equation, or the transport equation. Unfortunately except under simplifying assumptions solutions of the Boltzmann equation are not easily obtainable. However, often in applications of reactor theory, a simplified version of the transport equation, the diffusion equation, may be used.

In this section it is assumed that all the neutrons have the same energy. However, the more general case of finding a describing function solution to the multi-group diffusion theory equations is a straight forward extension of the theory developed here. The monoenergetic (one speed) diffusion theory can be derived as a special case of the neutron continuity equation.

IV.2. Derivation of the Neutron Diffusion Equation

The continuity equation can be derived by considering an arbitrary volume element V in a continuum containing monoenergetic neutrons. The rate of change of number of neutrons within the control volume is equal to the rate at which neutrons are produced within V minus the rate at which they are removed from V. The production is either by fission or a distributed or localized source of neutrons. The removal is either by absorption or leakage (neutrons leaving V). Thus

$$\frac{D}{Dt}\int_V n(\mathbf{r},t)dV = \text{rate of production} - \text{rate of absorption} - \text{rate of leakage}$$

$$= \int_V S(\mathbf{r},t)dV - \int_V \Sigma_a(\mathbf{r})\phi(\mathbf{r},t)dV - \int_A \mathbf{J}(\mathbf{r},t)\cdot\mathbf{n}\, dA, \qquad \text{IV.1}$$

where

$\frac{D}{Dt}$	= material derivative,
$n(\mathbf{r},t)$	= neutron number density n/cm^3 s,
$S(\mathbf{r},t)$	= source function n/cm^3 s,
$\Sigma_a(\mathbf{r})$	= macroscopic absorption cross section cm^{-1},
$\phi(\mathbf{r},t)$	= neutron flux n/cm^2 s,

$\mathbf{J}(\mathbf{r},t)\cdot\mathbf{n}$ = net rate of flow of neutrons through a unit area normal to n at r,

$\mathbf{r}$ = position vector,

$\mathbf{n}$ = normal unit vector.

Divergence theorem can be used to transform the leakage surface integral to a volume integral. Thus

$$\int_A \mathbf{J}(\mathbf{r},t)\cdot\mathbf{n}\,dA = \int_V \mathrm{div}(\mathbf{J}(\mathbf{r},t))dV,$$

and equation IV.1 becomes

$$\frac{D}{Dt}\int_V n(\mathbf{r},t)dV = \int_V S(\mathbf{r},t)dV - \int_V \Sigma_a(\mathbf{r})\phi(\mathbf{r},t)dV - \int_V \mathrm{div}(\mathbf{J}(\mathbf{r},t))dV.$$

Since V is an arbitrary portion of the continuum, it follows that

$$\frac{Dn(\mathbf{r},t)}{Dt} = S(\mathbf{r},t) - \Sigma_a(\mathbf{r})\phi(\mathbf{r},t) - \mathrm{div}(\mathbf{J}(\mathbf{r},t)). \qquad \text{IV.2}$$

Equation IV.2 is called the continuity equation. To find a solution for this equation it is necessary to define a relationship between $\mathbf{J}(\mathbf{r},t)$ and $\phi(\mathbf{r},t)$. The most common approximation is Fick's law of diffusion which states that the current density vector is proportional to the negative gradient of the flux (similar to Fourier's law of conduction). The proportionality constant is called the diffusion coefficient (similar to conductivity) and is denoted by the symbol D. Hence, Fick's law may be expressed as:

$$\mathbf{J} = - D \, \mathrm{grad}(\phi) = - D \, \nabla\phi.$$

If the volume is assumed to be stationary then the material derivative reduces to only a partial derivative. Thus, equation IV.2 becomes

$$\frac{\partial n(\mathbf{r},t)}{\partial t} = S(\mathbf{r},t) - \Sigma_a(\mathbf{r})\phi(\mathbf{r},t) - \mathrm{div}(D \, \mathrm{grad}(\phi(\mathbf{r},t))). \qquad \text{IV.3}$$

This equation is the monoenergetic (one speed) diffusion equation. For a complete discussion of the neutron diffusion equation and its properties see Lamarsh (1972), Duderstadt and Hamilton (1976), and Henry (1975).

IV.3. Application of the Describing Function Method

Equation IV.3 is a partial differential equation which can be transformed into a linear differential equation by proper variable substitutions. The diffusion coefficient plays a significant role in the variable substitution and integral transform steps of the solution technique. Hence, two cases are considered here.

IV.3.1. **Constant Diffusion Coefficient**

When D is a constant or if the problem is one dimensional (in position) then the following variable substitutions can be used:

$$-\mathbf{J} = \text{grad}(C) = D\ \text{grad}(\phi),$$

and

$$A(\mathbf{r},t) = \Sigma_a(\mathbf{r})\phi(\mathbf{r},t),$$

where

$$C = \text{current potential function},$$

and equation IV.3 becomes

$$\frac{\partial n(\mathbf{r},t)}{\partial t} = S(\mathbf{r},t) - A(\mathbf{r},t) - \text{Lap}(C), \qquad \text{IV.4}$$

where

$$\text{Lap}(C) = \text{div}(\text{grad } C).$$

For a monoenergetic (one speed) group of neutrons the flux can be expressed in terms of the number density and velocity as

$$\phi\ [\mathrm{n/cm^2\ s}] = n\ [\mathrm{n/cm^3}]\ .\ v\ [\mathrm{cm/s}],$$

and equation IV.4 becomes

$$\frac{1}{v}\frac{\partial\phi(\mathbf{r},t)}{\partial t} = S(\mathbf{r},t) - A(\mathbf{r},t) - \mathrm{Lap}(C(\mathbf{r},t)). \qquad \text{IV.5}$$

This equation can now be transformed to an algebraic form by using appropriate integral transformations. The two differential terms of concern are $\frac{\partial\phi(\mathbf{r},t)}{\partial t}$ and Lap(C($\mathbf{r}$,t)).

To transform the equation to an algebraic form a Laplace transform (or a T-transform) can be used in time domain and a B-transform in spatial domain (ref. appendix B), where, the transformations are defined as

$$L[f(t)] = \overline{f}(s) = \int_0^\infty e^{-st}\, f(t)\, dt,$$

and

$$B[g(\mathbf{r})] = \overline{g}(\boldsymbol{\rho}) = \int_{Vol} K(\mathbf{r},\boldsymbol{\rho})\ g(\mathbf{r})\ dV. \qquad \text{IV.6}$$

The kernel of spatial transformation is determined by solving Helmholtz's parabolic equation and applying the proper boundary conditions. For most diffusion problems the transformation of the Laplacian should yield the transformed function times a function of the transformation variable, i.e.

$$B[\mathrm{Lap}(g(\mathbf{r}))] = \Omega(\boldsymbol{\rho})\overline{g}(\boldsymbol{\rho}). \qquad \text{IV.7}$$

This transformation will yield an algebraic term in the governing equation. A requirement for the above transformation is that the kernel and the transformed function must satisfy the following boundary (surface) condition.

$$\int_{S} [K(\mathbf{r},\boldsymbol{\rho})\mathrm{grad}(g(\mathbf{r})) + g(\mathbf{r})\mathrm{grad}(K(\mathbf{r},\boldsymbol{\rho}))]\cdot\mathbf{n}\ dS = 0 \qquad \text{IV.8}$$

where **n** is a unit normal to the surface. If this boundary condition is satisfied then the transformation kernel can be determined from

$$\mathrm{Lap}(K(\mathbf{r},\boldsymbol{\rho})) = \Omega(\boldsymbol{\rho})K(\mathbf{r},\boldsymbol{\rho}). \qquad \text{IV.9}$$

After Laplace transformation equation IV.5 becomes

$$\frac{1}{v}\left[s\overline{\phi}(\rho,s) - \overline{\phi}(\rho,0)\right] = \overline{S}(\rho,s) - \overline{A}(\rho,s) - \Omega(\rho)\overline{C}(\rho,s). \qquad \text{IV.10}$$

Equation IV.10 contains three unknown variables $\overline{\phi}(\rho,s)$, $\overline{A}(\rho,s)$ and $\overline{C}(\rho,s)$. In order to solve this equation two more equations are needed. These equations are the describing function definitions for this problem. By definition, describing function is "the ratio of the double transformed output to the double transformed input." Thus

$$D_L = \frac{\overline{C}(\rho,s)}{\overline{\phi}(\rho,s)} \qquad \text{Leakage describing function} \qquad \text{IV.11}$$

$$D_A = \frac{\overline{A}(\rho,s)}{\overline{\phi}(\rho,s)} \qquad \text{Absorption describing function} \qquad \text{IV.12}$$

Introduction of these describing functions now enables us to solve for $\overline{\phi}(\rho,s)$ in equation IV.10

$$\overline{\phi}(\rho,s) = \frac{\overline{S}(\rho,s) + \frac{1}{v}\overline{\phi}(\rho,0)}{\frac{s}{v} + \Omega(\rho)D_L + D_A}. \qquad \text{IV.13}$$

The governing differential equation in the transformed space is therefore represented by a linear algebraic equation and two describing functions which provide a feedback to the linear part. A graphical representation of this system is illustrated in Figure IV.1.

IV.3.2. **Variable Diffusion Coefficient**

From the methods developed in Appendix B we can write

$$\mathrm{div}(D \ \mathrm{grad}\ \phi) = \frac{1}{h_1 h_2 h_3} \sum_{\ell=1}^{3} \frac{\partial}{\partial x^\ell} \left[\frac{h_1 h_2 h_3 D}{h_\ell^2} \frac{\partial \phi}{\partial x^\ell}\right] \qquad \text{IV.14}$$

where h_1, h_2 and h_3 are the nonzero elements of the metric tensor for an orthogonal coordinate system. Appropriate variable substitutions in this case are:

$$\frac{\partial L_i}{\partial x^i} = \frac{h_1 h_2 h_3 D}{h_i^2} \frac{\partial \phi}{\partial x^i} \qquad \text{for } i=1,2,3, \qquad \text{IV.15}$$

and

$$A(\mathbf{r},t) = \Sigma_a(\mathbf{r})\ \phi(\mathbf{r},t). \qquad \text{IV.16}$$

Again, using the results of appendix B for the B-transform of the second kind we get:

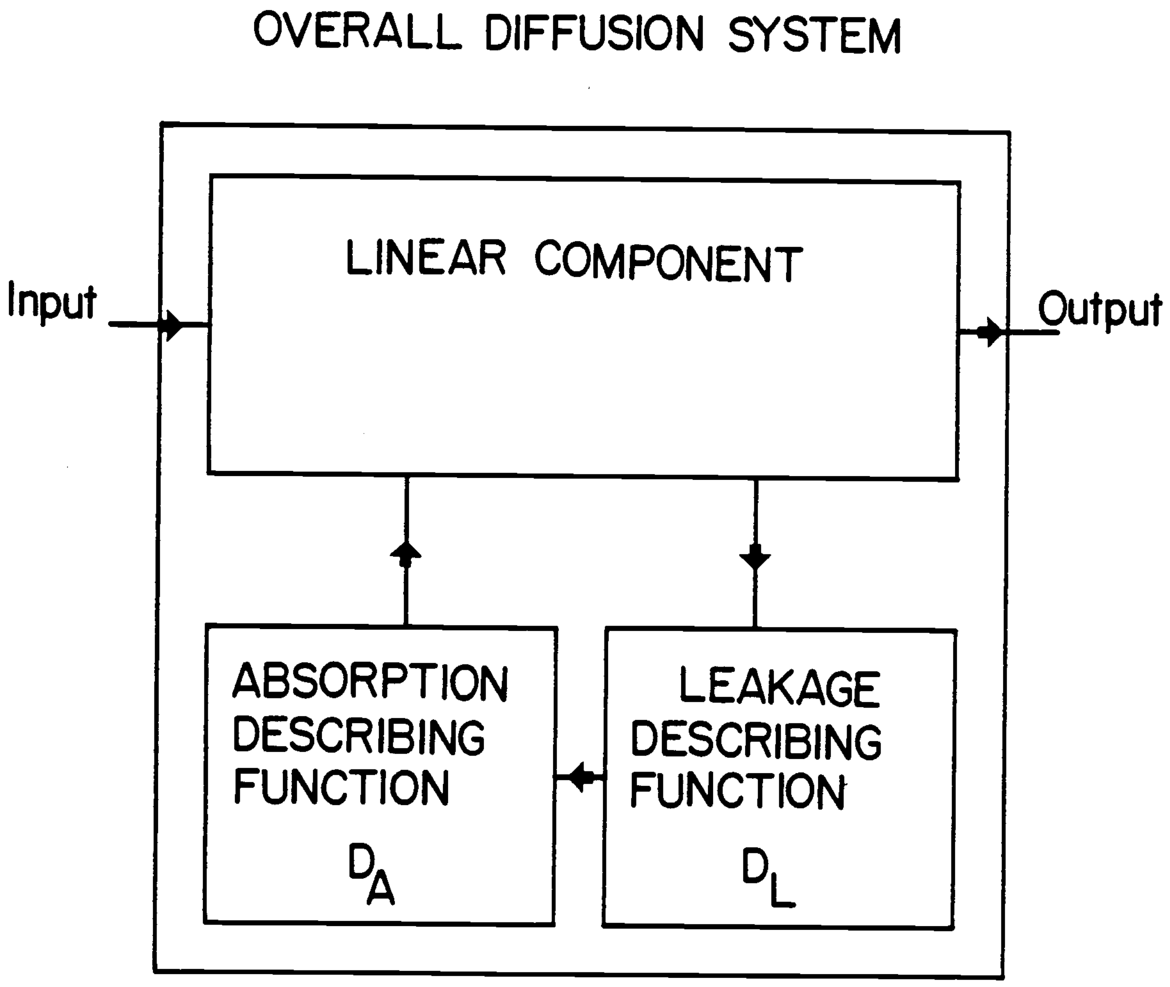


Figure IV.1

Separation of the diffusion system into linear and nonlinear components (for a constant diffusion coefficient)

$$B[\mathrm{div}(D \,\mathrm{grad}\, \phi(\mathbf{r},t))] = -\sum_{i=1}^{3} \rho_i^2 \, L_i(\boldsymbol{\rho},t).$$

After Laplace transformation the algebraic form of the diffusion equation becomes

$$\frac{1}{v}\,[s\overline{\phi}(\boldsymbol{\rho},s) - \overline{\phi}(\boldsymbol{\rho},0)] =$$

$$\overline{S}(\boldsymbol{\rho},s) - \overline{A}(\boldsymbol{\rho},s) - \sum_{i=1}^{3} \rho_i^2 \, \overline{L}_i(\boldsymbol{\rho},s). \qquad \text{IV.17}$$

This equation contains five unknown functions $\overline{\phi}(\boldsymbol{\rho},s)$, $\overline{A}(\boldsymbol{\rho},s)$, $\overline{L}_1(\boldsymbol{\rho},s)$, $\overline{L}_2(\boldsymbol{\rho},s)$ and $\overline{L}_3(\boldsymbol{\rho},s)$, therefore, four more constraints are needed. Four material describing functions (one for absorption and three for leakage) are defined as:

$$D_L^i = \frac{\overline{L}_i(\boldsymbol{\rho},s)}{\overline{\phi}(\boldsymbol{\rho},s)} \qquad i=1,2,3.$$

Introducing these material describing functions into equation IV.17 and solving for $\overline{\phi}(\boldsymbol{\rho},s)$ yields:

$$\bar{\Phi}(\rho,s) = \frac{\bar{S}(\rho,s) + \frac{1}{v}\bar{\Phi}(\rho,0)}{\frac{s}{v} + D_A + \sum_{i=1}^{3} \rho_i^2 D_L^i}. \qquad \text{IV.18}$$

As before an iterative method must be applied to this equation to obtain a solution. A graphical representation of this system is illustrated in Figure IV.2.

IV.4. Convergence

We can express the describing function solutions for cases one and two in a more general form as:

$$\bar{\Phi} = \frac{f(\rho,s)}{g(s) + D(\bar{\Phi})}, \qquad \text{IV.19}$$

where $f(\rho,s)$ is the function equal to the numerator of equations IV.17 and IV.13 and $g(s)$ is equal to s/v. The describing functions in the denominator are represented by $D(\bar{\phi})$. As mentioned before the iterative method starts by "guessing" a solution function ϕ_o. Therefore, for the nth iteration we can write

$$\bar{\Phi}_n = \bar{\Phi}_o + (\bar{\Phi}_1 - \bar{\Phi}_o) + (\bar{\Phi}_2 - \bar{\Phi}_1) + \ldots + (\bar{\Phi}_n - \bar{\Phi}_{n-1}),$$

IV.20

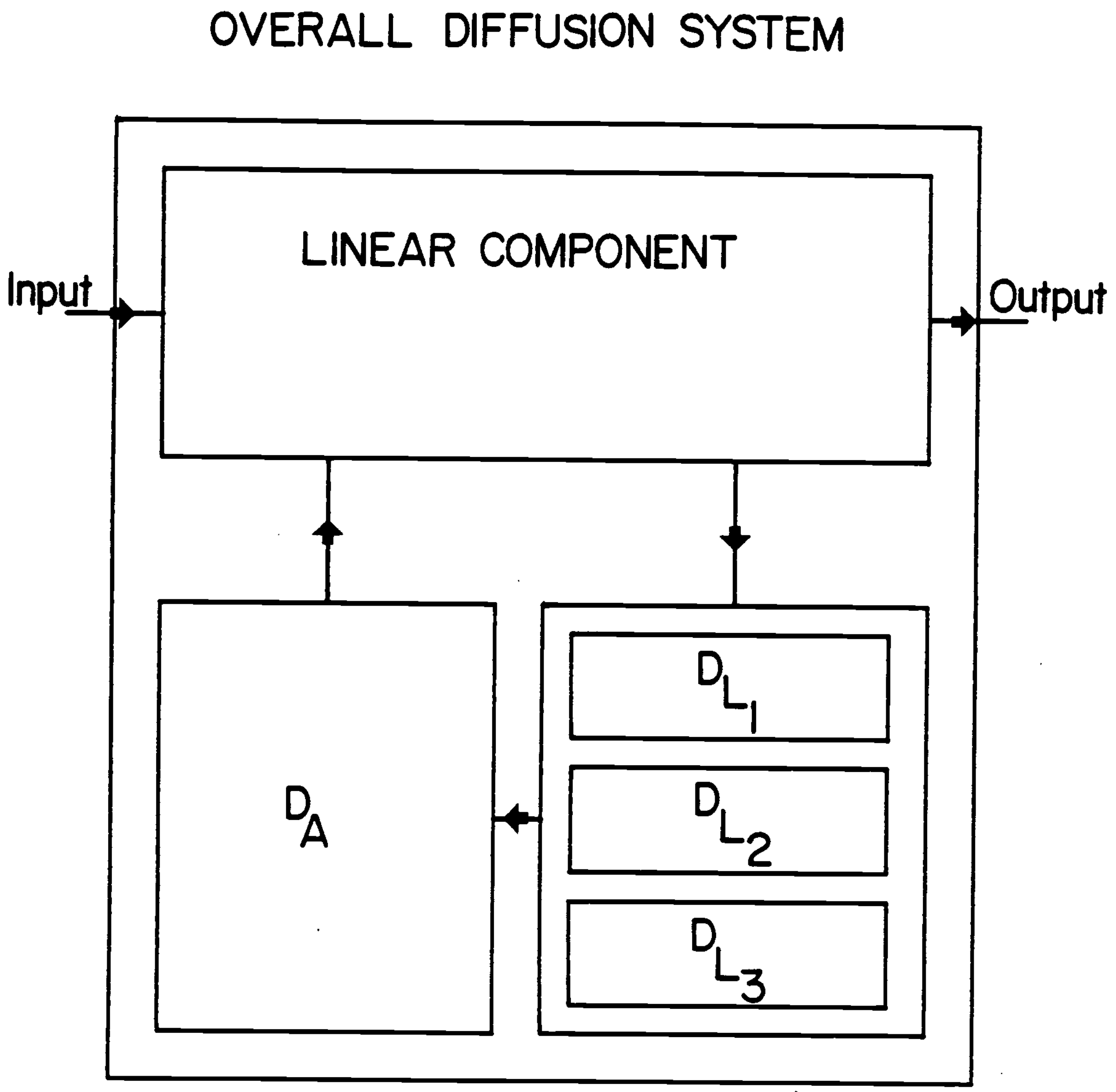


Figure IV.2

Separation of the diffusion system into
linear and nonlinear components
(for a variable diffusion coefficient)

or equivalently

$$\Phi_n = \Phi_o + \sum_{\ell=1}^{n} (\Phi_\ell - \Phi_{\ell-1}). \qquad \text{IV.21}$$

For convergence, we require that

$$\lim_{n \to \infty} \phi_n \text{ exists.}$$

This implies that the series in equation IV.21 must converge. A necessary and sufficient condition for convergence of the series is that

$$\lim_{i \to \infty} |\Phi_i - \Phi_{i-1}| \longrightarrow 0. \qquad \text{IV.22}$$

From equation IV.19 we can write

$$|\Phi_{i+1} - \Phi_i| = \left| \frac{f}{g + D_i} - \frac{f}{g + D_{i-1}} \right|$$

or

$$|\overline{\Phi}_{i+1} - \overline{\Phi}_i| = |f| \left| \frac{D_{i-1} - D_i}{(g + D_i)(g + D_{i-1})} \right| ,$$

by fundamental theorem of calculus we can write (Fulks, 1978)

$$|D_{i-1} - D_i| \leqq |D'| \; |\overline{\Phi}_i - \overline{\Phi}_{i-1}| ,$$

or

$$|\overline{\Phi}_{i+1} - \overline{\Phi}_i| \leqq \left| \frac{f\, D'}{(g + D_i)(g + D_{i-1})} \right| |\overline{\Phi}_i - \overline{\Phi}_{i-1}| ,$$

let

$$\alpha = \left| \frac{f\, D'}{(g + D_i)(g + D_{i-1})} \right| \qquad \text{radius of convergence,}$$

then

$$|\overline{\Phi}_{i+1} - \overline{\Phi}_i| \leqq \alpha^i \; |\overline{\Phi}_1 - \overline{\Phi}_o|.$$

Consequently, for convergence we need to show

$\alpha < 1$ for all ρ and s in domain.

It is deduced that convergence is strongly dependent on material properties, initial and boundary conditions, and the source term. The main difficulty in proving convergence is the evaluation of the D' term. We have

$$D' = \frac{dD}{d\phi}.$$

However, since the material property functions are usually given, the numerical evaluation of the radius of convergence, α, is possible. For a description of convergence of iterative methods see Blum (1972), Ortega and Rheinboldt (1970), and Lieberstein (1968).

IV.5. The Slab Reactor Problem

As a specific example a one dimensional steady state problem for a slab reactor is considered in this section. The geometry of the problem is shown in Figure IV.3.

The governing equation is the steady state diffusion equation expressed in the one dimensional cartesian coordinate form.

$$\frac{d}{dx}\left(D\frac{d\phi}{dx}\right) - \Sigma_a\phi + \frac{\nu\Sigma_f}{k}\phi = 0 \qquad \text{IV.23}$$

leakage - absorption + fission = 0,

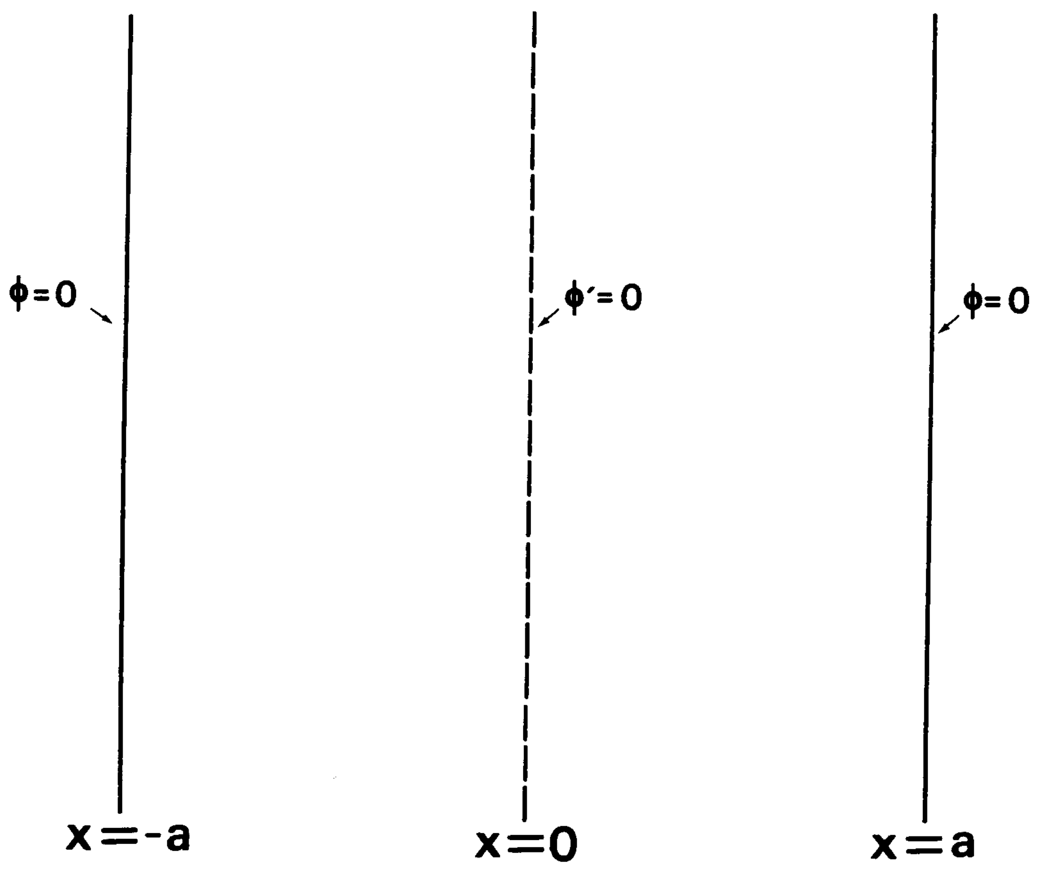


Figure IV.3

The slab reactor problem

where

D	= diffusion coefficient,
Σ_a	= macroscopic absorption cross section,
$\nu\Sigma_f/\Sigma_a$	= infinite multiplication factor = k_∞,
k	= multiplication factor.

The boundary conditions are

1. $\phi'(0) = 0$,
2. $\phi(\pm a) = 0$.

IV.5.1. **Analytical Solution**

An analytical solution to this problem can be obtained by using conventional methods for solution of ordinary differential equations, as:

$$\phi(x) = C_1 \sin(B_m x) + C_2 \cos(B_m x),$$

where B_m is known as the material buckling and C_1 and C_2 are arbitrary constants. Applying boundary conditions 1 and 2 respectively

$$\phi'(0) = C_1 B \cos(0) - C_2 B \sin(0) = 0.$$

Therefore

$$C_1 = 0,$$

and

$$\phi(a) = C_2 \cos(B_m a) = 0, \qquad B_n = \frac{n\pi}{2a}.$$

It is apparent that a solution will exist only if the values of B_n (geometric buckling) and B_m (material buckling) are equal. Hence, the analytical solutions to equation IV.23 may be expressed as

$$\phi(x) = C_2 \cos(B_n x) \quad \text{where} \quad B_n = \frac{n\pi}{2a},$$

$$\text{and} \quad n = 1, 3, 5, 7, \ldots .$$

It is observed that more than one solution to equation IV.23 exists. However, the fundamental mode of this solution set will dominate and all other harmonics will vanish (Lamarsh, 1972 and Duderstadt and Hamilton, 1976). Thus

$$\phi(x) = C_2 \cos(Bx) \quad \text{where} \quad B = \frac{\pi}{2a}.$$

The variable B is known as the geometric buckling.

IV.5.2. Describing Function Solution

For the describing function method of solution new variables are defined as

$$\frac{dC}{dx} = D\frac{d\phi}{dx} \qquad C = \int D\left(\frac{d\phi}{dx}\right) dx,$$

and

$$A = \lambda\phi,$$

where

$$\lambda = \frac{\nu\Sigma_f}{k} - \Sigma_a.$$

The spatial transformation in this case is a Fourier transform (refer to appendix B).

$$\bar{f}(\alpha_\ell) = \int_{-a}^{a} f(x)\cos(\alpha_\ell x)\, dx.$$

This transform converts the Laplacian operator into the desired form, i.e.,

$$I = \int_{-a}^{a} \frac{d^2f(x)}{dx^2} \cos(\alpha_\ell x)\ dx.$$

Integration by parts yields

$$I = \frac{df}{dx}\cos(\alpha_\ell x)\Bigg|_{-a}^{a} + \alpha_\ell \int_{-a}^{a} \frac{df}{dx} \sin(\alpha_\ell x)\ dx.$$

The first term introduces a constraint on the transformation variable (for example, we require the transformation kernel to vanish on the boundary).

$$\cos(\alpha_\ell a) = 0 \qquad\qquad \alpha_\ell = \frac{\ell\pi}{2a}, \quad \ell \text{ odd.}$$

Integrating by parts again, yields

$$I = \alpha_\ell[\ f(x)\ \sin(\alpha_\ell x)\Bigg|_{-a}^{a} - \alpha_\ell \int_{-a}^{a} f(x)\ \cos(\alpha_\ell x)\ dx\ ].$$

The first term must vanish, therefore, a second requirement is that $f(\pm a) = 0$. This second condition is identical to the second boundary condition that was imposed on the flux. Summarizing

$$\int_{-a}^{a} \frac{d^2f(x)}{dx^2} \cos(\alpha_\ell x)\, dx = -\alpha_\ell^2 \int_{-a}^{a} f(x) \cos(\alpha_\ell x)\, dx,$$

or equivalently

$$B\left[\frac{d^2f(x)}{dx^2}\right] = -\alpha_\ell^2\, \overline{f}(\alpha_\ell).$$

An equation for the inverse transformation is also required. Recall the definition for the transform

$$\overline{f}(\alpha_\ell) = \int_{-a}^{a} f(x) \cos(\alpha_\ell x)\, dx, \qquad \text{IV.24}$$

where f(x) is a smooth, differentiable function and can be expressed in terms of a Fourier series

$$f(x) = \sum_{\ell=0}^{\infty} [a_\ell \sin(\alpha_\ell x) + b_\ell \cos(\alpha_\ell x)].$$

Substituting from IV.24 into the above equation yields:

$$\overline{f}(\alpha_\ell) = \sum_{\ell=0}^{\infty} [a_\ell \int_{-a}^{a} \cos(\alpha_\ell x)\, \sin(\alpha_\ell x)\, dx$$

$$+ b_\ell \int_{-a}^{a} \cos(\alpha_\ell x)\, \cos(\alpha_\ell x)\, dx]. \qquad \text{IV.25}$$

Using orthogonality of the Fourier kernels the first term in the summation and all the perturbations of the second term vanish, except when ℓ=n in the second term. Therefore, after integration, equation IV.25 becomes

$$\overline{f}(\alpha_\ell) = b_\ell\, a.$$

Consequently, the inverse transformation becomes

$$f(x) = \frac{1}{a} \sum_{\substack{\ell=1 \\ \ell,\ \text{odd}}}^{\infty} \overline{f}(\alpha_\ell)\, \cos(\alpha_\ell x). \qquad \text{IV.26}$$

The algebraic form of the governing equation in the transformed space can be written as

$$- \alpha_\ell^2\, \overline{C}(\alpha_\ell) + \overline{A}(\alpha_\ell) = 0, \qquad \text{IV.27}$$

where, the describing functions are defined as before

$$D_L = \frac{\overline{C}(\alpha_\ell)}{\overline{\Phi}(\alpha_\ell)},$$

and

$$D_A = \frac{\overline{A}(\alpha_\ell)}{\overline{\Phi}(\alpha_\ell)}.$$

If D and λ are assumed to be constant then a simpler form for the describing functions can be obtained as:

$$D_L = \frac{D\overline{\Phi}(\alpha_\ell)}{\overline{\Phi}(\alpha_\ell)} = D, \qquad \text{IV.28}$$

and

$$D_A = \frac{\lambda\overline{\Phi}(\alpha_\ell)}{\overline{\Phi}(\alpha_\ell)} = \lambda = \frac{\nu\Sigma_f}{k} - \Sigma_a. \qquad \text{IV.29}$$

Interestingly enough, the describing functions have a direct relationship with material properties within the system. In a sense the describing function "describes" to us the inherent material characteristics, be it linear or nonlinear (hence, the name describing function).

An iterative method can now be applied to this system. Like all other iterative techniques an imbalance in the system is introduced by estimating an initial solution to the problem. Therefore,

$$\bar{\Phi}_i = \frac{(\lambda/D)}{\alpha_\ell^2}\bar{\Phi}_{i-1},$$

or

$$\bar{\Phi}_i = \left[\frac{(\lambda/D)}{\alpha_\ell^2}\right]^i \bar{\Phi}_o. \qquad \text{IV.30}$$

using the inverse transform we have:

$$\phi_i(x) = \frac{1}{a}\sum_{\ell=1}^{\infty}\frac{\lambda}{D\alpha_\ell^2}\bar{\Phi}_o(\alpha_\ell)\cos(\alpha_\ell x). \qquad \text{IV.31}$$

IV.5.3. Convergence

The convergence test in the iterative method must be performed over the entire domain of the problem to assure uniform convergence. An error function E(x) can be defined in the following manner

$$E(x) = \phi_{i+1} - \phi_i$$

$$= \frac{1}{a} \sum_{\ell=1}^{\infty} \left[\frac{B_m}{\alpha_\ell}\right]^{2i} [\overline{\Phi}_1(\alpha_\ell) - \overline{\Phi}_o(\alpha_\ell)] \cos(\alpha_\ell x)$$

$$= \frac{1}{a} \sum_{\ell=1}^{\infty} \left[\frac{B_m}{\alpha_\ell}\right]^{2i} \left[\frac{B_m^2}{\alpha_\ell^2} - 1\right] \overline{\Phi}_o(\alpha_\ell) \cos(\alpha_\ell x), \qquad \text{IV.32}$$

where

B_m^2 = material buckling.

It is apparent that for convergence $\overline{\phi}_o(\alpha_\ell)$ and $(B_m/\alpha_\ell)^{2i}$ must be finite for all values of n and i. The first condition can be satisfied by proper choice of $\phi_o(x)$. The second can only be satisfied if $B_m^2 <= \alpha_\ell^2$ for all values of n. Since

$$\alpha_\ell = \frac{\ell\pi}{2a},$$

it can be deduced that for a non-vanishing convergence ($\phi \neq 0$), B_m^2 must equal the minimum value of α_ℓ which is α_1. Thus

$$B_m = \alpha_1 = \frac{\pi}{2a}. \qquad \text{IV.33}$$

This condition is known as the criticality condition and enables one to solve for the multiplication factor. Thus

$$k = \frac{k_\infty}{1 + L^2B^2}. \qquad \text{IV.34}$$

The final form of the solution for ith iteration can be expressed as

$$\phi_i(x) = \frac{1}{a} \sum_{\substack{\ell=1 \\ \text{odd}}}^{\infty} \frac{1}{\ell^{2i}} \Phi_o(\alpha_\ell) \cos(\alpha_\ell x).$$

An initial estimate of the flux, $\phi_o(x)$ is necessary to start the iteration. Let

$$\phi_o(x) = E\frac{\pi}{4} \qquad \text{an arbitrary constant,}$$

then

$$\overline{\Phi}_o(\alpha_\ell) = E\,\frac{\pi}{4}\int_{-a}^{a}\cos(\alpha_\ell x)\;dx$$

$$= \frac{4a(-1)^\ell E}{\ell\pi} \qquad \text{for all } \ell\text{, odd.} \qquad \text{IV.35}$$

For this input guess the ith iteration solution can be written as:

$$\phi_i(x) = E\sum_{\substack{\ell=1\\ \text{odd}}}^{\infty}\frac{(-1)^{\ell+1}}{\ell^{2i+1}}\cos(\alpha_\ell x). \qquad \text{IV.36}$$

IV.5.4. **Numerical Comparison to the Analytical Solution**

A Pascal computer program was written to evaluate the analytical and describing function solutions (Appendix I) which also produces proper plot files for successive iterations (for a description of the Pascal programing language and system requirements see Jensen and Wirth, 1974 or Koffman, 1982). The HIPLOT plotting routine (Appendix F) is then used to produce a graphical display of the iterations.

Figure IV.4 shows the flux profile at each iteration. It is observed that the iterative method converges to the analytical solution in only two iterations. Figure IV.5 illustrates the number of terms required to obtain a numerical value of the flux at each grid point. The number of significant terms in the summation decreases as more iterations are performed, i.e., less computation time is spent at higher iteration levels. Hence, less and less effort is required to improve the answer as the iteration progresses.

Figures IV.6 and IV.7 illustrate the error function E(x) (equation IV.32) and the deviation of each iterative solution from the true solution, respectively. It is observed that these two functions resemble the same type of behavior, which is an indication of the stability of the iterative method. Figure IV.8 is a perspective illustration of the "iteration surface" for this problem. It is observed that convergence is achieved over the entire domain of the problem. It should also be mentioned that for a parabolic initial guess the solution would converge in only one iteration with a maximum error of 1×10^{-7}.

These results are obtained for the one dimensional steady state problem considered. The describing function method must be validated numerically for more sophisticated diffusion problems.

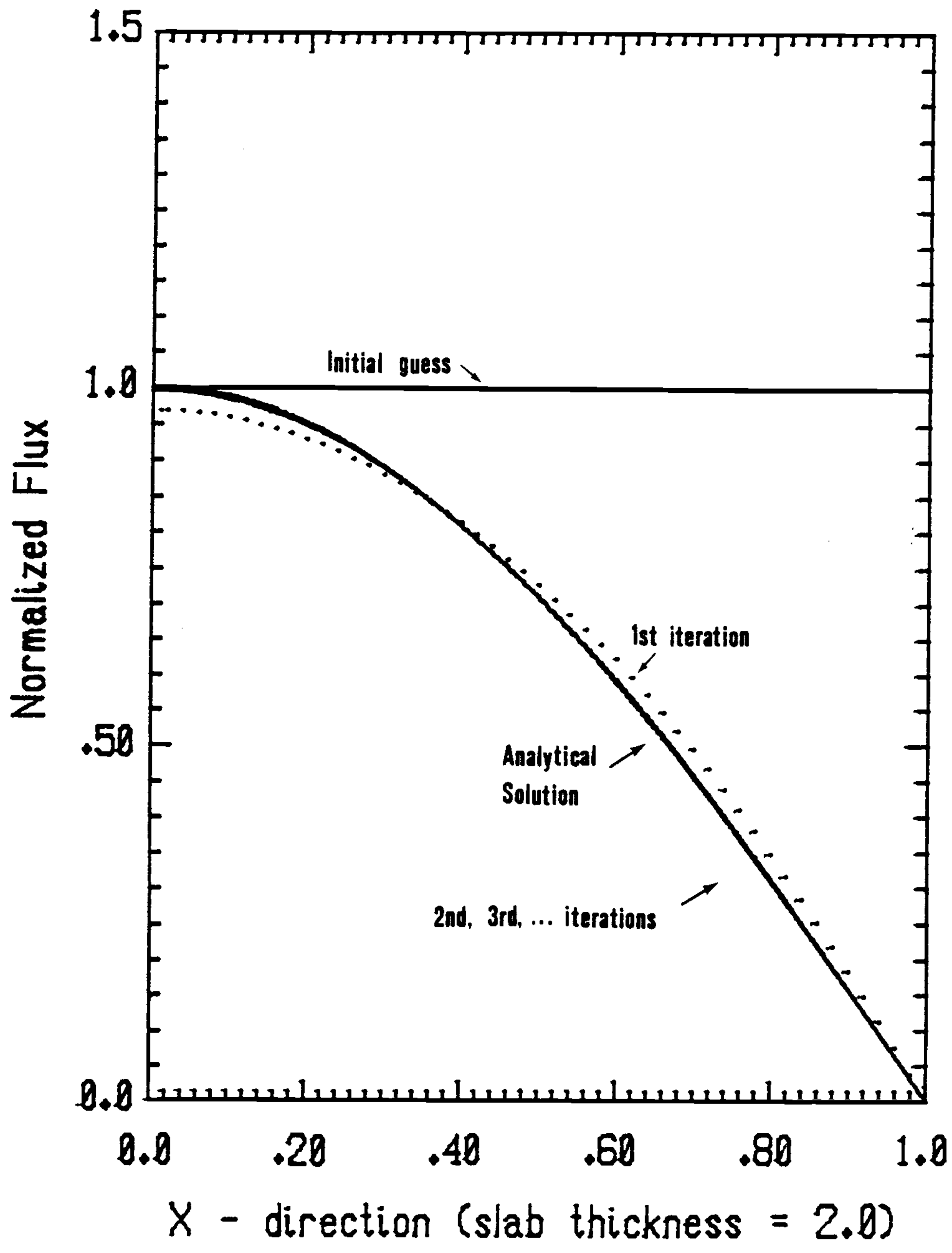


Figure IV.4

Comparison of analytical and describing function solutions for the slab reactor problem

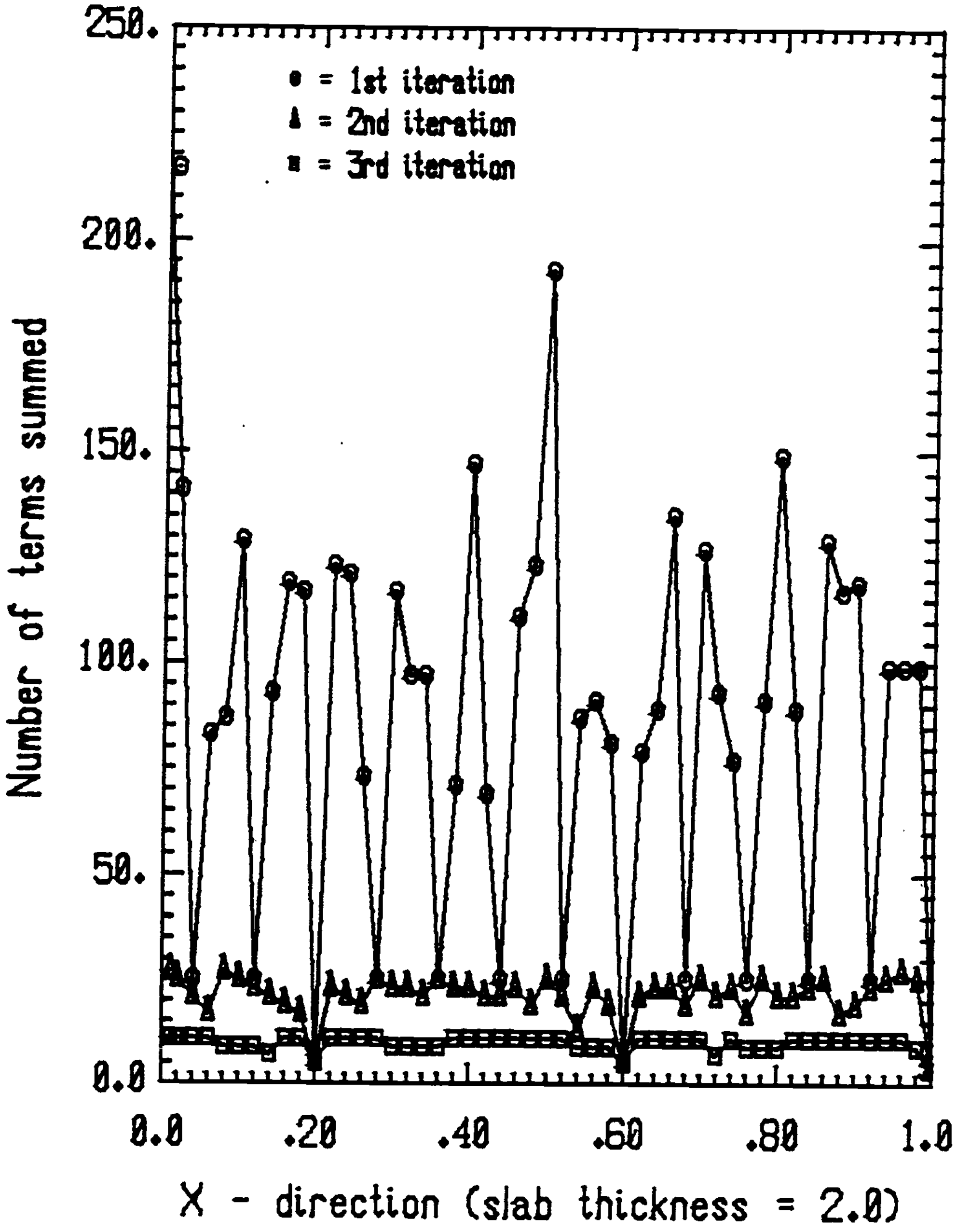


Figure IV.5

Number of terms summed in the describing function

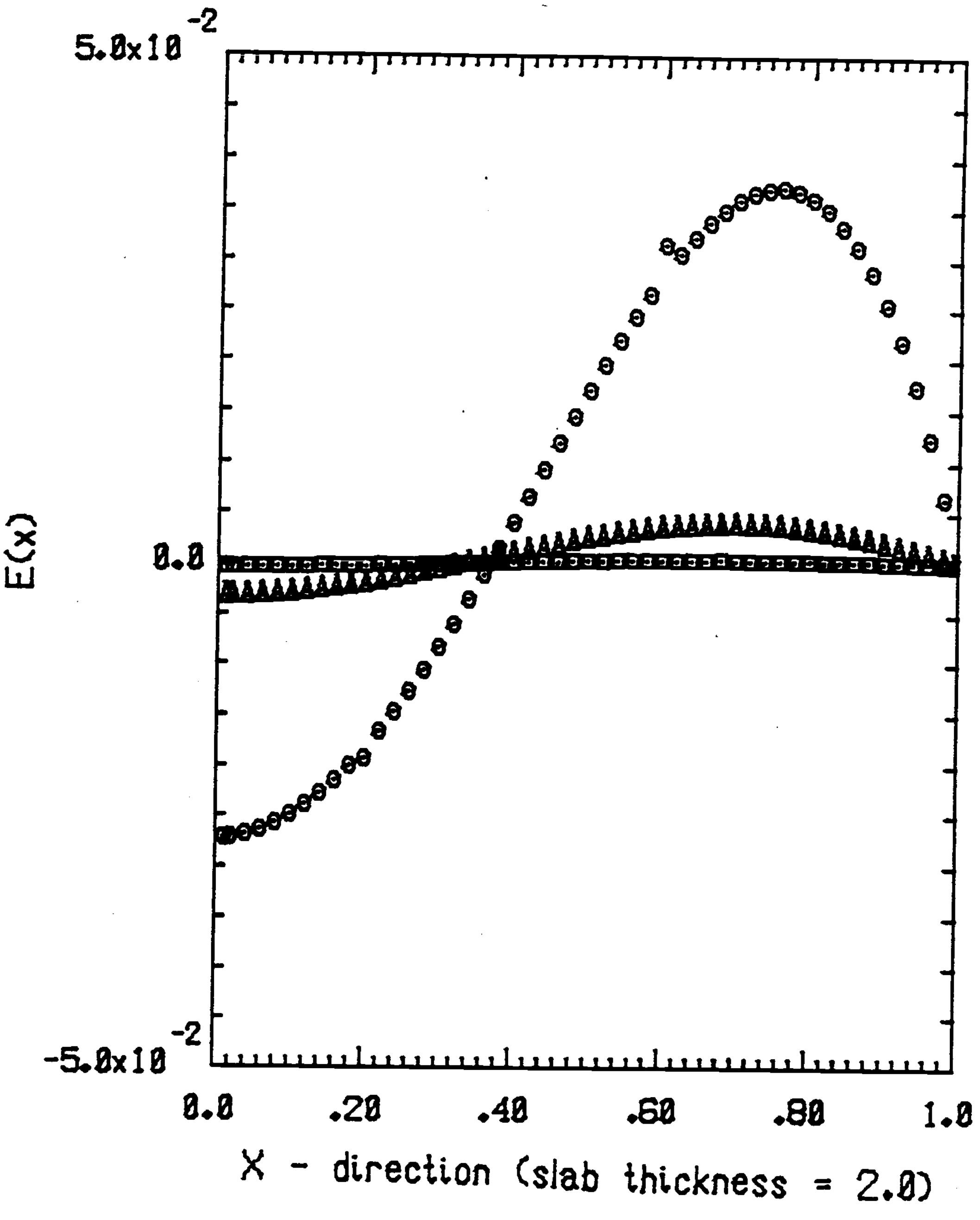


Figure IV.6

The error function E(x)

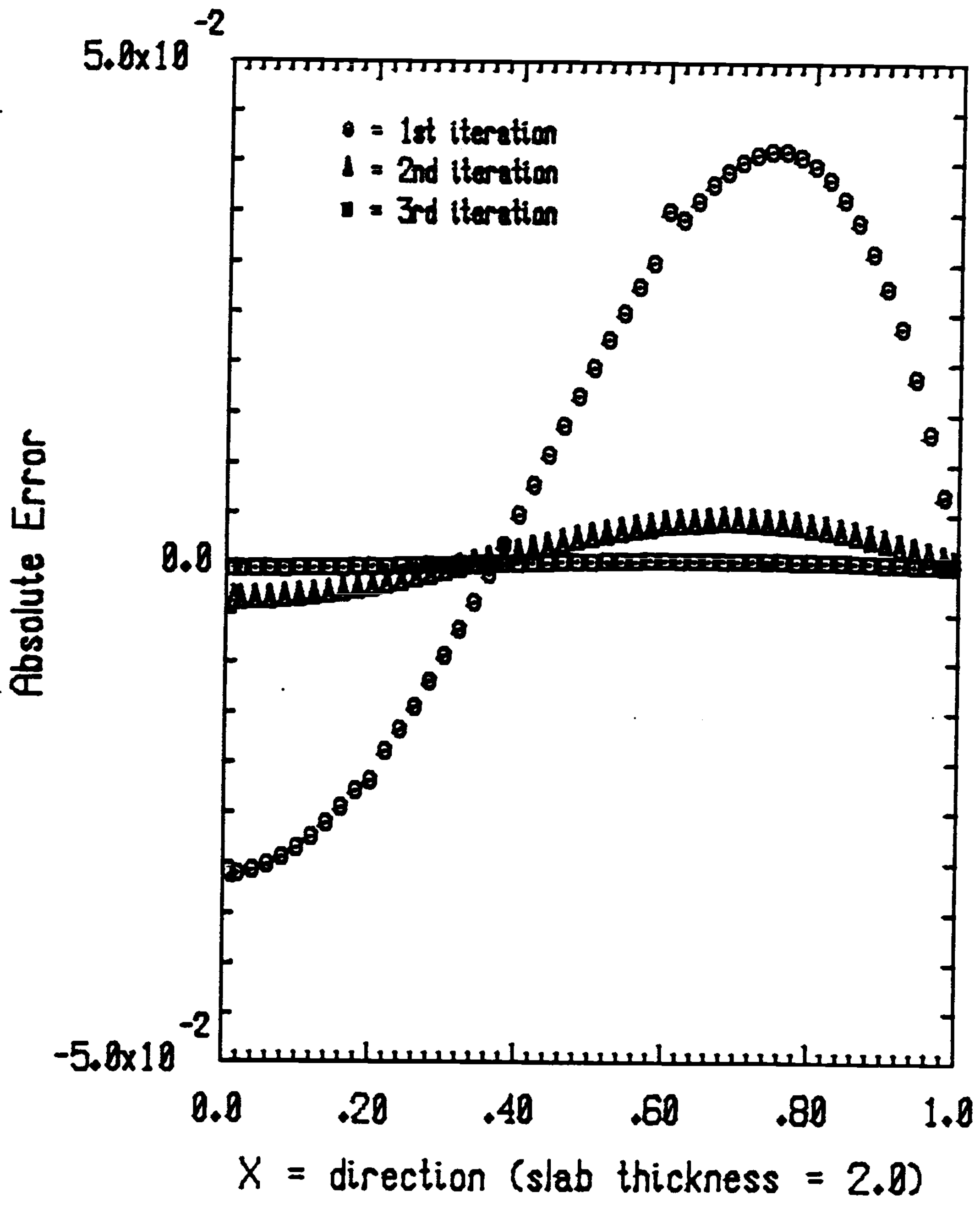


Figure IV.7

Deviation from true solution

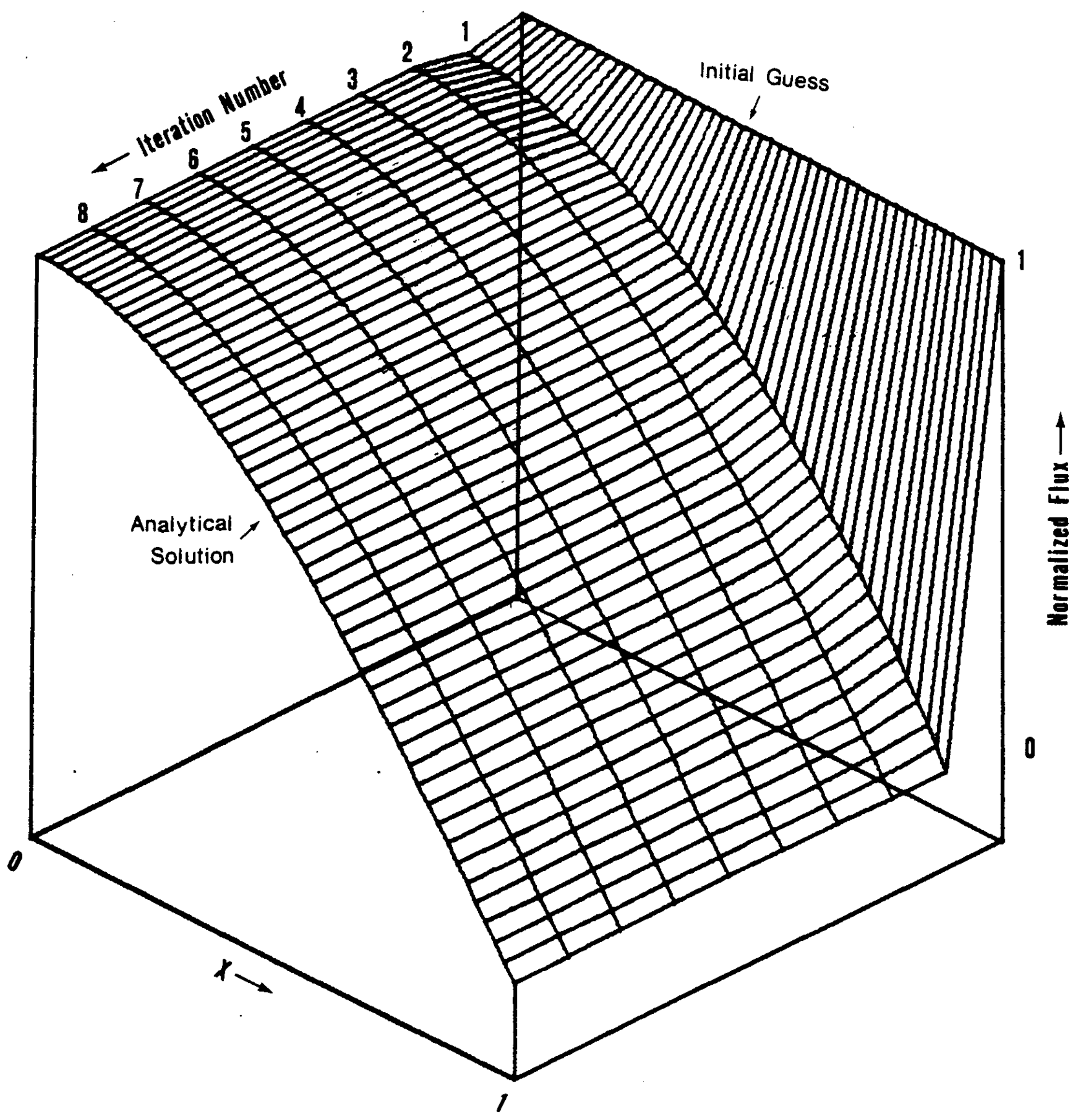


Figure IV.8

The iteration surface for
the neutron diffusion equation

V. CONCLUSIONS AND RECOMMENDATIONS

V.1. Summary

Various linearization methods have been used for solution of nonlinear integro-differential equations most of which assume a discrete interval approach. The transfer function method has traditionally been applied to systems of linear ordinary differential equations. The describing function for a nonlinear system is similar to the transfer function to a linear system. In this context the describing function has been applied to nonlinear partial differential equations (as well as linear ones). In the work developed in this paper a general three-step solution algorithm for partial differential equations is proposed as: variable substitutions, integral transformations, and an iterative solution by an approximation of the input function. The solution algorithm is then verified by two sample classes of problems.

The first example concerns the nonlinear heat conduction equation. A solution technique for the three dimensional transient problem is derived. The describing function solution technique is then demonstrated for a one dimensional nonlinear transient case, the nuclear fuel pin heat transfer problem. The computer program DFUNC (Appendix H) uses the analytical method to calculate fuel pin temperature field. For an exponential decay heat source (ANS 5.1 standard) the code produced excellent results compared to COYOTE (a finite element heat transfer code). For this case the describing function was derived based on an exponential input nominal. The same describing function produced poor results for a sinusoidal heat source. The results for the sinusoidal heat source were improved significantly by using a sinusoidal describing function. A fast convergence property was observed for the describing function method (2.5 CPU seconds for DFUNC as compared to 87 CPU seconds for COYOTE). For all the runs made with DFUNC the solution was obtained in only 4 to 6 iterations over the entire domain.

The second partial differential equation considered was the general form of the neutron diffusion equation. This equation, although similar to the heat conduction equation, has some different characteristics. The source term in this equation may be a function of the neutron flux (fission phenomenon) and the diffusion coefficient may or may not be a function of position. Therefore, two cases were considered. A solution for both cases was derived and convergence of the iterative technique discussed. As a specific example the slab reactor problem was considered. This problem was solved analytically and by using describing function method. A Pascal computer program DFdiffusion was developed to compare the analytical and describing function solutions. The describing function solution converged to the real solution in only two iterations. The convergence criteria in this case was identical to the criticality condition. An expression for the effective multiplication factor was derived. This equation is identical to the characteristic equation for criticality.

Two integral transform methods were developed as: the T-transform for time domain and the B-transform for the spatial domain. The T-transform is characterized by an exponential kernel. The B-transform is the Green's function solution to the Helmholtz equation and is therefore dependent on the geometry of the problem.

The plots of functions of single variables (two dimensional) and polynomial regression results were produced by the HIPLOT plotting and regression routine (Appendix F) developed as a part of this work. The perspective illustrations of surfaces were generated by the ThreeD plotting routine (Appendix G). The above codes were developed on an Apple II minicomputer in UCSD Pascal programming language.

V.2. Conclusions

The describing function method as studied in this work proved to be a viable approach for solution of second order nonlinear partial differential equations. It is an intermediate method between linearization and true nonlinear treatments of differential equations. True nonlinear methods of solution offer great accuracy but with the sacrifice of rapid computation time. These methods, usually iterative in nature, are indeed few and most are extremely involved and their adaptation for computer usage is usually difficult and requires special numerical techniques. True linearization methods on the other hand are extremely simple and convenient for computer adaptation. However, they require special treatment of the problem domain and selection of the proper finite grid over which linearization is valid. General methods, unless optimized for a specific application, may require significant amount of computation time.

The describing function methodology, being a quasi-linear technique, nevertheless, has the accuracy close to that of true nonlinear methods and speeds equal to and sometimes faster than those of linearization methods. The describing function solution is obtained over the entire domain of the problem, therefore, stepwise time consuming iterations are not necessary. The describing function method does not require any matrix inversion and all the matrix operations are of the multiplication and addition type. For well behaved heat transfer problems the describing function method is considerably more accurate than true linearization methods. By a well behaved problem we mean one in which the solution is a smooth well behaved mathematical function of time and position.

The describing function approach can be applied to any problem in which it is possible to determine a nominal input function about which the response of the system can be approximated. Physical insight or crude approximation techniques can be used in the selection of the input function. This becomes a difficult task when the solution contains large amplitude noise or spikes. For these problems a nonlinear treatment might be necessary. At the other end of the spectrum, transients with small amplitude changes, the nonlinear effects become less important and a true linearization approach may provide a good approximation. The convergence also plays a significant role in the selection of a solution method. The convergence in the describing function procedure is a function of the physical characteristics of the system, the initial and boundary conditions, the input form, and the describing function itself.

The applicability of the describing function method is that the describing function is derived for a given forcing function which then determines the behavior of the system and as a result is only valid for that system. Also, a knowledge of the system behavior is required in the solution process.

The describing function method can be interfaced to finite element or finite difference techniques (Nielsen, 1980). However, this would not necessarily be attractive because by nature the solution for the describing function method is obtained over the entire domain.

Indications are that the describing function theory as developed in this work is a promising technique in solution of nonlinear problems. In theory, it is possible to apply the describing function method (or a modified form of it) to any nonlinear system which can be set up as a feedback system. However, the mathematical and numerical methods for each application must be developed essentially in a case by case basis.

V.3. Recommendations for Future studies

There are several possibilities for further development and application of the describing function method. One would be to derive a solution technique for multigroup diffusion equations. This set of equations are very similar in nature to the neutron diffusion equation. The transformed equations will form a linear algebraic system of equations which is easily solvable for the transformed group flux vector. The basic method is similar to that presented in Chapter IV. Schwinkendorf has used a technique similar in nature to the describing function approach to solve the multigroup problem.

A second application would be to derive a describing function solution algorithm for the thermomechanical behavior of fluids and solids. The set of energy, mass and momentum equations derived in Appendix D can describe any classical continuum. As explained in that section solution of this set of equations is an extremely difficult task. The describing function methodology might be combined with finite difference techniques for solution of such classes of problems. As a more specific problem, heat transfer in fluids in an annular concentric and eccentric regions may be studied for which experimental results do exist (McNair, 1982).

The technique may also be applied to large nonlinear system to which an optimization solution is desired. This application of describing functions would speed up the search for an optimal solution for the system. In a manner this would be similar to an automated reasoning system where the system response is modeled by a set of weighting functions. The weighting factors can be represented by the describing function over the entire domain of the system.

The solution methods in this work were derived for an orthogonal curvilinear coordinate system. Using material presented in Appendix C it is possible to derive solutions for nonorthogonal curvilinear coordinates. Also useful, would be an accurate assessment of the use of describing function in application to integro-differential equations such as the neutron transport equation.

REFERENCES

Abramowitz, Milton and Stegun, Irene A., Handbook of Mathematical Functions with Formulas, Graphs, and Mathematical Tables, National Bureau of Standards Applied Mathematics Series, No. 55, Washington D. C., U. S. Government Printing Office, 1964.

Akcasu, A. Z., and Bost, C. J., Jr., "The Concept of Multiple-Input Zero Power Describing Functions in Nuclear Reactors," Nuclear Science and Engineering, Vol. 47, pp. 104-115, 1972.

Akcasu, A. Z., Shotkin, L. M., "Power Oscillations and the Describing Function in Reactors with Linear Feedback," Nuclear Science and Engineering, Vol. 28, pp. 72-81, 1967.

ANS/ANSI 5.1, American Nuclear Society/American National Standards Institute standard ANS 5.1, "Decay Heat Power in Light Water Reactors," June 1978, ANS, 555 N. Kensington Ave., Lagrange Park, IL 60525.

Arpaci, V. S., Conduction Heat Transfer, Addison-Wesley Publishing Co., 1966.

Bathe, K. and Wilson, E. L., Numerical Methods in Finite Element Analysis, Prentice-Hall, Inc., 1976.

Bellman, R., et. al., Numerical Inversion of the Laplace Transform, American Elsevier Publishing Co., Inc., 1966.

Bellman, R. E. and Kalaba, R. E., Qasilinearization and Nonlinear Boundary-Value Problems, The RAND Corporation, 1965.

Bernfeld, Stephen R. and Lakshmikantham, V., An Introduction to Nonlinear Boundary Value Problems, Academic Press, Inc., 1974.

Blum, E. K., Numerical Analysis and Computation Theory and Practice, Addison-Wesley Publishing Company, Inc., 1972.

Borisenko, A. I. and Tarapov, I. E., Vector and Tensor Analysis with Applications, Dover Publications, Inc., 1968.

Braum, M., Differential Equations and Their Applications, Springer-Verlag, 1978.

Brodie, K. W., Mathematical Methods in Computer Graphics and Design, Academic Press, Inc., 1980.

Carnahan, B., Luther, H. A. and Wilkes, J. O., Applied Numerical Methods, John Wiley & Sons, Inc., 1969.

Carslaw, H. S. and Jaeger, J. E., Conduction of Heat in Solids, Oxford University Press, 1959.

Carslaw, H. S. and Jaeger, J. E., Operational Methods in Applied Mathematics, Oxford University Press, 1943.

Churchill, R. V. and Brown, J. W., Fourier Series and Boundary Value Problems, McGraw-Hill Book Company, Inc., 1978

Churchill, R. V., Modern Operational Mathematics, McGraw-Hill Book Company, Inc., 1944.

Churchill, R. V., Operational Mathematics, 2nd ed., McGraw-Hill Book Co., Inc., 1958.

Corduneanu, C., Differential and Integral Equations, Chelsea Publishing Company, 1977.

Duderstadt, J. J., Hamilton, L. J., Nuclear Reactor Analysis, John Wiley & Sons, Inc., 1976.

El-wakil, M. M., Nuclear Heat Transport, American Nuclear Society, 1978.

Fulks, W., Advanced Calculus -- An Introduction to Analysis, John Wiley & Sons, Inc., 1978.

Fucik, S. and Kufner, A., Nonlinear Differential Equations, Elsevier Scientific Publishing Company, 1980.

Gelb, A., Vander Velde, W. E., Multiple-Input Describing Functions and NOnlinear System Design, McGraw-Hill Book Co., Inc., 1968.

Gartling, D. K., "COYOTE - A Finite Element Computer Program for Nonlinear Heat Conduction Problems," SAND77-1332, 1978.

Green, M. D., Kornfilt, J., "A Digital Transfer Matrix for Fuel Pin Heat Transfer," Nuclear Science and Engineering, Vol. 65, pp. 385-393, 1978.

Guidotti, T. E., Peddicord, K. L., Nielsen, L. H., "Transient Fuel Pin Temperature Calculations Using Describing Functions," EPRI NP-2278, 1982.

Gyftopoulos, E. P., Smets, H. B., "Transfer Functions of Distributed Parameter Nuclear Reactor Systems," Nuclear Science and Engineering, Vol. 5, pp. 405-414, 1959.

Hagrman, D. L., Reymann, G. A., ed., "MATPRO - Version 11, A Handbook of Materials Properties for Use in the Analysis of Light Water Reactor Fuel Rod Behavior," EG&G Idaho, Inc., NUREG/CR-0497, TREE-1280, 1979.

Hayes, J. G., Numerical Approximation to Functions and Data, Institute of Mathematics and Its Applications, 1970.

Henry, A. F., Nuclear-Reactor Analysis, The Massachusetts Institute of Technology, 1975.

Iriarte, M., Jr., "An Accurate Transfer Function for the Dynamic Analysis of Temperature and Heat Release in Cylindrical Fuel Elements," Nuclear Science and Engineering, Vol. 7, pp. 26-32, 1960.

Kelly, L. G., Handbook of Numerical Methods and Applications, Addison-Wesley Publishing Company, Inc., 1967.

Kendall, J. M., "Data Reduction by an Approximate Thermal Diffusion Solution," EPRI-SR-8, Electric Power Research Institute, 1975.

Koffman, E. B., Pascal -- A Problem Solving Approach, Addison-Wesley Publishing Company, Inc., 1982.

Kolstad, E., "Temperature and Stored Heat Decrease in Fuel Rods Following a Reactor Scram," OECD Halden Reactor Project, Halden Users Meeting, Session 4, Paper No. 22, Leon. Norway, 1975.

Kreyszig, E., Advanced Engineering Mathematics, John Wiley & Sons, Inc., 1979.

Lamarsh, J. R., Introduction to Nuclear Reactor Theory, Addison-Wesley Publishing Company, 1966.

Lieberstein, H. M., A Course in Numerical Analysis, Harper & Row, Publishers, 1968.

Maslov, V. P., Operational Methods, Mir Publishers, Moscow, 1976.

McNair, G. W., "Natural Convection Heat Transfer in Horizontal Annuli," Ph.D. thesis, November 1982.

Myint-U., T., Partial Differential Equations of Mathematical Physics, Elsevier North Holland, Inc., 1980.

Nielsen, L. H., "An Assessment of Describing Function Method Applied to Reactor Fuel Performance Analysis," OSU-NE-8007, June 1980.

Ortega, J. M. and Rheinboldt, W. E., Iterative Solution of Nonlinear Equations in Several Variables, Academic Press, Inc., 1970.

Ralston, A., Rabinowitz, P., A First Course in Numerical Analysis, 2nd ed., International Series in Pure and Applied Mathematics, McGraw-Hill Book Co., Inc., 1978.

Rashid, Y. R., Sharabi, M. N., "FREY - A Program For Two-Dimensional Transient Fuel Rod Analysis," ANA82-1, January 1982.

Sansone, G., Orthogonal Functions, Interscience Publishers, Inc., 1959.

Schlichting, H., Boundary-Layer Theory, McGraw-Hill Book Company, Inc., 1968.

Schmidt, T. R., Hetrick D. L., "Nonlinear Oscillations and Stability of a Nuclear Reactor with Two Reactivity Feedbacks," Nuclear Science and Engineering, Vol. 42, pp. 1-9, 1970.

Schwabik, S., Tvrdy, M. and Vejvoda, O., Differential and Integral Equations Boundary Valve Problems and Adjoints, B. Riedel Publishing Company, 1979.

Schwinkendorf, K., M.S. Thesis to be published, Oregon State University, 1983.

Smets, H. B., "The Describing Function of Nuclear Reactors," IRE Transactions of Nuclear Science, NS-6, pp. 8-12, 1959.

Sneddon, I. N., Fourier Transforms, International Series in Pure and Applied Mathematics, McGraw-Hill Book Company, 1951.

Trent, D. S., Budden, M. J., and Eyler, L. L., TEMPEST A Three-Dimensional Time-Dependent Computer Program for Hydrothermal Analysis, Pacific Northwest Laboratory, Richland, Washington, 1981.

Wasserman, A. A., "Contributions to Two Problems in Space-Independent Nuclear Reactor Dynamics," IDO-16755, Phillips Petroleum Co., 1962.

Wolf, K. B., Integral Transforms in Science and Engineering, Plenum Press, 1979.

Wylie, C. R., Advanced Engineering Mathematics, McGraw-Hill Book Company, 1975.

Zemanian, A. H., Generalized Integral Transformations, John Wiley α Sons, Inc., 1978.

APPENDICES

Appendix A

LIST OF SYMBOLS

Symbol	Description
Lower case:	
a	coefficient, general use
b	coefficient, general use
$\mathbf{b}$	body forces vector
ds	arc length
$\mathbf{e}^1, \mathbf{e}^2, \mathbf{e}^3$	coordinate vectors
$\mathbf{g}^i$	curvilinear coordinate vector
g_{ij}	components of the metric tensor
h	coolant heat transfer coefficient
$h(t)$	the Heaviside function
h_g	gap conductance
h_ℓ	diagonal elements of the fundamental metric for an orthogonal curvilinear coordinate system
$\mathbf{i}_k$	rectangular coordinate vector (Eulerian)
k	thermal conductivity, multiplication factor
k_c	clad thermal conductivity
k_f	fuel thermal conductivity
k_∞	infinite multiplication factor
m_i	moments for numerical integration
n	neutron number density
$\mathbf{n}$	normal unit vector

Symbol	Description
q''_f	fuel radial heat flux vector
q'''	volumetric heat source
q'''_s	fuel surface heat source
r	radial direction in cylindrical coordinates
$\mathbf{r}$	position vector
s	Laplace domain variable
t	time variable
$\tilde{\mathbf{t}}$	stress tensor
v	monoenergetic neutron speed
$\mathbf{v}$	vector field
v^i	vector components
w	frequency
w_i	weights for numerical integration
$\mathbf{x}$	curvilinear coordinates
x_i	base points for numerical integration
x^1,x^2,x^3	coordinate directions
z	volumetric plus radiative heat source
$\mathbf{z}$	Cartesian coordinate system (Eulerian)

Upper case:

A	absorption function, surface area
B	buckling
B_m	material buckling
B_n	Bessel function cross product, geometric buckling
C	heat capacity, current potential function

Symbol	Description
$\mathbf{C}$	heat flux vector
D	diffusion coefficient, deadband constant, describing function
D_A	absorption describing function
D_{th}	thermal diffusion describing function
D_L	leakage describing function
$\tilde{\mathbf{D}}$	deformation tensor
E	error function for neutron diffusion solution
G	determinant of the metric
G^{ij}	cofactor matrix for g_{ij}
I	integral conductivity
$\mathbf{I}_K$	Cartesian coordinate vector (Lagrangian)
I_s	fuel surface integral conductivity
J	the Jacobian
$\mathbf{J}$	neutron current vector, Jacobian matrix
J_n	ordinary Bessel function of the first kind
K	transformation kernel
L_i	leakage functions, i=1,2,3
$\mathbf{N}$	unit normal
R_1	center hole radius
R_2	fuel outer radius
R_3	clad inner radius
R_4	clad outer radius
S	neutron source function, surface area
S_t	deformed surface

Symbol	Description
T	temperature
T_c	clad temperature
T_f	fuel temperature
T_{ref}	reference temperature
T_∞	coolant temperature
U	integral heat capacity
U_s	fuel surface integral heat capacity
V	volume
V_o	undeformed volume
V_t	deformed volume
Y_n	ordinary Bessel function of the second kind
Z	Cartesian coordinate system (Lagrangian)
Greek:	
α	thermal diffusivity, general use coefficient
α_c	clad thermal diffusivity, radius of convergence
$\delta(t)$	Dirac delta function
Θ	angle, general use
$\pi(x)$	polynomial function
ρ	density, B-transform variable
$\boldsymbol{\rho}$	B-transform variable vector
ρ_ℓ	discrete values of the Hankel transform variable
Σ	summation
Σ_a	macroscopic absorption cross section
Σ_f	macroscopic fission cross section
ϕ	neutron flux

Symbol	Description
Ω	functional coefficient for B-transform of the Laplacian
Operators:	
B_1	B-transform operator of the first kind
B_2	B-transform operator of the second kind
div	divergence operator
grad	gradient operator also represented by ∇
L	Laplace transform operator
Lap	the Laplace operator also represented as Δ or ∇^2
T	time transform operator
Miscellaneous:	
;	covariant differentiation
:	diadic product
[i,jk]	Christoffel symbol of the first kind
$\left\{ \begin{matrix} i \\ j\ k \end{matrix} \right\}$	Christoffel symbol of the second kind
*	superscript for physical components
ℓ	integer subscript
$\bar{}$	overscore bar indicates integral transformed variable
$\sim$	overscore tilde indicates tensor field

Appendix B

INTEGRAL TRANSFORMS

B.1. Introduction

Among the main mathematical methods for solution of differential or integro-differential equations describing physical systems are integral transforms. The coupling between the elements which constitute a system, quite generally, prevents a straight forward solution. By describing the same system in an appropriate reference frame one can often bring about a mathematical uncoupling of the equations in such a way that the solution becomes that of noninteracting constituents. The "tilt" in the reference is introduced by an integral transformation.

A generalized form of integral transform is defined by a mathematical representation as

$$\bar{f}(\boldsymbol{\rho}) = \int_{\mathrm{Vol}} K(\mathbf{r},\boldsymbol{\rho})\ f(\mathbf{r})\ dV,$$

where, the kernel, $K(\mathbf{r},\boldsymbol{\rho})$, is a specified function of $\mathbf{r}$ and $\boldsymbol{\rho}$, and $\bar{f}(\boldsymbol{\rho})$ is the transform of the function $f(\mathbf{r})$. The transformation variable $\boldsymbol{\rho}$ is determined by proper initial or boundary conditions.

The differential equation can therefore be transformed into an algebraic form and it is possible to solve for the unknown transformed variable. Then if the transformation is invertible, the solution to the differential system can be obtained.

Two general integral transforms are often useful in physical problems. The first is a generalized time transformation and the second a parabolic type spatial transformation.

B.2. Time Transformation (the T-transform)

The generalized time transform may be defined as

$$\bar{f}(s) = T\,[f(t)] = \int_{\alpha}^{\beta} K(s,t)\, f(t)\, dt. \qquad \text{B.1}$$

For the first order differential time problems the following property is required.

$$T\left[\frac{df(t)}{dt}\right] = \Omega(\alpha,\beta,s)\,\bar{f}(s) + C(\alpha,\beta,s). \qquad \text{B.2}$$

Using definition B.1 and equation B.2 it is possible to find a relationship for determining the transformation kernel K(s,t) and the functions $\Omega(\alpha,\beta,s)$ and $C(\alpha,\beta,s)$. Integrating by parts:

$$\int_{\alpha}^{\beta} K(s,t) \frac{df}{dt} dt = K(s,t)\, f(t)\Big|_{\alpha}^{\beta} - \int_{\alpha}^{\beta} \frac{dK(s,t)}{dt} f(t)\, dt. \qquad \text{B.3}$$

By comparing equations B.1 and B.3 it is apparent that

$$C(\alpha,\beta,s) = K(s,t)\, f(t)\Big|_{\alpha}^{\beta} , \qquad \text{B.4}$$

and

$$\frac{dK(s,t)}{dt} = \Omega(\alpha,\beta,s)\, K(s,t). \qquad \text{B.5}$$

Equation B.5 is an ordinary differential equation and has a unique solution of the form

$$K(s,t) = A\, e^{-\Omega(\alpha,\beta,s)\, t} , \qquad \text{B.6}$$

where, A is an arbitrary constant and can be set equal to one. Therefore, the time transformation can be presented as

$$f(s) = T\,[f(t)] = \int_{\alpha}^{\beta} e^{-\Omega(\alpha,\beta,s)t}\, f(t)\, dt.$$

The function $\Omega(\alpha,\beta,s)$ is an arbitrary function. For instance, if we set

$$\Omega(\alpha,\beta,s) = s,$$

$$\alpha = 0,$$

and

$$\beta = \infty,$$

then the T-transform is identical to the Laplace transform. The T-transform may be used in a variety of problems where first order time differentials are present. To summarize,

$$\bar{f}(s) = T\,[f(t)] = \int_{\alpha}^{\beta} e^{-\Omega(\alpha,\beta,s)t}\, f(t)\, dt,$$

and

$$T\left[\frac{df(t)}{dt}\right] = e^{-\Omega(\alpha,\beta,s)\beta} f(\beta) - e^{-\Omega(\alpha,\beta,s)\alpha} f(\alpha)$$

$$- \Omega(\alpha,\beta,s)\,\bar{f}(s). \qquad \text{B.7}$$

Note that if α and β are both finite then the values of the function at the time interval end points must be known. The special case of the T-transform as Laplace transform requires only an initial condition. However, because of the infinite interval of integration the Laplace transform presents enormous numerical difficulties. The transformation matrix is extremely ill-conditioned (condition number of the order of 10^{17}) and the inverse transform is almost hopeless. The finite limits on the other hand are numerically well behaved.

B.3. Spatial Transformation (the B-transform)

The generalized spatial transform can be expressed as

$$\bar{f}(\boldsymbol{\rho}) = B\,[f(\mathbf{r})] = \int_{Vol} K(\mathbf{r},\boldsymbol{\rho})\, f(\mathbf{r})\, dV. \qquad \text{B.8}$$

Two cases arise for parabolic and hyperbolic equations (or any equation where a Laplacian term is present); namely, linear and nonlinear cases.

B.3.1. B-transform of the first kind (linear form)

For equations with a linear Laplacian term the following property will be useful

$$B\,[\mathrm{Lap}(f(\mathbf{r}))] = -\,\Omega(\boldsymbol{\rho})\,B\,[f(\mathbf{r})]. \qquad \text{B.9}$$

To find an explicit form for $K(\mathbf{r},\boldsymbol{\rho})$ and $\Omega(\boldsymbol{\rho})$ the left hand side of equation B.9 must be evaluated. In this connection the following vector identity is useful:

$$\mathrm{div}(a\mathbf{b}) = a\,\mathrm{div}\mathbf{b} + \mathbf{b}\cdot\mathrm{grad}(a). \qquad \text{B.10}$$

in which a and **b** are scalar and vector functions respectively. Then

$$\int_V K(\mathbf{r},\boldsymbol{\rho})\,\mathrm{Lap}(f(\mathbf{r}))\,dV =$$

$$\int_V \mathrm{div}[K(\mathbf{r},\boldsymbol{\rho})\,\mathrm{grad}(f(\mathbf{r}))]\,dV$$

$$-\int_V \mathrm{grad}(f(\mathbf{r}))\cdot\mathrm{grad}(K(\mathbf{r},\boldsymbol{\rho}))\,dV. \qquad \text{B.11}$$

From the divergence theorem, however,

$$\int_V \mathrm{div}[K(\mathbf{r},\boldsymbol{\rho})\ \mathrm{grad}(f(\mathbf{r}))]\ dV = \int_S K(\mathbf{r},\boldsymbol{\rho})\ \mathrm{grad}(f(\mathbf{r}))\cdot\mathbf{n}\ dS. \qquad \text{B.12}$$

Using identity B.10 again, it is observed that

$$\int_V \mathrm{grad}(f(\mathbf{r}))\ \cdot\ \mathrm{grad}(K(\mathbf{r},\boldsymbol{\rho}))\ dV =$$

$$\int_V \mathrm{div}[f(\mathbf{r})\ \mathrm{grad}(K(\mathbf{r},\boldsymbol{\rho}))]\ dV - \int_V \mathrm{div}(\mathrm{grad}(K(\mathbf{r},\boldsymbol{\rho})))\ f(\mathbf{r})\ dV. \qquad \text{B.13}$$

Substituting these results into equation B.11 and making use of the divergence theorem

$$\int_V K(\mathbf{r},\boldsymbol{\rho})\ f(\mathbf{r})\ dV =$$

$$\int_S [K(\mathbf{r},\boldsymbol{\rho})\ \mathrm{grad}(f(\mathbf{r})) - f(\mathbf{r})\ \mathrm{grad}(K(\mathbf{r},\boldsymbol{\rho}))]\cdot\mathbf{n}\ dS$$

$$+ \int_V f(\mathbf{r})\ \mathrm{div}(\mathrm{grad}(K(\mathbf{r},\boldsymbol{\rho})))\ dV. \qquad \text{B.14}$$

In order for property B.9 to exist, the following relations must hold:

$$\int_S [K(\mathbf{r},\boldsymbol{\rho}) \operatorname{grad}(f(\mathbf{r})) - f(\mathbf{r}) \operatorname{grad}(K(\mathbf{r},\boldsymbol{\rho}))]\cdot\mathbf{n}\, dS = 0, \qquad \text{B.15}$$

and

$$\operatorname{Lap}(K(\mathbf{r},\boldsymbol{\rho})) = - \Omega(\boldsymbol{\rho})\, K(\mathbf{r},\boldsymbol{\rho}). \qquad \text{B.16}$$

We shall call this transform the B-transform of the first kind.

B.3.2. **B-transform of the second kind (nonlinear form)**

For nonlinear parabolic equations the differential term of concern is the nonlinear form of the Laplacian. For example, in general

$$\operatorname{div}(\Omega \operatorname{grad} f),$$

where

Ω = function of the coordinate directions (or f),

and

f = function of the coordinate directions.

Next we define the quantities, grad f, and, div v, in a system of generalized coordinates x^1, x^2, x^3 with coordinate vectors e^1, e^2, $\mathbf{e}^3$. We use the generalized definitions in appendix C on tensor analysis to derive explicit expressions for the following operators.

B.3.2.1. **The gradient operator** - By the gradient of a scalar field $f = f(x^1,x^2,x^3)$ we mean the vector with coordinate components

$$\frac{\partial f}{\partial x^i} \cdot$$

Thus, introducing the "grad" operator

$$\text{grad} = \frac{\partial}{\partial x^i}\,\mathbf{g}^i.$$

We have

$$\text{grad } f = \frac{\partial f}{\partial x^i}\,\mathbf{g}^i = \nabla f. \qquad \text{B.17}$$

The physical components of grad f are

$$(\text{grad } f)^*_i = \frac{1}{\sqrt{g_{ii}}}\frac{\partial f}{\partial x^i} \text{ (no summation over i).} \qquad \text{B.18}$$

In the case of orthogonal coordinates the metric tensor is a diagonal matrix. In rectangular Cartesian coordinates,

$$(g_{ij}) = \begin{Bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{Bmatrix},$$

and hence

$h_1 = 1,\ h_2 = 1,\ h_3 = 1.$

In cylindrical coordinates,

$$(g_{ij}) = \begin{Bmatrix} 1 & 0 & 0 \\ 0 & (x^1)^2 & 0 \\ 0 & 0 & 1 \end{Bmatrix},$$

and hence

$h_1 = 1,\ h_2 = x^1 = r,\ h_3 = 1.$

In spherical coordinates,

$$(g_{ij}) = \begin{Bmatrix} 1 & 0 & 0 \\ 0 & (x^1)^2 & 0 \\ 0 & 0 & [x^1 \sin(x^2))]^2 \end{Bmatrix},$$

and hence

$h_1 = 1,\ h_2 = r,\ h_3 = r \sin(\theta).$

Equation B.18 becomes

$$(\text{grad } f)^*_i = \frac{1}{h_i} \frac{\partial f}{\partial x^i}. \qquad \text{B.19}$$

In terms of the metric coefficients h_i.

B.3.2.2. **The divergence operator** - The divergence of a vector field $\mathbf{v} = \mathbf{v}(x^1,x^2,x^3)$ is defined as the contraction of the (mixed) covarient derivative of **v** (div v is the first invariant of $v^i{}_{;j}$), i.e.,

$$\text{div } v = v^i{}_{;i} = \frac{\partial v^i}{\partial x^i} + \begin{Bmatrix} i \\ i\ j \end{Bmatrix} v^j. \qquad \text{B.20}$$

The sum

$$\begin{Bmatrix} i \\ i\ j \end{Bmatrix}$$

(Chrystoffel symbols of the second kind), can be expressed in terms of the metric tensor. In fact, using the symmetry of g^{ik} (the metric tensor), we have

$$\begin{Bmatrix} i \\ i\ j \end{Bmatrix} = g^{ik}\ [k,ij] = \frac{1}{2}\, g^{ik}\ ([k,ij] + [i,kj]), \qquad \text{B.21}$$

where, the Chrystoffel symbol of the first kind, [i,jk] is defined by

$$[i,jk] = \mathbf{g}_i \cdot \frac{\partial \mathbf{g}_j}{\partial x^k}, \qquad \text{B.22}$$

and

$$[k,ij] + [i,kj] = \frac{\partial g_{ik}}{\partial x^j}. \qquad \text{B.23}$$

Expanding the determinant $G = \det(g_{ik})$ with respect to elements of ith row we obtain

$$G = g_{ik}\, G^{ik} \qquad \text{(no summation over i)},$$

where G^{ik} is the cofactor of g_{ik} in the determinant G. But G^{ik} is the independent of g_{ik}, and hence

$$\frac{\partial G}{\partial g_{ik}} = G^{ik}. \qquad \text{B.24}$$

Therefore

$$g^{ik} = \frac{G^{ik}}{G} = \frac{1}{G}\,\frac{\partial G}{\partial g_{ik}}.$$

It follows from B.20 - B.24 that

$$\left\{ \begin{matrix} i \\ i\ j \end{matrix} \right\} = \frac{1}{2G} \frac{\partial G}{\partial g_{ik}} \frac{\partial g_{ik}}{\partial x^j} = \frac{1}{2G} \frac{\partial G}{\partial x^j} = \frac{1}{\sqrt{G}} \frac{\partial(\sqrt{G})}{\partial x^j}. \qquad \text{B.25}$$

Thus, finally,

$$\text{div } v = \frac{1}{G} \frac{\partial}{\partial x^i} (g^{ik} v_k \sqrt{G}) = \frac{1}{G} \sum_{i=1}^{3} \frac{\partial}{\partial x^i} [\frac{v^{*i} \sqrt{G}}{\sqrt{g_{ii}}}], \qquad \text{B.26}$$

where the v^{*i} are the physical components of the vector v. In particular,

$$\text{div } v = \frac{1}{h_1 h_2 h_3} \sum_{\ell=1}^{3} \frac{\partial}{\partial x^\ell} [\frac{v_\ell^* h_1 h_2 h_3}{h_\ell}], \qquad \text{B.27}$$

in orthogonal coordinates, where v_ℓ^* can be replaced by v_ℓ if the local basis is orthogonal. Using the above definitions

$$\text{div}(\Omega \text{ grad } f) = \frac{1}{h_1 h_2 h_3} \sum_{\ell=1}^{3} \frac{\partial}{\partial x^\ell} [\frac{h_1 h_2 h_3 \Omega}{h_\ell^2} \frac{\partial f}{\partial x^\ell}]. \qquad \text{B.28}$$

For an orthogonal coordinate system. Let

$$\frac{\partial L_i}{\partial x^i} = \frac{h_1 h_2 h_3 \Omega}{h_i^2} \frac{\partial f}{\partial x^i}, \qquad \text{B.29}$$

then

$$\operatorname{div}(\Omega \text{ grad } f) = \frac{1}{h_1 h_2 h_3} \sum_{i=1}^{3} \frac{\partial}{\partial x^i} \left[\frac{\partial L_i}{\partial x^i} \right], \qquad \text{B.30}$$

and define the component B-transform of the second kind as

$$\bar{f}(\boldsymbol{\alpha}) = \int_{a_1}^{b_1} \int_{a_2}^{b_2} \int_{a_3}^{b_3} f(\mathbf{x}) \left[\prod_{\ell=1}^{3} K_\ell(\alpha_\ell, x^\ell) \; h_\ell \right] dx^1 dx^2 dx^3, \qquad \text{B.31}$$

with the condition that

$$\int_V \frac{\partial}{\partial x^i} \left(\frac{\partial L_i}{\partial x^i} \right) K_i(\alpha_i, x^i) \; dx^i =$$

$$- \alpha_i^2 \int_V L_i(\mathbf{x}) \; K_i(\alpha_i, x^i) \; dx^i. \qquad \text{B.32}$$

Integrating the left hand side by parts

$$I = \int \frac{\partial}{\partial x^i} \left(\frac{\partial L_i}{\partial x^i} \right) K_i(\alpha_i, x^i) \; dx^i =$$

$$\frac{\partial L_i}{\partial x^i} K_i(\alpha_i, x^i) - \int \frac{\partial L_i}{\partial x^i} \frac{\partial K_i(\alpha_i, x^i)}{\partial x^i} dx^i.$$

Integrating by parts again

$$I = \frac{\partial L_i}{\partial x^i} K_i(\alpha_i, x^i) - L_i \frac{\partial K_i(\alpha_i, x^i)}{\partial x^i} + \int L_i \frac{\partial^2 K_i(\alpha_i, x^i)}{\partial x^{i2}} dx^i. \qquad \text{B.33}$$

The first two terms vanish by boundary conditions (or become a known constant) and it is apparent that

$$\frac{\partial^2 K_i(\alpha_i, x^i)}{\partial x^{i2}} = - \alpha_i^2 K_i(\alpha_i, x^i). \qquad \text{B.34}$$

Solutions to this equation are $\sin(\alpha_i x^i)$ or $\cos(\alpha_i x^i)$. Therefore the transformation becomes a Fourier transform. It is now possible to transform equation B.30 into an algebraic form

$$\int \text{div}(\Omega \text{ grad } f) \left[\prod_{\ell=1}^{3} K_\ell(\alpha_\ell, x^\ell) h_\ell \right] dx^1 dx^2 dx^3 =$$

$$\sum_{j=1}^{3} \frac{\partial}{\partial x^j} \left(\frac{\partial L_j}{\partial x^j} \right) \prod_{\ell=1}^{3} K_\ell(\alpha_\ell, x^\ell) \, dx^1 dx^2 dx^3 =$$

$$- \sum_{\ell=1}^{3} \alpha_\ell^2 \, \overline{L}_\ell(\alpha_1, \alpha_2, \alpha_3).$$

We shall call this type of transformation the B-transform of the second kind.

The above equations are used to determine the variable of transformation, ρ, and the transformation kernel $K(\mathbf{r},\boldsymbol{\rho})$. That is, the geometry of the problem together with the boundary (surface) conditions will determine the "type" of transformation. Once the integral transformation and its variable are determined, the existence of an inverse transformation must be investigated and an explicit form must be derived. The most common approach for orthogonal curvilinear coordinate systems is to expand the function in terms of the orthogonal kernel functions.

Appendix C

TENSOR ANALYSIS

C.1. Introduction

Tensors are abstract objects whose properties are independent of the reference frame used to describe them. In general, a tensor is determined with respect to a reference frame by its components and if the reference frame is changed, then the components change according to definite rules (which characterize the type of tensor under consideration). For a complete discussion of tensor analysis see Silverman (1968) and Mase (1970).

C.2. Curvilinear Coordinate Systems

Throughout this work regions (volumes) are considered in a three dimensional Euclidean space, R^3 and surfaces in R^2. We use

$$z^k, \qquad k = 1,2,3,$$

for the usual rectangular Cartesian coordinates of a geometric point.

A curvilinear coordinate system is a set of three functions

$$x^k = x^k(z^1,z^2,z^3), \qquad k = 1,2,3,$$

which are invertible, i.e., there are functions

$$z^k = z^k(x^1,x^2,x^3), \qquad k = 1,2,3,$$

and all the functions x^1, x^2, x^3, z^1, z^2 and z^3 are differentiable. Associated with a curvilinear coordinate system is the Jacobian matrix

$$\mathbf{J} = \left(\frac{\partial z^i}{\partial x^j}\right)_{\substack{i=1,2,3\\ j=1,2,3}} \qquad \text{C.1}$$

and the Jacobian (determinant)

$$J = \det(\mathbf{J}). \qquad \text{C.2}$$

Theorem - (The inverse function theorem) - If $x^k(z^1,z^2,z^3)$ is a curvilinear coordinate system it possesses a differentiable inverse if and only if the Jacobian does not vanish. Conversely, given functions $z^k(x^1,x^2,x^3)$ such that

$$\det\left(\frac{\partial z^k}{\partial x^\ell}\right) \neq 0,$$

then there is a curvilinear coordinate system, $x^k(z^1,z^2,z^3)$ inverse to the given functions.

Proof - Use the chain rule for functions of several variables.

Examples of curvilinear coordinate systems are spherical and cylindrical coordinates for which $J = (x^1)^2 \sin(x^2)$ and $J = x^1$, respectively.

In the case where $J > 0$ holds, the coordinate system is said to be positively oriented. In the case where $J < 0$ holds, the coordinate system is said to be negatively oriented. In this work we restricted our attention to positively oriented coordinate systems.

C.3. Summation Convention

An index which occurs exactly twice in a monomial, once as a subscript and once as a superscript is automatically to be summed from 1 to 3.

C.4. Basis Vectors for Curvilinear Coordinate Systems

First we establish the basis vectors associated with the rectangular Cartesian coordinates:

$$\mathbf{i}_k = \text{unit vector parallel to the } z^k \text{ axis.}$$

Given a point P we define the position vector (of P) to be

$$\mathbf{r} = \sum_{\ell=1}^{3} \mathbf{i}_\ell z^\ell = \mathbf{i}_\ell z^\ell. \qquad \text{C.3}$$

The position vector is the vector which begins at the origin and terminates at P.

Theorem - Suppose that in a neighborhood of P there is a curvilinear coordinate system, (x^1,x^2,x^3). We set

$$g_k(\mathbf{x}) = \frac{\partial \mathbf{r}}{\partial x^k}, \qquad \text{C.4}$$

for k=1,2,3. The particular triple $\mathbf{g}_1(\mathbf{x})$, $\mathbf{g}_2(\mathbf{x})$, $\mathbf{g}_3(\mathbf{x})$ form a linearly independent set.

Proof - We see from

$$\mathbf{r} = \mathbf{i}_k z^k,$$

that

$$\mathbf{g}_i = \frac{\partial \mathbf{r}}{\partial x^i} = \mathbf{i}_k \frac{\partial z^k}{\partial x^i},$$

holds for i=1,2,3. It follows that

$$\mathbf{g}_1 \cdot (\mathbf{g}_2 \times \mathbf{g}_3) = J \neq 0,$$

so $\mathbf{g}_1$, $\mathbf{g}_2$ and $\mathbf{g}_3$ are linearly independent.

C.5. Fundamental Metric (Tensor) on R^3

Consider a curve C given by $\mathbf{r} = \mathbf{r}(t)$ for a <= t <= b. One defines the differential of arc length, ds, also called the line element by

$$(ds)^2 = d\mathbf{r} \cdot d\mathbf{r}, \tag{C.5}$$

now, we have

$$d\mathbf{r} = \frac{\partial \mathbf{r}}{\partial x^k} dx^k = \mathbf{g}_k \, dx^k. \tag{C.6}$$

We compute

$$\begin{aligned}(ds)^2 &= (\mathbf{g}_k \, dx^k) \cdot (\mathbf{g}_\ell \, dx^\ell) \\ &= (\mathbf{g}_k \cdot \mathbf{g}_\ell) \, dx^k \, dx^\ell. \end{aligned} \tag{C.7}$$

Setting

$$g_{k\ell} = \mathbf{g}_k \cdot \mathbf{g}_\ell, \tag{C.8}$$

we have

$$(ds)^2 = g_{k\ell}\, dx^k\, dx^\ell. \qquad \text{C.9}$$

We note that

$$g_{\ell k} = g_{k\ell},$$

holds since the inner product is commutative. The matrix $(g_{k\ell})$ is the fundamental metric tensor which is a symmetric 3x3 matrix. Equation C.9 defines the metric of the space. The functions $g_{k\ell}(\mathbf{x})$ are called the fundamental metric coefficients.

A curvilinear coordinate system is termed orthogonal in case

$$\mathbf{g}_k \cdot \mathbf{g}_\ell = 0, \qquad \text{if } k \neq \ell.$$

In this case the matrix $(g_{k\ell})$ is a diagonal matrix

$$(g_{k\ell}) = \begin{bmatrix} g_{11} & 0 & 0 \\ 0 & g_{22} & 0 \\ 0 & 0 & g_{33} \end{bmatrix}$$

C.6. Differential Operators

C.6.1. The gradient operator

The gradient of a scalar function f(**x**) is the vector field

$$\text{grad } f = f_{,k}\, \mathbf{g}^k. \qquad \text{C.10}$$

In the case of scalar functions, partial and covariant differentiation are the same (by definition), so we write

$$\text{grad } f = f_{;k}\, \mathbf{g}^k. \qquad \text{C.11}$$

This leads us to define, for a vector field v(**x**),

$$\text{grad } \mathbf{v} = v_{i;k}\, \mathbf{g}^k\, \mathbf{g}^i\ , \qquad \text{C.12}$$

the gradient of v. Note that $\nabla \mathbf{v}$ is a tensor field of order 2. If $\tilde{\mathbf{A}}(\mathbf{x})$ is a tensor field, say of order 2, we define the gradient of $\tilde{\mathbf{A}}$, which is a tensor field of order 3, by

$$\text{grad } \tilde{\mathbf{A}} = A_{ij;k}\, \mathbf{g}^k\, \mathbf{g}^i\, \mathbf{g}^j\ . \qquad \text{C.13}$$

Similar definitions are used for tensor fields of orders 3, 4, ... The gradient operator, grad, raises the order of a tensor by 1. One also writes

$$\nabla \tilde{\mathbf{A}} = \text{grad } \tilde{\mathbf{A}} \text{ .}$$

C.6.2. The divergence operator

The divergence of a vector filed v(**x**) is defined to be the scalar function

$$\text{div } (\mathbf{v}) = v^k{}_{;k} \text{ .} \qquad \text{C.14}$$

The divergence of a tensor field $\tilde{\mathbf{A}}(\mathbf{x})$ of order 2 is the tensor field

$$\text{div}(\tilde{A}) = A^k{}_{i;k}\, \mathbf{g}^i \text{ .} \qquad \text{C.15}$$

Similarly, if $\tilde{B}(\mathbf{x})$ is of order 3 we set

$$\text{div}(\tilde{B}) = B^k{}_{ij;k}\, \mathbf{g}^i\, \mathbf{g}^j \text{ ,} \qquad \text{C.16}$$

and so on for tensor fields of order 4, 5, ... The divergence operator, div, lowers the order of a tensor by 1. One also writes

$$\nabla \cdot \tilde{A} = \text{div } \tilde{\mathbf{A}} \text{ .}$$

C.6.3. The Laplacian

The Laplace operator or Laplacian is denoted by Lap, ∇^2, or Δ and is defined by setting

$$\mathrm{Lap}(\tilde{A}) = \mathrm{div}(\mathrm{grad}\ \tilde{A}), \qquad \text{C.17}$$

for a tensor field $\tilde{A}$ (of any order including zero). The Laplacian leaves the order of a tensor field unchanged.

We note that all the operators defined in this section are linear; these are the gradient, divergence and Laplacian.

Appendix D

MATHEMATICAL DERIVATION OF FUNDAMENTAL LAWS OF CONTINUUM MECHANICS

D.1. Introduction

A continuum is subject to the same physical laws which govern the behavior of all matter. Consequently, there are certain equations which will hold in any continuum without regard to whether it is a fluid or solid, elastic or plastic, etc. For all practical purposes, we will be interested only in non-relativistic situations. For a more complete discussion on fundamental laws see Welty, Wicks and Wilson (1976), Arpaci (1966) and Mase (1970).

D.2. Conservation of Mass

We begin with the equation

$$\int_{V_o} \rho_o(\mathbf{x})\, dV = \int_{V_t} \rho(\mathbf{x},t)\, dv, \qquad \text{D.1}$$

which expresses the law of conservation of mass. Taking the material time derivative of both sides of this equation, we note that

$$\frac{D}{Dt}\int_{V_o} \rho_o(\mathbf{x})\, dV = 0, \qquad \text{D.2}$$

so we get

$$\frac{D}{Dt}\int_{V_t} \rho(\mathbf{x},t)\, dv = 0. \qquad \text{D.3}$$

We use rectangular Cartesian coordinate systems $\mathbf{z}$ and Z to work out an expression for D.3. We have

$$\int_{V_t} \rho(\mathbf{x},t)\, dv = \int_{V_t} \rho(\mathbf{z},t)\, dv. \qquad \text{D.4}$$

Since the volume integral of a scalar is a scalar, i.e. is independent of coordinate system. The motion can be expressed by equations

$$\mathbf{z} = \mathbf{z}(Z,t), \qquad \text{D.5}$$

and the inverse by

$$Z = Z(\mathbf{z},t). \qquad \text{D.6}$$

We have

$$\int_{V_t} \rho(\mathbf{z},t)\, dv = \int_{V_t} \rho(\mathbf{z},t)\, dz^1 dz^2 dz^3$$

$$= \int_{V_o} \rho[\mathbf{z}(Z,t),t] \left|\frac{\partial(z^1,z^2,z^3)}{\partial(Z^1,Z^2,Z^3)}\right| dZ^1 dZ^2 dZ^3, \qquad \text{D.7}$$

so

$$\frac{D}{Dt}\int_{V_t} \rho(\mathbf{z},t)\, dv =$$

$$\int_{V_o} \frac{\partial}{\partial t}\left[\rho[\mathbf{z}(Z,t),t] \left|\frac{\partial(z^1,z^2,z^3)}{\partial(Z^1,Z^2,Z^3)}\right|\right] dZ^1 dZ^2 dZ^3. \qquad \text{D.8}$$

Where we use

$$\frac{D}{Dt} = \frac{\partial}{\partial t},$$

once the integral has been expressed in Lagrangian variables and where we differentiate under the integral once the region of integration has become constant. Now,

$$\frac{\partial}{\partial t}\left\{\rho[z(Z,t),t]\;\left|\frac{\partial(z^1,z^2,z^3)}{\partial(Z^1,Z^2,Z^3)}\right|\right\} =$$

$$\frac{\partial}{\partial t}\Big[\rho[z(Z,t),t]\Big]\left|\frac{\partial(z^1,z^2,z^3)}{\partial(Z^1,Z^2,Z^3)}\right|$$

$$+\;\rho[z(Z,t),t]\frac{\partial}{\partial t}\left|\frac{\partial(z^1,z^2,z^3)}{\partial(Z^1,Z^2,Z^3)}\right|. \qquad \text{D.9}$$

Since at t=0

$$\left|\frac{\partial(z^1,z^2,z^3)}{\partial(Z^1,Z^2,Z^3)}\right| = 1 > 0, \qquad \text{D.10}$$

(because z and Z systems are coincident), we conclude that the Jacobian is always positive. After some tensor algebra we can write

$$\frac{\partial}{\partial t}\left|\frac{\partial(z^1,z^2,z^3)}{\partial(Z^1,Z^2,Z^3)}\right| = \frac{\partial v^i}{\partial z^i}\left|\frac{\partial(z^1,z^2,z^3)}{\partial(Z^1,Z^2,Z^3)}\right|. \qquad \text{D.11}$$

Substituting this result into equation D.9 yields

$$\frac{\partial}{\partial t}\left\{ \rho[z(Z,t),t] \left|\frac{\partial(z^1,z^2,z^3)}{\partial(Z^1,Z^2,Z^3)}\right|\right\} =$$

$$\frac{\partial}{\partial t}\left[\rho[z(Z,t),t] + \frac{\partial v^i}{\partial z^i}\right] \left|\frac{\partial(z^1,z^2,z^3)}{\partial(Z^1,Z^2,Z^3)}\right|. \qquad D.12$$

Noting that

$$\frac{\partial}{\partial t}\left(\rho[z(Z,t),t]\right) = \frac{D\rho}{Dt}, \qquad D.13$$

and changing variables back to the Eulerian form, we find

$$\frac{D}{Dt}\int_{V_t} \rho \; dv = \int_{V_t} \left(\frac{D\rho}{Dt} + \rho v^i{}_{;i} \right) dv. \qquad D.14$$

This is a tensor equation, hence, it holds in any coordinate system, not just the rectangular Cartesian system in which it was proved. Referring back to equation D.3 we see that

$$\int_{V_t} \left(\frac{D\rho}{Dt} + \rho v^i{}_{;i} \right) dv = 0, \qquad D.15$$

holds for each time t. Since V_o was an arbitrary portion of the undeformed continuum, we see that V_t is an arbitrary portion of the deformed continuum. It follows that

$$\frac{D\rho}{Dt} + \rho v^i{}_{;i} = 0, \qquad \text{D.16}$$

holds. This is the Eulerian form of the law of conservation of mass, and is called the continuity equation. We can rewrite the continuity equation as follows

$$\frac{D\rho}{Dt} + \rho v^i{}_{;i} = \frac{\partial\rho}{\partial t} + (\rho v^i)_{;i} = 0. \qquad \text{D.17}$$

We notice that if

$$\frac{D\rho}{Dt} = 0, \qquad \text{D.18}$$

then the continuity equation simplifies to

$$v^i{}_{;i} = 0. \qquad \text{D.19}$$

Equation D.18 can often be used:

(1) Many materials are nearly incompressible so that ρ is a constant; in particular, liquids are difficult to compress. So when studying their behavior, equation D.19 is typically used

for the continuity equation.

(2) A material may be compressible, but the motion under study may be one for which only negligible density changes actually occur. In this connection, we note that gases are extremely compressible, but they quickly reach a uniform density. Thus, in the study of the motion of a gas exterior to a region, as long as the velocities involved are small compared to the speed of sound, the density can be treated as constant.

In a case where equation D.18 is true (or is an appropriate simplification) the flow is called incompressible.

D.3. Conservation of Momentum

Let V_o be an arbitrary portion of a continuum in its initial configuration. Let V_t denote the particle initially in V_o. The total linear momentum of the particles in V_t, at time t, is

$$\int_{V_t} \rho \ v \ dv. \qquad \text{D.20}$$

In the absence of any forces, the momentum of a particle is a constant vector. Even in the absence of external forces, the particles within V_t may be expected to interact with each other. Thus the momentum of any specific particle within V_t should not be expected to be constant. We can assume, however, that the interactions between particles in V_t obey the law of action and reaction. Hence, the total momentum over V_t should stay constant. External forces will arise from the body forces and from the interaction of particles of the continuum across the surface, S_t, which bounds V_t. These forces will give the time rate of change of the total momentum in V_t. Let N denote the outward unit normal to S_t. Then we have a balance law for linear momentum

$$\int_{S_t} \tilde{\mathbf{t}} \cdot N \, da + \int_{V_t} \rho \, b \, dv = \frac{D}{Dt} \int_{V_o} \rho \, v \, dv. \qquad \text{D.21}$$

The integral

$$\int_{S_t} \tilde{\mathbf{t}} \cdot N \, da, \qquad \text{D.22}$$

can be rewritten using the divergence theorem:

$$\int_{S_t} \tilde{\mathbf{t}} \cdot N \, da = \int_{V_t} \operatorname{div}(\tilde{\mathbf{t}}) \, dv. \qquad \text{D.23}$$

Since $\tilde{t}$ is symmetric, we do not need to be concerned with the difference between $\tilde{t} \cdot N$ and $N \cdot \tilde{\mathbf{t}}$. We have

$$\int_{V_t} [\mathrm{div}(\tilde{\mathbf{t}}) + \rho\mathbf{b}]\, dv = \frac{D}{Dt} \int_{V_t} \rho\, v\, dv. \qquad \text{D.24}$$

The next step is the manipulation of the right hand side of this equation. Let us introduce Eulerian and Lagrangian Cartesian coordinate systems z and Z. We have

$$\int_{V_t} \rho(\mathbf{x},t)\, v(\mathbf{x},t)\, dv =$$

$$\mathbf{i}_k \int_{V_o} \rho[\mathbf{z}(Z,t),t]\, v^{*k}[\mathbf{z}(Z,t),t] \left|\frac{\partial(z^1,z^2,z^3)}{\partial(Z^1,Z^2,Z^3)}\right| dZ^1 dZ^2 dZ^3$$

$$= \mathbf{I}_K \int_{V_o} \rho(Z,t)\, v^{*K}(Z,t)\, J^*(Z,t)\, dZ^1 dZ^2 dZ^3. \qquad \text{D.25}$$

From this we get

$$\frac{D}{Dt} \left[\int_{V_t} \rho(\mathbf{x},t)\, v(\mathbf{x},t)\, dv \right] =$$

$$\mathbf{I}_K \int_{V_o} \frac{\partial}{\partial t} \left[\rho(Z,t)\, v^{*K}(Z,t)\, J^*(Z,t) \right] dZ^1 dZ^2 dZ^3. \qquad \text{D.26}$$

Since $\mathbf{I}_K$ is time independent. Now, note

$$\rho(Z,t)\, J^*(Z,t) = \rho_o(Z), \qquad \text{D.27}$$

so

$$\frac{\partial}{\partial t}[\,\rho(Z,t)\, J^*(Z,t)\,] = 0, \qquad \text{D.28}$$

and we have

$$\frac{D}{Dt}\left[\int_{V_t} \rho(\mathbf{x},t)\, \mathbf{v}(\mathbf{x},t)\, dv\,\right] =$$

$$\int_{V_t} \rho(\mathbf{x},t) \frac{D\mathbf{v}}{Dt}(\mathbf{x},t)\, dv\, . \qquad \text{D.29}$$

So the momentum balance law may be rewritten as

$$\int_{V_t} [\mathrm{div}(\tilde{\mathbf{t}}) + \rho\mathbf{b}]\, dv = \int_{V_t} \rho \frac{D\mathbf{v}}{Dt}\, dv,$$

or

$$\int_{V_t} [\,\mathrm{div}(\tilde{\mathbf{t}}) + \rho\mathbf{b} - \rho \frac{D\mathbf{v}}{Dt}\,]\, dv = 0. \qquad \text{D.30}$$

The latter equation is known as the integral balance law for linear momentum. Since V_t is an arbitrary portion of the continuum at time t, we conclude that the integrand must vanish. Thus we have

$$\mathrm{div}(\tilde{\mathbf{t}}) + \rho\mathbf{b} - \rho \frac{Dv}{Dt} = \mathbf{0}. \qquad \text{D.31}$$

In terms of components we have

$$t^j{}_{i;j} + \rho b_i = \rho \frac{Dv_i}{Dt}, \qquad \text{D.32}$$

or

$$t^{ji}{}_{;j} + \rho b^i = \rho \frac{Dv^i}{Dt}. \qquad \text{D.33}$$

These are known as the equations of motion. In case the acceleration is zero we have

$$\mathrm{div}(\tilde{\mathbf{t}}) + \rho\mathbf{b} = \mathbf{0} \qquad (\text{if } a = \mathbf{0}). \qquad \text{D.34}$$

In particular, in the case of static equilibrium we have

$$\mathrm{div}(\tilde{\mathbf{t}}) + \rho\mathbf{b} = \mathbf{0} \qquad \text{D.35}$$

D.4. Conservation of Energy

Energy can take many forms: mechanical, thermal, chemical and so on. We will need to introduce an internal energy variable to make conservation of energy hold. We postulate the existence of a quantity $u(\mathbf{x},t)$ known as the specific internal energy. In terms of the specific internal energy we define the internal energy, U, of the portion of the continuum occupying the (Eulerian) region V_t at time t by

$$U = \int_{V_t} \rho(\mathbf{x},t)\, u(\mathbf{x},t)\, dv. \qquad \text{D.36}$$

We observe that

$$\frac{DU}{Dt} = \int_{V_t} [\, \frac{D}{Dt} (\rho u) + \rho u\, \mathrm{div}(\mathbf{v})\,]\, dv$$

$$= \int_{V_t} \rho \frac{Du}{Dt}\, dv. \qquad \text{D.37}$$

The time rate of change of total energy is thus (in a thermomechanical continuum)

$$\frac{dK}{dt} + \frac{dU}{dt} = \int_{V_t} \rho \left[v \cdot \frac{Dv}{Dt} + \frac{Du}{Dt} \right] dv. \qquad D.38$$

This rate of change of total energy must equal the rate at which work is done on V_t plus the rate of increase of total heat in V_t. The heat flux per unit area per unit time by C, so that the rate at which heat transfer to V_t by conduction at the surface S_t is

$$- \int_{S_t} C \cdot N \, da. \qquad D.39$$

Heat can also be transferred to the continuum by radiation or produced within the continuum by internal sources. We denote the sum of radiant and internally produced heat transfer per unit mass per unit time by z. The rate of increase of total heat in V_t is

$$- \int_{S_t} C \cdot N \, da + \int_{V_t} \rho z \, dv. \qquad D.40$$

The law of conservation of energy is then

$$\frac{dK}{dt} + \frac{dU}{dt} = \int_{S_t} v \cdot \tilde{t} \cdot N \, da + \int_{V_t} \rho \mathbf{v} \cdot \mathbf{b} \, dv$$

$$+ \int_{V_t} \rho z \, dv - \int_{S_t} \mathbf{C} \cdot N \, da. \qquad \text{D.41}$$

We can convert the surface integrals on the right to volume integrals by the divergence theorem:

$$\int_{S_t} \mathbf{v} \cdot \tilde{t} \cdot N \, da = \int_{V_t} \operatorname{div}(\mathbf{v} \cdot \tilde{t}) \, dv, \qquad \text{D.42}$$

$$\int_{S_t} \mathbf{C} \cdot N \, da = \int_{V_t} \operatorname{dic}(\mathbf{C}) \, dv. \qquad \text{D.43}$$

If we also use equation D.38 we get

$$\int_{V_t} [\rho \mathbf{v} \cdot \frac{Dv}{Dt} + \rho \frac{Du}{Dt} - \operatorname{div}(\mathbf{v} \cdot \tilde{t}) - \rho \mathbf{v} \cdot \mathbf{b} - \rho z + \operatorname{div}(\mathbf{C})] \, dv = 0. \qquad \text{D.44}$$

Since this holds for an arbitrary portion V_t of the deformed continuum, we get the local form of the energy equation:

$$\frac{D}{Dt}\left[\frac{\mathbf{v}\cdot\mathbf{v}}{2} + u\right] = \frac{1}{\rho}\,\mathrm{div}(\mathbf{v}\cdot\tilde{\mathbf{t}}) + \mathbf{v}\cdot b - \frac{1}{\rho}\,\mathrm{div}(C) + z. \qquad D.45$$

After some tensor algebra we can arrive at

$$\frac{D}{Dt}\left[\frac{\mathbf{v}\cdot\mathbf{v}}{2}\right] = -\frac{1}{\rho}\,\tilde{\mathbf{D}} : \tilde{\mathbf{t}} + \frac{1}{\rho}\,\mathrm{div}(\mathbf{v}\cdot\tilde{\mathbf{t}}) + \mathbf{v}\cdot\mathbf{b}. \qquad D.46$$

When this is subtracted from equation D.45, we obtain

$$\frac{Du}{Dt} = \frac{1}{\rho}\,\tilde{\mathbf{D}} : \tilde{\mathbf{t}} - \frac{1}{\rho}\,\mathrm{div}(C) + z, \qquad D.47$$

where

D = deformation tensor,

$:$ = diadic product,

t = stress tensor.

D.5. The Missing Equations

We have derived five fundamental equations of conservation of mass (1), momentum (3) and energy (1). Typically, one assumes that the body force **b** and the distributed heat source z are known. This leaves the density ρ, the velocity v, the stress tensor $\tilde{t}$, the heat flux **C**, and the internal energy u as unknown. In other words there are fourteen independent unknowns ρ, v_1, v_2, v_3, t_{11}, t_{12}, t_{13}, t_{21}, t_{22}, t_{23}, t_{31}, t_{32}, t_{33}, C_1, C_2, C_3, and u. Two additional unknowns are in the Clausius-Duhem inequality; namely, the entropy s and the temperature T. The distributed entropy source e is assumed known. Thus we are left with sixteen unknowns and five equations. We need eleven more equations to hope to determine a solution. Generally speaking, three of these equations will relate temperature to heat conduction, two will be thermodynamic equations of state, and the remaining six will be the constitutive equations which describe the stress tensor.

The solution of this system differential equations is extremely difficult to obtain and there are relatively few general theorems in the field. The analytic methods to be applied to obtain exact solutions are limited. Usually, one tries to simplify the problem as much as possible before attempting the solution.

One popular simplification is to neglect the interaction between mechanical and thermal processes. One must always be aware that it is a simplification. The mechanical equations are the equations of continuity and motion. This gives four equations in ten unknowns. The needed six additional equations then relate the stress to the kinematic variables, i.e. to the strain or to the strain rate.

Appendix E

GENERALIZED NUMERICAL INTEGRATION

Analytical integration of functions is often a difficult or impossible task. Numerical methods for evaluation of definite integrals is a necessary part of the describing function method.

The numerical technique presented here is also known as the method of undetermined coefficients. We start by considering the definite integral

$$\int_a^b f(x)\,dx. \qquad \text{E.1}$$

The integration can be approximated by a summation as

$$\int_a^b f(x)\,dx \sim \sum_{i=1}^{N} w_i\, f(x_i). \qquad \text{E.2}$$

The numerical integration is based on a numerical approximation of the function f(x) by a polynomial $P_n(x)$. For a polynomial of order n the highest order term is x^n, hence

$$\int_a^b x^n \, dx = \sum_{i=1}^{N} w_i \; x_i^n. \qquad \text{E.3}$$

Note that the sides of this equation are exactly equal. If we let

$$m_i = \int_a^b x^i \, dx, \qquad \text{E.4}$$

to be the moments on the left hand side of equation E.3 then we can write

$$m_\ell = \sum_{k=1}^{N} w_k \; x_k^\ell. \qquad \text{E.5}$$

Writing this as a matrix equation, we have

$$\begin{aligned}
m_0 &= w_1 + w_2 + \dots + w_N \\
m_1 &= w_1x_1 + w_2x_2 + \dots + w_Nx_N \\
&\vdots \\
m_{2N-1} &= w_1(x_1)^{2N-1} + w_2(x_2)^{2N-1} + \dots + w_N(x_N)^{2N-1}
\end{aligned}$$

This is a 2N by 2N system of nonlinear equations, therefore, a special solution technique is required. We introduce a product function $\pi(x)$, as

$$\pi(x) = (x-x_1)(x-x_2) \dots (x-x_N), \qquad \text{E.6}$$

which is a polynomial of order N. Thus

$$\begin{aligned}
\pi(x) &= x^N + C_{N-1}x^{N-1} + \dots + C_1x + C_0 \\
&= \sum_{j=0}^{N} C_j\, x^j. \qquad \text{E.7}
\end{aligned}$$

We multiply both sides of equation E.5 by C_j and sum over j to get

$$\begin{aligned}
\sum_{j=0}^{N} m_j\, C_j &= \sum_{k=1}^{N} w_k \Big[\sum_{j=0}^{N} C_j\, x_k^j \Big] \\
&= \sum_{k=1}^{N} w_k\, \pi(x_k) = 0, \qquad \text{E.8}
\end{aligned}$$

and

$$\sum_{j=0}^{N-1} m_{j+k}\ C_j + m_{N+k} = 0 \quad \text{for all } k=0,1,\ \ldots\ ,N-1. \qquad \text{E.9}$$

This is a system of N equations and N unknowns, therefore, it is possible to solve for C_j, j=0,1,2, ... ,N-1. It is now possible to solve for the base points by finding the roots of the polynomial $\pi(x)$, since the coefficients are known. Then, using these base points it is possible to solve for the weights, w_k, by using

$$m_j = \sum_{k=1}^{N} w_k\ x_k^j \quad \text{for } j=0,1,2,\ \ldots\ ,2N-1. \qquad \text{E.10}$$

We note that only N of these equations are linearly independent, hence, a solution exists. The following two theorems provide a base for the above postulations about the integration method and existence of solutions.

Theorem - If

$$\int_a^b K(x)\ f(x)\ dx = \sum_{k=1}^{N} w_k\ f(x_k),$$

where K(x) >= 0 for a <= x <= b, and

$$m_j = \int_a^b K(x)\ x^j\ dx,$$

then the system of equations

$$\sum_{j=0}^{N-1} m_{j+k}\ C_j + m_{N+k} = 0,$$

has a solution.

Proof - Using the set of N homogeneous equations, we multiply the kth equation by C_k and sum over k, to get

$$\sum_{k=0}^{N-1}\ \sum_{j=0}^{N-1} m_{j+k}\ C_j\ C_k = 0.$$

Recalling the definition of m_j, we have

$$\int_a^b K(x)\ [\sum_{k=0}^{N-1} C_k\ x^k\ \sum_{j=0}^{N-1} C_j\ x^j\ ]\ dx = 0,$$

and

$$\int_a^b K(x) \left[\sum_{k=0}^{N-1} C_k \; x^k \right]^2 dx = 0.$$

Since $K(x) >= 0$, the above equation can be zero only if $\sum C_k \; x^k = 0$, that is, when all the $C_j = 0$. This means that the determinant cannot be zero and hence a solution exists.

Theorem - When $K(x) >= 0$ then the polynomial $\pi(x)$ has exactly N real, distinct zeros in the interval $a <= x <= b$.

Proof - We start by assuming the contrary, that is, it does not and possibly there are complex zeros, real zeros outside the interval or multiple zeros. For the odd-order zeros in the interval we can write

$$p(x) = (x-x_1) \; (x-x_2) \; \dots \; (x-x_k),$$

where $k < N$ by our assumption. From our integration formula we can write

$$\int_a^b K(x) \; \pi(x) \; p(x) \; dx = \sum_{\ell=1}^{N} w_\ell \; \pi(x_\ell) \; p(x_\ell) = 0.$$

We made the formula exact for all powers of x up to 2N and consequently for all polynomials of degree less than 2N, so, the integral must be zero. However, the integrand is always positive and we have a contradiction.

$$\int_a^b K(x)\ \pi(x)\ p(x)\ dx > 0$$

We must therefore have k = N and there are N real, distinct zeros in the interval a <= x <= b.

The FORTRAN subroutine RAW calculates the roots and weights for a given vector of moments. The PROOTS subroutine calculates the real or complex roots of a polynomial utilizing a three-stage algorithm for real polynomials using quadratic iterations. The RAW subroutine is used to calculate the roots and weights for the finite Hankel transform.

```
      PROGRAM RAW(INPUT,OUTPUT,TAPE6)
C
C  Program to calculate the roots and weights for
C  numerical Hankel transform
C
      DIMENSION M(51),A(51,51),B(51),C(52),X(51),WMAT(51,51),
     $RHS(51),W1(51),XIL(4),CR(52),ZR(51),ZI(51)
      DOUBLE PRECISION XIL,R,SUM,A,B,C,M,RHS,WMAT,W1,X,FACT
      LOGICAL FAIL
      DATA XIL/2.4048255576957727686216320D0,
     $         5.5200781102863106495966040D0,
     $         8.6537279129110122169541990D0,
     $         11.791534439014281613743050D0/
 100  PRINT*,'INPUT ROOT INDEX AND ORDER',
      READ*,IND,N
      N2=N*2
      MIND=N2-1
      DO 3 J=0,MIND
        SUM=(XIL(IND)**(J+2))/(J+2)
        DO 1 K=1,200
          NPOW=2*K+J+2
          R=(XIL(IND)**NPOW)/
     $         (((-4.0D0)**K)*(FACT(K)**2)*NPOW)
          IF(DABS(R).LT.1.0D-30)GO TO 2
          SUM=SUM+R
 1        CONTINUE
 2      M(J+1)=SUM
 3      CONTINUE
      DO 10 I=1,N
        B(I)=-M(N+I)
        DO 10 J=1,N
 10       A(I,J)=M(J+I-1)
      CALL HOUSRED(A,B,W1,0,1,N,N,51,51,1)
      NP=N+1
      C(1)=1.0D0
      DO 200 KK=1,52
 200    CR(KK)=0.0
      CR(1)=1.
      DO 13 I=2,NP
        K=NP-I+1
        C(I)=B(K)
        CR(I)=C(I)
 13     CONTINUE
      CALL ZRPOLY(CR,N,ZR,,ZI,FAIL)
      DO 20 I=1,N
 20     X(I)=DBLE(ZR(I))
      DO 25 J=1,N
        RHS(J)=M(J)
        WMAT(1,J)=1.0D0
        DO 25 I=2,N
          WMAT(I,J)=WMAT(I-1,J)*X(J)
```

```
  25      CONTINUE
      CALL HOUSRED(WMAT,RHS,W1,0,1,N,N,51,51,1)
      WRITE(6,31)XIL(IND)
  31  FORMAT(//20X,'XIL=',E25.16)
      WRITE(6,32)
  32  FORMAT(/10X,'WEIGHTS',35X,'BASE POINTS',/47X,'REAL',
     $18X,'IAMG',/)
      DO 40 I=1,N
  40    WRITE(6,41)RHS(I),ZR(I)
  41  FORMAT(3(E30.16))
      PRINT*,'AGAIN (0=YES)',
      READ*,ANSWER
      IF(ANSWER.EQ.0)GO TO 100
      STOP
      END
C
C
C
      SUBROUTINE HOUSRED(A,B,W1,IOPT,NRHS,M,N,MDIM,NDIM,NDIMRHS)
C
C  Solves a linear system of simultaneous equations
C  in a least squared sense
      DIMENSION A(MDIM,NDIM),B(MDIM,NDIMRHS),W1(NDIM)
      DOUBLE PRECISION A,B,PINR,SIGN,SUM,V0,W0,W1
      IF(IOPT.EQ.2) GO TO 30
C
C  Factor the A matrix
C
      NELM=N-1
      IF(M.GT.N) NELM=N
      DO 20 K=1,NELM
        SIGN=1.0D0
        IF(A(K,K).GT.0.0D0) SIGN=-1.0D0
        KP=K+1
C
C  Calculate the magnitude of the column to be eliminated
C
        V0=0.0D0
        DO 5 I=K,M
   5      V0=V0 + A(I,K)*A(I,K)
        V0=SIGN*DSQRT(V0)
        IF(V0.NE.0.0D0) GO TO 6
        WRITE(6,100)
 100    FORMAT(' *** HOUSRED ENCOUNTERED AN ALGORITHMICALLY SINGULAR',
     .  ' MATRIX')
        STOP
C
C  Calculate the unity normalized W vector and save all but
C  W1 in the lower triangle of A.
C
   6    W0=DSQRT(2.0D0*V0*(V0-A(K,K)))
```

```
      W1(K)=(A(K,K)-V0)/W0
      A(K,K)=V0
      DO 10 I=KP,M
  10    A(I,K)=A(I,K)/W0
C
C  Apply the Householder reduction to the remaining
C  cloumns of A.
C
      IF(KP.GT.N) GO TO 25
      DO 20 J=KP,N
C
C  Inner product and elimination
C
        PINR=W1(K)*A(K,J)
        DO 15 I=KP,M
 15       PINR=PINR + A(I,K)*A(I,J)
        A(K,J)=A(K,J) - 2.0D0*PINR*W1(K)
        DO 20 I=KP,M
 20       A(I,J)=A(I,J) - 2.0D0*PINR*A(I,K)
 25  IF(IOPT.EQ.1) RETURN
C
C  Transform the B vectors
C
 30  DO 40 K=1,NELM
      KP=K+1
      DO 40 J=1,NRHS
C
C  Inner product and elimination
C
        PINR=W1(K)*B(K,J)
        DO 35 I=KP,M
 35       PINR=PINR + A(I,K)*B(I,J)
        B(K,J)=B(K,J) - 2.0D0*PINR*W1(K)
        DO 40 I=KP,M
 40       B(I,J)=B(I,J) - 2.0D0*PINR*A(I,K)
C
C  Solve QAX = QB by back substitution
C
     DO 45 JRHS=1,NRHS
 45    B(N,JRHS)=B(N,JRHS)/A(N,N)
     DO 60 JRHS=1,NRHS
       DO 60 IOP=2,N
         I=N-IOP+1
         SUM=B(I,JRHS)
         IP=I+1
         DO 50 J=IP,N
 50        SUM=SUM - B(J,JRHS)*A(I,J)
 60      B(I,JRHS)=SUM/A(I,I)
     RETURN
     END
C
```

```
C  DOUBLE PRECISION factorial function
C
      DOUBLE PRECISION FUNCTION FACT(N)
      DOUBLE PRECISION PROD
      PROD=1.0D0
      IF((N.GT.1).AND.(N.LT.150))THEN
        DO 1 I=2,N
 1      PROD=PROD*I
        ENDIF
      FACT=PROD
      RETURN
      END
      SUBROUTINE ZRPOLY(OP,DEGREE,ZEROR,ZEROI,FAIL)
C
C  Finds the zeros of a real polynomial
C
C  Ref.  Jenkins, M. A., and Traub, J. F., "A Three-Stage
C          Algorithm for Real Polynomials using quadratic
C          iteration," SIAM J. Numer. Anal. 7 (1970), 545-566.
C
C        Jenkins, M. A., "Algorithm 493 - Zeros of a Real
C          Polynomial," ACM Transactions on Mathematical
C          Software, Vol. 1, No. 2, June 1975.
C
C
C  The parameters are:
C
C  OP      -- DOUBLE PRECISION vector of coefficients in
C             order of decreasing powers.
C  DEGREE  -- Integer degree of the polynomial
C  ZEROR,
C  ZEROI   -- Output DOUBLE PRECISION vectors of real and
C             and imaginary parts of the zeros.
C  FAIL    -- Output logical parameter, true only if
C             leading coefficient is zero or if RPOLY
C             has found fewer than DEGREE zeros.  In the
C             latter case DEGREE is reset to the number
C             of zeros found.
C
C  To change the size of the polynomials which can be solved,
C  reset the dimensions of the arrays in the common area and
C  in the following declarations.  The subroutine uses single
C  precision calculations for scaling, bounds and error
C  calculations.  All calculations for the iterations are done
C  in DOUBLE PRECISION.
C
      COMMON /GLOBAL/ P, QP, K, QK, SVK, SR, SI, U, V,
     * A, B, C, D, A1, A2, A3, A6, A7,
     * E, F, G, H, SZR, SZI, LZR, LZI,
     * ETA, ARE, MRE, N, NN
C¶D   DOUBLE PRECISION P(21), QP(21), K(21),
C¶D  $QK(21), SVK(21), SR, SI, U, V, A, B, C, D,
C¶D  $A1, A2, A3, A6, A7, E, F, G, H, SZR, SZI,
C¶D  $LZR, LZI
      REAL P(21), QP(21), K(21),
     $QK(21), SVK(21), SR, SI, U, V, A, B, C, D,
     $A1, A2, A3, A6, A7, E, F, G, H, SZR, SZI,
     $LZR, LZI
      REAL ETA, ARE, MRE
      INTEGER N, NN
C¶D   DOUBLE PRECISION OP(21), TEMP(21),
C¶D  $ZEROR(20), ZEROI(20), T, AA, BB, CC, FACTOR
```

```
       REAL OP(21), TEMP(21),
      $ZEROR(20), ZEROI(20), T, AA, BB, CC, FACTOR
       REAL PT(21), LO, MAX, MIN, XX, YY, COSR,
      $SINR, XXX, X, SC, BND, XM, FF, DF, DX, INFIN,
      $SMALNO, BASE
       INTEGER DEGREE, CNT, NZ, I, J, JJ, NM1
       LOGICAL FAIL, ZEROK
C
C  The following statements set machine constants used
C  in various parts of the program.  The meaning of the
C  four constants are ...
C
C  ETA      -- The maximum relative representation error
C              which can be described as the smallest
C              positive floating point number such that
C              1.0D0+ETA is greater than one.
C  INFINY   -- The largest floating point number.
C  SMALNO   -- The smallest positive floating point number.
C              If the exponent range differs in single and
C              double precision then SMALNO and INFINY
C              should indicate the smaller range.
C  BASE     -- The base of the floating point number
C              system used.
C
C  The values below correspond to the Apple II minicomputer
C
       BASE=10.0
       ETA=0.6*BASE**(1-8)
       INFIN=1.0E18
       SMALNO=1.0E-18
C
C  ARE and MRE refer to the unit error in + and *
C  respectively.  They are assumed to be the same as
C  ETA.
C
       ARE=ETA
       MRE=ETA
       LO=SMALNO/ETA
C
C  Intialization of constants for shift rotation.
C
       XX=0.70710678
       YY=-XX
       COSR=-0.069756474
       SINR=0.99756405
       FAIL=.FALSE.
       N=DEGREE
       NN=N+1
C
C  Algorithm fails if the leading coefficient is zero.
```

```
C
C¶D   IF(OP(1).NE.0.0D0)GO TO 10
      IF(OP(1).NE.0.0)GO TO 10
      FAIL=.TRUE.
      DEGREE=0
      RETURN
C
C  Remove the zeros at the origin if any.
C
C¶D 10  IF(OP(NN).NE.0.0)GO TO 20
   10 IF(OP(NN).NE.0.0)GO TO 20
      J=DEGREE-N+1
C¶D   ZEROR(J)=0.0D0
C¶D   ZEROI(J)=0.0D0
      ZEROR(J)=0.0
      ZEROI(J)=0.0
      NN=NN-1
      N=N-1
      GO TO 10
C
C  Make a copy of the coefficients.
C
   20 DO 30 I=1,NN
        P(I)=OP(I)
   30 CONTINUE
C
C  Start the algorithm for one zero.
C
   40 IF(N.GT.2)GO TO 60
      IF(N.LT.1)RETURN
C
C  Calculate the final zero or pair of zeros.
C
      IF(N.EQ.2)GO TO 50
      ZEROR(DEGREE)=-P(2)/P(1)
C¶D   ZEROI(DEGREE)=0.0D0
      ZEROI(DEGREE)=0.0
      RETURN
   50 CALL QUAD(P(1),P(2),P(3),ZEROR(DEGREE-1),
     $ ZEROI(DEGREE-1),ZEROR(DEGREE),ZEROI(DEGREE))
      RETURN
C
C  Find largest and smallest moduli of coefficients.
C
   60 MAX=0.0
      MIN=INFIN
      DO 70 I=1,NN
C¶D     X=ABS(SNGL(P(I)))
        X=ABS(P(I))
        IF(X.GT.MAX)MAX=X
        IF((X.NE.0.0).AND.(X.LT.MIN))MIN=X
   70  CONTINUE
C
C  Scale if there are large or very small coefficients.
C  Computes a scale factor to multiply the coefficients
C  of the polynomial.  The scaling is done to avoid
C  overflow and to avoid undetected underflow
C  interfering with the convergence criterion.
C  The factor is a power of the base.
C
       SC=LO/MIN
       IF(SC.GT.1.0)GO TO 80
       IF(MAX.LT.10.0)GO TO 110
       IF(SC.EQ.0.0)SC=SMALNO
       GO TO 90
   80  IF((INFIN/SC).LT.MAX)GO TO 110
   90  L=ALOG(SC)/ALOG(BASE)+0.5
C¶D    FACTOR=(BASE*1.0D0)**L
       FACTOR=(BASE*1.0)**L
C¶D    IF(FACTOR.EQ.1.0D0)GO TO 110
       IF(FACTOR.EQ.1.0)GO TO 110
       DO 100 I=1,NN
         P(I)=FACTOR*P(I)
  100  CONTINUE
C
C  Compute lower bound on moduli of zeros.
C
  110  DO 120 I=1,NN
C¶D      PT(I)=ABS(SNGL(P(I)))
         PT(I)=ABS(P(I))
  120  CONTINUE
       PT(NN)=-PT(NN)
C
C  Compute upper estimate of bound
C
       X=EXP((ALOG(-PT(NN))-ALOG(PT(1)))/FLOAT(N))
       IF(PT(N).EQ.0.0)GO TO 130
C
C  If Newton step at the origin is better, use it.
C
       XM=-PT(NN)/PT(N)
       IF(XM.LT.X)X=XM
C
C  Chop the interval (0,X) until FF.LE.0
C
  130  XM=X*0.1
       FF=PT(1)
       DO 140 I=2,NN
         FF=FF*XM+PT(I)
  140  CONTINUE
```

```
      IF(FF.LE.0.0)GO TO 150
      X=XM
      GO TO 130
  150 DX=X
C
C  Do Newton iteration until X converges to two
C  decimal places.
C
  160 IF(ABS(DX/X).LE.0.005)GO TO 180
      FF=PT(1)
      DF=FF
      DO 170 I=2,N
        FF=FF*X+PT(I)
        DF=DF*X+FF
  170 CONTINUE
      FF=FF*X+PT(NN)
      DX=FF/DF
      X=X-DX
      GO TO 160
  180 BND=X
C
C  Compute the derivative as the initial K polynomial
C  and do 5 steps with no shift.
C
      NM1=N-1
      DO 190 I=2,N
        K(I)=FLOAT(NN-1)*P(I)/FLOAT(N)
  190 CONTINUE
      K(1)=P(1)
      AA=P(NN)
      BB=P(N)
C¶D   ZEROK=K(N).EQ.0.0D0
      ZEROK=K(N).EQ.0.0
      DO 230 JJ=1,5
        CC=K(N)
        IF (ZEROK) GO TO 210
C
C  Use scaled form of recurrence if value of K at 0 is
C  nonzero
C
        T=-AA/CC
        DO 200 I=1,NM1
          J=NN-I
          K(J)=T*K(J-1)+P(J)
  200   CONTINUE
        K(1)=P(1)
C¶D     ZEROK=DABS(K(N)).LE.DABS(BB)*ETA*10.0
        ZEROK=ABS(K(N)).LE.ABS(BB)*ETA*10.0
        GO TO 230
C
```

```
C  Use unscaled form of recurrence.
C
  210   DO 220 I=1,NM1
          J=NN-I
          K(J)=K(J-1)
  220   CONTINUE
C¶D     K(1)=0.0D0
        K(1)=0.0
C¶D     ZEROK=K(N).EQ.0.0D0
        ZEROK=K(N).EQ.0.0
  230 CONTINUE
C
C  Save K for restarts with new shifts.
C
      DO 240 I=1,N
        TEMP(I)=K(I)
  240 CONTINUE
C
C  Loop to select the quadratic corresponding
C  to each new shift.
C
      DO 280 CNT=1,20
C
C  Quadratic corresponds to a double shift to a
C  non-real point and its complex conjugate.  The point
C  has modulus BND and amplitude rotated by 94 degrees
C  from the previous shift.
C
        XXX=COSR*XX-SINR*YY
        YY=SINR*XX+COSR*YY
        XX=XXX
        SR=BND*XX
        SI=BND*YY
C¶D     U=-2.0D0*SR
        U=-2.0*SR
        V=BND
C
C  Second stage calculation, fixed quadratic
C
        CALL FXSHFR(20*CNT,NZ)
        IF(NZ.EQ.0)GO TO 260
C
C  The second stage jumps directly to one of the third
C  stage iterations and returns here if successful.
C  Deflate the polynomial, store the zero or zeros and
C  return to the main algorithm.
C
        J=DEGREE-N+1
        ZEROR(J)=SZR
        ZEROI(J)=SZI
```

```
          NN=NN-NZ
          N=NN-1
          DO 250 I=1,NN
            P(I)=QP(I)
   250    CONTINUE
          IF(N.EQ.1)GO TO 40
          ZEROR(J+1)=LZR
          ZEROI(J+1)=LZI
          GO TO 40
C
C  If the iteration is unsuccessful another quadratic
C  is chosen after restoring K.
C
   260    DO 270 I=1,N
            K(I)=TEMP(I)
   270    CONTINUE
   280 CONTINUE
C
C  Return with failure if no convergence with 20
C  shifts.
C
       FAIL=.TRUE.
       DEGREE=DEGREE-N
       RETURN
       END
C
C
C
C
C
       SUBROUTINE FXSHFR(L2,NZ)
C
C  Computes up to L2 fixed shift K-polynomials,
C  testing for convergence in the linear or quadratic
C  case.  Initiates one of the variable shift
C  iterations and returns with the number of zeros
C  found.
C
C  L2   -- Limit of fixed shift steps.
C  NZ   -- Number of zeros found.
C
       COMMON /GLOBAL/ P, QP, K, QK, SVK, SR, SI, U, V,
      * A, B, C, D, A1, A2, A3, A6, A7,
      * E, F, G, H, SZR, SZI, LZR, LZI,
      * ETA, ARE, MRE, N, NN
C¶D    DOUBLE PRECISION P(21), QP(21), K(21),
C¶D   $QK(21), SVK(21), SR, SI, U, V, A, B, C, D,
C¶D   $A1, A2, A3, A6, A7, E, F, G, H, SZR, SZI,
C¶D   $LZR, LZI
       REAL P(21), QP(21), K(21),
     $QK(21), SVK(21), SR, SI, U, V, A, B, C, D,
     $A1, A2, A3, A6, A7, E, F, G, H, SZR, SZI,
     $LZR, LZI
      REAL ETA, ARE, MRE
      INTEGER N, NN
C¶D   DOUBLE PRECISION SVU, SVV, UI, VI, S
      REAL SVU, SVV, UI, VI, S
      REAL BETAS, BETAV, OSS, OVV, SS, VV, TS, TV,
     $OTS, OTV, TVV, TSS
      INTEGER L2, NZ, TYPE, I, J, IFLAG
      LOGICAL VPASS, SPASS, VTRY, STRY
      NZ=0
      BETAV=0.25
      BETAS=0.25
      OSS=SR
      OVV=V
C
C  Evaluate polynomial by synthetic devision.
C
      CALL QUADSD(NN,U,V,P,QP,A,B)
      CALL CALCSC(TYPE)
      DO 80 J=1,L2
C
C  Calculate next K-polynomial and estimate V.
C
        CALL NEXTK(TYPE)
        CALL CALCSC(TYPE)
        CALL NEWEST(TYPE,UI,VI)
        VV=VI
C
C  Estimate S
C
        SS=0.0
C¶D     IF(K(N).NE.0.0D0)SS=-P(NN)/K(N)
        IF(K(N).NE.0.0)SS=-P(NN)/K(N)
        TV=1.0
        TS=1.0
        IF((J.EQ.1).OR.(TYPE.EQ.3))GO TO 70
C
C  Compute relative measures of convergence of S and V
C  sequences.
C
        IF(VV.NE.0.0)TV=ABS((VV-OVV)/VV)
        IF(SS.NE.0.0)TS=ABS((SS-OSS)/SS)
C
C  If decreasing, multiply two most recent
C  convergence measures.
C
        TVV=1.0
        IF(TV.LT.OTV)TVV=TV*OTV
      TSS=1.0
      IF(TS.LT.OTS)TSS=TS*OTS
C
C  Compare with convergence criteria.
C
      VPASS=TVV.LT.BETAV
      SPASS=TSS.LT.BETAS
      IF(.NOT.(SPASS.OR.VPASS))GO TO 70
C
C  At least one sequence has passed the convergence
C  test.  Store variables before iterating.
C
      SVU=U
      SVV=V
      DO 10 I=1,N
        SVK(I)=K(I)
 10   CONTINUE
      S=SS
C
C  Choose iteration according to the fastest
C  converging sequence.
C
      VTRY=.FALSE.
      STRY=.FALSE.
      IF(SPASS.AND.((.NOT.VPASS).OR.(TSS.LT.TVV)))GO TO 40
 20   CALL QUADIT(UI,VI,NZ)
      IF(NZ.GT.0) RETURN
C
C  Quadratic iteration has failed.  Flag that it has
C  been tried and decrease the convergence criterion.
C
      VTRY=.TRUE.
      BETAV=BETAV*0.25
C
C  Try linear iteration if it has not been tried and
C  the S sequence is converging.
C
      IF(STRY.OR.(.NOT.SPASS))GO TO 50
      DO 30 I=1,N
        K(I)=SVK(I)
 30   CONTINUE
 40   CALL REALIT(S,NZ,IFLAG)
      IF(NZ.GT.0)RETURN
C
C  Linear iteration has failed.  Flag that it has been
C  tried and decrease the convergence criterion.
C
      STRY=.TRUE.
      BETAS=BETAS*0.25
      IF(IFLAG.EQ.0)GO TO 50
```

```
C
C  If linear iteration signals an almost double real
C  zero, attempt quadratic iteration.
C
         UI=-(S+S)
         VI=S*S
         GO TO 20
C
C  Restore variables.
C
  50     U=SVU
         V=SVV
         DO 60 I=1,N
           K(I)=SVK(I)
  60     CONTINUE
C
C  Try quadratic iteration, if it has not been tried
C  and the V sequence is converging.
C
         IF(VPASS.AND.(.NOT.VTRY))GO TO 20
C
C  Recompute QP and scalar values to continue the
C  second stage.
C
         CALL QUADSD(NN,U,V,P,QP,A,B)
         CALL CALCSC(TYPE)
  70     OVV=VV
         OSS=SS
         OTV=TV
         OTS=TS
  80  CONTINUE
      RETURN
      END

C
C
C
      SUBROUTINE QUADIT(UU,VV,NZ)
C
C  Variable-shift K-polynomial iteration for a
C  quadratic factor converges only if the zeros are
C  equimodular or nearly so.
C
C  UU, VV  -- Coefficients of starting quadratic.
C  NZ      -- Number of zeros found.
C
      COMMON /GLOBAL/ P, QP, K, QK, SVK, SR, SI, U, V,
     * A, B, C, D, A1, A2, A3, A6, A7,
     * E, F, G, H, SZR, SZI, LZR, LZI,
     * ETA, ARE, MRE, N, NN
```

```
C¶D   DOUBLE PRECISION P(21), QP(21), K(21),
C¶D  $QK(21), SVK(21), SR, SI, U, V, A, B, C, D,
C¶D  $A1, A2, A3, A6, A7, E, F, G, H, SZR, SZI,
C¶D  $LZR, LZI
      REAL P(21), QP(21), K(21),
     $QK(21), SVK(21), SR, SI, U, V, A, B, C, D,
     $A1, A2, A3, A6, A7, E, F, G, H, SZR, SZI,
     $LZR, LZI
      REAL ETA, ARE, MRE
      INTEGER N, NN
C¶D   DOUBLE PRECISION UI, VI, UU, VV
      REAL UI, VI, UU, VV
      REAL MS, MP, OMP, EE, RELSTP, T, ZM
      INTEGER NZ, TYPE, I, J
      LOGICAL TRIED
      NZ=0
      TRIED=.FALSE.
      U=UU
      V=VV
      J=0
C
C  Main loop...
C
  10  CALL QUAD(1.0,U,V,SZR,SZI,LZR,LZI)
C
C  Return if roots of the quadratic are real and not
C  clse to multiple or nearly equal and of opposite
C  sign.
C
C¶D   IF(DABS(DABS(SZR)-DABS(LZR)).GT.(0.01D0*DABS(LZR)))RETURN
      IF(ABS(ABS(SZR)-ABS(LZR)).GT.(0.01*ABS(LZR)))RETURN
C
C  Evaluate polynomial by quadratic synthetic devision.
C
      CALL QUADSD(NN,U,V,P,QP,A,B)
C¶D   MP=DABS(A-SZR*B)+DABS(SZI*B)
      MP=ABS(A-SZR*B)+ABS(SZI*B)
C
C  Compute a rigorous bound on the rounding error in
C  evaluating P.
C
C¶D   ZM=SQRT(ABS(SNGL(V)))
      ZM=SQRT(ABS(V))
C¶D   EE=2.0*ABS(SNGL(QP(1)))
      EE=2.0*ABS(QP(1))
      T=-SZR*B
      DO 20 I=2,N
C¶D     EE=EE*ZM+ABS(SNGL(QP(I)))
        EE=EE*ZM+ABS(QP(I))
  20  CONTINUE
```

```
C¶D   EE=EE*ZM+ABS(SNGL(A)+T)
      EE=EE*ZM+ABS(A+T)
C¶D   EE=(5.0*MRE+4.0*ARE)*EE-(5.0*MRE+2.0*ARE)*
C¶D  $(ABS(SNGL(A)+T)+ABS(SNGL(B))*ZM)+2.0*ARE*ABS(T)
      EE=(5.0*MRE+4.0*ARE)*EE-(5.0*MRE+2.0*ARE)*
     $(ABS(A+T)+ABS(B)*ZM)+2.0*ARE*ABS(T)
C
C  Iteration has converged sufficiently if the
C  polynomial value is less than 20 times this bound.
C
      IF(MP.GT.20.0*EE)GO TO 30
      NZ=2
      RETURN
   30 J=J+1
C
C  Stop iteration after 20 steps.
C
      IF(J.GT.20)RETURN
      IF(J.LT.2)GO TO 50
      IF((RELSTP.GT.0.01).OR.((MP.LT.OMP).OR.TRIED))GO TO 50
C
C  A cluster appears to be stalling the convergence.
C  Five fixed shift steps are taken with a U,V close
C  to the cluster.
C
      IF(RELSTP.LT.ETA)RELSTP=ETA
      RELSTP=SQRT(RELSTP)
      U=U-U*RELSTP
      V=V+V*RELSTP
      CALL QUADSD(NN,U,V,P,QP,A,B)
      DO 40 I=1,5
        CALL CALCSC(TYPE)
        CALL NEXTK(TYPE)
   40 CONTINUE
      TRIED=.TRUE.
      J=0
   50 OMP=MP
C
C  Calculate next K-polynomial and new U and V
C
      CALL CALCSC(TYPE)
      CALL NEXTK(TYPE)
      CALL CALCSC(TYPE)
      CALL NEWEST(TYPE,UI,VI)
C
C  If VI is zero the iteration is not converging
C
C¶D   IF(VI.EQ.0.0D0)RETURN
      IF(VI.EQ.0.0)RETURN
C¶D   RELSTP=DABS((VI-V)/VI)
```

```
      RELSTP=ABS((VI-V)/VI)
      U=UI
      V=VI
      GO TO 10
      END
C
C
C
C
C
      SUBROUTINE REALIT(SSS,NZ,IFLAG)
C
C  Variable-shift H polynomial iteration for a real
C  zero.
C
C  SSS     -- Starting iterate
C  NZ      -- Number of zero found
C  IFLAG   -- Flag to indicate a pair of zeros near real
C             axis.
C
      COMMON /GLOBAL/ P, QP, K, QK, SVK, SR, SI, U, V,
     * A, B, C, D, A1, A2, A3, A6, A7,
     * E, F, G, H, SZR, SZI, LZR, LZI,
     * ETA, ARE, MRE, N, NN
C¶D   DOUBLE PRECISION P(21), QP(21), K(21),
C¶D  $QK(21), SVK(21), SR, SI, U, V, A, B, C, D,
C¶D  $A1, A2, A3, A6, A7, E, F, G, H, SZR, SZI,
C¶D  $LZR, LZI
      REAL P(21), QP(21), K(21),
     $QK(21), SVK(21), SR, SI, U, V, A, B, C, D,
     $A1, A2, A3, A6, A7, E, F, G, H, SZR, SZI,
     $LZR, LZI
      REAL ETA, ARE, MRE
      INTEGER N, NN
C¶D   DOUBLE PRECISION PV, KV, T, S, SSS
      REAL PV, KV, T, S, SSS
      REAL MS, MP, OMP, EE
      INTEGER NZ, IFLAG, I, J, NM1
      NM1=N-1
      NZ=0
      S=SSS
      IFLAG=0
      J=0
C
C  Main loop ...
C
 10   PV=P(1)
C
C  Evaluate P at S
C
```

```
      QP(1)=PV
      DO 20 I=2,NN
        PV=PV*S+P(I)
        QP(I)=PV
   20 CONTINUE
C¶D   MP=DABS(PV)
      MP=ABS(PV)
C
C  Compute a rigorous bound on the error in evaluating P
C
C¶D   MS=DABS(S)
      MS=ABS(S)
C¶D   EE=(MRE/(ARE+MRE))*ABS(SNGL(QP(1)))
      EE=(MRE/(ARE+MRE))*ABS(QP(1))
      DO 30 I=2,NN
C¶D     EE=EE*MS+ABS(SNGL(QP(I)))
        EE=EE*MS+ABS(QP(I))
   30 CONTINUE
C
C  Iteration has converged sufficiently if the
C  polynomial value is less than 20 times this bound.
C
      IF(MP.GT.(20.0*((ARE+MRE)*EE-MRE*MP)))GO TO 40
      NZ=1
      SZR=S
C¶D   SZI=0.0D0
      SZI=0.0
      RETURN
   40 J=J+1
C
C  Stop iteration after 10 steps.
C
      IF(J.GT.10)RETURN
      IF(J.LT.2)GO TO 50
C¶D   IF((DABS(T).GT.(0.001*DABS(S-T))).OR.(MP.LE.OMP))
C¶D  $GO TO 50
      IF((ABS(T).GT.(0.001*ABS(S-T))).OR.(MP.LE.OMP))
     $GO TO 50
C
C  A cluster of zeros near the real axis has been
C  encountered.  Return with IFLAG set to indicate a
C  quadratic iteration
C
      IFLAG=1
      SSS=S
      RETURN
C
C  Return if the polynomial value has increased
C  significantly.
C
```

```
  50  OMP=MP
C
C  Compute T, the next polynomial, and the new iterate
C
      KV=K(1)
      QK(1)=KV
      DO 60 I=2,N
        KV=KV*S+K(I)
        QK(I)=KV
  60  CONTINUE
C¶D   IF(DABS(KV).LE.(DABS(K(N))*10.0*ETA))GO TO 80
      IF(ABS(KV).LE.(ABS(K(N))*10.0*ETA))GO TO 80
C
C  Use the scaled form of the recurrence if the value
C  of K at S is nonzero.
C
      T=-PV/KV
      K(1)=QP(1)
      DO 70 I=2,N
        K(I)=T*QK(I-1)+QP(I)
  70  CONTINUE
      GO TO 100
C
C  Use unscaled form.
C
C¶D 80  K(1)=0.0D0
  80  K(1)=0.0
      DO 90 I=2,N
        K(I)=QK(I-1)
  90  CONTINUE
  100 KV=K(1)
      DO 110 I=2,N
        KV=KV*S+K(I)
  110 CONTINUE
C¶D   T=0.0D0
      T=0.0
C¶D   IF(DABS(KV).GT.(DABS(K(N))*10.0*ETA))T=-PV/KV
      IF(ABS(KV).GT.(ABS(K(N))*10.0*ETA))T=-PV/KV
      S=S+T
      GO TO 10
      END
C
C
C
C
C
      SUBROUTINE CALCSC(TYPE)
C
C  This routine calculates scalar quantities used to
C  compute the next K polynomial and new estimates of
```

```
C  the quadratic coefficients.
C
C  TYPE  -- Integer variable set here indicating how the
C           calculations are normalized to avoid overflow.
C
      COMMON /GLOBAL/ P, QP, K, QK, SVK, SR, SI, U, V,
     * A, B, C, D, A1, A2, A3, A6, A7,
     * E, F, G, H, SZR, SZI, LZR, LZI,
     * ETA, ARE, MRE, N, NN
C¶D   DOUBLE PRECISION P(21), QP(21), K(21),
C¶D  $QK(21), SVK(21), SR, SI, U, V, A, B, C, D,
C¶D  $A1, A2, A3, A6, A7, E, F, G, H, SZR, SZI,
C¶D  $LZR, LZI
      REAL P(21), QP(21), K(21),
     $QK(21), SVK(21), SR, SI, U, V, A, B, C, D,
     $A1, A2, A3, A6, A7, E, F, G, H, SZR, SZI,
     $LZR, LZI
      REAL ETA, ARE, MRE
      INTEGER N, NN
      INTEGER TYPE
C
C  Synthetic division of K by the quadratic I,U,V
C
      CALL QUADSD(N,U,V,K,QK,C,D)
C¶D   IF(DABS(C).GT.(DABS(K(N))*100.0*ETA))GO TO 10
      IF(ABS(C).GT.(ABS(K(N))*100.0*ETA))GO TO 10
C¶D   IF(DABS(D).GT.(DABS(K(N-1))*100.0*ETA))GO TO 10
      IF(ABS(D).GT.(ABS(K(N-1))*100.0*ETA))GO TO 10
      TYPE=3
C
C  TYPE=3 indicates the quadratic is almost a factor
C  of K
C
      RETURN
C¶D 10  IF(DABS(D).LT.DABS(C))GO TO 20
 10   IF(ABS(D).LT.ABS(C))GO TO 20
      TYPE=2
C
C  TYPE=2 indicates that all formulas are divided by D
C
      E=A/D
      F=C/D
      G=U*B
      H=V*B
      A3=(A+G)*E+H*(B/D)
      A1=B*F-A
      A7=(F+U)*A+H
      RETURN
 20   TYPE=1
C
```

```
C  TYPE=1 indicates that all formulas are divided by C
C
      E=A/C
      F=D/C
      G=U*E
      H=V*B
      A3=A*E+((H/C)+G)*B
      A1=B-A*(D/C)
      A7=A+G*D+H*F
      RETURN
      END
C
C
C
C
C
      SUBROUTINE NEXTK(TYPE)
C
C  Computes the next K-polynomials using scalars
C  computed in CALCSC.
C
      COMMON /GLOBAL/ P, QP, K, QK, SVK, SR, SI, U, V,
     * A, B, C, D, A1, A2, A3, A6, A7,
     * E, F, G, H, SZR, SZI, LZR, LZI,
     * ETA, ARE, MRE, N, NN
C¶D   DOUBLE PRECISION P(21), QP(21), K(21),
C¶D  $QK(21), SVK(21), SR, SI, U, V, A, B, C, D,
C¶D  $A1, A2, A3, A6, A7, E, F, G, H, SZR, SZI,
C¶D  $LZR, LZI
      REAL P(21), QP(21), K(21),
     $QK(21), SVK(21), SR, SI, U, V, A, B, C, D,
     $A1, A2, A3, A6, A7, E, F, G, H, SZR, SZI,
     $LZR, LZI
      REAL ETA, ARE, MRE
      INTEGER N, NN
C¶D   DOUBLE PRECISION TEMP
      REAL TEMP
      INTEGER TYPE
      IF(TYPE.EQ.3)GO TO 40
      TEMP=A
      IF(TYPE.EQ.1)TEMP=B
C¶D   IF(DABS(A1).GT.(DABS(TEMP)*10.0*ETA))GO TO 20
      IF(ABS(A1).GT.(ABS(TEMP)*10.0*ETA))GO TO 20
C
C  If A1 is nearly zero then use a special form of the
C  recurrence.
C
C¶D   K(1)=0.0D0
      K(1)=0.0
      K(2)=-A7*QP(1)
      DO 10 I=3,N
        K(I)=A3*QK(I-2)-A7*QP(I-1)
   10 CONTINUE
      RETURN
C
C  Use scaled form of the recurrence.
C
   20 A7=A7/A1
      A3=A3/A1
      K(1)=QP(1)
      K(2)=QP(2)-A7*QP(1)
      DO 30 I=3,N
        K(I)=A3*QK(I-2)-A7*QP(I-1)+QP(I)
   30 CONTINUE
      RETURN
C
C  Use unscaled form of the recurrence if TYPE is 3
C
C¶D 40  K(1)=0.0D0
   40 K(1)=0.0
C¶D   K(2)=0.0D0
      K(2)=0.0
      DO 50 I=3,N
        K(I)=QK(I-2)
   50 CONTINUE
      RETURN
      END
C
C
C
C
C
      SUBROUTINE NEWEST(TYPE,UU,VV)
C
C  Computes new estimates of the quadratic coefficients
C  using the scalars computed in CALCSC.
C
      COMMON /GLOBAL/ P, QP, K, QK, SVK, SR, SI, U, V,
     * A, B, C, D, A1, A2, A3, A6, A7,
     * E, F, G, H, SZR, SZI, LZR, LZI,
     * ETA, ARE, MRE, N, NN
C¶D   DOUBLE PRECISION P(21), QP(21), K(21),
C¶D  $QK(21), SVK(21), SR, SI, U, V, A, B, C, D,
C¶D  $A1, A2, A3, A6, A7, E, F, G, H, SZR, SZI,
C¶D  $LZR, LZI
      REAL P(21), QP(21), K(21),
     $QK(21), SVK(21), SR, SI, U, V, A, B, C, D,
     $A1, A2, A3, A6, A7, E, F, G, H, SZR, SZI,
     $LZR, LZI
      REAL ETA, ARE, MRE
      INTEGER N, NN
C¶D   DOUBLE PRECISION A4, A5, B1, B2, C1, C2, C3,
C¶D  $C4, TEMP, UU, VV
      REAL A4, A5, B1, B2, C1, C2, C3,
     $C4, TEMP, UU, VV
      INTEGER TYPE
C
C  Use formulas appropriate to setting of TYPE.
C
      IF(TYPE.EQ.3)GO TO 30
      IF(TYPE.EQ.2)GO TO 10
      A4=A+U*B+H*F
      A5=C+(U+V*F)*D
      GO TO 20
  10  A4=(A+G)*F+H
      A5=(F+U)*C+V*D
C
C  Evaluate new quadratic coefficients.
C
  20  B1=-K(N)/P(NN)
      B2=-(K(N-1)+B1*P(N))/P(NN)
      C1=V*B2*A1
      C2=B1*A7
      C3=B1*B1*A3
      C4=C1-C2-C3
      TEMP=A5+B1*A4-C4
C¶D   IF(TEMP.EQ.0.0D0)GO TO 30
      IF(TEMP.EQ.0.0)GO TO 30
      UU=U-(U*(C3+C2)+V*(B1*A1+B2*A7))/TEMP
      VV=V*(1.0+(C4/TEMP))
      RETURN
C
C  If TYPE=3 the quadratic is zeroed.
C
C¶D 30  UU=0.0D0
  30  UU=0.0
C¶D   VV=0.0D0
      VV=0.0
      RETURN
      END
C
C
C
C
C
      SUBROUTINE QUADSD(NN,U,V,P,Q,A,B)
C
C  Divides P by the quadratic 1,U,V placing the
C  quotient in Q and the remainder in A,B
C
```

```
C¶D   DOUBLE PRECISION P(NN),Q(NN),U,V,A,B,C
      REAL P(NN),Q(NN),U,V,A,B,C
      INTEGER I
      B=P(1)
      Q(1)=B
      A=P(2)-U*B
      Q(2)=A
      DO 10 I=3,NN
        C=P(I)-U*A-V*B
        Q(I)=C
        B=A
        A=C
  10  CONTINUE
      RETURN
      END
C
C
C
      SUBROUTINE QUAD(A,B1,C,SR,SI,LR,LI)
C
C  Calculates the zeros of the quadratic A*Z**2+B1*Z+C.
C  The quadratic formula, modified to avoid
C  overflow, is used to find the larger zero if the
C  zeros are real and both zeros are complex.
C  The smaller real zero is found directly from the
C  product of the zeros C/A.
C
C
C¶D   DOUBLE PRECISION A,B1,C,SR,SI,LR,LI,B,D,E
      REAL A,B1,C,SR,SI,LR,LI,B,D,E
C¶D   IF(A.NE.0.0D0)GO TO 20
      IF(A.NE.0.0)GO TO 20
C¶D   SR=0.0D0
      SR=0.0
C¶D   IF(B1.NE.0.0D0)SR=-C/B1
      IF(B1.NE.0.0)SR=-C/B1
C¶D   LR=0.0D0
      LR=0.0
C¶D 10  SI=0.0D0
  10  SI=0.0
C¶D   LI=0.0D0
      LI=0.0
      RETURN
C¶D 20  IF(C.NE.0.0D0)GO TO 30
  20  IF(C.NE.0.0)GO TO 30
C¶D   SR=0.0D0
      SR=0.0
      LR=-B1/A
      GO TO 10
C
```

```
C  Compute descriminant avoiding overflow.
C
C¶D 30  B=B1/2.0D0
  30  B=B1/2.0
C¶D   IF(DABS(B).LT.DABS(C))GO TO 40
      IF(ABS(B).LT.ABS(C))GO TO 40
C¶D   E=1.0D0-((A/B)*(C/B))
      E=1.0-((A/B)*(C/B))
C¶D   D=DSQRT(DABS(E))*DABS(B)
      D=SQRT(ABS(E))*ABS(B)
      GO TO 50
  40  E=A
C¶D   IF(C.LT.0.0D0)E=-A
      IF(C.LT.0.0)E=-A
C¶D   E=B*(B/DABS(C))-E
      E=B*(B/ABS(C))-E
C¶D   D=DSQRT(DABS(E))*DSQRT(DABS(C))
      D=SQRT(ABS(E))*SQRT(ABS(C))
C¶D 50  IF(E.LT.0.0D0)GO TO 60
  50  IF(E.LT.0.0)GO TO 60
C
C  Real zeros
C
C¶D   IF(B.GE.0.0D0)D=-D
      IF(B.GE.0.0)D=-D
      LR=(D-B)/A
C¶D   SR=0.0D0
      SR=0.0
C¶D   IF(LR.NE.0.0D0)SR=(C/LR)/A
      IF(LR.NE.0.0)SR=(C/LR)/A
      GO TO 10
C
C  Complex conjugate zeros
C
  60  SR=-B/A
      LR=SR
C¶D   SI=DABS(D/A)
      SI=ABS(D/A)
      LI=-SI
      RETURN
      END
```

Appendix F

TWO-DIMENSIONAL GRAPHICS PROGRAM - HIPLOT

The HIPLOT plotting and regression program was developed on an Apple II plus micro-computer. The program requires 64 kilobytes of installed memory, one or more disk drives, the Apple Language System (UCSD Pascal), California Computer Systems 7710A serial asynchronous interface card and the HIPLOT DPM-2 digital plotter. The plotter is not necessary for regression analysis.

The regression package can perform linear, power, exponential, logarithmic and polynomial regressions up to and including order 9. The plotting section can produce box plots with x, y and top labels. The regression results may be plotted on the same graph as the data. Legends and multiplot options are also provided in the program in an interactive form.

The main program is written in a general form and in order to adapt it for a different plotting device only the PLOTSTUFF unit must be modified. The character set was generated by the CHARGEN program. The program is divided into a main section and three overlay sections: READFILE, PLOT and REGRESS. This allows for a maximum storage memory allocation for the data. A maximum of 500 data points pairs may be read and processed.

For general background information see Chapter 2 of the Apple Pascal Operating System Reference Manual.

```
§$S+†
PROGRAM BOXPLOT;

§Pascal program for regression and plotting†
§requirements: Apple ][/][+, two disk drives and Apple Pascal†

USES
   TRANSCEND,§$U BPPLOT:PLOTSTUFF.CODE†PLOTSTUFF;

CONST
   §The global constants are set for HIPLOT digital plotter†
   PLOTLENGTH=10.0;§Plot length determined by HI-PLOT limits†
   PLOTHEIGHT=7.0; §Plot height determined by HI-PLOT limits†
   TEKMAXLENGTH=8.5; §Maximum plot size for labels to fit†
   TEKMAXHEIGHT=5.5;
   §Grid spacing of 0.005 inches for the HIPLOT digital plotter†
   LGRID=0;
   RGRID=2000;
   BGRID=0;
   TGRID=1400;
   §Maximum number of data points†
   MAXCASE =  501;
   TABLELEN = 3;
   §Set up global constants†
   PI = 3.14159265;
   MISSING=-999.99;

TYPE
   DATAREC = PACKED RECORD
                POINTS:   0..MAXCASE;
                DATA:  ARRAY [0..MAXCASE] OF REAL;
             END;
   REGTYPE = (LIN, LOGARM, EXPO, PWR, POLY);
   POINTER = ©DATAREC;

VAR
   TABLE: ARRAY [0..TABLELEN] OF POINTER;
   LINK: POINTER;
   HEAP: ©INTEGER;
   PSLOT,FACTORS, MAXPTS, I: INTEGER;
   XLO, XHI, YLO, YHI: INTEGER;
   XLABEL, YLABEL, PLABEL1,PLABEL2: STRING;
   INNAME,OUTNAME : STRING;
   FIRSTTIME,MULTIPLOT,PRINTOUT: BOOLEAN;
   XFACT, YFACT, EFACT: INTEGER;
   REG : REGTYPE;
   OUTFILE : INTERACTIVE;
   POLYCOEF : ARRAY[0..10] OF REAL;
   XSCALE, YSCALE: REAL;
   TBOUND,BBOUND,LBOUND,RBOUND,D : INTEGER;
   XCHSIZE,YCHSIZE : REAL;
   A, B : REAL;
   R, RXLO, RXHI, RYLO, RYHI: REAL;
   XMIN, XMAX, XTEMP: REAL;
   YMIN, YMAX, YTEMP: REAL;
   X, Y, DX, DY, XTICS, YTICS: INTEGER;
   XDIVISION, YDIVISION: INTEGER;
   YLDIST:INTEGER;
   NUMS: STRING;
   POINT: CHAR;
   CONNECT: BOOLEAN;
   PARAMS: ARRAY[1..8] OF STRING;


PROCEDURE LABELPLOT;
  FORWARD;


FUNCTION REALVAL(S:STRING):REAL;
  FORWARD;


FUNCTION REGVALUE(X:REAL):REAL;
  FORWARD;


PROCEDURE PRINTONOFF;
  FORWARD;


§$I BPPLOT:DATPLOT.TEXT†

§$I BPPLOT:GETDATFILE.TEXT†

§$I BPPLOT:REGRESS.TEXT†


PROCEDURE PLOTSIZE;

VAR RMARG,LMARG,TMARG,BMARG : REAL;
    XSIZE, YSIZE : REAL;
    ANSWER : CHAR;


PROCEDURE GETSIZE;

BEGIN
  GOTOXY(0,5);
  WRITELN('Maximum X size is 8.5 inches');
  WRITELN('Maximum Y size is 5.5 inches');
  GOTOXY(0,8);
  WRITE('Input X size --> ');
  READLN(XSIZE);
  GOTOXY(0,9);
  WRITE('Input Y size --> ');
  READLN(YSIZE);
  RMARG:=(PLOTLENGTH-XSIZE)/2.0;
  LMARG:=RMARG;
  TMARG:=(PLOTHEIGHT-YSIZE)/2.0;
  BMARG:=TMARG;
END;


PROCEDURE GETMARG;

BEGIN
  GOTOXY(0,5);
  WRITE('Input top and bottom margins: ');
  READLN(TMARG,BMARG);
  WRITE('Input left and right margins: ');
  READLN(LMARG,RMARG);
  XSIZE:=PLOTLENGTH-(LMARG+RMARG);
  YSIZE:=PLOTHEIGHT-(TMARG+BMARG);
END;


BEGIN§getsize†
  REPEAT
    Repeat
      WRITE(HOME,EOSCLEAR);
      GOTOXY(0,2);
      WRITE('Center the plot? (Y/N)');
      READ(KEYBOARD,ANSWER);
      §$I-†
      IF (ANSWER = 'Y') OR (ANSWER = 'y') THEN GETSIZE
      ELSE GETMARG;
      §$I+†
    Until not BADIO;
  UNTIL ((XSIZE <= TEKMAXLENGTH) AND (YSIZE <= TEKMAXHEIGHT));
  WRITELN;
  WRITELN('Length...............',XSIZE:4:2);
  WRITELN('Height...............',YSIZE:4:2);
  WRITELN('Top margin...........',TMARG:4:2);
  WRITELN('Bottom margin........',BMARG:4:2);
  WRITELN('Left margin..........',LMARG:4:2);
  WRITELN('Right margin.........',RMARG:4:2);
  GOTOXY(0,23);
  WRITE(EOLNCLEAR,'Change character size? (Y/N)');
  READ(KEYBOARD,ANSWER);
  IF (ANSWER = 'Y') OR (ANSWER = 'y') THEN
    BEGIN
      REPEAT
        Repeat
          GOTOXY(0,20);
          §$I-†
          WRITE(EOLNCLEAR,'Input X character size -->');
          READLN(XCHSIZE);
          WRITE(EOLNCLEAR,'Input Y character size -->');
          READLN(YCHSIZE);
          §$I+†
        Until not BADIO;
      UNTIL ((XCHSIZE > 0) AND (XCHSIZE < 6)
         AND (YCHSIZE > 0) AND (YCHSIZE < 6));
    END;
  LBOUND:= TRUNC((LMARG/PLOTLENGTH)*(RGRID-LGRID));
  RBOUND:= TRUNC(((LMARG+XSIZE)/PLOTLENGTH)*(RGRID-LGRID));
  TBOUND:= TRUNC(((BMARG+YSIZE)/PLOTHEIGHT)*(TGRID-BGRID));
  BBOUND:= TRUNC((BMARG/PLOTHEIGHT)*(TGRID-BGRID));
END;§getsize†


PROCEDURE LABELPLOT;

VAR
   I, X, Y: INTEGER;
   S: STRING;

BEGIN
   WRITE (HOME,EOSCLEAR);
   WRITELN ('Select X and Y axis labels...');
   WRITELN ('<return> for default');
   GOTOXY (0,3);
   WRITELN ('Default labels:');
   WRITELN;
   WRITELN ('X axis = ',XLABEL);
   WRITELN ('Y axis = ',YLABEL);
   WRITELN ('First line of title =');
   WRITELN (PLABEL1);
   WRITELN ('Second line of title =');
   WRITELN (PLABEL2);
   REPEAT
     GOTOXY (0,11);
     WRITE (EOLNCLEAR,'X axis label:  ');
     READLN (S);
     IF LENGTH (S) > 0 THEN XLABEL:= S;
   UNTIL LENGTH (S) < 80;
   REPEAT
     GOTOXY (0,13);
     WRITE (EOLNCLEAR,'Y axis label:  ');
     READLN (S);
     IF LENGTH (S) > 0 THEN YLABEL:= S;
   UNTIL LENGTH (S) < 80;
   REPEAT
     GOTOXY (0,15);
     WRITELN (EOLNCLEAR,'Enter plot label: ');
     WRITELN ('First line: ');
     READLN (S);
     IF LENGTH (S) <> 0 THEN PLABEL1:=S;
     WRITELN ('Second line: ');
     READLN(S);
     IF LENGTH(S) <> 0 THEN PLABEL2:=S;
   UNTIL LENGTH (S) < 80;
END; §Selectt


PROCEDURE PRINTONOFF;

BEGIN
  §$I-†
  CLOSE (OUTFILE,LOCK);
  IF badio THEN exit(PRINTONOFF);
  §$I+†
  IF PRINTOUT THEN
    BEGIN
      PRINTOUT:=FALSE;
      OUTNAME:='CONSOLE:';
    END
  ELSE
    BEGIN
      PRINTOUT:=TRUE;
      WRITE(HOME,EOSCLEAR);
      GOTOXY(0,10);
      WRITE('Output file name -->');
      READLN(OUTNAME);
      IF POS('.TEXT',OUTNAME) = 0 THEN
         OUTNAME:=CONCAT(OUTNAME,'.TEXT');
    END;
  REWRITE(OUTFILE,OUTNAME);
END; §PRINTONOFF†


FUNCTION REGVALUE;

VAR
   R, Y: REAL;
   I: INTEGER;

BEGIN
  CASE REG OF
    LIN:    REGVALUE:= A + B * X;
    LOGARM: BEGIN
              IF X > 0 THEN REGVALUE:= A + B * LN(X)
              ELSE IF X = 0 THEN REGVALUE:= A+B
                   ELSE REGVALUE:= MISSING;
              IF X < 0 THEN WRITELN ('Undefined');
            END;
    EXPO:   IF B*X < 87 THEN REGVALUE:= A * EXP(B * X)
            ELSE
              BEGIN
                WRITELN ('Expotential overflow!');
                REGVALUE:= MISSING;
              END;
    PWR:    IF X=0 THEN REGVALUE:=0
            ELSE IF B=0 THEN REGVALUE:= A
                 ELSE REGVALUE:= A * (EXP(B * LN(X)));
    POLY:   BEGIN
              Y:= POLYCOEF[0];
              R:= X; I:= 1;
              REPEAT
                 Y:= Y + POLYCOEF[I] * R;
                 R:= R*X; I:= I+1;
              UNTIL (POLYCOEF[I] = 0) OR (I=11);
              REGVALUE:= Y;
            END;
  END; §case†
END; §predict†


FUNCTION REALVAL;

VAR
   RESULT: REAL;
   EXPON: INTEGER;
   REALCH: SET OF CHAR;
   I, PT, E: INTEGER;
   NEG, NEGEXP: BOOLEAN;

BEGIN
  REALCH := ['0','1','2','3','4','5','6','7','8','9',
             '.','+','-','E',' '];
  BAD:=FALSE;
  NEG:= FALSE;
  NEGEXP:= FALSE;
  EXPON:=0;
```

```
RESULT:=0.0;
I:=1;
REPEAT
  IF NOT (S[I] IN REALCH) THEN BAD:=TRUE;
  IF (S[I]='+') OR (S[I]=' ') THEN DELETE(S,I,1)
  ELSE I:=I+1;
UNTIL I > LENGTH(S);
IF POS('-',S) > 0 THEN
  BEGIN
    IF POS('-',S) = 1 THEN
      BEGIN
        NEG:= TRUE;
        DELETE(S,1,1);
      END;
  END;
IF (POS('-',S) > 0) THEN
  BEGIN
    IF POS('E-',S) > 0 THEN
      BEGIN
        NEGEXP:= TRUE;
        DELETE(S,POS('-',S),1);
      END
    ELSE BAD:= TRUE;
  END; §ift
E:= POS('E',S);
IF  (E > 0)
AND (POS('E',COPY(S,E+1,LENGTH(S)-E)) > 0) THEN
   BAD:= TRUE;
PT:= POS('.',S);
IF PT > 0 THEN
  BEGIN
    IF POS('.',COPY(S,PT+1,LENGTH(S)-PT)) > 0 THEN
       BAD:= TRUE;
  END
ELSE
  BEGIN
    IF E > 0 THEN
      BEGIN
        PT:= E;
        INSERT('.0',S,E);
        E:=E+2;
      END
    ELSE
      BEGIN
        PT:= LENGTH(S) + 1;
        S:= CONCAT(S,'.');
      END;
  END;
IF NOT BAD THEN
```

```
    BEGIN
      IF PT > 1 THEN FOR I:=1 TO PT-1 DO
        RESULT:= RESULT + PWROFTEN(PT-I-1) * (ORD(S[I]) - 48);
      IF E = 0 THEN FOR I:= PT+1 TO LENGTH(S)  DO
        RESULT:= RESULT + (ORD(S[I]) - 48) / PWROFTEN(I-PT)
      ELSE
        BEGIN
          FOR I:=PT+1 TO E-1 DO
            RESULT:= RESULT + (ORD(S[I])-48) / PWROFTEN(I-PT);
          FOR I:= E+1 TO LENGTH(S) DO
            EXPON:=EXPON +
              TRUNC(PWROFTEN(LENGTH(S)-I)) * (ORD(S[I])-48);
          IF NEGEXP THEN RESULT:=RESULT / PWROFTEN(EXPON)
          ELSE RESULT:=RESULT * PWROFTEN(EXPON);
        END;
      IF NEG THEN RESULT:= -RESULT;
      REALVAL:= RESULT;
    END
  ELSE
    BEGIN
      REALVAL:=-0.0;
      WRITELN (CHR(7),'Invalid Real Number');
    END;
END §valuet;


PROCEDURE INITIALIZE;

BEGIN
  XCHSIZE:=3.5;
  YCHSIZE:=3.5;
  LBOUND:=400;
  BBOUND:=400;
  RBOUND:=1800;
  TBOUND:=1200;
  D:=10;
  XFACT:=1;
  YFACT:=2;
  CHARSCALE(XCHSIZE,YCHSIZE);
  GETCRTINFO;
  MARK (HEAP);
  MULTIPLOT:= FALSE;
  PRINTOUT:=FALSE;
  RESET(OUTFILE,'CONSOLE:');
  FACTORS:=0;
  PSLOT:=2;
  PLOTSLOT(2);
  FIRSTTIME:=TRUE;
  Repeat
```

```
    §$I-tgetdatfile;§$I+t
   Until not BADIO;
   FIRSTTIME:=FALSE;
 END;


 BEGIN §main programt
   INITIALIZE;
   REPEAT
      GOTOXY(0,19);
      WRITELN(EOLNCLEAR,'HI-PLOT interface in slot # ',PSLOT);
      IF PRINTOUT
        THEN WRITE (EOLNCLEAR,'PRINTER ON ')
        ELSE WRITE (EOLNCLEAR,'PRINTER OFF');
      GOTOXY (0,21);
      WRITELN (EOLNCLEAR,FACTORS, ' Factors');
      IF FACTORS > 0 THEN
      WRITELN(EOLNCLEAR,'X = ', XLABEL);
      WRITE(EOLNCLEAR,'Y = ', YLABEL);
      GOTOXY (0,0);
      WRITE(EOLNCLEAR, 'HIPLOT: R)egress, I)nput, G)raph, ');
      WRITE('L)able, T)ogprint, S)ize, P)ort, N)ew, Q)uit.');
      READ (KEYBOARD, CH);
      CASE CH OF
         'R','r':  REGRESS;
         'T','t':  PRINTONOFF;
         'S','s':  PLOTSIZE;
         'L','l':  LABELPLOT;
         'I','i':  GETDATFILE;
         'G','g':  PLOTIT;
         'N','n':  BEGIN
                     PLOTLIM(LGRID,RGRID,BGRID,TGRID);
                     PAGE;
                     RINGIT;
                   END;
             ' ' : WRITE(HOME,EOSCLEAR);
         'P','p':  BEGIN
                     WRITE(HOME,EOSCLEAR);
                     GOTOXY (0,10);
                     WRITE ('Enter HIPLOT interface');
                     WRITE (' slot number:  ');
                     READLN (PSLOT);
                     PLOTSLOT (PSLOT);
                   END;
         'C','c':  BEGIN
                    REPEAT
                     Repeat
                       WRITE(HOME,EOSCLEAR);
                       GOTOXY(0,5);
```

```
                          §$I-†
                          WRITE('Input X character size -->');
                          READLN(XCHSIZE);
                          WRITELN;
                          WRITE('Input Y character size -->');
                          READLN(YCHSIZE);
                          §$I+†
                        Until not BADIO;
                       UNTIL ((XCHSIZE > 0) AND (YCHSIZE > 0)
                          AND (XCHSIZE < 6) AND (YCHSIZE < 6));
                      END;
     END; §Cases†
  UNTIL CH IN ['Q','q'];
  WRITE (HOME,EOSCLEAR);
  PAGE;
  §$I-†
  CLOSE(OUTFILE,LOCK);
  IF badio THEN WRITE(HOME,EOSCLEAR);
  §$I+†
  GOTOXY (0,10);
  WRITE ('That''s all folks...');
END.
```

```
SEGMENT PROCEDURE PLOTIT;

VAR CHNUMB : INTEGER;


Function sure: boolean;

Var ch: char;

Begin
  writeln;
  write('Are you sure? Y/N  ');
  read(keyboard,ch);
  If (ch = 'Y') or (ch = 'n') then sure:=true
  Else sure:=false;
End;


PROCEDURE WREAL(K,XCO,YCO:INTEGER;XCHSIZE,YCHSIZE,X:REAL);

VAR
   LENG,HEIGHT : INTEGER;
   D,P:STRING;
   XPCHSIZE: REAL;


PROCEDURE RSTR(X:REAL);

CONST
     LN10=2.3025851;

VAR
   I,N,M,M1,M2 : INTEGER;
   D1,D2:STRING;
   FACT,Y,DELTA : REAL;

BEGIN
  IF X <> 0.0 THEN
    BEGIN
      Y:=LOG(ABS(X));
      IF (Y >= 3.0) OR (Y <= -1.0) THEN
        BEGIN
          N:=TRUNC(Y);
          DELTA:=Y-N;
          IF Y <= -1.0 THEN DELTA:=DELTA+3.0
          ELSE DELTA:=DELTA+2.0;
          M:=ROUND(EXP(DELTA*LN10));
          FACT:=100.0;
          M2:=ROUND((M/FACT-TRUNC(M/FACT))*FACT);
          M1:=ROUND((M-M2)/FACT);
          STR(M1,D1);
          STR(M2,D2);
          IF X < 0.0 THEN D:=CONCAT('-',D1,'.',D2)
          ELSE D:=CONCAT(D1,'.',D2);
          IF N < 0 THEN N:=N-1;
          STR(N,P);
          IF N = 0 THEN P:='';
        END
      ELSE
        BEGIN
          Y:=TRUNC(X);
          IF Y = 0 THEN
            BEGIN
             STR(ROUND(ABS(X*100.0)),D);
             IF X >= 0.0 THEN D:=CONCAT('.',D)
             ELSE D:=CONCAT('-.',D);
             P:='';
            END
          ELSE
            BEGIN
             IF Y < 10.0 THEN
               BEGIN
                 STR(TRUNC(X),D);
                 M:=ROUND(ABS(X-TRUNC(X))*10.0);
                 STR(M,P);
                 D:=CONCAT(D,'.',P);
                 P:='';
               END
             ELSE
               BEGIN
                 STR(TRUNC(X),D);
                 D:=CONCAT(D,'.');
                 P:='';
               END;
            END;
        END;
    END
  ELSE
    BEGIN
      D:='0.0';
      P:='';
    END;
END;


BEGIN(*wreal*)
  XPCHSIZE:=XCHSIZE-0.5;
  RSTR(X);
  IF P <> '' THEN D:=CONCAT(D,'x10');
  LENG := ROUND(LENGTH(D)*XCHSIZE*8 + LENGTH(P)*XPCHSIZE*8);
  YLDIST:=LENG;
  HEIGHT:=ROUND((YCHSIZE*2-1)*8);
  IF K = 0 THEN MOVETO((XCO-LENG DIV 2),(YCO-HEIGHT-8))
  ELSE MOVETO((XCO-LENG),(YCO-HEIGHT DIV 2));
  CHARSCALE(XCHSIZE,YCHSIZE);
  PLOTSTR(D);
  IF K = 0 THEN
    MOVETO((XCO+LENG DIV 2 - LENGTH(P)*ROUND(YCHSIZE-0.5)*8),
            (YCO-ROUND(YCHSIZE*8)))
  ELSE
    MOVETO((XCO-ROUND(LENGTH(P)*XPCHSIZE*8)),
            (YCO+ROUND(YCHSIZE*4)));
  CHARSCALE((XCHSIZE-0.5),(YCHSIZE-0.5));
  PLOTSTR(P);
  CHARSCALE(XCHSIZE,YCHSIZE);
END;(*WREAL*)


PROCEDURE FINDMM;

BEGIN
  I:= 1;
  XMAX:= TABLE[XFACT]©.DATA[I];
  I:= 1;
  YMAX:= TABLE[YFACT]©.DATA[I];
  XMIN:= XMAX; YMIN:= YMAX;
  FOR I:= 1 TO TABLE[XFACT]©.POINTS DO
    BEGIN
      IF (TABLE[XFACT]©.DATA[I] > XMAX) THEN
        XMAX:= TABLE[XFACT]©.DATA[I];
      IF (TABLE[XFACT]©.DATA[I] < XMIN) THEN
        XMIN:= TABLE[XFACT]©.DATA[I];
    END;
  FOR I:= 1 TO TABLE[YFACT]©.POINTS DO
    BEGIN
      IF (TABLE[YFACT]©.DATA[I] > YMAX) THEN
        YMAX:= TABLE[YFACT]©.DATA[I];
      IF TABLE[YFACT]©.DATA[I] < YMIN THEN
        YMIN:= TABLE[YFACT]©.DATA[I];
    END;
END; §minmax†


PROCEDURE SCALEIT;

BEGIN
  XTEMP:= ABS(XMAX-XMIN);
```

```
YTEMP:= ABS(YMAX-YMIN);
XSCALE:= 1.0; YSCALE:= 1.0;
IF (XTEMP>1.0) AND (XTEMP<=10.0) THEN
  XSCALE:= PWROFTEN(TRUNC(LOG(XMAX)))
ELSE WHILE (XTEMP > 10.0) OR (XTEMP <= 1.0) DO
  IF XTEMP > 10.0 THEN
    BEGIN
      XTEMP:= XTEMP / 10.0;
      XSCALE:= XSCALE * 10.0;
    END
  ELSE
    BEGIN
      XTEMP:= XTEMP * 10.0;
      XSCALE:= XSCALE / 10.0;
    END;
IF (YTEMP>1.0) AND (YTEMP<=10.0) THEN
  YSCALE:= PWROFTEN(TRUNC(LOG(YMAX)))
ELSE WHILE (YTEMP > 10.0) OR (YTEMP <= 1.0) DO
  IF YTEMP > 10.0 THEN
    BEGIN
      YTEMP:= YTEMP / 10.0;
      YSCALE:= YSCALE * 10.0;
    END
  ELSE
    BEGIN
      YTEMP:= YTEMP * 10.0;
      YSCALE:= YSCALE / 10.0;
    END;
IF XMIN>=0.0 THEN XLO:= TRUNC(XMIN / XSCALE)
ELSE XLO:= TRUNC(XMIN / XSCALE) - 1;
IF YMIN>=0.0 THEN YLO:= TRUNC(YMIN / YSCALE)
ELSE YLO:= TRUNC(YMIN / YSCALE) - 1;
IF XMAX <= 30000.0 THEN
  BEGIN
    IF XMAX - TRUNC(XMAX) > 0.0 THEN
       XHI:= TRUNC(XMAX / XSCALE) + 1
    ELSE XHI:= TRUNC(XMAX / XSCALE);
  END
ELSE XHI:=TRUNC(XMAX / XSCALE);
IF YMAX <= 30000.0 THEN
  BEGIN
    IF YMAX - TRUNC(YMAX) > 0.0 THEN
       YHI:= TRUNC(YMAX / YSCALE) + 1
    ELSE YHI:= TRUNC(YMAX / YSCALE);
  END
ELSE YHI:= TRUNC(YMAX / YSCALE);
IF XHI-XLO = 1 THEN XTICS:= 20;
IF (XHI-XLO > 1) AND (XHI-XLO < 5) THEN XTICS:= 10;
IF (XHI-XLO >= 5) AND (XHI-XLO < 10) THEN XTICS:= 5;
```

```
   IF XHI-XLO >= 10 THEN XTICS:= 2;
   IF YHI-YLO = 1 THEN YTICS:= 20;
   IF (YHI-YLO > 1) AND (YHI-YLO < 5) THEN YTICS:= 10;
   IF (YHI-YLO >= 5) AND (YHI-YLO < 10) THEN YTICS:= 5;
   IF YHI-YLO >= 10 THEN YTICS:= 2;
   RXLO:=XLO*XSCALE;
   RXHI:=XHI*XSCALE;
   RYLO:=YLO*YSCALE;
   RYHI:=YHI*YSCALE;
   XDIVISION:=XHI-XLO;
   YDIVISION:=YHI-YLO;
END; §setscale†


PROCEDURE SETPARAMETERS;

BEGIN
  WRITE (HOME,EOSCLEAR);
  FINDMM;
  SCALEIT;
  WRITELN ('Set plot parameters');
  WRITELN;
  WRITELN('Minimum X value = ',XMIN:6:4);
  WRITELN('Maximum X value = ',XMAX:6:4);
  WRITELN('Minimum Y value = ',YMIN:6:4);
  WRITELN('Maximum Y value = ',YMAX:6:4);
  Y:= 9;
  FOR I:= 1 TO 8 DO
    BEGIN
      GOTOXY (0,Y+I);
      WRITE (I,'.',PARAMS[I]);
      GOTOXY (32,Y+I);
      CASE I OF
         1:  WRITE (RXLO:6:4);
         2:  WRITE (RXHI:6:4);
         3:  WRITE (RYLO:6:4);
         4:  WRITE (RYHI:6:4);
         5:  WRITE (XDIVISION);
         6:  WRITE (YDIVISION);
         7:  WRITE (XTICS);
         8:  WRITE (YTICS);
      END; §case†
    END;
  GOTOXY (0,20);
  WRITELN ('Enter <0>  when parameters are correct');
  WRITELN ('      <10> to exit');
  REPEAT
    GOTOXY (0,8);
    WRITE (EOLNCLEAR);
    READLN (I);
    REPEAT
      GOTOXY (32,Y+I);
      WRITE (EOLNCLEAR);
      CASE I OF
         1:  READLN (RXLO);
         2:  READLN (RXHI);
         3:  READLN (RYLO);
         4:  READLN (RYHI);
         5:  READLN (XDIVISION);
         6:  READLN (YDIVISION);
         7:  READLN (XTICS);
         8:  READLN (YTICS);
        10:  EXIT (PLOTIT);
      END; §case†
    UNTIL NOT BADIO;
  UNTIL I=0;
END; §setparams†


PROCEDURE GETDATACHAR;

BEGIN
  Repeat
    WRITE (HOME,EOSCLEAR);
    GOTOXY (0,2);
    WRITE('Change character size? (Y/N)');
    READ(KEYBOARD,CH);
  Until sure;
  IF (CH='Y') OR (CH = 'y') THEN
    BEGIN
      Repeat
        REPEAT
          GOTOXY(0,4);
          §I$-†
          WRITE('Input X character size -->');
          READLN(XCHSIZE);
          WRITE('Input X character size -->');
          READLN(YCHSIZE);
          §$I+†
        UNTIL ((XCHSIZE > 0) AND (XCHSIZE < 6)
          AND (YCHSIZE > 0) AND (YCHSIZE < 6));
      Until not BADIO;
    END;
  Repeat
    Repeat
      GOTOXY (0,6);
      WRITELN('Character #128 for a continuous line');
      WRITELN;
      WRITE ('Enter data point character # ');
      §$I-†READLN(CHNUMB);§$I+†
    Until sure;
  Until not BADIO;
  IF CHNUMB <> 128 THEN
    BEGIN
      IF (CHNUMB >127) OR (CHNUMB < 0) THEN POINT := CHR(0)
      ELSE POINT := CHR (CHNUMB);
    END;
  IF CHNUMB <> 128 THEN
    BEGIN
      Repeat
        GOTOXY (0,10);
        WRITE (EOLNCLEAR,'Connect points? (Y/N or A)bort)');
        READ (KEYBOARD, CH);
      Until sure;
      IF (CH = 'A') OR (CH = 'a') THEN
        BEGIN
          PLOTLIM(LGRID,RGRID,TGRID,BGRID);
          MOVETO (0,0);
          EXIT (PLOTIT);
        END;
      IF (CH = 'Y') OR (CH = 'y') THEN CONNECT:= TRUE
      ELSE CONNECT:= FALSE;
    END
  ELSE CONNECT:=TRUE;
END; §selectmode†


PROCEDURE LABELY;

VAR MIDDLE : INTEGER;

BEGIN
  IF LENGTH (YLABEL) > 0 THEN §Label Y axis†
    BEGIN
      CHARSCALE(XCHSIZE,YCHSIZE);
      MIDDLE:=(BBOUND+((TBOUND-BBOUND) DIV 2)) -
              (LENGTH (YLABEL) * ROUND(8*YCHSIZE) DIV 2);
      MOVETO ((LBOUND-YLDIST-50),MIDDLE);
      ANGLE (90);
      PLOTSTR (YLABEL);
      ANGLE (0);
    END; §if†
END; §labely†


PROCEDURE LABELX;
```

```
VAR MIDDLE : INTEGER;

BEGIN
  IF LENGTH (XLABEL) > 0 THEN §Label X axis†
    BEGIN
      CHARSCALE(XCHSIZE,YCHSIZE);
      MIDDLE:=(LBOUND+((RBOUND-LBOUND) DIV 2)) -
              ((LENGTH (XLABEL) * ROUND(8*XCHSIZE) ) DIV 2);
      MOVETO (MIDDLE,(BBOUND-ROUND(35*YCHSIZE)));
      PLOTSTR (XLABEL);
    END; §if†
END; §labelx†


PROCEDURE LABELPLOT;

VAR MIDDLE : INTEGER;


BEGIN
   IF (LENGTH(PLABEL2)>0) AND (NOT MULTIPLOT) THEN
      BEGIN
        CHARSCALE(XCHSIZE,YCHSIZE);
        MIDDLE:=(LBOUND+((RBOUND-LBOUND) DIV 2)) -
                (LENGTH(PLABEL2)*ROUND(8*XCHSIZE) DIV 2);
        MOVETO (MIDDLE,(TBOUND+40));
        PLOTSTR (PLABEL2);
      END;
   IF (LENGTH(PLABEL1)>0) AND (NOT MULTIPLOT) THEN
      BEGIN
        CHARSCALE(XCHSIZE,YCHSIZE);
        MIDDLE:=(LBOUND+((RBOUND-LBOUND) DIV 2)) -
                (LENGTH(PLABEL1)*ROUND(8*XCHSIZE) DIV 2);
        MOVETO (MIDDLE,(TBOUND+ROUND(YCHSIZE*8)+80));
        PLOTSTR (PLABEL1);
      END;
END;


PROCEDURE DRAWBOX;

VAR J: INTEGER;
    XSTEP,YSTEP : INTEGER;


PROCEDURE FINDRIGHT;

BEGIN
  DX:=(RBOUND - LBOUND) DIV XDIVISION;
  X:=LBOUND;
  REPEAT
    IF (X+DX <= RBOUND) AND (XTICS > 0) THEN
      FOR J:=1 TO XTICS-1 DO XSTEP:=ROUND(X+J*(DX/XTICS));
      X:=X+DX;
  UNTIL X>RBOUND;
  RBOUND := X-DX;
END;


PROCEDURE FINDTOP;

BEGIN
  DY:=(TBOUND - BBOUND) DIV YDIVISION;
  Y:=BBOUND;
  REPEAT
    IF (Y+DY <= TBOUND) AND (YTICS > 0) THEN
      FOR J:=1 TO YTICS-1 DO YSTEP:=ROUND(Y+J*(DY/YTICS));
      Y:=Y+DY;
  UNTIL Y > TBOUND;
  TBOUND:=Y-DY;
END;


PROCEDURE DRAWBR;

BEGIN
   FINDRIGHT;
   FINDTOP;
   MOVETO (RBOUND,BBOUND); §Draw bottom boundary of box†
   PEN(DOWN);
   PLOTTO (LBOUND,BBOUND);
   X:= LBOUND; Y:= BBOUND + D;
   MOVETO (X,(Y+D));
   I:= 0;
   DX:= (RBOUND - LBOUND) DIV XDIVISION;
   REPEAT §Number X Axis†
     IF (I <> 0) AND (X < RBOUND) THEN
       BEGIN
         ANGLE (270);
         PEN(DOWN);
         PLOTTO (X,BBOUND);
       END;
     ANGLE (180);
     R:=RXLO+I*(RXHI-RXLO)/XDIVISION;
     WREAL(0,X,BBOUND-10,XCHSIZE-0.5,YCHSIZE-0.5,R);
     MOVETO (X,Y);
     ANGLE (270);
     IF (X+DX <= RBOUND) AND (XTICS>0) THEN
```

```
        FOR J:= 1 TO XTICS-1 DO
          BEGIN
            XSTEP:=ROUND(X+J*(DX/XTICS));
            MOVETO (XSTEP,Y);
            PEN(DOWN);
            PLOTTO (XSTEP,BBOUND);
          END;
      X:= X+DX;
      IF X <= RBOUND THEN MOVETO (X,(Y+D));
      I:= I+1;
    UNTIL X > RBOUND;
    LABELX;
    MOVETO (RBOUND,TBOUND); §Draw right boundary of the box†
    PEN(DOWN);
    PLOTTO (RBOUND,BBOUND);
    X:= RBOUND-D; Y:= BBOUND;     §Number Y Axis†
    MOVETO ((X-D),Y);
    I:=0;
    DY:= (TBOUND-BBOUND) DIV YDIVISION;
    REPEAT
      IF (I <> 0) AND (Y < TBOUND) THEN
        BEGIN
          ANGLE (180);
          PEN(DOWN);
          PLOTTO (RBOUND,Y);
        END;
      IF (Y+DY <= TBOUND) AND (YTICS>0) THEN
        FOR J:= 1 TO YTICS-1 DO
          BEGIN
            YSTEP:=ROUND(Y+J*(DY/YTICS));
            MOVETO (X,YSTEP);
            PEN(DOWN);
            PLOTTO (RBOUND,YSTEP);
          END;
      Y:= Y+DY;
      IF Y <= TBOUND THEN MOVETO ((X-D),Y);
      I:= I+1;
    UNTIL Y > TBOUND;
END;§DRAWBR†


PROCEDURE DRAWTL;

VAR MIDDLE : INTEGER;

BEGIN
  MOVETO (RBOUND,TBOUND); §Draw top boundary of box†
  PEN(DOWN);
  PLOTTO (LBOUND,TBOUND);
```

```
X:= LBOUND; Y:= TBOUND - D;
MOVETO (X,(Y-D));
I:=0;
REPEAT §Number X axis†
  IF (I <> 0) AND (X < RBOUND) THEN
    BEGIN
     ANGLE (270);
     PEN(DOWN);
     PLOTTO (X,TBOUND);
    END;
  IF (X+DX <= RBOUND) AND (XTICS>0) THEN
    FOR J:= 1 TO XTICS-1 DO
      BEGIN
        XSTEP:=ROUND(X+J*(DX/XTICS));
        MOVETO (XSTEP,Y);
        PEN(DOWN);
        PLOTTO (XSTEP,TBOUND);
      END;
  X:= X+DX;
  IF X <= RBOUND THEN MOVETO (X,(Y-D));
  I:=I+1;
UNTIL X > RBOUND;
MOVETO (LBOUND,TBOUND); §Draw left boundary of box†
PEN(DOWN);
PLOTTO (LBOUND,BBOUND);
X:= LBOUND+D; Y:= BBOUND;     §Number Y axis†
I:= 0;
MOVETO ((X+D),Y);
REPEAT
  IF (I <> 0) AND (Y < TBOUND) THEN
    BEGIN
      ANGLE (180);
      PEN(DOWN);
      PLOTTO (LBOUND,Y);
    END;
  ANGLE(0);
  R:=RYLO+I*(RYHI-RYLO)/YDIVISION;
  WREAL(1,(LBOUND-10),(Y+ROUND((YCHSIZE-0.5)*4)),
        XCHSIZE-0.5,YCHSIZE-0.5,R);
  ANGLE (180);
  IF (Y+DY <= TBOUND) AND (YTICS>0) THEN
    FOR J:= 1 TO YTICS-1 DO
      BEGIN
        YSTEP:=ROUND(Y+J*(DY/YTICS));
        MOVETO (X,YSTEP);
        PEN(DOWN);
        PLOTTO (LBOUND,YSTEP);
      END;
  Y:= Y+DY;
```

```
    IF Y <= TBOUND THEN MOVETO ((X+D),Y);
    I:= I+1;
  UNTIL Y > TBOUND;
  LABELY;
  ANGLE (0);
END;§DRAWTL†

BEGIN§drawaxes†
  WRITE (HOME,EOSCLEAR);
  WRITELN('<return> when HIPLOT ready...');
  Write  ('<ESC> to abort ...');
  READ (KEYBOARD, CH);
  If CH = chr(27) then exit(PLOTIT);
  PLOTLIM(LGRID,RGRID,TGRID,BGRID);
  INITEK;
  DRAWBR;
  DRAWTL;
  LABELPLOT;
END; §drawaxes†


FUNCTION SCALEXAXIS (X: REAL): INTEGER;

BEGIN
  SCALEXAXIS:= ROUND(LBOUND+((RBOUND-LBOUND)*((X-RXLO)/
                     (RXHI-RXLO))));
END;

FUNCTION SCALEYAXIS (Y: REAL): INTEGER;

BEGIN
  SCALEYAXIS:= ROUND(BBOUND+((TBOUND-BBOUND)*((Y-RYLO)/
                     (RYHI-RYLO))));
END;

PROCEDURE PDATA;

BEGIN
  I:= 1;
  PLOTLIM(LBOUND,RBOUND,TBOUND,BBOUND);
  X:= SCALEXAXIS (TABLE[XFACT]©.DATA[I]);
  Y:= SCALEYAXIS (TABLE[YFACT]©.DATA[I]);
  IF CHNUMB <> 128 THEN
    BEGIN
      MOVETO (X-ROUND(3*XCHSIZE),Y-ROUND(4*YCHSIZE));
      PLOTCHAR (POINT);
    END;
  MOVETO (X,Y);
  I:= I+1;
```

```
    WHILE (I <= TABLE[XFACT]©.POINTS)
      AND (I <= TABLE[YFACT]©.POINTS) DO
      BEGIN
        X:= SCALEXAXIS (TABLE[XFACT]©.DATA[I]);
        Y:= SCALEYAXIS (TABLE[YFACT]©.DATA[I]);
        IF CONNECT THEN
          BEGIN
            IF CHNUMB <> 128 THEN
              BEGIN
                XTEMP:= SQRT(SQR(1.0*(X-PENX))+SQR(1.0*(Y-PENY)));
                DX:= ROUND(12.0*(X-PENX)/XTEMP);
                IF DX = 0 THEN DY:= 12
                ELSE DY:= ROUND(DX*((Y-PENY)/(X-PENX)));
                MOVETO (PENX+DX,PENY+DY);
                PEN(DOWN);
                PLOTTO (X-DX,Y-DY);
              END
            ELSE
              BEGIN
                PEN(DOWN);
                PLOTTO(X,Y);
              END;
          END; §if connect†
        IF CHNUMB <> 128 THEN
          BEGIN
            MOVETO (X-9,Y-12);
            PLOTCHAR (POINT);
            MOVETO (X,Y);
          END;
        I:= I+1;
      END; §while†
    PLOTLIM(LGRID,RGRID,TGRID,BGRID);
    MOVETO (0,0);
    ANGLE (0);
  END; §grafit†

  PROCEDURE PLOTFIT;

  VAR
     DELTA, XVAL: REAL;

  BEGIN
    PLOTLIM(LBOUND,RBOUND,TBOUND,BBOUND);
    DELTA:= (XMAX-XMIN)/400.0;
    XVAL:= XMIN;
    MOVETO (SCALEXAXIS(XVAL),SCALEYAXIS(REGVALUE(XVAL)));
    XVAL:= XVAL+DELTA;
    REPEAT
      PEN(DOWN);
    PLOTTO (SCALEXAXIS(XVAL),SCALEYAXIS(REGVALUE(XVAL)));
    XVAL:= XVAL+DELTA;
  UNTIL XVAL > XMAX;
  PLOTLIM(LGRID,RGRID,TGRID,BGRID);
  MOVETO (0,0);
  ANGLE (0);
END;


PROCEDURE GETLEGENDS;

VAR
   CHNUMBER, XCO, YCO : INTEGER;
   X, Y : REAL;
   S : STRING;


Procedure Movepen;

BEGIN
  Repeat
    REPEAT
      GOTOXY(0,7);
      WRITE(EOSCLEAR,'Input X & Y coordinates -->');
      §$I-†READLN(X,Y);§$I+†
      XCO:=SCALEXAXIS(X);
      YCO:=SCALEYAXIS(Y);
    UNTIL ((XCO >= LBOUND) AND (XCO <= RBOUND)
      AND  (YCO >= BBOUND) AND (YCO <= TBOUND));
  Until not BADIO;
END;


BEGIN
  REPEAT
    WRITE(HOME,EOSCLEAR);
    GOTOXY(0,0);
    WRITE('Legends: C)har size, N)umber, A)lphanumeric,');
    WRITE(' M)ove to, P)lot to, Q)uit.');
    READ(KEYBOARD,CH);
    CASE CH OF
       'C','c' : BEGIN
                   Repeat
                     REPEAT
                       GOTOXY(0,5);
                       §$I-†
                       WRITE('Input X character sizes : ');
                       READ(XCHSIZE);
                       WRITE('Input Y character sizes : ');
```

```
                    READ(YCHSIZE);
                    §$I+†
                  UNTIL ((XCHSIZE > 0) AND (XCHSIZE < 6)
                     AND (YCHSIZE > 0) AND (YCHSIZE < 6));
                Until not BADIO;
                CHARSCALE(XCHSIZE,YCHSIZE);
              END;
     'N','n' : BEGIN
                Repeat
                  REPEAT
                    GOTOXY(0,10);
                    WRITE(EOSCLEAR,'Input character #');
                    READLN(CHNUMBER);
                  UNTIL ((CHNUMBER > 0)
                     AND (CHNUMBER < 128));
                Until not BADIO;
                PLOTCHAR(CHR(CHNUMBER));
              END;
     'A','a' : BEGIN
                GOTOXY(0,10);
                WRITE(EOSCLEAR,'Input string -->');
                READLN(S);
                PLOTSTR(S);
              END;
     'M','m' : BEGIN
                Movepen;
                MOVETO(XCO,YCO);
              END;
     'P','p' : BEGIN
                Movepen;
                PEN(DOWN);
                PLOTTO(XCO,YCO);
              END;
    END;
  UNTIL CH IN ['Q','q'];
  WRITE(HOME,EOSCLEAR);
END;§GETLEGENDS†

BEGIN §plotit†
  WRITE (HOME,EOSCLEAR);
  WRITELN ('Graph ',XLABEL, ' Vs. ', YLABEL);
  PARAMS[1]:= 'Min X axis value';
  PARAMS[2]:= 'Max X axis value';
  PARAMS[3]:= 'Min Y axis value';
  PARAMS[4]:= 'Max Y axis value';
  PARAMS[5]:= 'X axis divisions';
  PARAMS[6]:= 'Y axis divisions';
  PARAMS[7]:= 'X axis sub-divisions';
  PARAMS[8]:= 'Y axis sub-divisions';
```

```
   IF NOT MULTIPLOT THEN
     BEGIN
       SETPARAMETERS;
       RINGIT;
       DRAWBOX;
     END;
   GETDATACHAR;
   PDATA;
   Repeat
     WRITE (HOME,EOSCLEAR);
     WRITE('Plot last regression? (Y/N)');
     READ(KEYBOARD,CH);
   Until sure;
   IF (CH = 'Y') OR (CH = 'y') THEN PLOTFIT;
   Repeat
     GOTOXY(0,3);
     WRITE (EOLNCLEAR,'Do you have any legends? (Y/N)');
     READ(KEYBOARD,CH);
   Until sure;
   IF (CH = 'Y') OR (CH = 'y') THEN GETLEGENDS;
   Repeat
     GOTOXY(0,6);
     WRITE(EOLNCLEAR,'More points on this same plot? (Y/N)');
     READ (KEYBOARD, CH);
   Until sure;
   IF (CH = 'Y') OR (CH = 'y') THEN MULTIPLOT:= TRUE
   ELSE MULTIPLOT:= FALSE;
 END; §plotit†
```

```
SEGMENT PROCEDURE GETDATFILE;

VAR
   J, I: INTEGER;
   BLANKREC: DATAREC;
   CH: CHAR;
   POINT: REAL;
   INFILE: TEXT;
   S, INNAME: STRING;
   F, COLS: INTEGER;
   DATAREAD: BOOLEAN;


PROCEDURE MINE;

BEGIN
  WRITELN; WRITELN;
  WRITELN ('             written by');
  WRITELN; WRITELN;
  WRITELN ('        Bahram Nassersharif');
  WRITELN ('  Department of Nuclear Engineering');
  WRITELN ('       Oregon State University');
  WRITELN ('       Corvallis, Oregon 97331');
  WRITELN ('           (503) 754-2341');
END;


PROCEDURE GETIT;

BEGIN
  WRITE (HOME,EOSCLEAR);
  WRITELN;WRITELN;WRITELN;
  WRITELN ('Interactive Apple II & HI-PLOT plotter');
  WRITELN ('    plotting & regression routine ');
  WRITELN ('           VERSION 1.0');
  IF FIRSTTIME THEN MINE;
  GOTOXY (0,20);
  WRITELN ('Read data from');
  WRITE   ('which textfile : ');
  READLN (INNAME);
  IF POS('.TEXT',INNAME) = 0 THEN
     INNAME:= CONCAT (INNAME, '.TEXT');
END;


BEGIN§readfile†
  GETIT;
  §$I-†
  RESET (INFILE,INNAME); §$I+†
```

```
IF BADIO THEN EXIT (GETDATFILE);
WRITE (HOME,EOSCLEAR);
WRITELN ('Reading ',INNAME);
RELEASE (HEAP);
FACTORS:=0;
F:= 0;
WITH BLANKREC DO
  BEGIN
    POINTS:= 0;
    FOR I:= 0 TO MAXCASE DO DATA[I]:= 0.0;
  END;
FOR I:= 0 TO TABLELEN DO TABLE[I]:= NIL;
READLN(INFILE,PLABEL1);
WRITELN(PLABEL1);
READLN(INFILE,PLABEL2);
WRITELN(PLABEL2);
READLN(INFILE,XLABEL);
WRITELN(XLABEL);
READLN(INFILE,YLABEL);
WRITELN(YLABEL);
§$I-†
WHILE NOT EOF (INFILE) DO
  BEGIN
    F:= F + 1;
    IF INFILE© = ' ' THEN GET (INFILE);
    WHILE (INFILE© <= ' ') AND NOT EOLN (INFILE) DO
       GET (INFILE);
    DATAREAD:= FALSE;
    WHILE NOT EOLN (INFILE) DO
      §read a line of text†
      BEGIN
        S:= '';
        CH:= ' ';
        IF TABLE[F] = NIL THEN
          BEGIN
            NEW (LINK);
            TABLE[F]:= LINK;
            TABLE[F]©:= BLANKREC;
          END;
        WHILE (INFILE© <= ' ') AND NOT EOLN (INFILE) DO
           GET (INFILE);
        CH:= INFILE©;
        IF NOT EOLN (INFILE) THEN
          BEGIN
            READ (INFILE, POINT);
            IF IORESULT = 0 THEN
              WITH TABLE[F]© DO
                BEGIN
                  POINTS:= POINTS + 1;
```

```
                    DATA[POINTS]:= POINT;
                    DATAREAD:= TRUE;
                    WRITE(DATA[POINTS]);
                  END;
              IF NOT EOLN (INFILE) THEN F:= F+1;
            END; §ift
        END; §while not eolnt
        WRITELN;
        IF DATAREAD THEN COLS:= F - FACTORS
        ELSE
          BEGIN
             FACTORS:= FACTORS + COLS;
             WRITELN;
          END;
        F:= FACTORS;
    END; §whilet
  CLOSE (INFILE);
  §$I+t
  MAXPTS:= 0;
  FOR I:= 1 TO FACTORS DO
    IF TABLE[I]©.POINTS > MAXPTS THEN
       MAXPTS:= TABLE[I]©.POINTS;
  WRITE(HOME,EOSCLEAR);
  GOTOXY(0,5);
  WRITELN (FACTORS, ' Factors read  ');
  WRITELN ('Maximum factor length = ', MAXPTS);
  WRITELN;
  WRITELN;
  WRITELN ('Memory available =',MEMAVAIL);
END; §getdatfilet
```

```
SEGMENT PROCEDURE REGRESS;

VAR
   I: INTEGER;
   CH, CH2: CHAR;
   S: STRING;


PROCEDURE FITVALUE;

VAR S: STRING;

BEGIN
  IF LENGTH(YLABEL) <= 10 THEN
     WRITELN ('Estimate values of ',YLABEL,'?')
  ELSE WRITELN ('Estimate values of Y?');
  WRITELN ('<return> to terminate');
  WRITELN;
  REPEAT
    IF LENGTH(XLABEL) <= 10 THEN WRITE (XLABEL, ' = ')
    ELSE WRITE ('X = ');
    READLN (S);
    IF LENGTH (S) > 0 THEN
      BEGIN
        IF LENGTH(YLABEL) <= 10 THEN
           WRITE ('Estimated ',YLABEL,' = ')
        ELSE WRITE ('Estimated Y = ');
        WRITELN (REGVALUE(REALVAL(S)));
      END;
  UNTIL LENGTH (S) = 0;
END; §estimate†


PROCEDURE REGIT;

VAR
   N, I, POINTS: INTEGER;
   SUMX,SUMY,SUMXSQ,SUMYSQ,SUMXY,RSQ,SESQ,X,Y: REAL;


PROCEDURE COEFFS;

BEGIN
  IF N = 0 THEN
    BEGIN
      A:=0; B:=0;
      WRITELN('FACTORS CONTAIN ZERO VALID DATA POINTS.....EXIT!');
      EXIT (COEFFS);
    END;
   B:= (N*SUMXY - SUMY*SUMX) / (N*SUMXSQ - SQR(SUMX));
   A:= (SUMY - B*SUMX) / N;
   IF (REG = EXPO) OR (REG = PWR) THEN A:= EXP(A);
   RSQ:= (B * (SUMXY - SUMX*SUMY/N))  / (SUMYSQ - SQR(SUMY)/N);
   SESQ:= ((SUMYSQ - SQR(SUMY)/N) - (B * (SUMXY-SUMX*SUMY/N)));
   WRITELN(OUTFILE,'A = ',A);
   WRITELN(OUTFILE,'B = ',B);
   WRITELN(OUTFILE,'N = ',N);
   WRITELN(OUTFILE);
   IF (N > 2) AND (SESQ > 0) THEN
   WRITELN(OUTFILE,'Stand error of esitmate = ',SQRT(SESQ/(N-2)));
   WRITELN(OUTFILE,'Coeff of corr (R)       = ',SQRT(RSQ));
   WRITELN(OUTFILE,'Coeff of deter (R**2)   = ',RSQ);
   WRITELN(OUTFILE);
END; §calculate†


PROCEDURE LINEAR;

BEGIN
  WRITE (HOME,EOSCLEAR);
  WRITELN (OUTFILE, 'Linear Regression');
  WRITELN (OUTFILE);
  SUMX:= 0; SUMY:=0; SUMXSQ:=0; SUMYSQ:=0; SUMXY:=0; N:=0;
  IF TABLE[XFACT]©.POINTS < TABLE[YFACT]©.POINTS THEN
     POINTS:= TABLE[XFACT]©.POINTS
  ELSE POINTS:= TABLE[YFACT]©.POINTS;
  FOR I:= 1 TO POINTS DO
    BEGIN
      X:= TABLE[XFACT]©.DATA[I];
      Y:= TABLE[YFACT]©.DATA[I];
      IF (X <> MISSING) AND (Y <> MISSING) THEN
        BEGIN
          SUMX:= SUMX + X;
          SUMY:= SUMY + Y;
          SUMXSQ:= SUMXSQ + SQR(X);
          SUMYSQ:= SUMYSQ + SQR(Y);
          SUMXY:= SUMXY + X*Y;
          N:= N + 1;
        END;
    END; §for†
  COEFFS;
  IF LENGTH(YLABEL) <= 10 THEN WRITE (OUTFILE,YLABEL,' = ')
  ELSE WRITE (OUTFILE,'Y = ');
  IF ABS(A) < 1 THEN WRITE (OUTFILE,A)
  ELSE WRITE (OUTFILE, A:6:3);
  IF B > 0 THEN WRITE (OUTFILE,' + ');
  IF ABS(B) < 1 THEN WRITE (OUTFILE,B)
  ELSE WRITE (OUTFILE, B:6:3);
  IF LENGTH(XLABEL) <= 10 THEN
     WRITELN (OUTFILE, '(', XLABEL, ')')
  ELSE WRITELN (OUTFILE, '(X)');
  WRITELN (OUTFILE);
  IF NOT PRINTOUT THEN FITVALUE;
END;§lineart


PROCEDURE LOGARITHMIC;

BEGIN
  WRITE (HOME,EOSCLEAR);
  WRITELN (OUTFILE, 'Logarithmic Regression');
  WRITELN (OUTFILE);
  SUMX:= 0; SUMY:=0; SUMXSQ:=0; SUMYSQ:=0; SUMXY:=0; N:=0;
  IF TABLE[XFACT]©.POINTS < TABLE[YFACT]©.POINTS THEN
     POINTS:= TABLE[XFACT]©.POINTS
  ELSE POINTS:= TABLE[YFACT]©.POINTS;
  FOR I:= 1 TO POINTS DO
    BEGIN
       X:= TABLE[XFACT]©.DATA[I];
       Y:= TABLE[YFACT]©.DATA[I];
       IF (X <> MISSING) AND (Y <> MISSING) THEN
         BEGIN
           IF X < 0 THEN
             BEGIN
               WRITELN ('Negative argument for LOG');
               EXIT (LOGARITHMIC);
             END;
           IF X = 0 THEN X:= 1
           ELSE X:= LN(X);
           SUMX:= SUMX + X;
           SUMY:= SUMY + Y;
           SUMXSQ:= SUMXSQ + SQR(X);
           SUMYSQ:= SUMYSQ + SQR(Y);
           SUMXY:= SUMXY + X*Y;
           N:= N + 1;
         END;
    END; §fort
  COEFFS;
  IF LENGTH(YLABEL) <= 10 THEN
     WRITE (OUTFILE,YLABEL,' = ')
  ELSE WRITE (OUTFILE,'Y = ');
  WRITE (OUTFILE, A:6:3);
  IF B > 0 THEN WRITE (' + ');
  WRITE (OUTFILE, B:6:3);
  IF LENGTH(XLABEL) <= 10 THEN
     WRITELN (OUTFILE, '(Ln (', XLABEL, ') )')
  ELSE WRITELN (OUTFILE, '(Ln(X))');
  WRITELN (OUTFILE);
  IF NOT PRINTOUT THEN FITVALUE;
END; §logarithmict


PROCEDURE EXPONENTIAL;

BEGIN
  WRITE (HOME,EOSCLEAR);
  WRITELN (OUTFILE, 'Exponential Regression');
  WRITELN (OUTFILE);
  SUMX:= 0; SUMY:=0; SUMXSQ:=0; SUMYSQ:=0; SUMXY:=0; N:=0;
  IF TABLE[XFACT]©.POINTS < TABLE[YFACT]©.POINTS THEN
     POINTS:= TABLE[XFACT]©.POINTS
  ELSE POINTS:= TABLE[YFACT]©.POINTS;
  FOR I:= 1 TO POINTS DO
    BEGIN
      X:= TABLE[XFACT]©.DATA[I];
      Y:= TABLE[YFACT]©.DATA[I];
      IF (X <> MISSING) AND (Y <> MISSING) THEN
        BEGIN
          SUMX:= SUMX + X;
          IF Y < 0 THEN
            BEGIN
              WRITELN ('Negative argument for LOG');
              EXIT (EXPONENTIAL);
            END;
          IF Y = 0 THEN Y:= 1
          ELSE Y:= LN(Y);
          SUMY:= SUMY + Y;
          SUMXSQ:= SUMXSQ + SQR(X);
          SUMYSQ:= SUMYSQ + SQR(Y);
          SUMXY:= SUMXY + X*Y;
          N:= N + 1;
        END;
    END; §fort
  COEFFS;
  IF LENGTH(YLABEL) <= 10 THEN WRITE (OUTFILE,YLABEL,' = ')
  ELSE WRITE (OUTFILE,'Y = ');
  WRITE (OUTFILE, A);
  WRITE (OUTFILE, ' Exp(');
  WRITE (OUTFILE, B);
  IF LENGTH(XLABEL) <= 10 THEN
     WRITELN (OUTFILE, ' ',XLABEL, ')')
  ELSE WRITELN (OUTFILE, ' X)');
  WRITELN (OUTFILE);
  IF NOT PRINTOUT THEN FITVALUE;
END; §exponentialt
```

```
PROCEDURE POWER;

BEGIN
  WRITE (HOME,EOSCLEAR);
  WRITELN (OUTFILE, 'Power Regression');
  WRITELN (OUTFILE);
  SUMX:= 0; SUMY:=0; SUMXSQ:=0; SUMYSQ:=0; SUMXY:=0; N:=0;
  IF TABLE[XFACT]©.POINTS < TABLE[YFACT]©.POINTS THEN
     POINTS:= TABLE[XFACT]©.POINTS
  ELSE POINTS:= TABLE[YFACT]©.POINTS;
  FOR I:= 1 TO POINTS DO
    BEGIN
      X:= TABLE[XFACT]©.DATA[I];
      Y:= TABLE[YFACT]©.DATA[I];
      IF (X<0) OR (Y<0) THEN
        BEGIN
          WRITELN (OUTFILE, 'Negative argument for LOG');
          EXIT(POWER);
        END;
      IF (X <> MISSING) AND (Y <> MISSING) THEN
        BEGIN
          IF X=0 THEN X:= 1
          ELSE X:= LN(X);
          IF Y=0 THEN Y:= 1
          ELSE Y:= LN(Y);
          SUMX:= SUMX + X;
          SUMY:= SUMY + Y;
          SUMXSQ:= SUMXSQ + SQR(X);
          SUMYSQ:= SUMYSQ + SQR(Y);
          SUMXY:= SUMXY + X*Y;
          N:= N + 1;
        END;
    END; §fort
  COEFFS;
  IF LENGTH(YLABEL) <= 10 THEN WRITE (OUTFILE,YLABEL,' = ')
  ELSE WRITE (OUTFILE,'Y = ');
  WRITE (OUTFILE, A);
  IF LENGTH(XLABEL) <= 10 THEN
     WRITE (OUTFILE, ' (', XLABEL, ') Power ')
  ELSE WRITE (OUTFILE, ' (X) Power ');
  WRITELN (OUTFILE, B);
  WRITELN (OUTFILE);
  IF NOT PRINTOUT THEN FITVALUE;
END; §powert


BEGIN §doregst
  CASE REG OF
```

```
    LIN: LINEAR;
    LOGARM: LOGARITHMIS;
    EXPO: EXPONENTIAL;
    PWR: POWER;
  END;
END; §doregst


PROCEDURE POLYNOMIAL;

VAR
   DEGREE, POINTS, I, J, K, N: INTEGER;
   A: ARRAY [0..21] OF REAL;
   R: ARRAY [0..11,0..12] OF REAL;
   T: ARRAY [0..12] OF REAL;
   X, Y, Z: REAL;
   S: STRING;


FUNCTION INTPOWER (X: REAL; N: INTEGER): REAL;

VAR
  I: INTEGER;
  Y: REAL;

BEGIN
  IF N = 0 THEN Y:= 1.0
  ELSE
    BEGIN
      Y:= X;
      IF N > 1 THEN FOR I:= 1 TO N - 1 DO Y:= X * Y;
    END;
  INTPOWER:= Y;
END; §intpowert


PROCEDURE ZIPEM;

BEGIN
  FOR I:= 0 TO 21 DO A[I]:= 0;
  FOR I:= 0 TO 12 DO
    BEGIN
      T[I]:= 0;
      FOR J:= 0 TO 11 DO R[J,I]:= 0;
    END;
  FOR I:= 0 TO 10 DO POLYCOEF[I]:= 0;
END; §cleararrayst
```

```
PROCEDURE POLYFIT;

BEGIN
  FOR J:= 1 TO DEGREE +1 DO
    FOR I:= 1 TO DEGREE + 1 DO
      R[J,I]:= A[J+I-1];
  FOR J:= 1 TO DEGREE + 1 DO
    BEGIN
      K:= J;
      REPEAT
        Z:= R[K,J];
        IF Z = 0 THEN K:= K + 1;
      UNTIL (K > DEGREE + 1) OR (Z <> 0);
      IF K > DEGREE +1 THEN
        BEGIN
          WRITELN ('No unique solution!');
          EXIT (POLYNOMIAL);
        END;
      FOR I:= 1 TO DEGREE + 2 DO
        BEGIN
          Z:= R[J,I];
          R[J,I]:= R[K,I];
          R[K,I]:= Z;
        END;
      Z:= 1 / R[J,J];
      FOR I:= 1 TO DEGREE + 2 DO R[J,I]:= Z * R[J,I];
      FOR K:= 1 TO DEGREE + 1 DO
        IF K <> J THEN
          BEGIN
            Z:= -R[K,J];
            FOR I:= 1 TO DEGREE + 2 DO
              R[K,I]:= R[K,I] + Z * R[J,I];
          END;
    END; §fort
  FOR I:= 0 TO DEGREE DO POLYCOEF[I]:= R[I+1,DEGREE+2];
  WRITELN (OUTFILE, '----- Polynomial Coefficients -----');
  WRITELN (OUTFILE);
  WRITELN (OUTFILE, 'Degree', 'Coefficient':21);
  WRITELN (OUTFILE, '------', '-----------':21);
  FOR I:= 0 TO DEGREE DO
     WRITELN (OUTFILE, I:2, POLYCOEF[I]:24);
  WRITELN (OUTFILE);
END; §polyfitt


BEGIN §polynomialt
  WRITE (HOME,EOSCLEAR);
  REG:= POLY;
  WRITELN (OUTFILE, 'Polynomial Regression');
```

```
IF LENGTH(YLABEL) <= 10 THEN
  WRITE(OUTFILE,YLABEL,' = C(0) + SUM§ C(N) * (')
ELSE WRITE (OUTFILE,'Y = C(0) + SUM§ C(N) * (');
IF LENGTH(XLABEL) <= 10 THEN
   WRITELN (OUTFILE,XLABEL,')**N †')
ELSE WRITELN (OUTFILE,'X) to the power N †');
WRITELN;
ZIPEM;
REPEAT
  WRITE ('Degree of polynomial ?');
  READLN (S);
  IF LENGTH (S) = 0 THEN EXIT (POLYNOMIAL);
  DEGREE:= TRUNC(REALVAL(S));
UNTIL (DEGREE > 0) AND (DEGREE <=10);
WRITELN;
IF TABLE[XFACT]©.POINTS < TABLE[YFACT]©.POINTS THEN
  POINTS:= TABLE[XFACT]©.POINTS
ELSE POINTS:= TABLE[YFACT]©.POINTS;
N:= 0;
FOR I:= 1 TO POINTS DO
  BEGIN
    X:= TABLE[XFACT]©.DATA[I];
    Y:= TABLE[YFACT]©.DATA[I];
    IF NOT ((X = MISSING) OR (Y = MISSING)) THEN
      BEGIN
        FOR J:= 2 TO 2 * DEGREE +1 DO
           A[J]:= A[J] + INTPOWER(X, J-1);
        FOR J:= 1 TO DEGREE + 1 DO
          BEGIN
            R[J,DEGREE+2]:= T[J] + Y*(INTPOWER(X,J-1));
            T[J]:= T[J] + Y*(INTPOWER(X,J-1));
          END;
        T[DEGREE+2]:= T[DEGREE+2] + Y*Y;
        N:= N + 1;
      END;
  END;
A[1]:= N;
POLYFIT;
X:= 0;
FOR J:= 2 TO DEGREE + 1 DO
  X:= X + R[J,DEGREE+2] * (T[J] - A[J] * T[1]/N);
Y:= X / (T[DEGREE+2] - T[1]*T[1] / N);
IF (N - DEGREE - 1) > 0 THEN
  BEGIN
    Z:= ((T[DEGREE+2] - T[1]*T[1]/N) - X) / (N - DEGREE -1);
    WRITELN (OUTFILE, 'Stand error of estimate = ', Z);
  END;
WRITELN (OUTFILE, 'Coeff of corr (R)       = ', SQRT(Y));
WRITELN (OUTFILE, 'Coeff of deter (R**2)   = ', Y);
```

```
   WRITELN (OUTFILE);
   IF NOT PRINTOUT THEN FITVALUE;
 END; §polynomial†


 BEGIN §Main Program†
   WRITE (HOME,EOSCLEAR);
   REPEAT
     GOTOXY (0,21);
     IF PRINTOUT THEN WRITE (EOLNCLEAR,'Printer On ')
     ELSE WRITE (EOLNCLEAR,'Printer Off');
     WRITELN (EOLNCLEAR,FACTORS:4, ' Factors');
     IF FACTORS > 0 THEN
       BEGIN
         WRITELN(EOLNCLEAR,'X = ', XLABEL);
         WRITE(EOLNCLEAR,'Y = ', YLABEL);
       END;
     GOTOXY (0,0);
     WRITE (EOLNCLEAR,'Regress: Li)n, Lo)g, E)xp, Pw)r,');
     WRITE (' Po)ly, Pr)int, La)bel, Q)uit.');
     READ (KEYBOARD, CH);
     CASE CH OF
       'P','p': BEGIN
                  READ (KEYBOARD, CH2);
                  CASE CH2 OF
                    'O','o':  POLYNOMIAL;
                    'W','w':  REG:= PWR;
                    'R','r':  PRINTONOFF;
                  END;
                  IF (CH2 = 'W') OR (CH2 = 'w') THEN REGIT;
                END;
       'L','l': BEGIN
                  READ (KEYBOARD, CH2);
                  CASE CH2 OF
                    'I','i':  REG:= LIN;
                    'O','o':  REG:= LOGARM;
                    'A','a':  LABELPLOT;
                  END;
                  IF (CH2 = 'I') OR (CH2 = 'O')
                  OR (CH2 = 'i') OR (CH2 = 'o') THEN REGIT;
                END;
       'E','e': BEGIN
                  REG:= EXPO;
                  REGIT;
                END;
       ' '    : WRITE(HOME,EOSCLEAR);
     END; §Cases†
   UNTIL CH IN ['Q','q'];
   §$I-†
```

```
  CLOSE(OUTFILE,LOCK);
  IF badio THEN exit(REGRESS);
  §$I+†
  WRITE(HOME,EOSCLEAR);
END;
```

```
§$S+†
UNIT PLOTSTUFF;

§HIPLOT Plotstuff unit†

INTERFACE

TYPE

   wordbit = RECORD CASE BOOLEAN OF
               true: (word: CHAR);
               false: (bit: PACKED ARRAY[0..15] OF BOOLEAN);
             END;

   PENSTATE = (UP,DOWN);

   PENCOORDS = PACKED RECORD
                   XVEC: 0..23;
                   YVEC: 0..23;
                   XNEG: BOOLEAN;
                   YNEG: BOOLEAN;
                   LASTPOINT: BOOLEAN;
                   PENDOWN: BOOLEAN;
               END;

   CHARACTER = ARRAY[0..20] OF PENCOORDS;

VAR
   CHXSCALE, CHYSCALE: REAL;
   bad: BOOLEAN;
   PENX, PENY, PENANGLE: INTEGER;
   CHARFILE: FILE OF CHARACTER;
   EOLNCLEAR, EOSCLEAR, HOME : PACKED ARRAY [0..1] OF CHAR;
   CH : CHAR;


FUNCTION BADIO : BOOLEAN;
PROCEDURE getcrtinfo;
PROCEDURE PEN (MODE: PENSTATE);
PROCEDURE PLOTTO (X,Y: INTEGER);
PROCEDURE MOVETO (X,Y: INTEGER);
PROCEDURE PLOT (DIST: INTEGER);
PROCEDURE MOVE (DIST: INTEGER);
PROCEDURE ANGLE (X: INTEGER);
PROCEDURE PLOTCHAR (CH: CHAR);
PROCEDURE PLOTSTR (S: STRING);
PROCEDURE PLOTLIM (L,R,T,B: INTEGER);
PROCEDURE PLOTSLOT (I: INTEGER);
PROCEDURE CHARSCALE (X,Y: REAL);
PROCEDURE INITEK;
PROCEDURE CURSORON;
PROCEDURE PAGE;
PROCEDURE RINGIT;

IMPLEMENTATION

CONST
   LBOUND = 0;
   RBOUND = 2000;
   TBOUND = 1400;
   BBOUND = 0;

VAR
   LEFTB, RIGHTB, TOPB, BOTTOMB: INTEGER;
   TABLE: ARRAY[0..127] OF CHARACTER;
   SYMBOL: CHARACTER;
   GS, BELL, FF : CHAR;
   I : INTEGER;
   ESC, SUB : CHAR;
   DELPEN, DELMOVE: INTEGER;


FUNCTION INITCARD (I: INTEGER): INTEGER;
   EXTERNAL;

PROCEDURE WPLOTTER (CH: CHAR);
   EXTERNAL;

PROCEDURE getcrtinfo;

TYPE
   syscomrec =  PACKED ARRAY[0..511] OF CHAR;
   syscomptr = ©syscomrec;

VAR
   syscom: syscomptr;
   trix: RECORD CASE BOOLEAN OF
            false: (address: INTEGER);
            true:  (pointer: ©syscomptr);
         END;
   prefixed: wordbit;
   leadin: CHAR;

BEGIN
   trix.address:= 248;
   syscom:= trix.pointer©;
   leadin:= syscom©[62];
   eosclear[1]:= syscom©[64];
   eolnclear[1]:= syscom©[65];
   home[1]:= syscom©[63];
   WITH prefixed DO
      BEGIN
         word:= syscom©[72];
         IF bit[6]
            THEN home[0]:= leadin
            ELSE home[0]:= chr(0);
         IF bit[3]
            THEN eosclear[0]:= leadin
            ELSE eosclear[0]:= chr(0);
         IF bit[2]
            THEN eolnclear[0]:= leadin
            ELSE eolnclear[0]:= chr(0);
      END; §witht
END; §getcrtinfot

FUNCTION BADIO;

VAR
   IOCODE: 0..16;

BEGIN
   IOCODE:= IORESULT;
   BADIO:= FALSE;
   IF IOCODE > 0 THEN
      BEGIN
         BADIO:= TRUE;
         GOTOXY (0,21);
         WRITE ('I/O ERROR:  ');
         CASE IOCODE OF
            1: WRITELN ('Bad Block, Parity Read Error');
            2: WRITELN ('Bad Unit Number');
            3: WRITELN ('Bad Mode, Illegal Operation');
            4: WRITELN ('Undefined Hardware Error');
            5: WRITELN ('Lost Unit, Unit is no longer on line');
            6: WRITELN ('Lost File, File is no longer in Directory');
            7: WRITELN ('Bad Title, Illegal File Name');
            8: WRITELN ('No Room on Disk, Insufficient space');
            9: WRITELN ('No Unit, No such volume on line');
            10: WRITELN ('No File, No such file on volume');
            11: WRITELN ('Duplicate File');
            12: WRITELN ('File not Closed, Attempt to open an open file');
            13: WRITELN ('Not Open, Attempt to access a closed file');
            14: WRITELN ('Bad Format, Error in reading real or integer');
            15: WRITELN ('Ring Buffer Overflow');
            16: WRITELN ('Write Protected');
         END; §Casest
         WRITELN;
         WRITE ('PRESS <SPACEBAR> TO CONTINUE');
         READ (KEYBOARD,CH);
         GOTOXY (0,21);
         WRITE (EOSCLEAR);
      END; §Ift
END; §Badiot

PROCEDURE PLOTTO;

VAR
   F,D,I,T,E,Z,DX,DY: INTEGER;
   A: STRING[16];

BEGIN
  IF X > RIGHTB THEN X:=RIGHTB;
  IF X < LEFTB  THEN X:=LEFTB;
  IF Y > TOPB   THEN Y:=TOPB;
  IF Y < BOTTOMB THEN Y:=BOTTOMB;
  DX:=X-PENX;
  DY:=Y-PENY;
  A:='pqrqrststuvuvwpw';
  F:=ABS(DX)+ABS(DY);
  IF F > 0 THEN
    BEGIN
      D:=ABS(DY)-ABS(DX);
      I:=0;
      IF DY >= 0 THEN I:=2;
      T:=DX+DY;
      IF T >= 0 THEN I:=I+2;
      T:=DY-DX;
      IF T >= 0 THEN I:=I+2;
      IF DX < 0 THEN I:=I+10
      ELSE I:=8-I;
      IF D < 0 THEN T:=ABS(DY)
      ELSE
        BEGIN
          T:=ABS(DX);
          D:=-D;
        END;
      E:=0;
      REPEAT
        Z:=T+D+E+E;
        IF Z < 0 THEN
          BEGIN
            E:=E+T;
            F:=F-1;
            WPLOTTER(A[I-1]);
          END
        ELSE
```

```
            BEGIN
              E:=E+D;
              F:=F-2;
              WPLOTTER(A[I]);
            END;
        UNTIL F <= 0;
        PENX:=X;
        PENY:=Y;
        FOR I:= 1 TO DELMOVE DO
          BEGIN
            (*DELAY*)
          END;
    END;
END; §plottot

PROCEDURE PEN;

VAR I: INTEGER;

BEGIN
   IF MODE = UP THEN WPLOTTER('y')
   ELSE WPLOTTER('z');
   FOR I:= 1 TO DELPEN DO
     BEGIN
       (*DELAY*)
     END;
END; §pent

PROCEDURE MOVETO;

BEGIN
  PEN(UP);
  PLOTTO (X,Y);
END;

FUNCTION SINE (ANGLE: INTEGER): REAL;

VAR
   RADS, RESULT, POWER, FACT, EPSILON: REAL;
   I, SIGN: INTEGER;

BEGIN
   RADS:= 3.14159265*ANGLE/180.0;
   POWER:= RADS * RADS * RADS;
   RESULT:= RADS;
   RADS:= RADS * RADS;
   FACT:= 6.0;
   SIGN:= -1;
   I:= 1;
```

```
      REPEAT
         EPSILON:= POWER/FACT;
         RESULT:= RESULT + SIGN * EPSILON;
         SIGN:= -SIGN;
         POWER:= POWER * RADS;
         FACT:= FACT * (2*I+2) * (2*I+3);
         I:= I+1;
      UNTIL ABS(EPSILON) < 1.0E-6;
   SINE:= RESULT;
END; §sine†

PROCEDURE PLOTSLOT;

BEGIN
   IF (I>0) AND (I<=7)
      THEN
         BEGIN
            IF INITCARD (I) <> 0 THEN
               WRITELN ('IO ERROR:  Tektronix terminal not ready');
         END
      ELSE
         WRITELN ('INVALID SLOT NUMBER:  Tektronix terminal not ready');
END; §plotslot†

PROCEDURE PLOTLIM;

BEGIN
   IF (R>L) AND (T>B) THEN
      BEGIN
         IF (L >= LBOUND) AND (L <= RBOUND) THEN
            LEFTB:= L;
         IF (R >= LBOUND) AND (R <= RBOUND) THEN
            RIGHTB:= R;
         IF (B >= BBOUND) AND (B <= TBOUND) THEN
            BOTTOMB:= B;
         IF (T >= BBOUND) AND (B <= TBOUND) THEN
            TOPB:= T;
      END;
END; §plotlim†

PROCEDURE PLOT;

VAR
   X,Y:INTEGER;

BEGIN
   X:= PENX + ROUND (DIST * SINE (90-PENANGLE));
   Y:= PENY + ROUND (DIST * SINE (PENANGLE));
   PLOTTO (X,Y);
```

```
END; §plott

PROCEDURE MOVE;

BEGIN
  PEN(UP);
  PLOT(DIST);
END;

PROCEDURE ANGLE;

BEGIN
   PENANGLE:= X;
   WHILE PENANGLE >= 360 DO PENANGLE:= PENANGLE-360;
   WHILE PENANGLE <= -360 DO PENANGLE:= PENANGLE+360;
END; §anglet

PROCEDURE CHARSCALE;

BEGIN
   IF X < 1.0 THEN X:= 1.0;
   IF X > 100.0 THEN X:= 100.0;
   CHXSCALE:= X;
   IF Y < 1.0 THEN Y:= 1.0;
   IF Y > 100.0 THEN Y:= 100.0;
   CHYSCALE:= Y;
END; §charscalet


PROCEDURE PLOTCHAR;

VAR
   I,X,Y: INTEGER;
   LINE: BOOLEAN;

BEGIN
   SYMBOL:= TABLE[ORD(CH)];
   I:= 0;
   LINE:= FALSE;
   REPEAT
      IF SYMBOL[I].PENDOWN <> LINE THEN
         BEGIN
            IF SYMBOL[I].PENDOWN
               THEN
                  BEGIN
                     PEN (DOWN);
                     LINE:= TRUE;
                  END
               ELSE
```

```
                  BEGIN
                     PEN (UP);
                     LINE:= FALSE;
                  END;
            END;
         IF PENANGLE = 90
            THEN
               BEGIN
                  IF SYMBOL[I].YNEG
                     THEN X:= PENX + ROUND(SYMBOL[I].YVEC*CHYSCALE)
                     ELSE X:= PENX - ROUND(SYMBOL[I].YVEC*CHYSCALE);
                  IF SYMBOL[I].XNEG
                     THEN Y:= PENY - ROUND(SYMBOL[I].XVEC*CHXSCALE)
                     ELSE Y:= PENY + ROUND(SYMBOL[I].XVEC*CHXSCALE);
               END
            ELSE
               BEGIN
                  IF SYMBOL[I].XNEG
                     THEN X:= PENX - ROUND(SYMBOL[I].XVEC*CHXSCALE)
                     ELSE X:= PENX + ROUND(SYMBOL[I].XVEC*CHXSCALE);
                  IF SYMBOL[I].YNEG
                     THEN Y:= PENY - ROUND(SYMBOL[I].YVEC*CHYSCALE)
                     ELSE Y:= PENY + ROUND(SYMBOL[I].YVEC*CHYSCALE);
               END;
         PLOTTO (X,Y);
         I:= I+1;
   UNTIL SYMBOL[I-1].LASTPOINT;
END; §plotchart

PROCEDURE PLOTSTR;

VAR I: INTEGER;

BEGIN
   IF LENGTH (S) > 0 THEN
      FOR I:= 1 TO LENGTH (S) DO
         PLOTCHAR (S[I]);
END; §plotstrt

PROCEDURE READCHRSET;

VAR I: INTEGER;

BEGIN
   §$I-t
   RESET (CHARFILE,'#4:CHRSET');
   IF IORESULT<>0 THEN
      BEGIN
         RESET (CHARFILE,'#5:CHRSET');
```

```
          IF IORESULT<>0 THEN
             BEGIN
                GOTOXY (0,5);
                WRITELN ('CHARACTER SET NOT AVAILABLE....ABORT');
                EXIT (PROGRAM);
             END;
      END; §$I+†
   FOR I:= 0 TO 127 DO
      BEGIN
         TABLE[I]:= CHARFILE©;
         GET (CHARFILE);
      END;
   CLOSE (CHARFILE);
END; §readchrset†

PROCEDURE INITEK;

BEGIN
  MOVETO(0,0);
END;

PROCEDURE CURSORON;

BEGIN
END;

PROCEDURE PAGE;

BEGIN
  MOVETO(0,0);
END;

PROCEDURE RINGIT;

BEGIN
END;

BEGIN
   DELPEN:=300;
   DELMOVE:=40;
   PLOTSLOT(2);
   BELL:=CHR(7);
   FF:=CHR(12);
   GS:=CHR(29);
   SUB:=CHR(26);
   ESC:=CHR(27);
   PENX:= 0;
   PENY:= 0;
   PENANGLE:= 0;
   PLOTLIM (LBOUND,RBOUND,TBOUND,BBOUND);
   CHARSCALE (3,3);
   READCHRSET;
END.
```

```
PROGRAM CHARGEN;

USES TURTLEGRAPHICS;

CONST  MAXCOORDS = 20;

TYPE
   COORDS = PACKED RECORD
                 XCOORD: 0..23;
                 YCOORD: 0..23;
                 NEGXMOVE: BOOLEAN;
                 NEGYMOVE: BOOLEAN;
                 LAST: BOOLEAN;
                 PENDOWN: BOOLEAN;
              END;
   CHARACTER = ARRAY[0..MAXCOORDS] OF COORDS;

VAR
   CHARFILE: FILE OF CHARACTER;
   TABLE: ARRAY[0..127] OF CHARACTER;
   SYMBOL: CHARACTER;
   X, Y: INTEGER;
   NAME:  STRING;
   CH: CHAR;

PROCEDURE HOME;

BEGIN
   WRITE (CHR(12));
   PAGE (OUTPUT);
END;

PROCEDURE READFILE;

VAR
   I:  INTEGER;

BEGIN
   TEXTMODE;
   HOME;
   GOTOXY (0,5);
   WRITE ('Read which file : ');
   READLN (NAME);
   §$I-†
   RESET (CHARFILE, NAME);
   §$I+†
   IF IORESULT <> 0 THEN
      BEGIN
         WRITELN ('IO ERROR:  file not found');
         EXIT (READFILE);
      END;
   FOR I:= 0 TO 127 DO
      BEGIN
         TABLE[I]:= CHARFILE©;
         GET (CHARFILE);
      END;
   CLOSE (CHARFILE);
END; §readfile†

PROCEDURE SAVE;

VAR
   I: INTEGER;

BEGIN
   TEXTMODE;
   HOME;
   GOTOXY (0,5);
   WRITE ('Save as what file : ');
   READLN (NAME);
   §$I-†
   REWRITE (CHARFILE, NAME);
   §$I+†
   IF IORESULT <> 0 THEN
      BEGIN
         WRITELN ('IO ERROR:  file not found');
         EXIT (SAVE);
      END;
   FOR I:= 0 TO 127 DO
      BEGIN
         CHARFILE©:= TABLE[I];
         PUT (CHARFILE);
      END;
   CLOSE (CHARFILE, LOCK);
END; §save†

PROCEDURE NEWFILE;

VAR
   I:  INTEGER;

BEGIN
   TEXTMODE;
   HOME;
   FOR I:= 0 TO MAXCOORDS DO
      WITH SYMBOL[I] DO
         BEGIN
            XCOORD:= 0;
            YCOORD:= 0;
            NEGXMOVE:= FALSE;
            NEGYMOVE:= FALSE;
            LAST:= FALSE;
            PENDOWN:= FALSE;
         END;
   FOR I:= 0 TO 127 DO TABLE[I]:= SYMBOL;
   GOTOXY (0,5);
   WRITELN (' --- Memory cleared ---');
END; §newfilet

PROCEDURE GRID;

BEGIN
   GRAFMODE;
   FILLSCREEN (BLACK);
   X:= 0;
   Y:= 0;
   MOVETO (X,Y);
   REPEAT
      PENCOLOR (GREEN);
      MOVETO (270,Y);
      PENCOLOR (NONE);
      Y:= Y+10;
      MOVETO (X,Y);
   UNTIL Y>190;
   Y:= 0;
   MOVETO (X,Y);
   REPEAT
      PENCOLOR (GREEN);
      MOVETO (X,190);
      PENCOLOR (NONE);
      X:= X+10;
      MOVETO (X,Y);
   UNTIL X>270;
   VIEWPORT (69,131,39,121);
   FILLSCREEN (BLACK);
   VIEWPORT (69,151,39,45);
   FILLSCREEN (BLACK);
   VIEWPORT (0,279,0,191);
   X:= 70;
   Y:= 40;
   MOVETO (X,Y);
   PENCOLOR (WHITE);
   MOVE (0);
   PENCOLOR (NONE);
END; §gridt

PROCEDURE DISPLAY;
```

```
VAR
   I, J:  INTEGER;

BEGIN
   TEXTMODE;
   HOME;
   GOTOXY (0,5);
   REPEAT
      WRITE ('Display which character : ');
      §$I-†
      READ (I);
      §$I+†
      IF IORESULT <> 0 THEN EXIT (DISPLAY);
      WRITELN;
      IF (I>=0) AND (I<128) THEN
         BEGIN
            SYMBOL:= TABLE[I];
            GRID;
            J:= 0;
            REPEAT
               WITH SYMBOL[J] DO
                  BEGIN
                     IF NEGXMOVE
                        THEN X:= X - XCOORD*10
                        ELSE X:= X + XCOORD*10;
                     IF NEGYMOVE
                        THEN Y:= Y - YCOORD*10
                        ELSE Y:= Y + YCOORD*10;
                     IF PENDOWN
                        THEN PENCOLOR (WHITE)
                        ELSE PENCOLOR (NONE);
                     MOVETO (X,Y);
                     J:= J+1;
                  END; §witht
            UNTIL SYMBOL[J-1].LAST;
            PENCOLOR (WHITE);
            MOVE (0);
            PENCOLOR (NONE);
         END; §ift
   UNTIL (I<0) OR (I>127);
END; §displayt

PROCEDURE CREATE;

VAR
   I,J: INTEGER;
   X0, Y0, XSUM, YSUM: INTEGER;
   XPOS, YPOS: INTEGER;
```

```
PROCEDURE MARKIT;

BEGIN
   IF  (XPOS >-9)
   AND (XPOS < 9)
   AND (YPOS >-9)
   AND (YPOS < 9) THEN
     BEGIN
        XSUM:= XSUM+XPOS;
        YSUM:= YSUM+YPOS;
        WRITE ('Point',J:3,'  X =',XSUM:3,'   Y = ',YSUM:2);
        IF SYMBOL[J].PENDOWN THEN WRITELN ('  Pendown')
        ELSE WRITELN ('  Penup');
        MOVETO (X0,Y0);
        X:= X0 + XPOS*10;
        Y:= Y0 + YPOS*10;
        X0:= X;
        Y0:= Y;
        IF SYMBOL[J].PENDOWN THEN
           BEGIN
              PENCOLOR (WHITE);
              MOVETO (X,Y);
           END
        ELSE
           BEGIN
              MOVETO (X,Y);
              PENCOLOR (REVERSE);
              MOVE (0);
           END;
        PENCOLOR (NONE);
        IF XPOS < 0 THEN
           BEGIN
              SYMBOL[J].NEGXMOVE:= TRUE;
              SYMBOL[J].XCOORD:= ABS(XPOS);
           END
        ELSE SYMBOL[J].XCOORD:= XPOS;
        IF YPOS < 0 THEN
           BEGIN
              SYMBOL[J].NEGYMOVE:= TRUE;
              SYMBOL[J].YCOORD:= ABS(YPOS);
           END
        ELSE SYMBOL[J].YCOORD:= YPOS;
        IF J=19 THEN
           BEGIN
             WRITE   (CHR(7),'One move remaining....');
             WRITELN ('Terminate character');
           END;
        J:= J+1;
        IF J<21 THEN SYMBOL[J].PENDOWN:= SYMBOL[J-1].PENDOWN;
        XPOS:= 0;
        YPOS:= 0;
     END
   ELSE WRITELN ('Invalid point....Try again');
END; §markit†

PROCEDURE POINT;

BEGIN
   IF CH IN ['1'..'4','6'..'9'] THEN
      BEGIN
         PENCOLOR (REVERSE);
         MOVE (0);
         PENCOLOR (NONE);
         X:= X0 + XPOS*10;
         Y:= Y0 + YPOS*10;
         MOVETO (X,Y);
         PENCOLOR (REVERSE);
         MOVE (0);
         PENCOLOR (NONE);
      END;
END; §point†

BEGIN §create†
   TEXTMODE;
   HOME;
   WRITELN ('Create a new character');
   WRITELN;
   WRITELN ('Movement commands are:');
   writeln;
   writeln ('1  2  3');
   writeln ('4  5  6');
   writeln ('7  8  9');
   writeln (',  0  .');
   writeln;writeln;
   REPEAT
      WRITELN;
      WRITE ('Character number : ');
      §$I-†
      READ (I);
      §$I+†
      IF IORESULT <> 0 THEN EXIT (CREATE);
      WRITELN;
      IF (I>=0) AND (I<128) THEN
         BEGIN
            SYMBOL:= TABLE[I];
            FOR J:= 0 TO MAXCOORDS DO
               WITH SYMBOL[J] DO
```

```
          BEGIN
             XCOORD:= 0;
             YCOORD:= 0;
             NEGXMOVE:= FALSE;
             NEGYMOVE:= FALSE;
             LAST:= FALSE;
             PENDOWN:= FALSE;
          END;
    GRID;
    X0:= X;
    Y0:= Y;
    XPOS:= 0;
    YPOS:= 0;
    XSUM:= 0;
    YSUM:= 0;
    J:= 0;
    REPEAT
       READ (KEYBOARD, CH);
       CASE CH OF
          '.':  SYMBOL[J].PENDOWN:= TRUE;
          ',':  SYMBOL[J].PENDOWN:= FALSE;
          '6':  XPOS:= XPOS+1;
          '4':  XPOS:= XPOS-1;
          '8':  YPOS:= YPOS+1;
          '2':  YPOS:= YPOS-1;
          '9':  BEGIN
                   XPOS:= XPOS+1;
                   YPOS:= YPOS+1;
                END;
          '1':  BEGIN
                   XPOS:= XPOS-1;
                   YPOS:= YPOS-1;
                END;
          '3':  BEGIN
                   XPOS:= XPOS+1;
                   YPOS:= YPOS-1;
                END;
          '7':  BEGIN
                   XPOS:= XPOS-1;
                   YPOS:= YPOS+1;
                END;
          '0':  BEGIN
                   SYMBOL[J].LAST:= TRUE;
                   MARKIT;
                END;
          '5':  MARKIT;
          'G':  GRAFMODE;
          'T':  TEXTMODE;
       END; §case†
```

```
                POINT;
             UNTIL CH = '0';
             TABLE[I]:= SYMBOL;
          END; §ift
   UNTIL (I<0) OR (I>127);
END;§createt

BEGIN
   INITTURTLE;
   TEXTMODE;
   HOME;
   REPEAT
      GOTOXY (0,0);
      WRITE('Chargen: R)ead, S)ave, D)isplay, ');
      WRITE('C)reate, N)ew, Q)uit.');
      READ (CH);
      CASE CH OF
         'R','r':  READFILE;
         'S','s':  SAVE;
         'D','d':  DISPLAY;
         'C','c':  CREATE;
         'N','n':  NEWFILE;
         'Q','q':  TEXTMODE;
         'T','t':  TEXTMODE;
         'G','g':  GRAFMODE;
      END;
   UNTIL CH IN ['Q','q'];
END.
```

Appendix G

PERSPECTIVE REPRESENTATION OF FUNCTIONS OF TWO VARIABLES

In this section a technique for representation of arbitrary continuous single-valued functions of two variables of bounded variation is suggested. A Pascal computer program is then developed which uses the method for perspective representation of functions.

The planar projection that best represents real objects as viewed by the naked eye through a medium is perspective. This type of projection maps an arbitrary point of the space occupied by the object to a point on the plane such that all lines connecting the two points of intersect in a common point called the center of projection which corresponds to the eye of the observer. The plane is called the plane of projection and is oriented so that its normal is parallel to the line of sight (see Figure G.1).

Let $f(x,y)$ be the function to be plotted and suppose that the observer's eye is situated at the point C with coordinates (c_x, c_y, c_z) referred to the same rectangular coordinate system. Let α, β, and λ be the angles which the line of sight makes with the x, y, and z axes, respectively, and further, a distance, d, is given and define $Q(q_x,q_y,q_z)$ as a point such that CQ is the direction of sight and $|CQ| = d$. Let the plane of projection, π, be constructed through Q and normal CQ. The straight line from C to an arbitrary point in space $P(x,y,z)$ will intersect π in some point $P'(x',y',z')$. P' is the perspective image of P in π with respect to C.

After some tedious trigonometry and algebra we obtain the following equations

$$q_x = c_x + d(\cos\alpha)$$

$$q_y = c_y + d(\cos\beta)$$

$$q_z = c_z + d(\cos\lambda)$$

$$K = \frac{d}{(x-c_x)\cos\alpha + (y-c_y)\cos\beta + (z-c_z)\cos\lambda}$$

$$x' = c_x + K(x-c_x)$$

$$y' = c_y + K(y-c_y)$$

$$z' = c_z + K(z-c_z)$$

$$X = \frac{(x'-q_x)\cos\beta - (y'-q_y)\cos\alpha}{\sin\lambda}$$

$$Y = \frac{z'-q_z}{\sin\lambda}$$

For a vertical line of sight $\sin\lambda = 0$ and the above equations must be modified. If up corresponds to an increasing y-component, positive X is to the right, and positive Y is up, then the perspective transformation is

$$X = \frac{-(x'-q_x)\cos\lambda + (z'-q_z)\cos\alpha}{\sin\beta}$$

$$Y = \frac{y'-q_y}{\sin\beta}$$

A subset of the infinite number of points on the surface must be selected that will conveniently and effectively represent the surface. Most convenient for this purpose are families of orthogonal curves on the surface. This orthogonal property is generally desirable, as perspective views generated from such families tend to be well defined and aesthetically pleasing.

The most simple pair of mutually orthogonal families is

$$u = x, \qquad x_{min} <= x <= x_{max},$$

$$v = y, \qquad y_{min} <= y <= y_{max},$$

with generation of the rectangular array of (x,y,z) triples being trivial for this case.

The above technique is used by the program ThreeD which generates a perspective image.

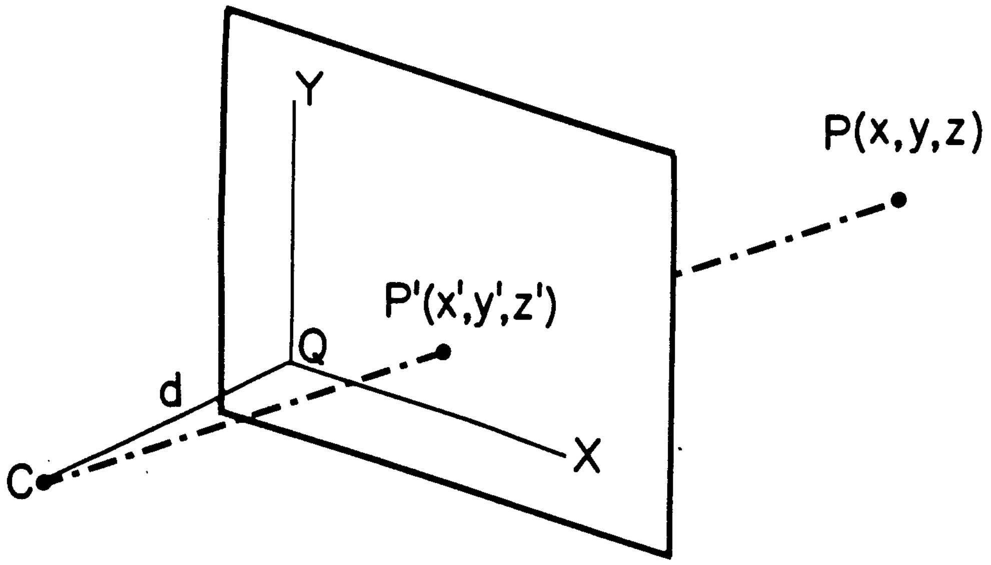


Figure G.1

Mapping of a Point in Space onto the Projection Plane

```
(*$S+*)
Program ThreeD;

(*
Pascal program to generate perspective image of
a function of two variables.  The input is in the
form of a two dimensional matrix.
*)

USES TRANSCEND,(*$U #5:3D-PS.CODE*)PLOTSTUFF;

Const
  pi=3.141592654;
  XMAX=51;
  YMAX=30;

Type
  Realmatrix=Array [1..XMAX,1..YMAX] OF real;

Var
   surface,x,y:Realmatrix;
   Zscale,xco,yco,cx,cy,cz:real;
   alpha,beta,gamma:real;
   qx,qy,qz,K,d,xi,eta,zeta:real;
   pslot,RowNo,ColumnNo,i,j:integer;
   xscale,yscale,ixco,iyco:real;
   Sinbeta,Singamma:real;
   DirAlpha,DirBeta,DirGamma:real;
   xminimum,xmaximum,yminimum,ymaximum,zmin,zmax:real;
   xcomin,xcomax,ycomin,ycomax:real;
   inname:string;
   infile:text;
   IntoGrid,xsize,ysize:real;
   Auto: boolean;


Procedure scale(xco,yco:real);

Begin
  ixco:=(1.0+xscale*(xco-xcomin))*IntoGrid;
  iyco:=(1.0+yscale*(yco-ycomin))*IntoGrid;
End;


Procedure MaxMin;

Begin
  xcomin:=x[1,1];
  xcomax:=x[1,1];
  ycomin:=y[1,1];
  ycomax:=y[1,1];
  For i:=1 to RowNo Do
    For j:=1 to ColumnNo Do
      Begin
        If x[i,j] < xcomin then xcomin:=x[i,j];
        If x[i,j] > xcomax then xcomax:=x[i,j];
        If y[i,j] < ycomin then ycomin:=y[i,j];
        If y[i,j] > ycomax then ycomax:=y[i,j];
      End;
End;


Procedure CalcQvector;

Begin
  If not Auto Then
    Begin
      DirAlpha:=Cos(alpha);
      DirBeta :=Cos(beta);
      DirGamma:=Cos(gamma);
    End;
  SinBeta :=SQRT(1.0-sqr(DirBeta));
  SinGamma:=SQRT(1.0-sqr(DirGamma));
  qx:=cx+d*DirAlpha;
  qy:=cy+d*DirBeta;
  qz:=cz+d*DirGamma;
End;


Procedure Projection(x,y,z:real);

Begin
  If SinGamma<>0.0 then
    Begin
      K:=d/((x-cx)*DirAlpha+(y-cy)*DirBeta+
                            (z-cz)*DirGamma);
      xi:=cx+K*(x-cx);
      eta:=cy+K*(y-cy);
      zeta:=cz+K*(z-cz);
      xco:=((xi-qz)*DirBeta-(eta-qy)*DirAlpha)/SinGamma;
      yco:=(zeta-qz)/SinGamma;
    End
  Else
    Begin
      xco:=((zeta-qz)*DirAlpha-(xi-qx)*DirGamma)
                                         /SinBeta;
      yco:=(eta-qy)/SinBeta;
    End;
End;
```

```
Procedure Setup;

Var xstep,ystep,xval,yval,zval:real;

Begin
  gotoxy(0,22);
  write(EOLNCLEAR,'Projecting...');
  CalcQvector;
  xstep:=(xmaximum-xminimum)/(RowNo-1);
  ystep:=(ymaximum-yminimum)/(ColumnNo-1);
  For i:=1 to RowNo Do
    For j:=1 to ColumnNo Do
      Begin
        xval:=xminimum+xstep*(i-1);
        yval:=yminimum+ystep*(j-1);
        zval:=surface[i,j]/Zscale;
        Projection(xval,yval,zval);
        x[i,j]:=xco;
        y[i,j]:=yco;
      End;
End;


Procedure Convert;

Begin
  gotoxy(0,22);
  write(EOLNCLEAR,'Scaling...');
  MaxMin;
  If xcomax<>xcomin then xscale:=xsize/(xcomax-xcomin)
  Else xscale:=0.0;
  If ycomax<>ycomin then yscale:=ysize/(ycomax-ycomin)
  Else yscale:=0.0;
  For i:=1 to RowNo Do
    For j:=1 to ColumnNo Do
      Begin
        scale(x[i,j],y[i,j]);
        x[i,j]:=ixco;
        y[i,j]:=iyco;
      End;
End;


Procedure GetFile;

Begin
  write(HOME,EOSCLEAR);
  Repeat
    gotoxy(0,5);
    write(EOLNCLEAR);
    write('Input row dimension of the matrix -->');
    readln(RowNo);
    write(EOLNCLEAR);
    write('Input column dimension of the matrix -->');
    readln(ColumnNo);
    gotoxy(0,10);
    write(EOLNCLEAR,'Input file name -->');
    readln(inname);
    If inname='' then exit(GetFile);
    If pos(inname,'.TEXT') = 0 then
           inname:=concat(inname,'.TEXT');
    (*$I-*)
    Reset(infile,inname);
    (*$I+*)
  Until not badio;
  write(HOME,EOSCLEAR);
  write('Reading...',inname);
  For i:=1 to RowNo Do
    Begin
      gotoxy(0,10);
      write(EOLNCLEAR,i);
      For j:=1 to (ColumnNo-1) Do
        Begin
          Read(infile,surface[i,j]);
          If (i=1) and (j=1) then
            Begin
              zmin:=surface[1,1];
              zmax:=surface[1,1];
            End;
          If surface[i,j] < zmin then
             zmin:=surface[i,j];
          If surface[i,j] > zmax then
             zmax:=surface[i,j];
          gotoxy(10,10);
          write(j);
        End;
      Readln(infile,surface[i,ColumnNo]);
      gotoxy(10,10);
      write(ColumnNo);
    End;
  close(infile);
End;


Procedure GetXvalues;

Begin
  write(HOME,EOSCLEAR);
  gotoxy(0,5);
  write(EOLNCLEAR,'Input min x value -->');
  readln(xminimum);
  write(EOLNCLEAR,'Input max x value -->');
  readln(xmaximum);
End;


Procedure GetYvalues;

Begin
  write(HOME,EOSCLEAR);
  gotoxy(0,5);
  write(EOLNCLEAR,'Input min y value -->');
  readln(yminimum);
  write(EOLNCLEAR,'Input max y value -->');
  readln(ymaximum);
End;


Procedure GetZvalues;

Begin
  write(HOME,EOSCLEAR);
  gotoxy(0,5);
  write(EOLNCLEAR,'Input min z value -->');
  readln(zmin);
  write(EOLNCLEAR,'Input max z value -->');
  readln(zmax);
End;


Procedure GetSize;

Begin
  write(HOME,EOSCLEAR);
  gotoxy(0,5);
  write(EOLNCLEAR,'Input x-size in inches -->');
  readln(xsize);
  write(EOLNCLEAR,'Input y-size in inches -->');
  readln(ysize);
End;


Procedure GetReference;

Begin
  write(HOME,EOSCLEAR);
  gotoxy(0,5);
  write(EOLNCLEAR);
  write('Input reference x coordinate -->');
  readln(cx);
  write(EOLNCLEAR);
  write('Input reference y coordinate -->');
  readln(cy);
  write(EOLNCLEAR);
  write('Input reference z coordinate -->');
  readln(cz);
End;


Procedure GetAngles;

Begin
  write(HOME,EOSCLEAR);
  gotoxy(0,5);
  write(EOLNCLEAR);
  write('Input reference angle alpha -->');
  readln(alpha);
  alpha:=alpha*pi/180.0;
  write(EOLNCLEAR);
  write('Input reference angle beta  -->');
  readln(beta);
  beta:=beta*pi/180.0;
  write(EOLNCLEAR);
  write('Input reference angle gamma -->');
  readln(gamma);
  gamma:=gamma*pi/180.0;
  write(EOLNCLEAR);
  write('Input the sight distance "d" -->');
  readln(d);
End;


Procedure Map;

Begin
  Setup;
  Convert;
  gotoxy(0,22);
  write(EOLNCLEAR);
  write('<return> when plotting device ready...');
  gotoxy(0,22);
  write(EOLNCLEAR,'Plotting...');
  For i:=1 to RowNo Do
    Begin
      PEN(up);
      For j:=1 to ColumnNo Do
        Begin
          PLOTTO(Round(x[i,j]),Round(y[i,j]));
          PEN(down);
        End;
      End;
    For j:=1 to ColumnNo Do
      Begin
        PEN(up);
        For i:=1 to RowNo Do
          Begin
            PLOTTO(Round(x[i,j]),Round(y[i,j]));
            PEN(down);
          End;
      End;
  End;


Procedure ToggleDest;

Begin
  INITEK;
  If TEK Then TEK:=false
  Else TEK:=true;
  If TEK Then
    Begin
      IntoGrid:=128.0;
      PLOTLIM(0,1023,780,0);
    End
  Else
    Begin
      IntoGrid:=200.0;
      PLOTLIM(0,2000,1400,0);
    End;
End;


Procedure WrOptions;

Begin
  If not Auto then CalcQvector;
  write  ('[A]==Set Auto Mode');
  writeln('        [N]==Clear Auto Mode');
  write  ('[D]==Plot dimension is');
  writeln(xsize:4:2,' x ',ysize:4:2);
  write  ('[X]==Xmin = ',xminimum);
  write  ('     Xmax = ',xmaximum);
  writeln('     Xdiv = ',RowNo);
  write  ('[Y]==Ymin = ',yminimum);
  write  ('     Ymax = ',ymaximum);
  writeln('     Ydiv = ',ColumnNo);
  write  ('[Z]==Zmin = ',zmin);
  writeln('     Zmax = ',zmax,'     d = ',d);
  write  ('[C]==Cx = ',cx,'     Cy = ',cy);
  writeln('     Cz = ',cz);
  write  ('[<]==DirAlpha = ',DirAlpha:4:2);
  write  ('     DirBeta = ',DirBeta:4:2);
  writeln('     DirGamma = ',DirGamma:4:2);
  write  ('[T]==Toggle destination (Tek/HiPlot)');
  writeln('     [I]==Goto (0,0)');
  writeln('[S]==Change plotter slot');
  write  ('[R]==Read datafile     ');
  writeln('[M]==Scale in z-direction Zscale = ',Zscale);
  writeln('[P]==Plot      [B]==Box the Plot (axes)');
  writeln('[Q]==Quit.');
End;


Procedure Wrtitles;

Begin
  write(HOME,EOSCLEAR);
  write  ('Prospective image of a surface');
  writeln(' in 3-D _______ VERSION 1.0');
  write  ('-------------------------------------');
  writeln(' memory = ',MEMAVAIL,'  words');
  If TEK then
    Begin
      write('Plot to Tektronix 4010');
      write('  --  ',XMAX,' x ',YMAX);
    End
  Else
    Begin
      write('Plot to HiPlot pen-plotter');
      write('  --  ',XMAX,' x ',YMAX);
    End;
  If Auto Then writeln('    **Auto Mode**')
  Else writeln('   **Manual Mode**');
  writeln;
  WrOptions;
  gotoxy(0,20);
  write(EOLNCLEAR,'Input option:');
  Read(keyboard,ch);
End;

Procedure SetAuto;

VAR Xp,Yp,Zp:real;

Begin
  Auto:=true;
  i:=RowNo div 2;
  j:=ColumnNo div 2;
  Xp:=0.5*(xmaximum+xminimum);
```

```
   Yp:=0.5*(ymaximum+yminimum);
   Zp:=0.5*(zmin+zmax);
   d:=SQRT(sqr(Xp-Cx)+sqr(Yp-Cy)+sqr(Zp-Cz));
   DirAlpha:=(Xp-Cx)/d;
   DirBeta:=(Yp-Cy)/d;
   DirGamma:=(Zp-Cz)/d;
   If xmaximum > xminimum then Cx:=10.0*xmaximum
   Else  Cx:=10.0*xminimum;
   If ymaximum > yminimum then Cy:=10.0*ymaximum
   Else  Cy:=10.0*yminimum;
   If zmax > zmin then Cz:=10.0*zmax
   Else  Cz:=10.0*zmin;
 End;


 Procedure initialize;

 Begin
   Zscale:=1.0;
   Auto:=true;
   xsize:=5.0;
   ysize:=5.0;
   IntoGrid:=128.0;
   GETCRTINFO;
   GetFile;
   xminimum:=zmin;
   yminimum:=zmin;
   xmaximum:=zmax;
   ymaximum:=zmax;
   SetAuto;
 End;


 Procedure Movein3D(x,y,z:real; switch:PENSTATE);

 Begin
   Projection(x,y,z);
   scale(xco,yco);
   PEN(switch);
   PLOTTO(Round(ixco),Round(iyco));
 End;


 Procedure BoxIt;

 Var Xorg,Yorg:integer;
 Begin
   Movein3D(xminimum,yminimum,zmin,UP);
   Movein3D(xmaximum,yminimum,zmin,DOWN);
   Movein3D(xmaximum,ymaximum,zmin,DOWN);
```

```
  Movein3D(xminimum,ymaximum,zmin,DOWN);
  Movein3D(xminimum,yminimum,zmin,DOWN);
  Movein3D(xminimum,yminimum,zmax,DOWN);
  Movein3D(xminimum,yminimum,zmin,UP);
 End;


Begin
  initialize;
  Repeat
    Wrtitles;
    CASE ch OF
      'A','a': SetAuto;
      'N','n': Begin
                 Auto:=false;
               End;
      'B','b': BoxIt;
      'D','d': GetSize;
      'X','x': GetXvalues;
      'Y','y': GetYvalues;
      'Z','z': GetZvalues;
      'C','c': GetReference;
      '<','>': GetAngles;
      ',','.': GetAngles;
      'T','t': ToggleDest;
      'S','s': Begin
                 write(HOME,EOSCLEAR);
                 gotoxy(0,12);
                 write('Input plotter slot #');
                 readln(pslot);
                 PLOTSLOT(pslot);
               End;
      'M','m': Begin
                 gotoxy(0,12);
                 write(EOLNCLEAR);
                 write('Input Z scale factor -->');
                 Readln(Zscale);
               End;
      'I','i': INITEK;
      'R','r': GetFile;
      'P','p': Map;
    End;
  Until ch in ['Q','q'];
  Write(HOME,EOSCLEAR);
  gotoxy(0,12);
  write('That''s all Folks !');
END.
```

```
(*$S+*)
UNIT PLOTSTUFF;

(*Tektronix plot stuff unit*)

INTERFACE

  TYPE
      wordbit = RECORD CASE BOOLEAN OF
                   true: (word: CHAR);
                   false: (bit: PACKED ARRAY[0..15] OF BOOLEAN);
                 END;
      PENSTATE = (UP,DOWN);
      PENCOORDS = PACKED RECORD
                        XVEC: 0..23;
                        YVEC: 0..23;
                        XNEG: BOOLEAN;
                        YNEG: BOOLEAN;
                        LASTPOINT: BOOLEAN;
                        PENDOWN: BOOLEAN;
                  END;
      CHARACTER = ARRAY[0..20] OF PENCOORDS;

  VAR
      TEK, bad: BOOLEAN;
      PENX,PENY,PENANGLE: INTEGER;
      CHARFILE: FILE OF CHARACTER;
      EOLNCLEAR,EOSCLEAR,HOME : PACKED ARRAY [0..1] OF CHAR;
      CH : CHAR;


   PROCEDURE RINGIT;
   PROCEDURE PAGE;
   PROCEDURE getcrtinfo;
   FUNCTION BADIO : BOOLEAN;
   PROCEDURE PEN (MODE: PENSTATE);
   PROCEDURE PLOTTO (X,Y: INTEGER);
   PROCEDURE MOVETO (X,Y: INTEGER);
   PROCEDURE PLOT (DIST: INTEGER);
   PROCEDURE MOVE (DIST: INTEGER);
   PROCEDURE ANGLE (X: INTEGER);
   PROCEDURE PLOTLIM (L,R,T,B: INTEGER);
   PROCEDURE PLOTSLOT (I: INTEGER);
   PROCEDURE INITEK;

IMPLEMENTATION

VAR
   LBOUND,RBOUND,TBOUND,BBOUND:INTEGER;
   CHXSCALE, CHYSCALE: REAL;
   LEFTB,RIGHTB,TOPB,BOTTOMB: INTEGER;
   TABLE: ARRAY[0..127] OF CHARACTER;
   SYMBOL: CHARACTER;
   ESC, GS, BELL, FF : CHAR;
   DELPEN,i : integer;

FUNCTION INITCARD (I: INTEGER): INTEGER;
   EXTERNAL;

PROCEDURE WPLOTTER (CH: CHAR);
   EXTERNAL;

PROCEDURE getcrtinfo;

TYPE
   syscomrec =  PACKED ARRAY[0..511] OF CHAR;
   syscomptr = ^syscomrec;

VAR
   syscom: syscomptr;
   trix: RECORD CASE BOOLEAN OF
           false: (address: INTEGER);
           true:  (pointer: ^syscomptr);
         END;
   prefixed: wordbit;
   leadin: CHAR;

BEGIN
   trix.address:= 248;
   syscom:= trix.pointer^;
   leadin:= syscom^[62];
   eosclear[1]:= syscom^[64];
   eolnclear[1]:= syscom^[65];
   home[1]:= syscom^[63];
   WITH prefixed DO
      BEGIN
         word:= syscom^[72];
         IF bit[6]
            THEN home[0]:= leadin
            ELSE home[0]:= chr(0);
         IF bit[3]
            THEN eosclear[0]:= leadin
            ELSE eosclear[0]:= chr(0);
         IF bit[2]
            THEN eolnclear[0]:= leadin
            ELSE eolnclear[0]:= chr(0);
      END; (*with*)
END; (*getcrtinfo*)

FUNCTION BADIC;
```

```
VAR
   IOCODE: 0..16;

BEGIN
  IOCODE:= IORESULT;
  BADIO:= FALSE;
  IF IOCODE > 0 THEN
    BEGIN
      BADIO:= TRUE;
      GOTOXY (0,21);
      WRITE ('I/O ERROR:  ');
      CASE IOCODE OF

 1: WRITELN ('Bad Block, Parity Read Error');
 2: WRITELN ('Bad Unit Number');
 3: WRITELN ('Bad Mode, Illegal Operation');
 4: WRITELN ('Undefined Hardware Error');
 5: WRITELN ('Lost Unit, Unit is no longer on line');
 6: WRITELN ('Lost File, File is no longer in Directory');
 7: WRITELN ('Bad Title, Illegal File Name');
 8: WRITELN ('No Room on Disk, Insufficient space');
 9: WRITELN ('No Unit, No such volume on line');
10: WRITELN ('No File, No such file on volume');
11: WRITELN ('Duplicate File');
12: WRITELN ('File not Closed, Attempt to open an open file');
13: WRITELN ('Not Open, Attempt to access a closed file');
14: WRITELN ('Bad Format, Error in reading real or integer');
15: WRITELN ('Ring Buffer Overflow');
16: WRITELN ('Write Protected');

        END; (*Cases*)
      WRITELN;
      WRITE ('PRESS <SPACEBAR> TO CONTINUE');
      READ (KEYBOARD,CH);
      GOTOXY (0,21);
      WRITE (EOSCLEAR);
    END; (*If*)
END; (*Badio*)

PROCEDURE PLOTTO;

VAR
   LOWX,LOWY,HIGHX,HIGHY : CHAR;
   F,D,I,T,E,Z,DX,DY: INTEGER;
   A: STRING[16];

BEGIN
  IF TEK THEN
    BEGIN
```

```
        IF X > RIGHTB THEN X:=RIGHTB;
        IF X < LEFTB  THEN X:=LEFTB;
        IF Y > TOPB   THEN Y:=TOPB;
        IF Y < BOTTOMB THEN Y:=BOTTOMB;
        LOWX:=CHR((X MOD 32) + 64);
        LOWY:=CHR((Y MOD 32) + 96);
        HIGHX:=CHR((X DIV 32) +32);
        HIGHY:=CHR((Y DIV 32) +32);
        WPLOTTER(HIGHY);
        WPLOTTER(LOWY);
        WPLOTTER(HIGHX);
        WPLOTTER(LOWX);
        PENX:=X;
        PENY:=Y;
      END
    ELSE
      BEGIN
        IF X > RIGHTB THEN X:=RIGHTB;
        IF X < LEFTB  THEN X:=LEFTB;
        IF Y > TOPB   THEN Y:=TOPB;
        IF Y < BOTTOMB THEN Y:=BOTTOMB;
        DX:=X-PENX;
        DY:=Y-PENY;
        A:='pqrqrststuvuvwpw';
        F:=ABS(DX)+ABS(DY);
        IF F > 0 THEN
          BEGIN
            D:=ABS(DY)-ABS(DX);
            I:=0;
            IF DY >= 0 THEN I:=2;
            T:=DX+DY;
            IF T >= 0 THEN I:=I+2;
            T:=DY-DX;
            IF T >= 0 THEN I:=I+2;
            IF DX < 0 THEN I:=I+10
            ELSE I:=8-I;
            IF D < 0 THEN T:=ABS(DY)
            ELSE
              BEGIN
                T:=ABS(DX);
                D:=-D;
              END;
            E:=0;
            REPEAT
              Z:=T+D+E+E;
              IF Z < 0 THEN
                BEGIN
                  E:=E+T;
                  F:=F-1;
                  WPLOTTER(A[I-1]);
              END
            ELSE
              BEGIN
                E:=E+D;
                F:=F-2;
                WPLOTTER(A[I]);
              END;
          UNTIL F <= 0;
          PENX:=X;
          PENY:=Y;
        END;
    END;
END; (*plotto*)


PROCEDURE PEN;

VAR I: INTEGER;

BEGIN
   IF TEK THEN
     BEGIN
       IF MODE = UP THEN WPLOTTER (GS);
     END
   ELSE
     BEGIN
        IF MODE = UP THEN WPLOTTER('y')
        ELSE WPLOTTER('z');
        FOR I:= 1 TO DELPEN DO
          BEGIN
            (*DELAY*)
          END;
     END;
END; (*pen*)

PROCEDURE MOVETO;

BEGIN
  PEN(UP);
  PLOTTO (X,Y);
END;

FUNCTION SINE (ANGLE: INTEGER): REAL;

VAR
   RADS, RESULT, POWER, FACT, EPSILON: REAL;
   I, SIGN: INTEGER;

BEGIN
   RADS:= 3.14159265*ANGLE/180.0;
   POWER:= RADS * RADS * RADS;
   RESULT:= RADS;
   RADS:= RADS * RADS;
   FACT:= 6.0;
   SIGN:= -1;
   I:= 1;
      REPEAT
         EPSILON:= POWER/FACT;
         RESULT:= RESULT + SIGN * EPSILON;
         SIGN:= -SIGN;
         POWER:= POWER * RADS;
         FACT:= FACT * (2*I+2) * (2*I+3);
         I:= I+1;
      UNTIL ABS(EPSILON) < 1.0E-6;
   SINE:= RESULT;
END; (*sine*)

PROCEDURE PLOTSLOT;

BEGIN
  IF (I>0) AND (I<=7) THEN
    BEGIN
      IF INITCARD (I) <> 0 THEN
        WRITELN ('IO ERROR: Tek terminal not ready');
    END
  ELSE
    WRITELN ('INVALID SLOT NUMBER: Tek terminal not ready');
END; (*plotslot*)

PROCEDURE PLOTLIM;

BEGIN
   IF (R>L) AND (T>B) THEN
      BEGIN
         IF (L >= LBOUND) AND (L <= RBOUND) THEN
            LEFTB:= L;
         IF (R >= LBOUND) AND (R <= RBOUND) THEN
            RIGHTB:= R;
         IF (B >= BBOUND) AND (B <= TBOUND) THEN
            BOTTOMB:= B;
         IF (T >= BBOUND) AND (B <= TBOUND) THEN
            TOPB:= T;
      END;
END; (*plotlim*)

PROCEDURE PLOT;

VAR
   X,Y:INTEGER;
```

```
BEGIN
   X:= PENX + ROUND (DIST * SINE (90-PENANGLE));
   Y:= PENY + ROUND (DIST * SINE (PENANGLE));
   PLOTTO (X,Y);
END; (*plot*)

PROCEDURE MOVE;

BEGIN
  PEN(UP);
  PLOT(DIST);
END;

PROCEDURE ANGLE;

BEGIN
   PENANGLE:= X;
   WHILE PENANGLE >= 360 DO PENANGLE:= PENANGLE-360;
   WHILE PENANGLE <= -360 DO PENANGLE:= PENANGLE+360;
END; (*angle*)


PROCEDURE RINGIT;

BEGIN
  IF TEK THEN WPLOTTER(BELL);
END;

PROCEDURE PAGE;

BEGIN
  IF TEK THEN
    BEGIN
      WPLOTTER(ESC);
      WPLOTTER(FF);
      WPLOTTER(GS);
      WPLOTTER(BELL);
      WPLOTTER(BELL);
    END;
  PEN(UP);
  PLOTTO(0,0);
END;


PROCEDURE INITEK;

BEGIN
  IF TEK THEN
    BEGIN
      WPLOTTER(ESC);
```

```
        WPLOTTER(FF);
        WPLOTTER(GS);
        WPLOTTER(BELL);
        WPLOTTER(BELL);
      END;
    PEN(UP);
    PLOTTO(0,0);
  END;

  BEGIN
     IF TEK THEN
       BEGIN
         LBOUND := 0;
         RBOUND := 1023;
         TBOUND := 780;
         BBOUND := 0;
       END
     ELSE
       BEGIN
         LBOUND := 0;
         RBOUND := 2000;
         TBOUND := 1400;
         BBOUND := 0;
       END;
     DELPEN:=200;
     TEK:=true;
     PLOTSLOT(1);
     BELL:=CHR(7);
     FF:=CHR(12);
     ESC:=CHR(27);
     GS:=CHR(29);
     INITEK;
     PENX:= 0;
     PENY:= 0;
     PENANGLE:= 0;
     PLOTLIM (LBOUND,RBOUND,TBOUND,BBOUND);
     WPLOTTER(BELL);
     WPLOTTER(BELL);
  END.
```

Appendix H

DESCRIBING FUNCTION TEMPERATURE ANALYSIS CODE - DFUNC

The listed program was developed for numerical evaluation of the describing function method. The program is written in FORTRAN V symbolic programming language. The code was developed on a CDC-6600 (CYBER 170/720) main frame computer system.

```
      PROGRAM DFUNC(INPUT,OUTPUT,TAPE5=INPUT,TAPE6)
C
C  FORTRAN V program for fuel pin transient temperature
C  calculation using describing function method.
C
      DIMENSION SSCT(10),FUELTO(10),CIO(10),CIPO(10)
      DIMENSION FST(15),HCST(15),FLUXT(15),CITT(15)
      DIMENSION ZJ1R(20),GAPCON(15),QT(15),COOLTT(15)
      DIMENSION XIRBS(15),XIRBT(15),D(10),C(2),FSTOLD(15)
      DIMENSION AMAT(15,2),RHS(15,1),W1(2),ZJ0(20)
      COMMON RFUEL,NP,NS,S(15),TIME(15),PSI(10),
     *       XJ1R(10),CONINT(15,10)
      COMMON /DFC/ DESFUN(15,10),A(10),B(10),CIST(15)
      COMMON /LAPVAR/ COOLTS(15),FLUXS(15),CITS(15),
     *       HCSS(15),HSS(15)
      COMMON /HANVAR/ HSO,HSP(10),HCSO,HCPO(10)
      EQUIVALENCE (FSTOLD,QT),(HCST,COOLTT)
      DATA C/143895., 114.284/
      DATA D/0.5636368282704,.2277596778211,.4810856741816,
     *       .2697005128316,.4547568332689,.2881648264654,
     *       .4408953991634,.2990620051465,.4320383209448,
     *       .3064446575845/
      DATA ZJ0/2.4048255577, 5.5200781103, 8.6537279129,
     *       11.7915344391,14.9309177086,18.0710639679,
     *       21.2116366299,24.3524715308,27.4934791320,
     *       30.6346064684,33.7758202136,36.9170983537,
     *       40.0584257646,43.1997917132,46.3411883717,
     *       49.4826098974,52.6240518411,55.7655107550,
     *       58.9069839261,62.0484691902/
      DATA ZJ1R/.5191474973,-.3402648065, .2714522999,
     *        -.2324598314, .2065464331,-.1877288030,
     *         .1732658942,-.1617015507, .1521812138,
     *        -.1441659777, .1372969434,-.1313246267,
     *         .1260694971,-.1213986248, .1172111989,
     *        -.1134291926, .1099911430,-.1068478883,
     *         .1039595729,-.1012934989/
      REWIND 5
C
C  Read the input data and write it out.
C
      CALL RWDATA(RFUEL,RICLAD,ROCLAD,POWERL,PEAKAV,CLTHDF,
     *CCOND,GAPCNO,COOLTO,HFILM,NS,NP,TMAX,TEMERR,ACCFAC,
     *MITNO,WITERS,NPFUEL,NPCLAD)
C
C  Calculate initial conditions.
C
      CALL QUADHS(NPFUEL,NPCLAD,PEAKAV,POWERL,RFUEL,RICLAD,
     *ROCLAD,CCOND,HFILM,GAPCNO,COOLTO,CITO,HCSO,CIO,FLUXO,
     *SSCT,FUELTO)
C
```

```
C  Set Hankel domain constants.
C
      DO 100 IP=1,NP
        PSI(IP)=ZJ0(IP)/RFUEL
        XJ1R(IP)=ZJ1R(IP)
 100    CONTINUE
      CALL HANKI(PEAKAV,POWERL,CIO(NPFUEL),CIPO)
      AFUELT=0.
      DO 400 I=1,NPFUEL
 400    AFUELT=AFUELT + FUELTO(I)
      AFUELT=AFUELT/NPFUEL
      AVFTD=FTHDIF(AFUELT)
C
C  Set input function parameters.
C
      DO 500 IP=1,NP
        A(IP)=2.*CIPO(IP)/(RFUEL*XJ1R(IP))**2
        B(IP)=AVFTD*PSI(IP)**2
 500    CONTINUE
C
C  Set Laplace domain constants.
C
      CALL LAPMAT(TMAX)
      DO 600 IS=1,NS
        FST(IS)=FUELTO(NPFUEL)
        COOLTT(IS)=COOLNT(TIME(IS))
        QT(IS)=SOURCE(TIME(IS))
        GAPCON(IS)=CONRES(TIME(IS),FST(IS),GAPCNO)
 600    CONTINUE
      CALL LTRANS(NS,COOLTT,COOLTS)
      CALL LTRANS(NS,QT,HSS)
      CALL CLADCO(RICLAD,ROCLAD,CCOND,CLTHDF,HFILM)
C
C  Begin iterative solution.
C
      ONEMAF=1-ACCFAC
      ITERNO=0
 700  ITERNO=ITERNO+1
      IF(ITERNO.GT.MITNO) GO TO 1900
      DO 800 IS=1,NS
 800    FSTOLD(IS)=FST(IS)
      CALL FICONV(NS,15,1,FST,CIST)
      CALL FIHCV(NS,15,FST,HCST)
      CALL LTRANS(NS,HCST,HCSS)
      DO 1000 IP=1,NP
C
C  Calculate fuel surface integral conductivity in
C  Laplace domain.
C
        DO 900 IS=1,NS
```

```
          XIRBT(IS)=0.
          IF(B(IP)*TIME(IS).LT.50) XIRBT(IS)=
     .         EXP(-B(IP)*TIME(IS))*CIST(IS)
 900      CONTINUE
        CALL LTRANS(NS,XIRBT,XIRBS)
C
C  Calculate the describing function and the
C  double transformed integral conductivity.
C
        CONST=A(IP)*D(IP)/(RFUEL*XJ1R(IP)*PSI(IP))**2
        DO 1000 IS=1,NS
          DESFUN(IS,IP)=C(1)+2.*C(2)*(S(IS)+B(IP))*
     *    (XIRBS(IS)+CONST/(S(IS)+2.*B(IP)))
1000      CONTINUE
       DO 1050 IP=1,NP
        P2=PSI(IP)*PSI(IP)
        CONST1=RFUEL*XJ1R(IP)/PSI(IP)
        DO 1050 IS=1,NS
1050      CONINT(IS,IP)=((HSP(IP)+CONST1*HSO)*HSS(IS)+
     *    HCPO(IP)-CONST1*(S(IS)*HCSS(IS)-HCSO))/
     *    (S(IS)*DESFUN(IS,IP)+P2)
C
C  Use a least squares solution for fitting input
C  coefficients.
C
      DO 1100 IP=1,NP
        DO 1070 I=1,NS
          AMAT(I,1)=CONINT(I,IP)
          AMAT(I,2)=-1
1070      RHS(I,1)=-CONINT(I,IP)*S(I)
        CALL SOLVE(AMAT,RHS,W1,0,1,NS,2,15,2,1)
        B(IP)=RHS(1,1)
        A(IP)=RHS(2,1)*2./(RFUEL*XJ1R(IP))**2
1100    CONTINUE
C
C  Calculate the surface flux.
C
      DO 1160 IS=1,NS
        SUM=0.0
        DO 1150 IP=1,NP
1150      SUM=SUM + CONINT(IS,IP)*PSI(IP)/XJ1R(IP)
1160      FLUXS(IS)=2.0*SUM/(RFUEL*RFUEL)
      CALL CIRTS(CITO,FLUXO,COOLTO,HFILM)
      CALL LTINV(NS,FLUXS,FLUXT)
      CALL LTINV(NS,CITS,CITT)
      CALL SMOOTH(NS,FLUXO,FLUXT)
      CALL SMOOTH(NS,CITO,CITT)
      DO 1200 IS=1,NS
1200    FST(IS)=FLUXT(IS)/GAPCON(IS) + CITT(IS)
      IF(WITERS.EQ.1) CALL WITER(ITERNO,CIO(1),FLUXT,CITT,FST)
```

```
C
C  Check convergence and accelerate.
C
      DO 1300 IS=1,NS
        DELTAT=ABS(FST(IS)-FSTOLD(IS))
        IF(DELTAT.GT.TEMERR) GO TO 1500
1300  CONTINUE
      GO TO 1900
1500  DO 1700 IS=1,NS
1700    FST(IS)=ONEMAF*FSTOLD(IS)+ACCFAC*FST(IS)
      GO TO 700
C
C  Calculate the temperature field and print it.
C
1900  CONTINUE
      IF(ITERNO.GT.MITNO) TIME(6)=TIME(5)/FLOAT(ITERNO-1)
      CALL FICONV(NS,15,1,FST,CIST)
      CALL FTEMP(NPFUEL,CIO)
      CALL WTEMP(WITERS,ITERNO,MITNO,NPFUEL,FUELTO)
      CALL CTEMP(NPCLAD,RICLAD,ROCLAD,CLTHDF,FLUXO,
     *           COOLTO,HFILM,SSCT)
      CALL WCTEMP(NPCLAD,RICLAD,ROCLAD,SSCT)
      STOP
      END
C
C
      FUNCTION COOLNT(T)
C
C  Calculates coolant temperature as a function of time.
C
      COOLNT=509.
      RETURN
      END
C
C
      FUNCTION SOURCE(TIME)
C
C  Calculates the normalized volumetric heat source
C  as a function of time.
C
      OPTIME=1.E13
      QR=200.
      PSI=.35
      T=OPTIME+TIME
      PD=1.02/QR*(F(TIME)-F(T))
      G=1.+(3.24E-6+5.23E-10*TIME)*OPTIME**.4*PSI
      SOURCE=PD*G
      RETURN
      END
C
```

```
      FUNCTION F(T)
      DIMENSION A(23),XLAM(23)
      DATA A/6.5057E-01,  5.1264E-01,  2.4384E-01,
     *       1.3850E-01,  5.5440E-02,  2.2225E-02,  3.3088E-03,
     *       9.3015E-04,  8.0943E-04,  1.9567E-04,  3.2535E-05,
     *       7.5595E-06,  2.5232E-06,  4.9948E-07,  1.8531E-07,
     *       2.6608E-08,  2.2398E-09,  8.1641E-12,  8.7797E-11,
     *       2.5131E-14,  3.2176E-16,  4.5038E-17,  7.4791E-17 /
      DATA XLAM/2.2138E+01,  5.1587E-01,  1.9594E-01,
     *          1.0314E-01,  3.3656E-02,  1.1681E-02,  3.5870E-03,
     *          1.3930E-03,  6.2630E-04,  1.8906E-04,  5.4988E-05,
     *          2.0958E-05,  1.0010E-05,  2.5438E-06,  6.6361E-07,
     *          1.2290E-07,  2.7213E-08,  4.3714E-09,  7.5780E-10,
     *          2.4786E-10,  2.2384E-13,  2.4600E-14,  1.5699E-14 /
      XT=1.E13
      SUM=0.
      DO 50 I=1,23
      XLT=XLAM(I)*T
      XLXT=XLAM(I)*XT
      IF(XLT.GT.50.)GO TO 10
      EX1=EXP(-XLT)
      GO TO 20
   10 EX1=0.
   20 IF(XLXT.GT.50.)GO TO 30
      EX2=EXP(-XLXT)
      GO TO 40
   30 EX2=0.
   40 SUM=SUM+A(I)/XLAM(I)*EX1*(1.-EX2)
   50 CONTINUE
      F=SUM
      RETURN
      END
C
C
      FUNCTION CONRES(TIME,TFSURF,GAPCNO)
C
C  Calculates gap conductance (contact resistance) as a
C  function of time, surface temperature and initial
C  gap conductance.
C
      CONRES=GAPCNO
      RETURN
      END
C
C
      SUBROUTINE RWDATA(RFUEL,RICLAD,ROCLAD,POWERL,PEAKAV,
     * CLTHDF,CCOND,GAPCNO,COOLTO,HFILM,NS,NP,TMAX,TEMEPS,
     * ACCFAC,MITNO,WITERS,NPFUEL,NPCLAD)
C
C  Subroutine to read the input data from a free
```

```
C  format data file.  The data is then printed
C  and labeled.
C
      DIMENSION FST(1)
      READ(5,*)RFUEL,RICLAD,ROCLAD,POWERL,PEAKAV,
     *         CLTHDF,CCOND,GAPCNO,COOLTO,HFILM
      READ(5,*)NS,NP,TMAX,TEMEPS,ACCFAC,MITNO,WITERS,
     *         NPFUEL,NPCLAD
      WRITE(6,100)
      WRITE(6,101)
      WRITE(6,102)
      WRITE(6,103)
      WRITE(6,104)RFUEL
      WRITE(6,105)RICLAD
      WRITE(6,106)ROCLAD
      WRITE(6,107)CLTHDF
      WRITE(6,108)CCOND
      WRITE(6,109)GAPCNO
      WRITE(6,110)COOLTO
      WRITE(6,111)HFILM
      WRITE(6,112)POWERL
      WRITE(6,113)PEAKAV
      WRITE(6,200)
      WRITE(6,201)TMAX
      WRITE(6,202)TEMEPS
      WRITE(6,203)ACCFAC
      WRITE(6,204)NS
      WRITE(6,205)NP
      WRITE(6,206)MITNO
      WRITE(6,207)WITERS
      WRITE(6,208)NPFUEL
      WRITE(6,209)NPCLAD
      WRITE(6,210)
      IF(WITERS.EQ.1)WRITE(6,300)
 100  FORMAT(//30X,'D F U N C'//)
 101  FORMAT(10X,'DESCRIBING FUNCTION FUEL PIN ',
     *'TEMPERATURE CALCULATION')
 102  FORMAT(//1X,23('*'),' I N P U T  ',
     *' P A R A M E T E R S ',25('*'))
 103  FORMAT(//10X,'PARAMETERS FOR THE PROBLEM'
     */10X,26('-'))
 104  FORMAT(10X,'FUEL PELLET RADIUS                        '
     *,F10.7,' (M).')
 105  FORMAT(10X,'CALD INNER RADIUS                         '
     *,F10.7,' (M).')
 106  FORMAT(10X,'CALD OUTER RADIUS                         '
     *,F10.7,' (M).')
 107  FORMAT(10X,'CLAD THERMAL DIFFUSIVITY                  '
     *,E10.4,' (M**2/S).')
 108  FORMAT(10X,'CLAD CONDUCTIVITY                         '
     *,F10.3,' (W/M-K).')
 109  FORMAT(10X,'INITIAL GAP CONDUCTANCE              '
     *,F10.1,' (W/M**2-K).')
 110  FORMAT(10X,'INITIAL COOLANT TEMPERATURE          '
     *,F10.1,' (K).')
 111  FORMAT(10X,'COOLANT HEAT TRANSFER COEFFICIENT    '
     *,F10.1,' (W/M**2-K).')
 112  FORMAT(10X,'LINEAR POWER                         '
     *,F10.1,' (W/M).')
 113  FORMAT(10X,'PEAK-TO-AVERAGE RATIO                '
     *,F10.4,' .')
 200  FORMAT(/10X,'PARAMETERS FOR DFUNC'/10X,20('-'))
 201  FORMAT(10X,'FINAL TIME                            ',
     *'                 ',F10.3,' (S).')
 202  FORMAT(10X,'TEMPERATURE CONVERGENCE EPSILON    ',
     *'                 ',F10.3,' (K).')
 203  FORMAT(10X,'RELAXATION (CONVERGENCE ACCELERATION)',
     *' FACTOR      ',F10.3)
 204  FORMAT(10X,'NUMBER OF TIME POINTS (REAL AND ',
     *'LAPLACE DOMAINS)  ',I10)
 205  FORMAT(10X,'NUMBER OF TERMS IN BESSEL SERIES',
     *'                   ',I10)
 206  FORMAT(10X,'MAXIMUM NUMBER OF ITERATIONS ALLOWED',
     *'               ',I10)
 207  FORMAT(10X,'OPTION FOR ITERATION OUTPUT (1-YES)',
     *'                 ',I10),
 208  FORMAT(10X,'NUMBER OF FUEL TEMPERATURES NODES ',
     *'FOR OUTPUT      ',I10)
 209  FORMAT(10X,'NUMBER OF CLAD TEMPERATURES NODES ',
     *'FOR OUTPUT      ',I10)
 210  FORMAT(///1X,23('*'),' O U T P U T ',23('*')/)
 300  FORMAT(5X,'ITERATION OUTPUT'/5X,16('*'))
      RETURN
      END
C
C
      SUBROUTINE QUADHS(NPFUEL,NPCLAD,PEAKAV,POWERL,RFUEL,
     *RICLAD,ROCLAD,CCOND,HFO,GAPCNO,COOLTO,CITO,HCSO,CIO,
     *FLUXO,TC,TF)
C
C  Subroutine to calculate the steady state temperatures
C  for a quadratic heat source.
C
      DIMENSION TC(1),TF(1),CIO(1)
      DATA PI/3.14159265/
      QO=POWERL*(2.-PEAKAV)/(PI*RFUEL*RFUEL)
      A=(PEAKAV-1.)/(1.-PEAKAV/2.)
      FLUXO=QO*RFUEL*(1.+A/2.)/2.
C
C  Calculate the clad temperature.
```

```
C
      DO 100 I=1,NPCLAD
        RC=RICLAD + (ROCLAD-RICLAD)*FLOAT(I-1)/FLOAT(NPCLAD-1)
        TC(I)=RFUEL*FLUXO*(ALOG(ROCLAD/RC)/CCOND+1./
     *        (ROCLAD*HFO))+COOLTO
 100    CONTINUE
      CITO=TC(1)
C
C  Calculate the fuel temperature.
C
      CIO(NPFUEL)=FICON(FLUXO/GAPCNO+TC(1))
      HCSO=FIHCAP(FLUXO/GAPCNO+TC(1))
      XIFAC=QO*RFUEL*RFUEL/4.
      CIO(1)=XIFAC*(1.+A/4.)+CIO(NPFUEL)
      TF(1)=FICON(-CIO(1))
      TF(NPFUEL)=FICON(-CIO(NPFUEL))
      NM=NPFUEL-1
      DO 200 I=2,NM
        RF=FLOAT(I-1)/FLOAT(NM)
        CIO(I)=XIFAC*(1.-RF*RF+A*(1.-RF**4)/4.)+CIO(NPFUEL)
 200    TF(I)=FICON(-CIO(I))
      RETURN
      END
C
      SUBROUTINE HANKI(PEAKAV,POWERL,XIRO,CIPO)
C
C  Subroutine to calculate the Hankel transformed
C  integral conductivity, integral heat capacity and
C  the heat source derived analytically for a
C  quadratic heat source.
C
      COMMON RFUEL,NP,NS,S(15),TIME(15),PSI(10),
     *       XJ1R(10),CONINT(15,10)
      COMMON /HANVAR/ HSO,HSP(10),HCSO,HCPO(10)
      DIMENSION CIPO(10)
      DATA PI/3.14159265/
      DATA C1/143895./,C2/114.284/
      QO=POWERL*(2-PEAKAV)/(PI*RFUEL*RFUEL)
      A=(PEAKAV-1.)/(1.-PEAKAV/2.)
      HSO=QO*(1.+A)
C
C  Hankel transformed heat source
C
      CQ=-4.*QO*A/RFUEL
      DO 100 IP=1,NP
 100    HSP(IP)=CQ*XJ1R(IP)/PSI(IP)**3
C
C  Hankel transformed integral conductivity
C
      DO 200 IP=1,NP
 200   CIPO(IP)=Q0*((1+A)*RFUEL/PSI(IP)**3
     * - 4.*A/(RFUEL*PSI(IP)**5))*XJ1R(IP)
C
C  Hankel transformed integral heat capacity
C
      CU1=C1+2.*C2*XIRO
      CU2=C2*Q0*Q0
      P1=(A*A + 4.*A + 4.)*RFUEL**3/8.
      P2=(17.*A*A + 32.*A + 8.)*RFUEL/2.
      P3=A*(144.*A + 72.)/RFUEL
      P4=576.*A*A/RFUEL**3
      DO 300 IP=1,NP
        HCPO(IP)=CU1*CIPO(IP) + CU2*(-P1/PSI(IP)**3
     *  +P2/PSI(IP)**5-P3/PSI(IP)**7+P4/PSI(IP)**9)*XJ1R(IP)
 300    CONTINUE
      RETURN
      END
C
C
      SUBROUTINE WITER(ITERNO,XICLO,FLUXT,CITT,FST)
C
C  Subroutine to print each iteration results.
C
      COMMON RFUEL,NP,NS,S(15),TIME(15),PSI(10),XJ1R(10),
     *       CONINT(15,10)
      COMMON /DFC/ DESFUN(15,10),A(10),B(10),CIST(15)
      COMMON /LAPVAR/ COOLTS(15),FLUXS(15),CITS(15),
     *                HCSS(15),HSP(15)
      DIMENSION XICLS(15),XICLT(15),TCLT(15),FLUXT(1),
     *          CITT(1),FST(1)
      WRITE(6,101)ITERNO
      WRITE(6,102)
      WRITE(6,106)(A(I),I=1,NP)
      WRITE(6,104)
      WRITE(6,106)(B(I),I=1,NP)
      WRITE(6,105)
      DO 10 IS=1,NS
  10    WRITE(6,103) IS,(DESFUN(IS,IP),IP=1,NP)
      WRITE(6,107)
      DO 20 IS=1,NS
  20    WRITE(6,103) IS,(CONINT(IS,IP),IP=1,NP)
C
C  Calculate centerline temperature.
C
      CALL CLICS(XICLS)
      CALL LTINV(NS,XICLS,XICLT)
      DO 30 IS=1,NS
  30    XICLT(IS)=XICLT(IS) + CIST(IS)
      CALL SMOOTH(NS,XICLO,XICLT)
      CALL FICONV(NS,15,-1,TCLT,XICLT)
      WRITE(6,108)
      WRITE(6,109)(S(I),FLUXS(I),CITS(I),XICLS(I),I=1,NS)
      WRITE(6,110)
      WRITE(6,111)(TIME(I),FLUXT(I),CITT(I),XICLT(I),FST(I)
     *              ,TCLT(I),I=1,NS)
  101 FORMAT(//5X,18('*')/5X,'* ITERATION - ',I2,
     *' *'/5X,18('*'))
  102 FORMAT(/' NEW ESTIMATE OF DESCRIBING FUNCTION
     *CONSTANTS -',/' COEFFICIENTS (IP=1,NP)')
  103 FORMAT(' ROW ',I2,':   ',10E12.4)
  104 FORMAT(' EXPONENTS (IP=1,NP)')
  105 FORMAT(/' ***** MATERIAL DESCRIBING FUNCTION MATRIX',
     * ' (NS BY NP) *****')
  106 FORMAT(1X,10G13.6)
  107 FORMAT(/' ***** DOUBLE TRANSFORMED INTEGRAL CONDUCTIVITY
     * MATRIX (NS BY NP) *****')
  108 FORMAT(/' ***** LAPLACE DOMAIN VARIABLES *****'
     * /4X,'S',12X,'FLUXS',7X,'CITS',8X,'XICLS'/1X,48('*'))
  109 FORMAT(1X,G12.5,3E12.5)
  110 FORMAT(/' ***** TIME DOMAIN VARIABLES *****'/2X,'TIME',7X,
     *'FLUXT',7X,'CITT',8X,'XICLT',7X,'FST',9X,'TCLT'/1X,68('*'))
  111 FORMAT(1X,F8.3,5G12.5)
      RETURN
      END
C
C
      SUBROUTINE FTEMP(NPTF,CIO)
C
C  Calculates fuel temperature field.
C
      COMMON RFUEL,NP,NS,S(15),TIME(15),PSI(10),XJ1R(10),CONINT(15,10)
      COMMON /DFC/ CM1(170),CIST(15)
      COMMON /TEMPER/ TEMP(15,10)
      DIMENSION TEM(15),XIRRS(15),XIRTOT(15),XIRRT(15),CIO(1)
      EQUIVALENCE (TEM,XIRRS),(XIRTOT,XIRRT)
      DO 50 IR=1,NPTF
        RR=RFUEL*FLOAT(IR-1)/FLOAT(NPTF-1)
        DO 10 IS=1,NS
   10     XIRRS(IS)=0.
        DO 20 IP=1,NP
          BESSEL=2.*BJ(PSI(IP)*RR,0)/(RFUEL*XJ1R(IP))**2
          DO 20 IS=1,NS
   20       XIRRS(IS)=XIRRS(IS) + CONINT(IS,IP)*BESSEL
        CALL LTINV(NS,XIRRS,XIRRT)
        DO 30 IS=1,NS
   30     XIRTOT(IS)=CIST(IS)+XIRRT(IS)
        CALL SMOOTH(NS,CIO(IR),XIRTOT)
        CALL FICONV(NS,15,-1,TEM,XIRTOT)
        DO 40 IS=1,NS
   40     TEMP(IS,IR)=TEM(IS)
```

```
 50     CONTINUE
      RETURN
      END
C
C
      SUBROUTINE WTEMP(ITERWR,ITERNO,MITNO,NPFUEL,FUELTO)
C
C  Writes the converged material describing function,
C  integral conductivity, and transient fuel temperature
C  matrices (NS x NP)
C
      COMMON RFUEL,NP,NS,S(15),TIME(15),PSI(10),XJ1R(10),CONINT(15,10)
      COMMON /DFC/ DESFUN(15,10)
      COMMON /TEMPER/ TEMP(15,10)
      DIMENSION RAD(10),FUELTO(1)
      IF(ITERWR.EQ.1)GO TO 20
      IF(ITERNO.GT.MITNO)WRITE(6,450)MITNO
      IF(ITERNO.LE.MITNO)WRITE(6,500)ITERNO
      WRITE(6,100)
      DO 10 IS=1,NS
 10     WRITE(6,200) IS,(DESFUN(IS,IP),IP=1,NP)
      WRITE(6,150)
      DO 15 IS=1,NS
 15     WRITE(6,200) IS,(CONINT(IS,IP),IP=1,NP)
 20   WRITE(6,250)
      DO 30 IR=1,NPFUEL
 30     RAD(IR)=RFUEL*FLOAT(IR-1)/FLOAT(NPFUEL-1)
      WRITE(6,300)(RAD(I),I=1,NPFUEL)
      WRITE(6,350)
      TSS=0.
      WRITE(6,400)TSS,(FUELTO(J),J=1,NPFUEL)
      DO 40 IS=1,NS
        JS=NS-IS+1
 40     WRITE(6,400)TIME(JS),(TEMP(JS,J),J=1,NPFUEL)
      RETURN
100   FORMAT(/5X,45('*')/5X,'* FINAL MATERIAL DESCRIBING
     * FUNCTION MATRIX *'/5X,45('*'))
150   FORMAT(/5X,38('*')/5X,'* FINAL INTEGRAL CONDUCTIVITY
     * MATRIX *',/5X,38('*'))
200   FORMAT(' ROW ',I2,':   ',10E12.4)
250   FORMAT(//5X,39('*')/5X,'* FUEL TRANSIENT TEMPERATURE',
     *' PROFILES *',/5X,39('*'))
300   FORMAT(' RADIUS (M) ',10(F9.6,3X))
350   FORMAT(' TIME (SEC)')
400   FORMAT(1X,F9.4,2X,10G12.5)
450   FORMAT(//5X,'THE SOLUTION WAS NOT CONVERGED AFTER ',
     *I2,' ITERATIONS.')
500   FORMAT(//5X,'THE SOLUTION WAS CONVERGED AFTER ',I2,
     * ' ITERATIONS.')
      END
```

```
C
C
      SUBROUTINE CTEMP(NPCLAD,RICLAD,ROCLAD,ALPHA,FLUXO,
     *                 COOLTO,HFILM,SSCT)
C
C  Calculate cald transient temperature field from converged
C  Laplace domain heat flux.
C
      COMMON RFUEL,NP,NS,S(15)
      COMMON /CCLAD/ CM1(30),CMAT(2,2,15)
      COMMON /LAPVAR/ COOLTS(15),FLUXS(15)
      COMMON /TEMPER/ CLDTEM(15,10)
      DIMENSION CTEMPS(15),CTEMPT(15),SSCT(1)
      DO 100 IR=1,NPCLAD
        RFUEL=RICLAD + (ROCLAD-RICLAD)*FLOAT(IR-1)/FLOAT(NPCLAD-1)
        DO 50 IS=1,NS
          IF(S(IS).LT.0.)GO TO 10
          P=SQRT(S(IS)/ALPHA)
          BES1=BI(P*RFUEL,0)
          BES2=BK(P*RFUEL,0)
          GO TO 20
   10     Q=SQRT(-S(IS)/ALPHA)
          BES1=BJ(Q*RFUEL,0)
          BES2=BY(Q*RFUEL,0)
   20     DFS=FLUXS(IS)-FLUXO/S(IS)
          TCOOLS=(COOLTS(IS)-COOLTO/S(IS))*(-HFILM)
          CTEMPS(IS)=SSCT(IR)/S(IS)
     *       +BES1*(CMAT(1,1,IS)*DFS+CMAT(1,2,IS)*TCOOLS)
     *       +BES2*(CMAT(2,1,IS)*DFS+CMAT(2,2,IS)*TCOOLS)
   50     CONTINUE
        CALL LTINV(NS,CTEMPS,CTEMPT)
        CALL SMOOTH(NS,SSCT(IR),CTEMPT)
        DO 60 IS=1,NS
   60     CLDTEM(IS,IR)=CTEMPT(IS)
  100   CONTINUE
      RETURN
      END
C
C
      SUBROUTINE WCTEMP(NPCLAD,RICLAD,ROCLAD,SSCT)
C
C  Write clad temperature field.
C
      COMMON RFUEL,NP,NS,S(15),TIME(15)
      COMMON /TEMPER/ CLDTEM(15,10)
      DIMENSION RAD(10),SSCT(1)
      WRITE(6,100)
      DO 10 IR=1,NPCLAD
        RAD(IR)=RICLAD +
     *     (ROCLAD-RICLAD)*FLOAT(IR-1)/FLOAT(NPCLAD-1)
  10    CONTINUE
      WRITE(6,200)(RAD(IR),IR=1,NPCLAD)
      WRITE(6,300)
      TSS=0.
      WRITE(6,400) TSS,(SSCT(IR),IR=1,NPCLAD)
      DO 20 IS=1,NS
        JS=NS-IS+1
  20    WRITE(6,400)TIME(JS),(CLDTEM(JS,IR),IR=1,NPCLAD)
      RETURN
 100  FORMAT(//5X,39('*')/5X,'* CLAD TRANSIENT
     *TEMPERATURE PROFILES *',/5X,39('*'))
 200  FORMAT(' RADIUS (M) ',10(F9.6,3X))
 300  FORMAT(' TIME (SEC)')
 400  FORMAT(1X,F9.4,2X,10G12.5)
      END
C
C
      FUNCTION FTHDIF(T)
C
C  Uranium dioxide thermal diffusivity as a function of
C  temperature.
C
      RHO=10400.
      FTHDIF=FTHCON(T)/(RHO*FHCAP(T))
      RETURN
      END
C
C
      FUNCTION FHCAP(T)
C
C  Uranium dioxide heat capacity as a function of
C  temperature.
C
C         T - TEMPERATURE (K)
C         FHCAP - HEAT CAPACITY (J/KG-K)
C
      IF((T.LT.273.).OR.(T.GT.3113.))WRITE(6,20)T
      EX=EXP(535.285/T)
      FHCAP=(8.5005E+07)*EX/(T*T*(EX-1)**2)+0.02432*T
     *  +(1.6588E+12)*EXP(-18968./T)/(T*T)
      RETURN
  20  FORMAT(' ***** WARNING *****  TEMP. EXCEEDS
     *"FHCAP" LIMITS -- T=',E11.4)
      END
C
C
      FUNCTION FIHCAP(T)
C
C  Uranium dioxide integral heat capacity
C  as a function of temperature (for zero reference
C  temperature).
C
      IF (T.LE.273.15)THEN
        FIHCAP=0.
      ELSE
        IF(T.GT.3113.)WRITE(6,20)T
        H=10248./(EXP(535.285/T)-1.)+7.8473E-04*T*T
     *   + 5.6437E+06*EXP(-18968/T)-1739.4
        RHO=10400.
        FIHCAP=15.496*H*RHO
      ENDIF
      RETURN
   20 FORMAT(' ***** WARNING *****  TEMP. EXCEEDS FIHCAP
     * LIMITS -- T=',E11.4)
      END
C
C
      SUBROUTINE FIHCV(N,NDIM,T,FICAP)
C
C  Uranium dioxide integral heat capacity as a function
C  of time.  This subroutine accepts a vector of values
C  for temperature and returns a vector of values for
C  integral heat capacity.
C
      DIMENSION T(NDIM),FICAP(NDIM)
      DO 10 I=1,N
        IF(T(I).LE.273.15)THEN
          FICAP(I)=0.
          GO TO 10
        ELSE
          IF(T(I).GT.3113.)WRITE(6,20)T(I)
          H=10248./(EXP(535.285/T(I))-1.)+7.8473E-04
     *    *T(I)*T(I) + 5.6437E+06*EXP(-18968/T(I))-1739.4
          RHO=10400.
          FICAP(I)=15.496*H*RHO
        ENDIF
   10   CONTINUE
      RETURN
   20 FORMAT(' ***** WARNING *****  TEMP. EXCEEDS FIHCV
     * LIMITS -- T=',E11.4)
      END
C
C
      FUNCTION FTHCON(T)
C
C  Uranium dioxide thermal conductivity as a function of
C  temperature.
C
      TC=T-273.15
      IF(TC.LE.1650.)THEN
```

```
          IF(TC.LE.0.)WRITE(6,20)T
          FTHCON=4040./(464.+TC)+0.01216*EXP(0.001867*TC)
        ELSE
          IF(TC.GT.2840.)WRITE(6,20)T
          FTHCON=1.91+0.01216*EXP(0.001867*TC)
        ENDIF
        RETURN
   20   FORMAT(' ***** WARNING *****  TEMP. EXCEEDS "FTHCON"
       * LIMITS -- T=',E11.4)
        END
C
C
        FUNCTION FICON(TK)
C
C  Uranium dioxide integral conductivity as
C  as a function of temperature.
C
        IF(TK.GE.0)THEN
          T=TK-273.15
          IF(T.LE.0.)THEN
            FICON=0.
          ELSE
            IF (T.LT.1650.)THEN
              FICON=4040.*ALOG(464.+T)+
       *            6.513*EXP(.001867*T)-24812.
            ELSE
              IF(T.GT.2840.)WRITE(6,20)TK
              FICON=1.91*T+6.513*EXP(.001867*T)+2968.4
            ENDIF
          ENDIF
        ELSE
          CI=ABS(TK)/100.
          FICON=464.8*EXP(CI/40.4)-212.611+1.340*CI
       *        -0.0351*CI*CI
          IF(FICON.LE.1923.15) RETURN
          FICON=-2396.67+91.22*CI-0.355*CI*CI
        ENDIF
        RETURN
   20   FORMAT(" ***** WARNING *****  TEMP. EXCEEDS
       * FICON LIMITS -- T=",E11.4)
        END
C
C
        SUBROUTINE FICONV(N,NDIM,NOPT,T,I)
C
C  Uranium dioxide integral conductivity is computed for
C  NOPT greater than or equal to zero.  For NOPT less than
C  zero the inverse is computed.  The input is the
C  temperature vector and the output is the integral
C  conductivity vector.
```

```
C
      DIMENSION T(NDIM),I(NDIM)
      REAL I, IJ
      IF(NOPT.GE.0)THEN
        DO 5 J=1,N
          TJ=T(J)-273.15
          IF(TJ.LE.0.)THEN
            I(J)=0.
          ELSE
            IF (TJ.LT.1650.)THEN
              I(J)=4040.*ALOG(464.+TJ)+
     *           6.513*EXP(.001867*TJ)-24812.
            ELSE
              IF(TJ.GT.2840.)WRITE(6,20)T(J)
              I(J)=1.91*TJ+6.513*EXP(.001867*TJ)+2968.4
            ENDIF
          ENDIF
    5     CONTINUE
      ELSE
        DO 10 J=1,N
          IJ=I(J)/100.
          T(J)=464.8*EXP(IJ/40.4)-212.611+
     *         1.340*IJ-0.0351*IJ*IJ
          IF(T(J).LE.1923.15) GO TO 10
          T(J)=-2396.67+91.22*IJ-0.355*IJ*IJ
          IF(T(J).GT.3113.15)WRITE(6,20)T(J)
   10     CONTINUE
      ENDIF
      RETURN
   20 FORMAT(' ***** WARNING *****  TEMP. EXCEEDS
     * FICONV LIMITS -- T=',E11.4)
      END
C
C
      FUNCTION CLADSH (CTEMP)
C
C  Zircaloy specific heat as a function of temperature.
C
      DIMENSION CPDATA (26)
      DATA CPDATA/281.,  300., 302.,  400., 331.,  640.,
     *            375., 1090., 502., 1093., 590., 1113.,
     *            615., 1133., 719., 1153., 816., 1173.,
     *            770., 1213., 619., 1213., 469., 1233.,
     *            356., 1248./
      IF(CTEMP.LT.1248.)THEN
        CLADSH=FINDY(CPDATA,CTEMP,13,1)
      ELSE
        CLADSH=356.
      ENDIF
      RETURN
      END
C
C
      FUNCTION CTHCON(CTEMP)
C
C  Zircaloy thermal conductivity as a function of temperatre.
C
      CTHCON=7.511+CTEMP*(2.088E-2+CTEMP*(-1.45E-5+CTEMP
     1   *7.668E-09))
      RETURN
      END
C
C
      SUBROUTINE LAPMAT(TMAX)
C
C  Determine the explicit matrix form for Laplace transform
C  and its inverse.
C
      COMMON RAD,NHANK,N,S(15),TIME(15),CM(170)
      COMMON /LAPLC/ SHIFT,A(15,15),EMAT(15,15)
      DIMENSION RFUEL(15),W(15),C(17,17),B(15,15),P(15)
      DOUBLE PRECISION RFUEL,W,C,X1,X2,X,CONINT,B,PZERO,
     * PREV2,PREV1,Z1,Z2,Z3,PREV3,P,A,DENOM
      DATA RFUEL/6.003740989757422 3D-03, 3.1363303799647246D-02,
     *           7.5896708294786505D-02, 1.3779113431991500D-01,
     *           2.1451391369573062D-01, 3.0292432646121834D-01,
     *           3.9940295300128272D-01, 5.0000000000000000D-01,
     *           6.0059704699871723D-01, 6.9707567353878161D-01,
     *           7.8548608630426927D-01, 8.6220886568008490D-01,
     *           9.2410329170521343D-01, 9.6863669620035270D-01,
     *           9.9399625901024251D-01/
      DATA W/1.5376620998058628D-02, 3.5183023744054046D-02,
     *        5.3579610233585954D-02, 6.9785338963077129D-02,
     *        8.3134602908496945D-02, 9.3080500007781079D-02,
     *        9.9215742663555754D-02, 1.0128912096278060D-01,
     *        9.9215742663555754D-02, 9.3080500007781079D-02,
     *        8.3134602908496945D-02, 6.9785338963077129D-02,
     *        5.3579610233585954D-02, 3.5183023744054046D-02,
     *        1.5376620998058628D-02/
      DO 5 KIN=1,17
      DO 5 JIN=1,17
    5   C(KIN,JIN)=0.0D0
C
C  Shift time scale by TMAX.
C
      SHIFT=-TMAX/ALOG(SNGL(RFUEL(1)))
      TIME(1)=TMAX
      S(1)=1./SHIFT
      DO 10 I=2,N
        S(I)=FLOAT(I)/SHIFT
```

```
  10     TIME(I)=-SHIFT*ALOG(SNGL(RFUEL(I)))
C
C  Determine explicit matrix form.
C
      DO 12 J=1,N
        EMAT(1,J)=W(J)
        RIM=1.
        DO 12 I=2,N
          RIM=RIM*RFUEL(J)
  12      EMAT(I,J)=RIM*W(J)
      NPLUS=N+1
      C(1,1)=1.0D0
      DO 20 M=1,NPLUS
        MP=M+1
        DO 15 N=MP,NPLUS
          X1=N+M-2
          X2=N-M
  15      C(N,M)=C(N-1,M)*X1/X2
        N=MP
        X=M
  20    C(N,N)=-C(N,N-1)*2.0D0/X
      DO 30 I=1,16
        DO 30 M=1,NPLUS
  30      C(I,M)=C(I+1,M)
      N=NPLUS-1
      DO 100 I=1,N
        CONINT=RFUEL(I)
        K=N
        B(I,K)=C(N,N+1)
        DO 50 II=2,N
          K=K-1
  50      B(I,K)=C(N,K+1)+CONINT*B(I,K+1)
 100    CONTINUE
      NMINUS=N-1
      DO 180 I=1,N
        X=RFUEL(I)
        PZERO=1.0D0 - 2.0D0*X
        PREV2=PZERO
        PREV1=1.0D0
        DO 150 M=2,NMINUS
          Z1=M-1
          Z2=2*M-1
          Z3=M
          PREV3=-(Z1*PREV1-Z2*PZERO*PREV2)/Z3
          PREV1=PREV2
 150      PREV2=PREV3
        Z1=N
        Z2=2.0D0*X*(X-1.0D0)
 180    P(I)=PREV3*Z1/Z2
      DO 220 M=1,N
```

```
        DENOM=W(M)*P(M)
        DO 220 K=1,N
 220      A(M,K)=B(M,K)/DENOM
      RETURN
      END
C
C
      SUBROUTINE LTRANS(N,FR,FT)
C
C  Performs Laplace transformation.
C
      COMMON /LAPLC/ SHIFT,A(15,15),B(15,15)
      DIMENSION FR(15),FT(15)
      DOUBLE PRECISION A, SUM
      DO 20 I=1,N
        SUM=0.0D0
        DO 10 J=1,N
  10      SUM=SUM + B(I,J)*FR(J)
  20    FT(I)=SNGL(SUM)*SHIFT
      RETURN
      END
C
C
      SUBROUTINE LTINV(N,FT,FR)
C
C  Performs inverse Laplace transform.
C
      COMMON /LAPLC/ SHIFT,A(15,15),B(15,15)
      DIMENSION FT(15),FR(15)
      DOUBLE PRECISION A, SUM
      DO 20 I=1,N
        SUM=0.D0
        DO 10 J=1,N
  10      SUM=SUM + A(I,J)*FT(J)
  20    FR(I)=SNGL(SUM)/SHIFT
      RETURN
      END
C
C
      SUBROUTINE SMOOTH(N,FO,F)
C
C  Smooths Laplace transform inverse by filtering.
C
      DIMENSION F(N)
      NM=N-1
      FM2=(1.25*F(1)+F(2))/2.25
      FM1=(F(1)+2.*F(2)+F(3))/4.
      DO 10 I=3,NM
        F(I-2)=FM2
        FM2=FM1
```

```
         FM1=(F(I-1)+2.*F(I)+F(I+1))/4.
   10    CONTINUE
       F(N)=(2.*F0+F(N)+F(N-1))/4.
       F(N-1)=FM1
       F(N-2)=FM2
       RETURN
       END
C
C
       SUBROUTINE CLADCO(RICLAD,ROCLAD,CCOND,ALPHA,HFILM)
C
C  Determines constants needed for clad solution.
C
       COMMON RFUEL,NP,NS,S(15)
       COMMON /CCLAD/ XKORI(15),XIORI(15),CLDMAT(2,2,15)
       DIMENSION XJORI(15),XYORI(15),B(2,2)
       EQUIVALENCE (XIORI,XJORI),(XKORI,XYORI)
       REAL CCOND
       RIR=RICLAD/RFUEL
       DO 100 IS=1,NS
         IF(S(IS).LT.0.) GO TO 20
         P=SQRT(S(IS)/ALPHA)
         PK=P*CCOND
         XIORI(IS)=BI(P*RICLAD,0)
         XKORI(IS)=BK(P*RICLAD,0)
         B(1,1)=-PK*BI(P*RICLAD,1)*RICLAD/RFUEL
         B(2,1)=-PK*BI(P*ROCLAD,1) - HFILM*BI(P*ROCLAD,0)
         B(1,2)=PK*BK(P*RICLAD,1)*RICLAD/RFUEL
         B(2,2)=PK*BK(P*ROCLAD,1) - HFILM*BK(P*ROCLAD,0)
         GO TO 40
   20    Q=SQRT(-S(IS)/ALPHA)
         QK=Q*CCOND
         XJORI(IS)=BJ(Q*RICLAD,0)
         XYORI(IS)=BY(Q*RICLAD,0)
         B(1,1)=QK*BJ(Q*RICLAD,1)*RIR
         B(2,1)=QK*BJ(Q*ROCLAD,1) - HFILM*BJ(Q*ROCLAD,0)
         B(1,2)=QK*BY(Q*RICLAD,1)*RIR
         B(2,2)=QK*BY(Q*ROCLAD,1) - HFILM*BY(Q*ROCLAD,0)
   40    DET=B(1,1)*B(2,2) - B(1,2)*B(2,1)
         IF(DET.NE.0.) GO TO 50
         WRITE(6,200)
         STOP
   50    CLDMAT(1,1,IS)=B(2,2)/DET
         CLDMAT(2,1,IS)=-B(2,1)/DET
         CLDMAT(1,2,IS)=-B(1,2)/DET
         CLDMAT(2,2,IS)=B(1,1)/DET
  100    CONTINUE
       RETURN
  200  FORMAT(' *** SINGULAR MATRIX IN CLADCO ***')
       END
```

```
C
C
      SUBROUTINE CIRTS(CITO,FLUXO,COOLTO,HFILM)
C
C  Calculates inner clad temperature in Laplace domain.
C
      COMMON RFUEL,NP,NS,S(15)
      COMMON /CCLAD/ XBES2(15),XBES1(15),CLDMAT(2,2,15)
      COMMON /LAPVAR/ COOLTS(15),FLUXS(15),CITS(15),
     *                HCSS(15),HSS(15)
      DO 10 IS=1,NS
        DFS=FLUXS(IS) - FLUXO/S(IS)
        TCOOLS=(COOLTS(IS) - COOLTO/S(IS))*(-HFILM)
        CITS(IS)=CITO/S(IS)
     *  + XBES1(IS)*(CLDMAT(1,1,IS)*DFS
     *  + CLDMAT(1,2,IS)*TCOOLS)
     *  + XBES2(IS)*(CLDMAT(2,1,IS)*DFS
     *  + CLDMAT(2,2,IS)*TCOOLS)
  10    CONTINUE
      RETURN
      END
C
C
      FUNCTION BJ(X,N)
C
C  Zeroth and first order ordinary Bessel functions of
C  the first kind.
C
      IF(X.LT.0.)THEN
        WRITE(6,20)
        STOP
      ELSE
        IF(N.EQ.0)THEN
          IF(X.LE.3)THEN
            Y=X*X/9.
            BJ=(((((.00021*Y-.0039444)*Y+.0444479)*Y-
     *          .3163866)*Y+1.2656208)*Y-2.2499997)*Y+1.
          ELSE
            Y=3./X
            FO=(((((.00014476*Y-.00072805)*Y+.00137237)
     *      *Y-.00009512)*Y-.0055274)*Y-.00000077)*Y+.79788456
            THETO=(((((.00013558*Y-.00029333)*Y-.00054125)*Y
     *      +.00262573)*Y-.00003954)*Y-.04166397)*Y-.78539816+X
            BJ=FO*COS(THETO)/SQRT(X)
          ENDIF
        ELSE
          IF(X.LE.3.)THEN
            Y=X*X/9.
            BJ=X*((((((.00001109*Y-.00031761)*Y+.00443319)*Y
     *      -.03954289)*Y+.21093573)*Y-.56249985)*Y+.5)
```

```
          ELSE
            Y=3./X
            F1=(((((-.00020033*Y+.00113653)*Y-.00249511)*Y
     *      +.00017105)*Y+.01659667)*Y+.00000156)*Y+.79788456
            THET1=(((((-.00029166*Y+.00079824)*Y+.00074348)*Y
     *      -.00637879)*Y+.0000565)*Y+.12499612)*Y-2.35619449+X
            BJ=F1*COS(THET1)/SQRT(X)
          ENDIF
        ENDIF
      ENDIF
      RETURN
   20 FORMAT('*** ERROR -- NEGATIVE BJ ARGUMENT')
      END
C
C
      FUNCTION BY(X,N)
C
C  Zeroth and first order ordinary Bessel functions of
C  the second kind.
C
      DATA PI/3.14159265/
      IF(X.LT.0.)THEN
        WRITE(6,20)
        STOP
      ELSE
        IF(N.EQ.0)THEN
          IF(X.LE.3)THEN
            Y=X*X/9.
            BY=(((((-.00024846*Y+.00427916)*Y-.04261214)*Y
     *         +.25300117)*Y-.74350384)*Y+.60559366)*Y
     *         +.36746691+2.*ALOG(X/2.)*BJ(X,0)/PI
          ELSE
            Y=3./X
            F0=(((((.00014476*Y-.00072805)*Y+.00137237)*Y
     *         -.00009512) *Y-.0055274)*Y-.00000077)*Y
     *         +.79788456
            THETO=(((((.00013558*Y-.00029333)*Y-.00054125)*Y
     *              +.00262573)*Y-.00003954)*Y-.04166397)*Y
     *              -.78539816 + X
            BY=F0*SIN(THETO)/SQRT(X)
          ENDIF
        ELSE
          IF(X.LE.3.)THEN
            Y=X*X/9.
            BY=((((((.0027873*Y-.0400976)*Y+.3123951)*Y
     *         -1.3164827)*Y+2.1682709)*Y+.2212091)*Y
     *         -.6366198)/X+2.*ALOG(X/2.)*BJ(X,1)/PI
          ELSE
            Y=3./X
            F1=(((((-.00020033*Y+.00113653)*Y-.00249511)*Y
     *          +.00017105)*Y+.01659667)*Y+.00000156)*Y+.79788456
            THET1=(((((-.00029166*Y+.00079824)*Y+.00074348)*Y
     *             -.00637879)*Y+.0000565)*Y+.12499612)*Y
     *             -2.35619449 + X
            BY=F1*SIN(THET1)/SQRT(X)
          ENDIF
        ENDIF
      ENDIF
      RETURN
   20 FORMAT('*** ERROR -- NEGATIVE BY ARGUMENT')
      END
C
C
      FUNCTION BI(X,N)
C
C  Zeroth and first order modified Bessel functions of
C  the first kind.
C
      IF(X.LT.0.)THEN
        WRITE(6,20)
        STOP
      ELSE
        T=X/3.75
        IF(N.EQ.0)THEN
          IF(X.LE.3.75)THEN
            TS=T*T
            BI=(((((.0045813*TS+.0360768)*TS+.2659732)*TS
     *         +1.2067492)*TS+3.0899424)*TS+3.5156229)*TS+1.
          ELSE
            TI=1./T
            XI0=((((((((.00392377*TI-.01647633)*TI
     *         +.02635537)*TI-.02057706)*TI+.00916281)*TI
     *         -.00157565)*TI+.00225319)*TI+.01328592)*TI
     *         +.39894228
            BI=XI0*EXP(X)/SQRT(X)
          ENDIF
        ELSE
          IF(X.LE.3.75)THEN
            TS=T*T
            BI=X*((((((.00032411*TS+.00301532)*TS
     *         +.02658733)*TS+.15084934) *TS+.51498869)*TS
     *         +.87890594)*TS+.5)
          ELSE
            TI=1./T
            XI1=((((((((-.00420059*TI+.01787654)*TI
     *          -.02895312)*TI+.02282967) *TI-.01031555)*TI
     *          +.00163801)*TI-.00362018)*TI-.03988024)*TI
     *          +.39894228
            BI=XI1*EXP(X)/SQRT(X)
          ENDIF
          ENDIF
        ENDIF
        RETURN
   20   FORMAT('*** ERROR -- NEGATIVE BI ARGUMENT')
        END
C
C
        FUNCTION BK(X,N)
C
C  Zeroth and first order modified Bessel functions of
C  the second kind.
C
        IF(X.LT.0.)THEN
          WRITE(6,20)
          STOP
        ELSE
          T=X/2.
          IF(N.EQ.0)THEN
            IF(X.LE.2.)THEN
              TS=T*T
              BK=(((((.0000074*TS+.00010750)*TS+.00262698)*TS
     *           +.0348859)*TS+.23069756)*TS+.42278420)*TS
     *           -.57721566-ALOG(T)*BI(X,0)
            ELSE
              TI=1./T
              XK0=(((((.00053208*TI-.00251540)*TI+.00587872)*TI
     *            -.01062446)*TI+.02189568)*TI-.07832358)*TI
     *            +1.25331414
              BK=XK0*EXP(-X)/SQRT(X)
            ENDIF
          ELSE
            IF(X.LE.2)THEN
              TS=T*T
              BK=((((((-.00004686*TS-.00110404)*TS-.01919402)*TS
     *           -.18156897)*TS-.67278579)*TS+.15443144)*TS+1.)/X
     *           + ALOG(T)*BI(X,1)
              RETURN
            ELSE
              TI=1./T
              XK1=(((((-.00068245*TI+.00325614)*TI-.00780353)*TI
     *            +.01504268) *TI-.0365562)*TI+.23498619)*TI
     *            +1.25331414
              BK=XK1*EXP(-X)/SQRT(X)
            ENDIF
          ENDIF
        ENDIF
        RETURN
   20   FORMAT('*** ERROR -- NEGATIVE BK ARGUMENT')
        END
C
```

```
C
      SUBROUTINE CLICS(DICLS)
C
C  Calculates centerline value of integral conductivity
C  in the Laplace domain (relative to fuel surface value).
C
      COMMON RFUEL,NP,NS,CM1(40),XJ1R(10),CONINT(15,10)
      DIMENSION DICLS(15)
      DO 10 IS=1,NS
        SUM=0.0
        DO 5 IP=1,NP
    5     SUM=SUM + CONINT(IS,IP)/XJ1R(IP)**2
   10   DICLS(IS)=2.0*SUM/(RFUEL*RFUEL)
      RETURN
      END
C
C
      SUBROUTINE SOLVE(A,B,W1,IOPT,NRHS,M,N,MDIM,NDIM,NDRHS)
C
C  Solves an M x N system of linear equations in a least
C  squares sense using Householder's reduction method.
C
      DIMENSION A(MDIM,NDIM),B(MDIM,NDRHS),W1(NDIM)
      IF(IOPT.EQ.2) GO TO 30
C
C  Factor the A matrix
C
      NELM=N-1
      IF(M.GT.N) NELM=N
      DO 20 K=1,NELM
        SIGN=1.
        IF(A(K,K).GT.0.) SIGN=-1.
        KP=K+1
C
C  Calculate the magnitude of the column to be eliminated
C
        V0=0.
        DO 5 I=K,M
    5     V0=V0 + A(I,K)*A(I,K)
        V0=SIGN*SQRT(V0)
        IF(V0.NE.0.) GO TO 6
        WRITE(6,100)
        STOP
C
C  Calculate the normalized W unit vector and save all but
C  W1 in the lower triangle of A.
C
    6   W0=SQRT(2.*V0*(V0-A(K,K)))
        W1(K)=(A(K,K)-V0)/W0
        A(K,K)=V0
        DO 10 I=KP,M
  10      A(I,K)=A(I,K)/WO
C
C  Apply Housholder's reduction to the remaining
C  columns of A.
C
        IF(KP.GT.N) GO TO 25
        DO 20 J=KP,N
C
C  Inner product and elimination
C
          PINR=W1(K)*A(K,J)
          DO 15 I=KP,M
  15        PINR=PINR + A(I,K)*A(I,J)
          A(K,J)=A(K,J) - 2.*PINR*W1(K)
          DO 20 I=KP,M
  20        A(I,J)=A(I,J) - 2.*PINR*A(I,K)
  25  IF(IOPT.EQ.1) RETURN
C
C  Transform the B vectors
C
  30  DO 40 K=1,NELM
        KP=K+1
        DO 40 J=1,NRHS
          PINR=W1(K)*B(K,J)
          DO 35 I=KP,M
  35        PINR=PINR + A(I,K)*B(I,J)
          B(K,J)=B(K,J) - 2.*PINR*W1(K)
          DO 40 I=KP,M
  40        B(I,J)=B(I,J) - 2.*PINR*A(I,K)
C
C  Solve QAX=QB by back substitution
C
      DO 45 JRHS=1,NRHS
  45    B(N,JRHS)=B(N,JRHS)/A(N,N)
      DO 60 JRHS=1,NRHS
        DO 60 IOP=2,N
          I=N-IOP+1
          SUM=B(I,JRHS)
          IP=I+1
          DO 50 J=IP,N
  50        SUM=SUM - B(J,JRHS)*A(I,J)
  60      B(I,JRHS)=SUM/A(I,I)
      RETURN
 100  FORMAT(' *** "SOLVE" ENCOUNTERED AN ALGORITHMICALLY
     * SINGULAR',' MATRIX')
      END
C
C
      FUNCTION FINDY (XY,XX,NN,KK)
```

```
C
C  Finds a Y value for a given X from a table of points
C  by interpolation.
C
      DIMENSION XY(2)
      DATA NERR,NPRMAX / 1, 20 /
      DATA ZERO,DM10 / 0., 1.E-10 /
      X=XX
      N=NN
      M=IABS(N)
      K=KK
      IF(K.LT.1)K=1
      IF(K.GE.M)K=M-1
      IF(M-1)5,6,10
    5 FINDY=ZERO
      RETURN
    6 FINDY=XY(1)
      RETURN
   10 IF(XY(2*K)-X)20,20,11
   11 K=K-1
   12 IF(K)30,30,10
   20 IF(X-XY(2*K+2)-DM10)100,100,21
   21 K=K+1
      IF(K-M)20,40,40
C
C  Test for extrapolation
C
   30 IF(N)31,5,180
   31 K=1
      GO TO 100
   40 IF(N)41,5,190
   41 K=M-1
C
C  Everything OK, get answer
C
  100 KK=K
      FINDY=XY(2*K-1)+(X-XY(2*K))*(XY(2*K+1)-XY(2*K-1))
     1  /(XY(2*K+2)-XY(2*K))
      RETURN
C
C  Failure, search out of bounds
C
  180 FINDY=XY(1)
      GO TO 200
  190 FINDY=XY(2*M-1)
  200 IF(NERR.GT.NPRMAX)GO TO 202
      WRITE(6,210)KK,K,N,X,(XY(2*I),I=1,N)
C     CALL ERRTRA
      IF(NERR.EQ.NPRMAX)WRITE(6,212)
      NERR=NERR+1
```

```
202  RETURN
210  FORMAT(' ERROR IN FINDY',/,'INITIAL INDEX =',I6,10X,
    */,'FINAL INDEX =',I6,10X,/,'ARRAY LENGTH =',I6,10X,
    */,'ARGUMENT =',E14.6,/,'TABLE OF X VALUES =',/,(8E15.6))
 212 FORMAT (//'***** FURTHER MESSAGES SUPPRESSED')
     END
```

Appendix I

DIFFUSION EQUATION CODE - DFDIFFUSION

The listed program was developed for numerical evaluation of the describing function and analytical solutions to the neutron diffusion equation. The program is written in Pascal (USCD) symbolic programming language. The code was developed on an APPLE II minicomputer.

```
Program DFdiffusion;

(*
   Pascal program to evaluate the analytical and
   describing function solutions to the slab reactor
   problem.  This program writes the plot files for
   the HiPlot plotting and regression routine in
   appendix F.
*)

USES transcend;

Const
   pi=3.141592654;

Type
   vector = array [0..100] of real;

Var
   x: vector;
   iterfile, plotfile: text;
   itername, plotname: string;
   FF: char;
   i, Ngrid, iteration, maxiter: integer;
   B, a, step, error: real;


Function Phi_analytical (x: real): real;

(*   This function evaluates the analytical solution
     for the flux at a given value of x and returns
     the normalized flux   *)

   Begin
     Phi_analytical := Cos( B * x);
   End;


Function sign (n: integer): integer;

   Begin
     If (n mod 4) = 1 then sign := 1
     Else sign := -1;
   End;


Function intpow (x, n: integer): real;

(*    This function computes the value of a real variable
      x, to a positive integer power n     *)
   Var k: integer;
       product: real;

   Begin
     If n > 0 then
       Begin
         product:=1.0;
         for k := 1 to n do product := product * x;
         intpow := product;
       End
     Else
       Begin
         If n = 0 then intpow := 1.0
         Else
           Begin
             write('*** intpow error - value one returned');
             intpow := 1.0;
           End;
       End;
   End;


Function Phi_DF (x: real; iter: integer): real;

(*    This function computes the describing function
      solution for the normalized flux with a maximum
      error of 1.0E-7    *)

   Var xin, sum, term, normterm, xil: real;
       n: integer;

   Begin
     sum := 0.0;
     n := 1;
     Repeat
       xin := n * B;
       term := (sign(n)/intpow(n, 2*iter+1)) * Cos( x * xin );
       sum := sum + term;
       normterm:= ABS(term);
       n := n + 2;
     Until normterm < error;
     n:=n - 2;
     writeln(iterfile,x,'  ',n);
     Phi_DF := sum;
   End;


Procedure titles;
```

```
   Var thickness: real;

   Begin
     thickness:=2.0*a;
     write  (plotfile,'Comparison of Analytical');
     writeln(plotfile,' and Describing Function');
     write  (plotfile,'Solutions for the Slab ');
     writeln(plotfile,'Reactor Problem');
     write  (plotfile,'X - direction (slab ');
     writeln(plotfile,'thickness = ',thickness:2:1,')');
     writeln(plotfile,'Normalized Flux');
     write  (iterfile,'Number of terms summed in');
     writeln(iterfile,' the describing function');
     writeln(iterfile,'Maximum error = 1.0 E-7');
     write  (iterfile,'X - direction (slab ');
     writeln(iterfile,'thickness = ',thickness:2:1,')');
     writeln(iterfile,'Number of terms summed');
   End;


Begin (*main program*)
   error:= 1.0E-7;
   FF:= Chr(12);
   write(FF,'Input slab thickness -->');
   readln(a);
   B:= pi / (2 * a);
   write('Input # of grid points -->');
   readln(Ngrid);
   step := a / Ngrid;
   write('Input max # of iterations -->');
   readln(maxiter);
   For i:= 0 to Ngrid do x[i]:= i * step;
   Repeat
      write('Enter file name -->');
      readln(plotname);
      If pos('.TEXT',plotname) = 0 then
         plotname:= concat(plotname,'.TEXT');
      write('Enter iteration file name--> ');
      readln(itername);
      If pos('.TEXT',itername) = 0 then
         itername:= concat(itername,'.TEXT');
      itername:=concat('#5:',itername);
      (*$I-*)
      (*  Turn off error checking by the compiler and
          open the files  *)
      rewrite(plotfile,plotname);
      rewrite(iterfile,itername);
      (*$I+*)
   Until ioresult = 0;
   titles;
   For i:= 0 to Ngrid do
      writeln(plotfile,x[i],'    ',Phi_analytical(x[i]));
   For iteration:=1 to maxiter do
      Begin
        writeln(iterfile,'iteration #',iteration);
        writeln(plotfile,'Iteration #',iteration);
        For i:=0 to Ngrid do
          write  (plotfile,x[i],'    ');
          writeln(Phi_DF(x[i],iteration));
      End;
   Close(plotfile,lock);
   Close(iterfile,lock);
   write(FF);
   gotoxy(0,12);
   writeln('That''s all folks...');
End. (*of program*)
```