This is to certify that the

dissertation entitled

Correlation of Stress, Strain, Morphological Shifts and Permeation-Rate Changes in Polymer Films

presented by

Scott Allan Morris

has been accepted towards fulfillment of the requirements for

Ph.D. degree in Agricultural Engineering

Larry J. Segerlind
Major professor

Date Nov 20, 1992

Correlation of Stress, Strain, Morphological Shifts and Permeation-Rate Changes in Polymer Films.

By

Scott Allan Morris

A DISSERTATION

Submitted to
Michigan State University
in Partial Fulfillment of the Requirements
for the Degree of

Doctor of Philosophy

Department of Agricultural Engineering

1992

ABSTRACT

CORRELATION OF STRESS, STRAIN, MORPHOLOGICAL SHIFTS AND PERMEATION-RATE CHANGES IN POLYMER FILMS.

By

Scott Allan Morris

This study constructs a quantitative link between stress, strain, morphology changes and changes in the rate of permeation in a polymer film sample. The purpose of this is to provide a simple method of correlating mechanical input and performance changes in polymeric material using a series of simple tests rather than the repetitive construction and evaluation of prototypes.

A step-loading test fixture was devised to apply varying levels of stress to a cruciform film sample. An optimization scheme based on the COMPLEX algorithm was devised to determine the material constants of the polymer used in the material and a finite element model for viscoelastic materials was constructed to resolve the state of strain, thinning and change in free volume occurring in the sample material. Small Angle Light Scattering (SALS), and Scanning Electron Microscopy (SEM) were used to inspect the material for gross changes in morphology as a result of mechanical input. Finally, CO_2 permeation was tested via the quasi-

isostatic method to check the film for changes in permeation rate.

The study revealed an increase of up to 20% in the rate of permeation in response to a small (less than 6%) increase in free-volume and an even smaller (less than 1%) amount of thinning in the sample. The correlation of these changes with a specific state of strain in the region tested represents a significant step forward since the stress-strain model may be used to resolve similar occurances in other structures made of the same material. More generally, the study represents the inititation of a modelling methodology which may be used to predict overall permeation changes in a particular structure before beginning prototype construction.

Dedication

This work is dedicated to

Dr. Allan J. Morris
Frances M. Stearns-Morris
Pamela F. Morris

for their pertinacious support.

ACKNOWLEDGEMENTS

I would like to acknowledge the persistent and crucial support of Dr. Bruce Harte in the construction, funding and realization of this project. Thanks also to my major professor, Dr. Larry Segerlind, and the other committee members; Dr. John Gerrish and Dr. Gary Cloud, all of whom have made valuable and knowledgeable contributions to this study.

Other major contributors to this study are the A. H. Case Center for Computer-Aided Engineering and Manufacturing, and particularly Greg Fell and Fred Hall for their persistent good humor in the face of endless simple questions. At the Center for Electron Optics, Michigan State University, Stan Flegel and Peg Hogan are to be thanked for their help with the electron microscopy portion of the study. Dr. Dashin Liu, Department of Metallurgy, Mechanics, and Material Science, deserves thanks for his assistance with the vagaries of viscoelastic solid mechanics. Dr. Jack Giacin of the MSU School of Packaging donated crucial advice and equipment for the permeation analysis.

Beyond that, the entire staff, faculty and graduate student bodies of both the School of Packaging and the Department of Agricultural Engineering are to be thanked for providing a great place to "push the envelope" and for providing a tough act to follow. Also I would like to thank Edna for fielding many penetrating insights.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

1.0 INTRODUCTION

The use of barrier polymers in packages designed for long shelf-life products is increasing. With this increase comes the need for a quantitative assessment of the effects of stress and strain imposed during package manufacture, filling, and distribution on the gas-transport properties of a material. Mechanically induced changes in barrier properties can result in the reduced shelf-life of a packaged product, or the reduced efficacy of a pharmaceutical product even if there is no overt sign of deterioration. An improved understanding of the relationship between mechanical, morphological, and gas-barrier properties of materials will allow producers and users to optimize their use of these materials by obtaining desired barrier properties with lower volumes of more carefully engineered materials.

The relationships between loading, deformation, structural orientation, and changes in gas transport properties (diffusion and permeation) in polymers are poorly correlated. Some studies have characterized microstructural changes and changes in gas transport properties of highly crystalline polymers (Polycarbonate, PVC, and polyimide) under simple uniaxial tensile strain. These results often

conflict with existing theories of free-volume reduction, however, and do not apply to cases of multiaxial strain, nor to the semicrystalline polymers prevalent in the packaging industry such as PE and PP [O'Brian et al.,1987; Smith and Adam, 1981; El-Hibri and Paul 1985].

Studies have been conducted to characterize changes in the crystalline/amorphous structure of deformed semi-crystalline polymers in terms of an accurately measured state of strain. These studies did not consider the gas-transfer properties of the polymers. [Segula and Rietsch, 1985; Hendra et al., 1985; Burke and Weatherly, 1987; Pan et al., 1987; Schultz,1984].

Other investigators have attempted to correlate strain and gas transport properties, but they have only considered the strain field imposed on the samples in terms of simple uniaxial elasticity/plasticity relationships. They did not consider viscoelastic (time-dependent) relaxation effects, nor did they make an effort to characterize the resultant changes in morphology [Yasuda and Peterlin, 1974; Paulos and Thomas, 1980; El-Hibri and Paul, 1986; Morris and Lee, 1987].

The relationship between stress, strain, changes in morphology, and changes in gas transport properties is in need of a comprehensive, multifaceted study in order to approach mechanically caused barrier property change problems with the proper analytical tools. The significance of the quantitative understanding of this relationship reaches

beyond food and pharmaceutical packaging into the fields of:

- Separation and purification operations (optimization of membrane structure during the fabrication of separator/filter cartridge structures.
- Biomedical processes (such as dialysis and oxygenation via the strain modification of the requisite semipermeable membrane structures).
- Extended storage of whole blood/ blood platelets (modification of the polymeric container for optimum gas transport) (NASA MSC-21157).

2.0 OBJECTIVES

This study was designed to determine the correlation between mechanical stresses, viscoelastic response, shifts in polymer morphology, and changes in gas-transport properties in the semicrystalline polymers used in packaging. A quantitative link between mechanical stress and shifts in the gas-barrier properties in these materials was to have been defined.

Specifically, the objectives of the study are:

1. To experimentally determine the viscoelastic properties of a material sample and apply the coefficients to a numerical model of the biaxially strained sample in order to predict the stress, strain and time relationships of the material.
2. To characterize changes, if any, in the morphology of the polymer.
3. To measure the gas-transport rate of the material before and after deformation.
4. To link the information gathered from the previous objectives into a comprehensive picture of the functional changes occurring in the polymer film as a result of two-dimensional mechanical stress.

3.0 LITERATURE REVIEW

3.1 Mechanics and Viscoelasticity

The primary quantities which are considered in the mechanics of materials are stresses, strains, and material properties. Stresses result from forces acting within the body being considered. Strains are the result of differential movements within the body, and are usually described in terms of percentage of a referential measure (eg. final length divided by original length in the linear case). Displacements are the movement of referential point(s) within the body relative to some external coordinate system and are related to strain.

The field of the analytical mechanics of deformable bodies is primarily concerned with the relationship of these three conceptual quantities, and constructs three broad classifications of families of equations to relate the quantities to one another: equilibrium conditions (or equations of motion for dynamic problems), kinematic relations, and constitutive equations. Equilibrium conditions are material independent, and define the stresses (or conditions of motion) acting on the body in question. Kinematic relations, which are also material-independent

relate the strains within a body to its displacements. The constitutive equations are material dependent and relate the stress and strain within the body. Since the materials may vary widely in their response, there are many types of constitutive equations falling into four broad categories; elastic, viscous, plastic, and viscoelastic.

Elastic materials may be defined as those materials which store energy without significant loss (as with a stretched metal spring), and stress is linearly proportional to strain. Viscous materials, by contrast, dissipate energy without significant energy-storage capacity (demonstrated by pouring water from one glass to another) and exhibit stress in proportion to strain rate. Plastic behavior combines the preceding two concepts in the following manner: A plastic material will store energy in an elastic fashion until the "yield point" is reached, after which the material begins to irreversibly deform without further energy storage. Plastic behavior usually does not account for the rate of strain on the material (Flugge, 1967).

Viscoelasticity is concerned with the study of the materials which have a stress-strain relationship with characteristics that do not exactly fit the concepts of plasticity, viscosity or elasticity. Viscoelastic materials can both store and dissipate energy in a manner where the rate of strain is dependent on the time-history applied stress. Viscoelastic constitutive equations are often

composed of both viscous and elastic equation elements (hence the term "visco-elasticity"), and the constitutive equations of these types of materials may be linear, with constant material coefficients, or non-linear (where the material coefficients contain terms which are a function of stress, strain, or their derivatives) (Ferry, 1980). In this study, for the sake of simplicity, the response of the material was assumed to be linearly viscoelastic.

For most constitutive equations arising from the study of deformable bodies, the two basic elements of viscoelastic models are the elastic spring element where

$$\begin{aligned} Stress &= Constant \cdot Strain \\ \sigma &= E \cdot \varepsilon \end{aligned} \tag{1}$$

(The constant in this equation is usually the elastic [Young's] Modulus, E). The viscous dashpot element used is

$$\begin{aligned} Stress &= Constant \cdot Strain\ Rate \\ \sigma &= constant \cdot \dot{\varepsilon} \end{aligned} \tag{2}$$

In fluid mechanics, the constant is often taken to be η, the viscosity coefficient relating shear stress and shear rate.

The more complex material responses are often modelled by combinations of simple elements, some of which are reproduced in Figure 1.

The three-parameter solid is the constitutive model that

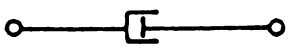
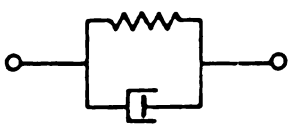
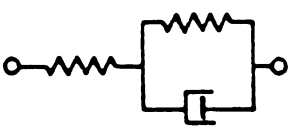
Model	*Name*	*Differential equation* / *Inequalities*
	elastic solid	$\sigma = q_0\epsilon$
	viscous fluid	$\sigma = q_1\dot{\epsilon}$
	Maxwell fluid	$\sigma + p_1\dot{\sigma} = q_1\dot{\epsilon}$
	Kelvin solid	$\sigma = q_0\epsilon + q_1\dot{\epsilon}$
	3-parameter solid	$\sigma + p_1\dot{\sigma} = q_0\epsilon + q_1\dot{\epsilon}$ $q_1 > p_1 q_0$

Figure 1. Chart of Constitutive Equations.
(Flugge, 1968)

will be used in this study since its compliance curve most nearly matches that of the types of polymers under consideration (Ferry, 1980), and it encompasses all of the preceding models in the table.

3.1.1 Effects of Loading

The type and degree of loading is critical in predicting the response of viscoelastic materials. Since the analytical solution of the constitutive differential equations is often accomplished using Laplace transforms loading regimes considered are usually those which can be constructed of the more common functions considered within the context of Laplace transforms: Dirac delta (spike) functions, Heaviside unit (stepped) functions and "ramp" functions. Superposition and the time-shift principles (the so-called t- and s-shifts) may be used to combine these functions to mimic many "real-world" situations (Speigel, 1965). In this study, the loading will be limited to a single step function.

3.1.2 Material Response to a Single Step Function.

The response of a three-parameter solid may be shown by solving the model's governing equation,

$$\sigma + p_1\dot{\sigma} = q_0\varepsilon + q_1\dot{\varepsilon} \tag{3}$$

using the Laplace Transform, this operation gives

$$\bar{\varepsilon} = \sigma_0\left(\frac{p_1 + \frac{1}{s}}{q_0 + q_1 s}\right) \tag{4}$$

or

$$\bar{\varepsilon} = \frac{\sigma_0}{q_1}\left(\frac{p_1 s + 1}{s(\lambda + s)}\right) \qquad where\ \frac{q_0}{q_1} = \lambda \tag{5}$$

where σ_0 is the initial step loading value. When rearranged and operated on by the inverse transform, the equation becomes

$$\varepsilon = \frac{\sigma_0}{q_1}\left[\frac{1}{\lambda}(1 - e^{-\lambda t}) + p_1 e^{-\lambda t}\right] \tag{6}$$

Note that

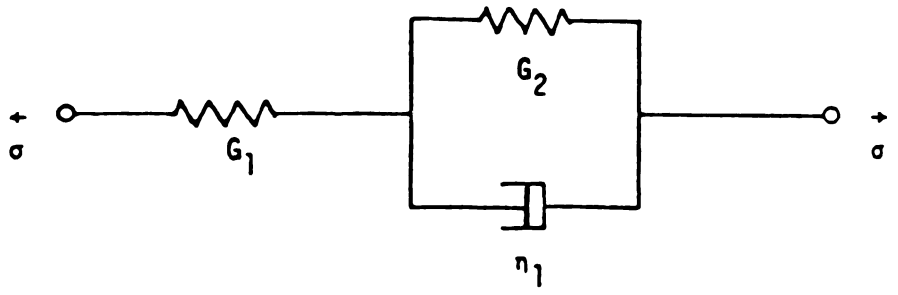
$$\varepsilon(t=0^{+}) = \frac{\sigma_0 p_1}{q_1}$$
$$\varepsilon(t=\infty) = \frac{\sigma_0}{q_0} \tag{7}$$

which is illustrated in Figure(2).

3.1.3 Numerical Solutions of the Stress-Strain Equations

In all but the most simple geometrical configurations, the exact analysis of the state of stress and strain at a chosen point within the body to be considered verges on the impossible. Sections with irregular geometries, sharp corners, inclusions such as holes or slots, or boundary conditions that are less than ideal all render the analytical formulations of the states of stress and strain unsolvable in any practical sense (Cook & Young, 1985).

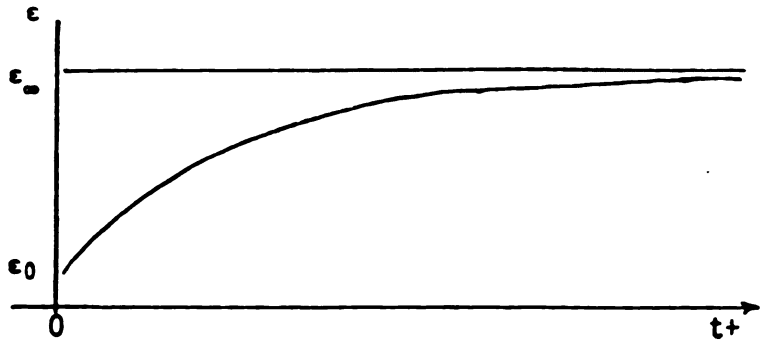
Numerical methods are viewed as a means to closely approximate the solutions of the partial differential equations of stress and strain of the body to be analyzed. The finite difference method replaces the partial differential equation (PDE) and boundary condition terms with a system of equivalent difference equations. The solution of this system of simultaneous equations then yields the solution to the PDE at each node. This method is useful in



$$p_1 = \frac{\eta_1}{G_1+G_2}$$

$$q_0 = \frac{G_1 G_2}{G_1+G_2}$$

$$q_1 = \frac{G_1 \eta_1}{G_1+G_2}$$



For a stepped load σ_0 applied at time t=0

$$\varepsilon_\infty = \frac{\sigma_0}{q_0}$$

Figure 2. Response of a Three-Parameter Solid to an Applied Step Loading.

that the formulation is relatively simple. Unfortunately, the rectangular grid geometry for the points used in the analysis is simple as well--and inflexible (Ugural, 1981). Although it is possible to produce a mesh of triangular elements, or re-map them into circular or spherical coordinates, extremely irregular boundaries or unusual shapes will cause the model to match the spatial coordinates of the actual object very poorly (Paulsen, 1992).

The finite-element method, although burdened with a more difficult formulation, has the advantage of not being limited to a strictly rectangular or triangular approximation of the body's geometry. Rectangular and triangular elements may be mixed, curved elements may be produced (or closely approximated), and the dimensions of the elements may be allowed to vary in order to more closely approximate the geometry of the subject.

In the system used to model the viscoelastic response of the materials in this study, the finite element formulation also will be shown to conveniently accommodate the time-dependency of the material as well as the irregular boundary conditions of the test samples (Segerlind, 1984, Boresi and Sidebottom, 1984).

3.2 Optimization and Parameter Estimation

Determining the material coefficients for the defining differential equation of the three-parameter model is a troublesome aspect of the mechanics of viscoelastic materials for all but the simplest types of tests and sample geometries. There are several standard ASTM tests for these types of materials, but the results are quite simplified and are of limited use for the modelling of complex systems. A great deal of effort has been devoted to producing a good set of material coefficients for many types of constitutive models, both linear and nonlinear (Augl and Land, 1985) (Ferry, 1980). Most methods require simplified test protocols or sample geometries and may not return values that are useful in large-scale modelling of the two-dimensional films used in this study.

Investigators in the system optimization field have occasionally used finite element modelling to describe the objective function of the optimization scheme at hand (Jehle & Mlejveck, 1990). The usual scheme in such optimization models is to computationally fit the material coefficients of the model so that the difference between the output of the model and a set of observed data, or desired outcomes, is minimized. This scheme, although computationally intensive, is useful in the optimization of a system that is too

complex to describe analytically, or does not possess the simple geometries so often used in standard measurement techniques. This method lends itself well to the problem at hand since the already developed finite element model can be used as the objective function. The data taken from optical measurements in existing test fixtures may be used as the data to be matched.

The method developed for this study was that of minimizing the difference between the finite element model's calculated values for strain and the experimentally observed strain data over a series of equally spaced time-steps. The advantage of this method, particularly in a biaxially stressed system, was that the material parameters can be retrofitted to the observed data without the need for additional testing equipment. Additionally, this method shows great promise for the analysis of material properties in other situations where the material is in some unusual configuration, is in use and may not be removed for analysis, or may only be studied via some non-contact methodology.

3.3.1 The Complex Method of Function Minimization

The Complex method, an extension of the Simplex method, is an efficient and versatile tool for finding global optima within the domain of possible solutions (Box, 1969). It

avoids some of the convergence problems which occur with simplex methods and may be used with almost any number of variables or type of objective function. It also has the notable advantage of being able to accommodate restrictions on both the limits of the estimated parameters and the regions to be considered feasible. It has the further advantage of not requiring the calculation of derivatives.

In operation, the Complex method randomly seeds a number of vertex points about the allowable domain of the function to be minimized (creating a complex polygon) and calculates the value of the objective functions at each of these points. The difference between these points is considered, and if the difference exceeds a preset parameter value then the vertex returning the lowest value is replaced by another point located a distance along a line located through the rejected vertex and the centroid of the complex polygon. This process repeats until the difference between the vertex points is less than the minimum (β). At this point, the centroid of the isoplethic polygon is considered to be the minimum (ringed, in a sense, by the vertices of the complex polygon), and the computation stops, returning the values of the parameters at the centroid, and the centroid objective function value.

3.3 Gas Permeation in Deformed Polymer Films

The rate of gas permeation through materials can be correlated to several phenomena which occur during the deformation process. The simplest is the result of changes in section thickness which occur as a material is strained (half the thickness yields roughly twice the permeation). In most polymers, however the processes which occur are much more complex, and the changes are not nearly as simple as with simple thinning phenomenon.

Under large strains, polymer films deform visco-elastically, changing their morphology from a mass of nearly random macromolecular chains to a well oriented series of parallel fibrils. Experimental research has shown that the fibrillar component of these oriented materials has a permeation rate that is orders of magnitude lower than that of the amorphous material surrounding it (Williams and Peterlin, 1971). From this, one may correctly assume that extensively orienting a film and creating a high fibril content will cause the permeation rate of the film to decrease drastically.

This phenomenon is commonly used in industry to tailor the permeation rates of polymer films to a specific application, and to reduce the amount of raw material needed to perform the barrier function of a particular package.

Currently, the most visible example of this method is in the polyethylene terpthalate (PET) bottles, used by the soft-drink industry, which are oriented both axially and radially in order to retain carbonation for the required period of time. Careful manipulation of both the molding and orientation process during production has resulted in a substantial material reduction since the introduction of the bottle in the 1980's.

Under lesser strains, the material behaves much differently than in the large-strain orientation described above. Under small strains, polymer free-volume[1] and diffusion increase (provided that Poisson's ratio is less than 0.5, which is approximately the case for most organic polymers other than rubber) until a plateau is reached at relatively low levels of strain ($0.05 < \varepsilon < 0.20$ for polyethylene). The increase in free volume is accounted for in the small-strain equation for thinning

$$\begin{aligned} \varepsilon_z &= -\frac{\mu}{E}(\sigma_x+\sigma_y) \\ &= -\mu(\varepsilon_x+\varepsilon_y) \end{aligned} \tag{8}$$

(Gere and Timoshenko, 1984).

---

[1]The "Free Volume" of a polymer may be thought of as the volume of the void-free solid versus the space occupied by the actual molecular chains comprising it. A long chain polymer may be folded and bundled such that there is sufficient space between the folds to allow the slow passage of sufficiently small gas molecules.

For a square area of material

$$x=y$$
$$\mu=0.25 \tag{9}$$
$$\varepsilon_x=\varepsilon_y=0.1$$
$$\varepsilon_x=-0.25(0.1+0.1)=-0.05$$

$$\begin{aligned} \textit{New Area} &= (x+\varepsilon_x)\cdot(y+\varepsilon_y) \\ &= (x+0.1x)\cdot(y+0.1y) \\ &= 1.21(\textit{Original Area}) \end{aligned} \tag{10}$$

$$\begin{aligned} \textit{New Thickness} &= (z+\varepsilon_z) \\ &=(z-0.5z) \\ &=(0.95z) \\ &= 0.95\ (\textit{Original Thickness}) \end{aligned} \tag{11}$$

$$\textit{New Volume}=(1.21\ (\textit{Original Area})\cdot 0.95\ (\textit{Original Thickness})$$
$$=1.14\ \textit{Original Volume}$$

Note that for large values of the elastic modulus, E, the effect is more pronounced to a limiting value equal to the ratio of new surface area to original surface area. Additionally E may be replaced with E(t) by superposition to give a small-strain linear viscoelastic formulation (Flugge, 1968) with similar results.

After the "plateau" of maximum free-volume expansion has been reached, the aforementioned effects of orientation begin to dominate the system and permeation will drop off to well

below its unstrained value. Sorption steadily increases as well and appears to approach an asymptote as the levels of strain increase (Yasuda et al, 1964; Yasuda and Peterlin, 1974).

The studies mentioned above consider the viscoelastic polymer in terms of an elastic-plastic model and do not account for the time-dependance of the material response in terms of any overall material constitutive model or system.

3.4 Material Analysis

3.4.1 Scanning Electron Microscopy

Scanning electron microscopy (SEM) is an often-used technique for the characterization of the surface structure of polymeric materials (Roulin-Maloney, 1990). The usual procedure utilizes either an as-produced surface or a surface created for the analysis (through cryomicrotomy or similar method) as the specimen over which a replicant surface is cast using a high resolution casting material. Often the surface to be examined is treated with some type of etchant such as p-xylene or chromic acid to increase the resolution of the differing regions within the distorted material (Jang et al, 1985, Hashimoto et al, 1976). Once the surface (or its replica) is prepared, the material is desiccated using critical point extraction to remove the

solvents and residual moisture which may contaminate the interior of the microscope. The surface to be examined is electroplated (to dissipate any charge that may be built up from the scanning electron beam), and an image may then be recorded.

Recent developments in scanning force electron microscopy (SFEM) and scanning tunnelling electron microscopy (STEM) have led to very high resolution of surface characteristics and have made quantifiable roughness measurements possible (Reiss et al, 1991; Howells et al, 1991). The equipment for these SFEM and STEM, however, was not available for use in this study. Although the surface profile information ws not be as good as with SFEM and STEM methods, the geometry and degree of asymmetry of the spherulites should have been accurately measurable.

3.4.2 Small Angle Light Scattering, Spherulite Size and Orientation.

Small angle light scattering (SALS) is commonly used to determine the degree and direction of orientation of many types of materials. In its simplest form, the apparatus is simply a highly collimated (preferably coherent) source of monochromatic light which is passed through a polarizer then through the material to be analyzed and through a second, adjustable polarizer (analyzer) (Figures 3 and 4). In the

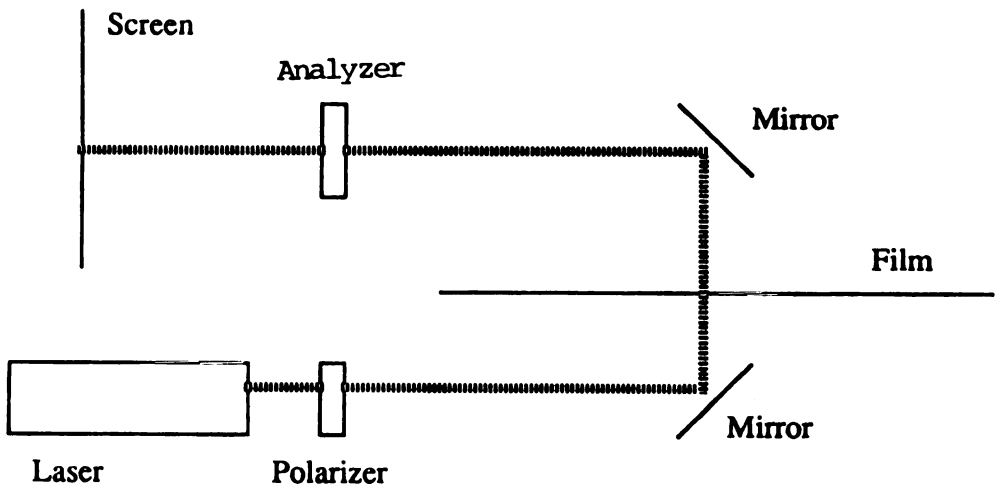


Figure 3. Schematic of Small-Angle Light Scattering Apparatus.

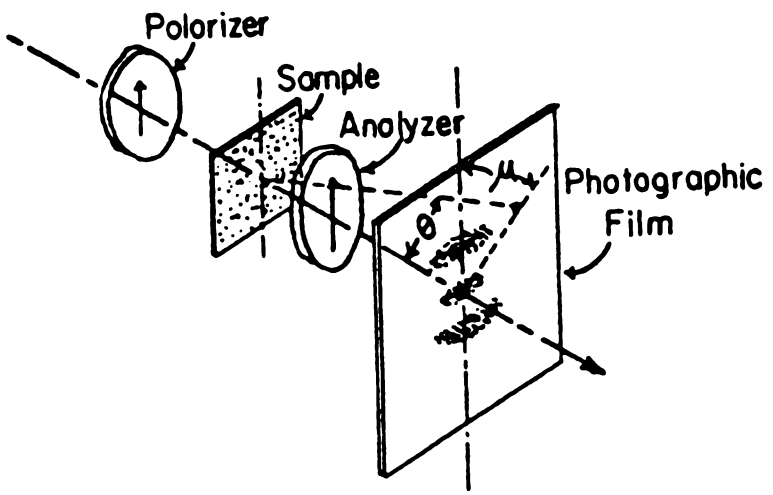


Figure 4. The Experimental Arrangement for Photographic Light Scattering From Films (Stein and Rhodes, 1960).

case of high molecular weight polymers, the spherulite size may be estimated (using a HeNe laser with λ=632.8 nm) as

$$R_0 = 4.08 / 4\pi \sin(\theta_m/2) \qquad (12)$$

where R_0 = average radius of the spherulite and θ_m = maximum scattering angle (Stein & Rhodes, 1969).

Orientation may be obtained by observing that the drawing of the polymer will pull the polymer chains in both the crystalline and amorphous regions of the polymer into an oriented state which is most often compared to a series of oriented rods. The scattering that occurs from this fibrillar component will produce an bias in the scattering pattern normal to the direction of orientation of the polymer (Rhodes and Stein, 1961; Stein and Rhodes, 1960) which may be used (when corrected for sample thickness) to correlate the degree of orientation at the particular sampling area to the rotation of the biased maximum.

The usual reason for attempting to measure these changes in orientation is to observe any fibril formation or re-formation which might be occurring in the polymer during deformation. Since the onset of a decrease of permeation rate is linked to this fibril formation, the determination of the point at which these changes begin to occur will be useful in the construction of an accurate predictive model for permeation changes.

4.0 DERIVATION OF THE FINITE ELEMENT FORMULATION

Starting with the equilibrium equation

$$\sigma_{ij,j}+F_i=0 \qquad i,j=1,2,3 \tag{13}$$

and the viscoelastic constitutive equation

$$\sigma_{ij}=\int_0^t (G_{ijkl}(t-\tau))\left(\frac{\delta\varepsilon_{kl}(\tau)}{\delta\tau}\right)d\tau \tag{14}$$

where

$$G_{ijkl}=\frac{1}{3}[G_2(t)-G_1(t)]\delta_{ij}\delta_{kl}+\frac{1}{2}G_1(t)[\delta_{ik}\delta_{jl}+\delta_{il}\delta_{kl}] \tag{15}$$

$$i, j, k, l = 1, 2, 3, 4$$

Using the general convolution notation, which states that for any functions $\mathbf{A}'$ and $\mathbf{B}'$,

$$\int_0^t \mathbf{A}'(t-\tau)\left(\frac{\delta\mathbf{B}'(\tau)}{\delta\tau}\right)d\tau = \mathbf{A}'*\mathbf{B}' \tag{16}$$

the viscoelastic constitiutive equation may be expressed as

$$\sigma_{ij}=G_{ijkl}*\varepsilon_{kl} \tag{17}$$

where

$$G_{ijkl}=\frac{1}{3}\left[G_2(t)-G_1(t)\right]\delta_{ij}\delta_{kl}+\frac{1}{2}G_1(t)\left[\delta_{jl}\delta_{ij}+\delta_{il}\delta_{jk}\right] \tag{18}$$

Rearranging the terms of equation (17) gives

$$\begin{aligned}\sigma_{ij}&=\int_0^t G_{ijkl}(t-\tau)\left(\frac{\delta e_{kl}(\tau)}{\delta\tau}\right)d\tau\\ &=\frac{1}{3}\int_0^t\left[G_2(t)-G_1(t)\right]\delta_{ij}\delta_{kl}\left(\frac{\delta e_{kl}(\tau)}{\delta\tau}\right)d\tau \qquad (19a)\\ &+\frac{1}{2}\int_0^t G_1(t)\left[\delta_{ik}\delta_{jl}+\delta_{il}\delta_{jk}\right]\left(\frac{\delta e_{kl}(\tau)}{\delta\tau}\right)d\tau \qquad (19b)\end{aligned}$$

For equation (19a)

$$\begin{aligned}\delta_{ij}F_{ik}&=F_{jk} \qquad \text{for } j=k\\ \delta_{ij}F_{ij}&=F_{jj} \qquad (=F_{kk})\\ &=\sum_{a=1}^{3}F_{aa}\end{aligned} \tag{20}$$

Using the identity

$$\delta_{kl}\left(\frac{\delta e_{kl}(\tau)}{\delta\tau}\right)=\delta_{kl}F_{kl} \tag{21}$$

equation (19a) becomes

$$\frac{1}{3}\int_0^t [G_2(t)-G_1(t)]\,\delta_{ij}\delta_{kl}\left(\frac{\delta\varepsilon_{kl}(\tau)}{\delta\tau}\right)d\tau$$
$$= \frac{1}{3}\int_0^t [G_2(t)-G_1(t)]\,\delta_{ij}\left(\frac{\delta(\varepsilon_{11}+\varepsilon_{22}+\varepsilon_{33})}{\delta\tau}\right)d\tau \tag{22}$$

Using the identities

$$\left.\begin{array}{l}\varepsilon_{kl}=\varepsilon_{lk}\\ \delta_{ij}=\delta_{ji}\end{array}\right\}\ \textit{symmetry}$$
$$\delta_{ij}F_{ik}=F_{jk} \qquad j=k \tag{23}$$

and producing the identity

$$\begin{aligned}(\delta_{ik}\delta_{jl}+\delta_{il}\delta_{jk})\,F_{kl} &= \delta_{ik}\delta_{jl}F_{kl}+\delta_{il}\delta_{jk}F_{kl}\\ &= \delta_{ik}\delta_{lj}F_{lk}+\delta_{il}\delta_{kj}F_{kl}\\ &= \delta_{ik}F_{jk}+\delta_{il}F_{jl}\\ &= \delta_{kl}F_{kj}+\delta_{li}F_{lj}\\ &= (F_{ij}+F_{ij})\\ &= 2F_{ij}\end{aligned} \tag{24}$$

allows (19b) to be rewritten as

$$\frac{1}{2}\int_0^t G_1(t)\,[\delta_{ik}\delta_{ij}+\delta_{il}\delta_{jk}]\,\frac{\delta\varepsilon_{kl}(\tau)}{\delta\tau}\,d\tau = \int_0^t G_1(t)\,\frac{\delta\varepsilon_{ij}}{\delta\tau}\,d\tau \qquad (25)$$

combining (22) and (25) gives

$$\sigma_{ij}=\frac{1}{3}\int_0^t [G_2(t)-G_1(t)]\,\delta_{ij}\frac{\delta\varepsilon_{kk}}{\delta\tau} + \int_0^t G_1(t)\,\frac{\delta\varepsilon_{ij}}{\delta\tau}\,d\tau \qquad (26)$$

which may be expressed in convolution notation as

$$\sigma_{ij}=\frac{1}{3}\,[G_2(t)-G_1(t)]\,\delta_{ij}*\varepsilon_{kk}+g_1(t)*\varepsilon_{ij} \qquad (27)$$

Defining

$$\begin{aligned}
\varepsilon_{1\cdot} &= \varepsilon_{11} = \varepsilon_{xx} = \frac{\partial v_x}{\partial_x}\\
\varepsilon_{2\cdot} &= \varepsilon_{22} = \varepsilon_{yy} = \frac{\partial v_y}{\partial_y}\\
\varepsilon_{3\cdot} &= \varepsilon_{33} = \varepsilon_{zz} = \frac{\partial v_z}{\partial_z}\\
\varepsilon_{4\cdot} &= \varepsilon_{12} = \varepsilon_{xy} = \frac{1}{2}\left(\frac{\partial v_x}{\partial_y}+\frac{\partial v_y}{\partial_x}\right)
\end{aligned} \qquad (28)$$

and

$$\begin{aligned}
\sigma_{1\cdot} &= \sigma_{11} = \sigma_{xx}\\
\sigma_{2\cdot} &= \sigma_{22} = \sigma_{yy}\\
\sigma_{3\cdot} &= \sigma_{33} = \sigma_{zz}\\
\sigma_{4\cdot} &= \sigma_{12} = \sigma_{xy}
\end{aligned} \qquad (29)$$

(27) can be written as

$$\sigma_k = A_{km} * \varepsilon_m \qquad k,m = 1^\circ, 2^\circ, 3^\circ, 4^\circ \tag{30}$$

where

$$[A] = \begin{bmatrix} \frac{1}{3}(G_2+2G_1) & \frac{1}{3}(G_2-G_1) & \frac{1}{3}(G_2-G_1) & 0 \\ & \frac{1}{3}(G_2+2G_1) & \frac{1}{3}(G_2-G_1) & 0 \\ & & \frac{1}{3}(G_2+2G_1) & 0 \\ Sym. & & & \frac{1}{2}G_1 \end{bmatrix} \tag{31}$$

Using the functional

$$I = \int_V [\tfrac{1}{2} G_{ijkl} * \varepsilon_{ij} * \varepsilon_{kl} - F_i * u_i]\, dV - \int_{b_\sigma} (S * u_i)\, dA \tag{32}$$

(Christensen, 1971) the solution to (32) may be found,

provided that the following boundary conditions are satisfied:

$$\sigma_{ij}n_j=S_i \text{ on } B_\sigma$$
$$u_i=\Delta i \text{ on } B_u \tag{33}$$

where B_σ is the boundary over which the tractions S_i are specified, B_u is the boundary where the displacements Δ_i are specified, and n_j describes the boundary unit normal vector (positive outward). When the boundary conditions are satisfied, the first variation, δI, vanishes and the solution may be found by finding the stationary value of I.

The functional I may be discretized over the relevant region, as a set of subregions:

$$I=\sum_{e=1}^{p}\frac{1}{2}\int_{V^e}(\varepsilon_K^e*A_{KM}*\varepsilon_M^e)\,dV-\sum_{e=1}^{p}\int_{V^e}(u_\alpha^e*F_\alpha^e)\,dV-\sum_{e=1}^{p}\int_{B_\sigma^e}(u_\alpha^e*S_\alpha^e)\,dA \tag{34}$$

$$k=1^e,2^e,3^e,4^e \qquad \alpha=1,2$$

where e represents the subregions. The displacements may be found via the matrix formulations (Zienkiewicz, 1971)

$$\{\varepsilon^e\} = [B^e]\{U\} \tag{35a}$$
$$\{u^e\} = [N^e]\{U\} \tag{35b}$$

where [N] is the shape function matrix, {u} are the elemental displacements, {U} are the nodal displacements, $[B^e]$ is the matrix relating strains to nodal displacements, and $\{\epsilon^e\}$ is the strain vector.

Substituting the equations (35a) and (35b) into (34), integrating, then performing the convolution gives

$$I=\frac{1}{2}\{U\}^T*[K]*\{U\}-\{U\}^T*R \tag{36}$$

where the stiffness matrix [K] and force vector {R} are, respectively,

$$\begin{aligned}[K]&=\sum[k^e]=\sum_{e=1}^{p}\int_{V^e}[B^e]^T[A]\,[B^e]\,dV\\ \{R\}&=\sum\{r^e\}=\sum_{e=1}^{p}\left(\int_{V^e}[N^e]^TF^e dV+\int_{B^e}[N^e]^T\{S^e\}\,dA\right)\end{aligned} \tag{37}$$

and $[\ ]^T$ represents the transpose of a matrix

The first variation of (36) gives a result similar in appearance to the elasticity finite element formulation:

$$\begin{aligned}\delta I=\delta\{U\}^T*[K]*\{U\}-\delta\{U\}^T*\{R\}&=0\\ [K]*\{U\}-\{R\}&=0\\ [K]*\{U\}&=\{R\}\end{aligned} \tag{38}$$

From the preceding derivation the elasticity finite-element formulation

$$[K]\{Displacement\} = \{Force\}$$
$$[K]\{U\} = \{F\} \tag{39}$$

then can be shown to have a corresponding time-dependent formulation,

$$[K] * \{U\} = \{R\} \tag{40}$$

where (40) represents the integral

$$\int_{\tau=0}^{t} [K(t-\tau)]\, d\{u(\tau)\}\, d\tau = \{R(t)\} \tag{41}$$

Discretization of (41) over n equally spaced time-steps

$$\sum_{m=1}^{n} [K(t_n - t_m)]\{\Delta U(t_m)\} = \{R(t_n)\} \tag{42}$$

where $\{\Delta U(t_m)\}$ represents individual displacements from time t_m to t_{m+1}.

This formulation gives each displacement as a function of all previous displacements, plus a possible initial step displacement although this is often taken to be zero. The other individual components of equation (42) are [K] The stiffness matrix which contains spatial data and the viscoelastic modulii

$$[K] = \sum [K^e]$$
$$= \sum_{e=1}^{n_e} \int_{V^e} [B^e]^T [A] \, [B^e] \, dV \tag{43}$$

Where $[B^e]$ is the shape function for element e and [A] is the matrix containing the time dependent material properties derived from the material constitutive equations. The coefficients in [A] are

$$a_{11} = a_{22} = \frac{E(t)}{1-\mu^2}$$
$$a_{12} = a_{21} = \frac{E(t)\,\mu}{1-\mu^2} \tag{44}$$
$$a_{33} = \frac{a_{11}(1-\mu)}{2}$$

and {R(t)} is the force vector at time t.

4.1 Derivation of Material Constitutive Equations

The three-parameter (Maxwell) model for viscoelastic solids, has the defining differential equation

$$\sigma + p_1 \dot{\sigma} = q_0 \epsilon + q_1 \dot{\epsilon} \tag{45}$$

operating on (45) with the Laplace-transformed Heaviside unit step function

$$\begin{aligned} \mathcal{L}[\sigma_0 u(t)] &= \sigma_0(s) \\ &= \int_0^\infty \sigma_0 u(t)\, e^{-st} dt \\ &= \frac{1}{s}\sigma_0 \\ &= \overline{\sigma_0} \end{aligned} \tag{46}$$

produces the transformed differential equation

$$[\tfrac{1}{s} + p_1]\,\overline{\sigma_0} = [q_0 + q_1 s]\,\overline{\epsilon} \tag{47}$$

From this, and the superposition principle, one may define a viscoelastic shear modulus (Flugge, 1968) which may be used in place of the elastic shear modulus in the material properties matrix of the finite element formulations

$$2G(s) = \frac{\frac{1}{s} + p_1}{q_0 + q_1 s} \tag{48}$$

The shear modulus may be expressed (after rearrangement and taking the inverse transform) as a function of three groups of coefficients:

$$G(t) = \frac{1}{2q_0}\left[1 - (1 - \frac{p_1 q_0}{q_1})\, e^{\frac{-q_0(t)}{q_1}}\right] \tag{49}$$

substituting

$$\begin{aligned} K_1 &= \frac{1}{2q_0} \\ K_2 &= (1 - \frac{p_1 q_0}{q_1}) \\ K_3 &= \frac{q_0}{q_1} \end{aligned} \tag{50}$$

gives

$$G(t) = K_1 [1 - K_2 e^{-K_3(t)}] \tag{51}$$

It is worth noting that the value for G(t) when t = ∞ is simply

$$G(\infty) = \frac{1}{2q_0} = K_1 \tag{52}$$

This equation for G(t) may be substituted into the general

equation for the elastic modulus

$$E = 2G(1+\mu) \tag{53}$$

where Poisson's ratio, μ, is constant to give

$$E(t) = (K_1[1-k_2 e^{-k_3(t)}]\,(1+\mu) \tag{54}$$

which are used in the previously described material properties matrix [A] defined in (44).

4.2 Implementation of the Finite-Element Viscoelastic Solid Modelling Program

To implement the previously derived finite element method, the program VISCO2 was written in FORTRAN-88, and implemented on the Michigan State University Case Engineering Center VAX-8650 as well as the CRAY Y-MP4/464 at the National Center for Supercomputing Applications at the University of Illinois, Urbana-Champaign[2].

The implementation of this method is rather straight forward since many of the components of the algorithm are taken directly from elasticity finite element formulations. The substantial difference is that the residual term must be calculated for each time-step, and is a function of all of the preceding time-steps. This particular version calculates many of the components of these terms at the beginning of the program, then holds them in memory arrays, so that the program is not constantly recalculating the same values for the residual term components.

It should be noted that the large number of residual terms generated to account for the stress/strain "history" of

---

[2] The code for VISCO2 is included as part of Appendix A as is an earlier version, VISCO1, which uses disk memory to store the residual matrices, and is usable (although very slow) on smaller computers.

the material demands a great deal of storage capacity. These requirements can be mitigated somewhat by the use of banded storage techniques and other data-compression algorithms, but the final result often remains memory intensive.

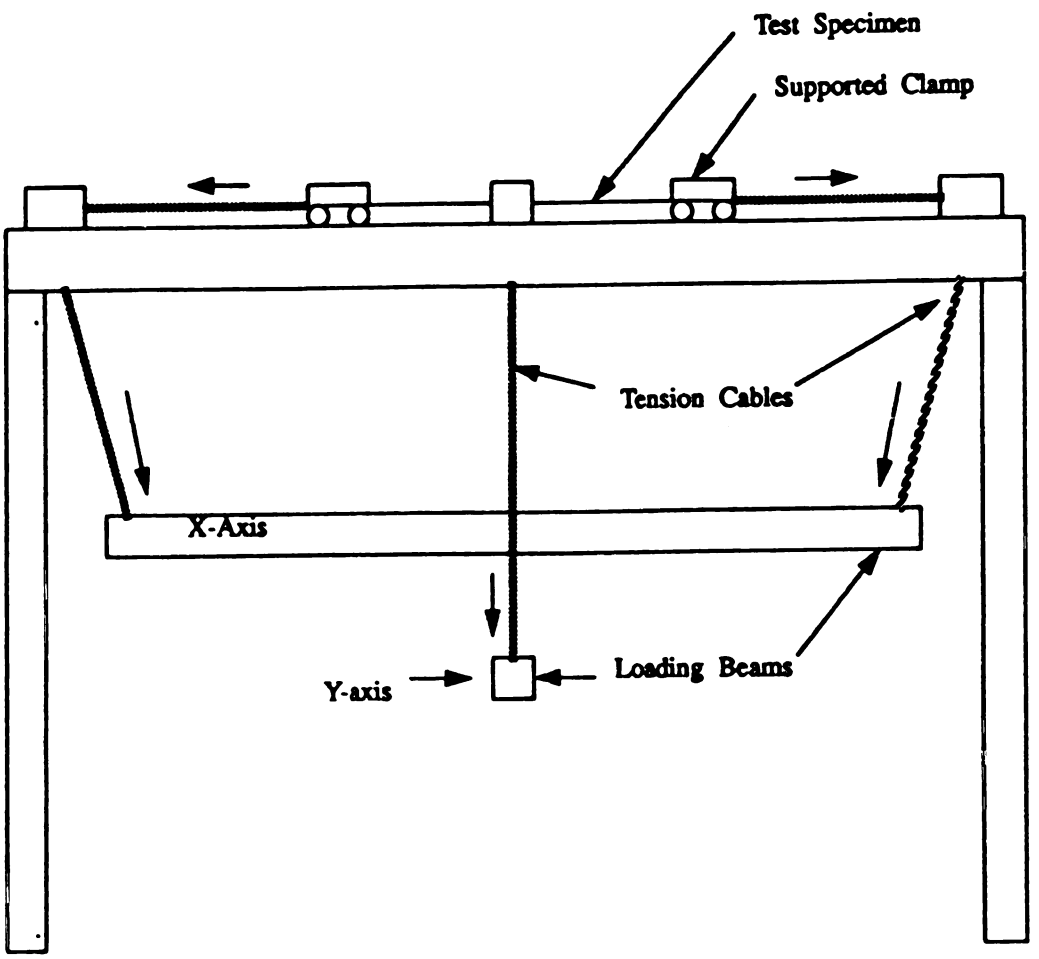
5.0 MATERIALS AND METHODS

5.1 Derivation of Material Properties Constants

The primary material used in this study was 1 mil cast polyethylene film (PW-242 "Flex-O-Film", Flex-O-Glass, Inc. Chicago, IL 60651). Actual thickness of the film was checked using a TMI 549 M micrometer (Testing Machines Inc. Amityville, NY) and was found to be .85 mil (0.00085"; 0.000216cm) over all parts of the film.

Material properties were derived using a cruciform sample subjected to a stepped loading in a biaxial tensile fixture (Figures 5 and 6). The samples were marked with a non-interactive acrylic ink (Hunt Mfg. Co., Statesville, NC 28677) in a pattern to conform to the grid design selected for use with the P21.FOR and P21a.FOR software (Figure 7). This pattern was chosen as a compromise between minimizing the number of data points to be collected and conformity to the sample during deformation.

Since the practical limit for the finite element software is approximately 15% total strain, this limit was found by trial and error (using linear samples) to be approximately 2800 psi (19.30 MPa) and was taken to be the limiting factor in the strain level applied to the film. The



Test Fixture Diagram

Figure 5. Tensile Loading Fixture Diagram.

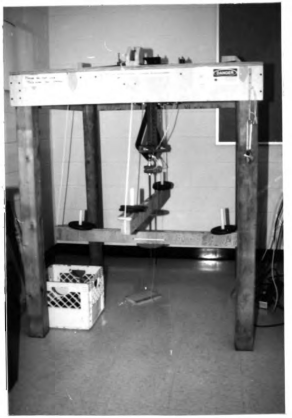


Figure 6. Photograph of Biaxial Tensile Loading Fixture.

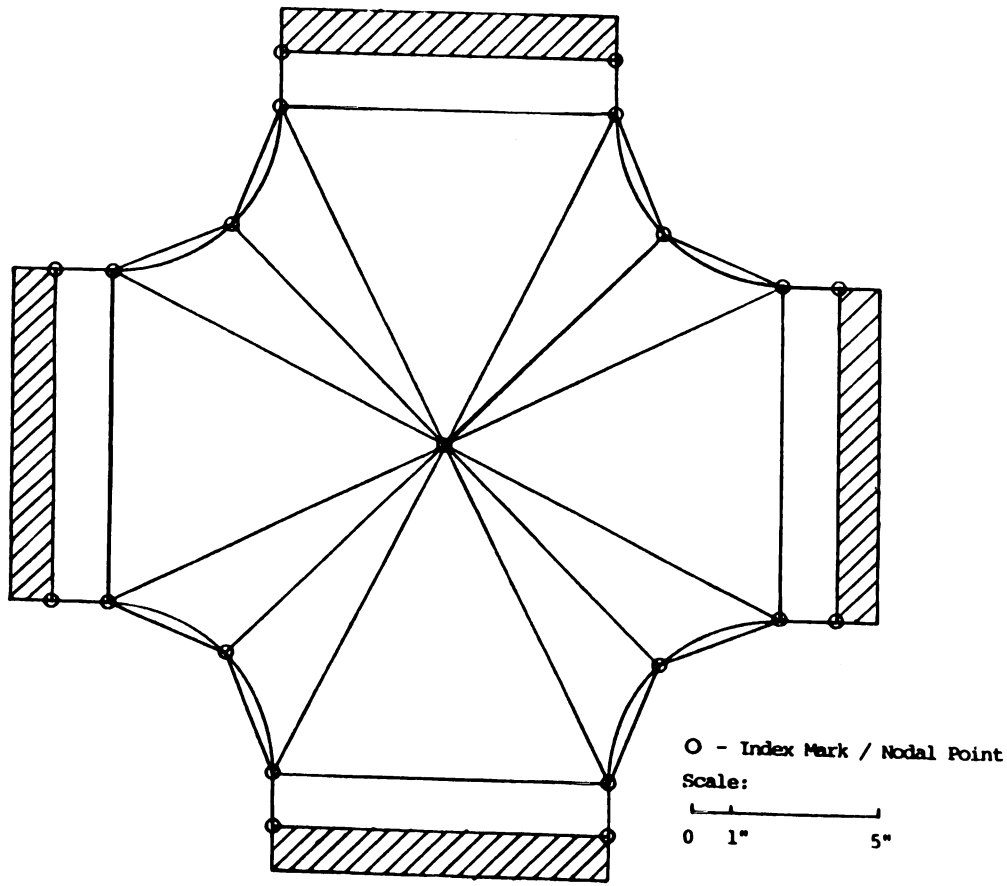


Figure 7. Diagram of Film Sample and Finite Element Grid Used for Material Parameter Estimation.

samples were then loaded to 47%, 82%, and 100% of maximum strain (1316, 2296 and 2800 psi or 9.08, 15.83 and 19.30 MPa respectively). The choice of loading levels are a result of the standardized size of the dead-weights used to load the fixture's cross-beams. The deflection of the material was recorded against a background grid of 0.2" x 0.2" (0.51 x 0.51 cm) squares using a VHS format camcorder (Panasonic VHS Reporter) and then played back using the still-frame feature of an RCA-500 VCR. Data was recorded manually from the film markings and measurement grid previously described. The VHS format records images at 1/30 sec. intervals which allows the deformation history to be recorded over a span of 6 frames of tape (1/5 sec.). Since the start of the test was not synchronized with the frame recording of the camera, the exact start and end-points of the deflection history were taken by extrapolation.

Initial tests showed a significant amount of rebound in the film due to an unknown factor in the test fixture. By experimenting with the mass distribution on the loading beams of the tensile fixture it was found that the rebound could be nearly eliminated by placing the dead-weights near the ends of the beams. This acted to reduce the amount of beam flexure and rebound during material deflection without affecting the level of load placed on the sample.

Once the data points had been recorded, the deflections were calculated and averaged, exploiting the symmetry present

in the test sample to minimize the calculation necessary to construct models for the mapping of strain fields.

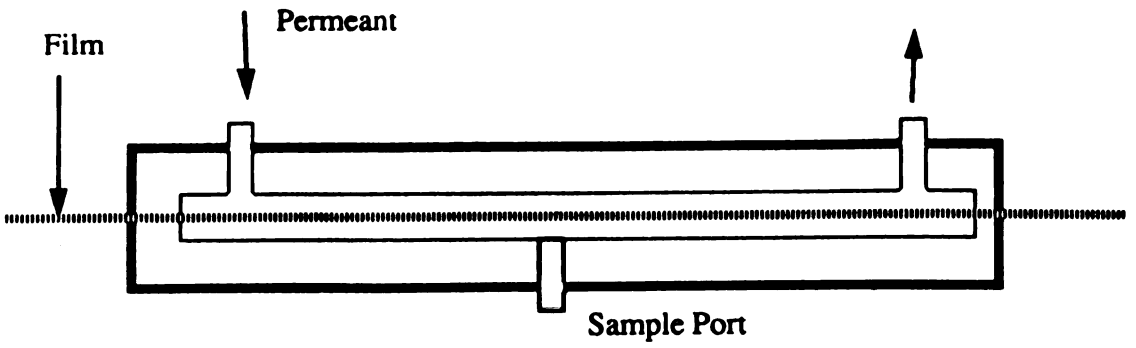
5.2 Permeation Testing

Permeation testing was done using the quasi-isostatic method. A permeation cell was devised with extended clamping arms which can accommodate the sample held in the tensile fixture (Figure 8). Carbon dioxide flowed through the lower chamber of the cell at a regulated rate of approximately 50cc/min. The upper chamber of the permeation cell (figures 9 and 10) was ventilated with low pressure compressed air for a minimum of 60 minutes after the cell was clamped on the film specimen to allow the CO_2 in the lower chamber to reach 100% concentration. At this point the upper cell was sealed and the headspace of the upper chamber was assayed at approximately seven minute intervals for CO_2 concentration by analyzing 1 cc. syringe-drawn aliquots of the headspace gas with a Carle 2153-B Gas Chromatograph and Spectra-Physics 2400 Computing Integrator.

The concentration values were plotted using the integrator's least-squares[3] curve-fitting software, and the rate of increase ascertained in order to calculate the rate

---

[3]It should be noted that the least-squares fit is used as a matter of convenience since it was part of the integrator software. The actual curve is more closely defined by an exponential time curve [$Y=C_f(1-e^{-kt})$], but for the purpose of rate determination either will suffice.



Permeation Cell

Figure 8. Permeation Cell Schematic.

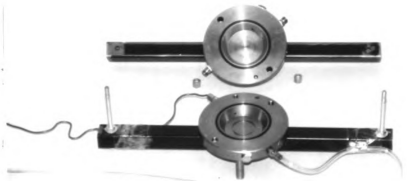


Figure 9. Permeation Cell Used
In the Study.

Figure 10. Permeation Cell With Film Sample Under Tension.

of permeation. A standard gas sample (5.05 % CO_2, 21.3% O_2, balance N_2: AGA Specialty Gasses, Maumee OH 43537) was used to calibrate the detector response, but due to the availability of only the single concentration all calculations of permeation rate are done at that concentration in order to minimize error due to detector non-linearity. The curve-fitting software gave the equation of the line plotting the rise in concentration versus time as

$$Ax^2 + Bx + C = y \tag{55}$$

where x represents time and y is the concentration at time x.

The lesser root of the quadratic equation may be used to construct the point x′ at which the concentration passes through the standard value y′

$$x = \frac{-B-\sqrt{B^2 - 4A(C-y')}}{2A} \tag{56}$$

The time-rate of concentration change is taken at this point as

$$Dc/Dt = Dy/Dx = 2A\,x' + B \tag{57}$$

and used for permeation calculations.

5.3 Small Angle Light Scattering Analysis

The Small Angle Light Scattering (SALS) analysis fixture was constructed such that material under tension in the tensile fixture could be analyzed. The Helium-Neon laser source produces a polarized beam of light which was used with an analyzer to record the H_h and H_v patterns photographically. The θ_0 was measured against a grid on the projection screen and the μ_v, or rotation of the flare pattern, was measured using a slit photometer.

5.4 Scanning Electron Microscopic Analysis

Surface features of the film in both its unstressed and maximally stressed state were cast in the tensile fixture using Reprosil Type 1 light (L.D. Caulk Co., Milford DE 19963) and then a transfer casting was made using Spurr's epoxy resin (Klomparens et al, 1986). The replica was then sputter coated with gold, and examined in the JEOL-35 Scanning Electron Microscope (Center For Electron Optics, Michigan State University). The samples were checked for change of surface features at the center within the area of the sample covered by the permeation cell where the largest strains were predicted to occur.

6.0 RESULTS AND DISCUSSION

6.1 Parameter-Estimation Method Evaluation

Two parameter estimation programs, P21.FOR and p21a.FOR were written to extract material property constants from deflection data using tests with known levels of applied loads. The programs utilize the COMPLEX function minimization methods incorporating the VISCO2 finite element program as the objective function, attempting to minimize the difference between the deflection predicted by the finite element software and the "real world" deflection data.

In order to evaluate the accuracy and utility of the parameter estimation software P21.FOR and P21A.FOR, the previously-described finite element software used dummy material property constants to generate nodal deflection values as a substitute for observed data over an arbitrary 10 time-step period (figure 11). The parameter estimation software was then used to try and re-extract the original constant values. Records were kept of the number of function-evaluations required for the model to converge.

The effect of varying the operating parameters on the final accuracy of the returned estimate was ascertained

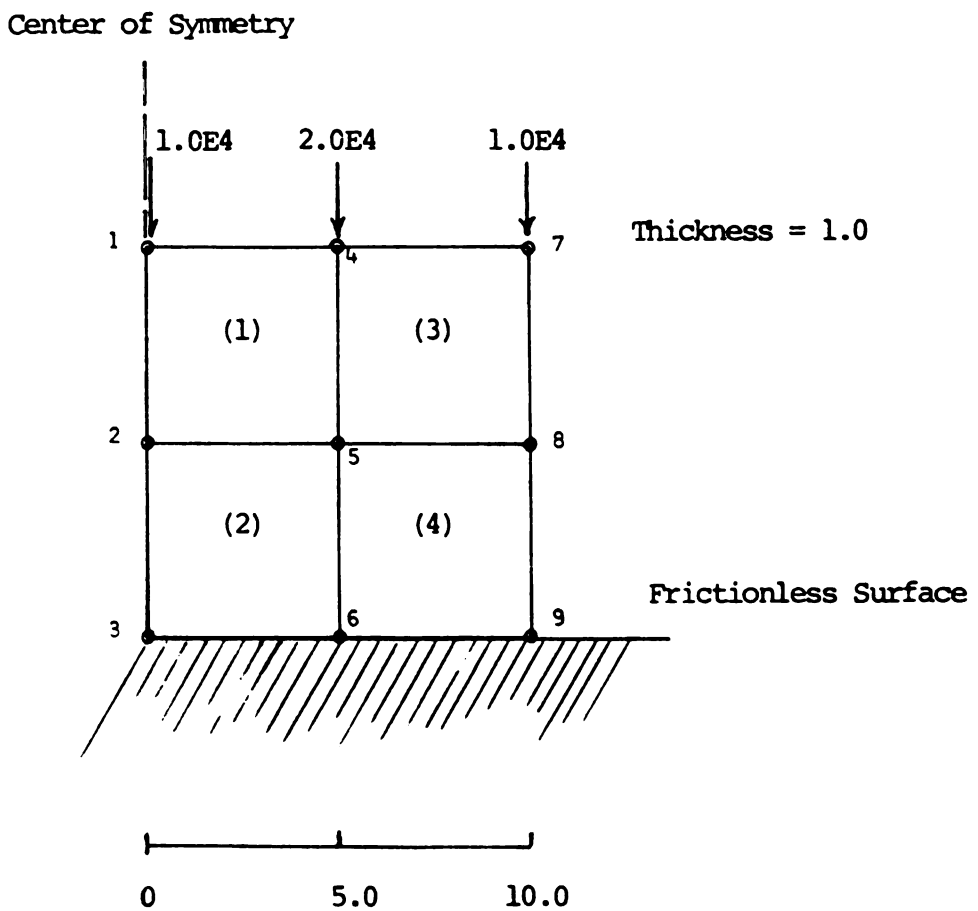


Figure 11. Finite Element Grid Used to Evaluate the Parameter Estimation Method.

(table 1). Different numbers of complex polygon vertices were used in the optimization scheme, and the value of β (the intrapolygonal variation of the vertex values, below which the model is assumed to have converged) was varied as well.

Table 1.
Parameter Estimation Evaluation Results

Number of Elements: 4
Number of Nodes: 9

Estimation of:	**K_1**
Upper Limit of Estimation:	**1.0×10^6**
Lower Limit of Estimation:	**1.0×10^2**
Starting Value:	**5.0×10^4**

3 Vertices

β	Nodal Error at Centroid	Number of Evaluations
10.0	0.240%	30
5.0	0.002	38
2.5	0.002	38
1.0	0.002	38
0.5	0.002	38

4 Vertices

β	Nodal Error at Centroid	Number of Evaluations
10.0	0.082%	34
5.0	0.074	38
2.5	0.003	56
1.0	0.008	64
0.5	0.004	75

5 Vertices

β	Nodal Error at Centroid	Number of Evaluations
10.0	0.001%	56
5.0	0.003	56
2.5	0.003	62
1.0	0.002	89
0.5	0.003	93

6 Vertices

β	Nodal Error at Centroid	Number of Evaluations
10.0	0.063%	81
5.0	0.029	89
2.5	0.008	99
1.0	0.003	111
0.5	0.000	126

Table 1 (cont'd).

Estimation of:	K_2	K_3
Upper Limit of Estimation:	1.0×10^6	1.0
Lower Limit of Estimation:	1.0×10^2	15.0
Starting Value:	5.0×10^4	12.0

3 Vertices

β	Nodal Error at Centroid	Number of Evaluations
10.0	2.214%	25
5.0	0.319	59
2.5	0.113	75
1.0	0.056	86
0.5	0.051	92

4 Vertices

β	Nodal Error at Centroid	Number of Evaluations
10.0	1.125%	54
5.0	0.024	85
2.5	0.042	91
1.0	0.006	107
0.5	0.003	112

5 Vertices

β	Nodal Error at Centroid	Number of Evaluations
10.0	0.107%	99
5.0	0.189	102
2.5	0.189	102
1.0	0.036	154
0.5	0.005	187

6 Vertices

β	Nodal Error at Centroid	Number of Evaluations
10.0	0.060%	128
5.0	0.021	138
2.5	0.021	147
1.0	0.017	176
0.5	0.001	201

As is shown by the tabulated values, the model returned a very accurate estimate of the data with relatively few evaluations of the finite element objective function and a very coarse model grid. It must be noted that the "error" value to be minimized is a sum of the errors of all points over all time-steps, and thus is extremely situation dependent. The simple study shown illustrates the relative efficiency of the method even when constraint values are used.

This computation-intensive type of parameter estimation becomes slow as the number of nodes in the objective function model increases, so the software was implemented on the Cray Y/MP 4-464 supercomputer at the National Center for Supercomputing Applications at the University of Illinois at Urbana-Champaign. The results of these trials indicated that with the parallel processor environment available the computation time shrunk by several orders of magnitude (from tens of minutes to less than three seconds).

The software is extremely efficient at estimating simpler cases such as elastic (time-independent) deformation. Although many simpler methods exist for some elastic problems, this method may find use in evaluating unknown materials of unusual geometry.

6.2 Material Parameter Estimation

The film used in this study was evaluated at three different load levels (the 100%, 82% and 47% levels described in section 5.1) using the biaxial tensile fixture. Although the fixture is capable of axially independent loading of a sample, the small amount of material available for the study confined the tests to regimes where both axes' were loaded at the samelevel simultaneouly. Initial tests showed that the entire deflection of the material occurred within approximately one fifth of a second (6 frames of videotape) and no further deformation was recorded over periods of up to a week. As will be subsequently shown, this is due to the load-time history of the material.

Data taken from the video record (Appendix B) was normalized and converted into displacement files for use with the parameter estimation software P21.FOR and P21A.FOR. Results of the material parameter-estimation are shown in Table 2.

Table 2.
Material Parameter Estimation Results

Loading	K_1	K_2	K_3
100%	43335	136323	5.55
Error[4]	5.63%	81.10%	
82%	53742	142661	12.51
Error	4.15%	79.18%	
47%	40473	155974	96.02
Error	4.72%	40.41%	

It should be noted that this parameter estimation method was designed for use where the time history of the material would be a significant factor in the results of the rest of the test (eg. that the permeation testing would occur before the material had reached an equilibrium level of deflection). In this particular case, all of the strain occurred over such a short time period and other material measurements occurred

[4]Note that the errors referred to in table (2) are the average absolute cumulative deviation of all points over all timesteps relevant to that parameter estimation run. The K_1 parameter is derived from a single timestep as previously explained, and the K_2 and K_3 parameters are derived from 6 timesteps. Further, the error in K_1 is carried through into the estimation run for K_2 and K_3 so these values may seem high.

after such a relatively long time period, that the time-dependency of the material is almost irrelevant. A parameter-estimation method of this complexity would be a much more efficient analytical tool for a material which exhibits much slower deformation time history (on the order of days or weeks).

6.2.1 Error in Material Models and Estimated Parameters Due to Loading History Deviations From the Assumed Heaviside Function

Since the tensile testing fixture used with a material with an extraordinarily short deformation history is essentially an impact loading device, the assumption that the load onset in the sample forms a perfect step-function becomes somewhat questionable since the load in a perfectly elastic (undamped) system forms an oscillator with a load amplitude twice that of the dead weight loading, and a system which is overdamped to the point of not exceeding the applied load will show a load onset more closely akin to a ramp function. Both of these conditions violate the assumptions upon which the material model is based, and call the validity of the parameter estimation method into question.

In order to check the load vs. time curve of the film in the test fixture, a strain gauge load cell was fabricated (Figure 12) and put in series with one of the test fixture tension cables (Figure 13). The output of the strain gauge was conditioned with a Daytronic 3170 Strain Gauge Conditioner and displayed on a Hewlett-Packard 54504A Digital Oscilloscope. The load vs. time history for the first 500 milliseconds of loading was measured at each of

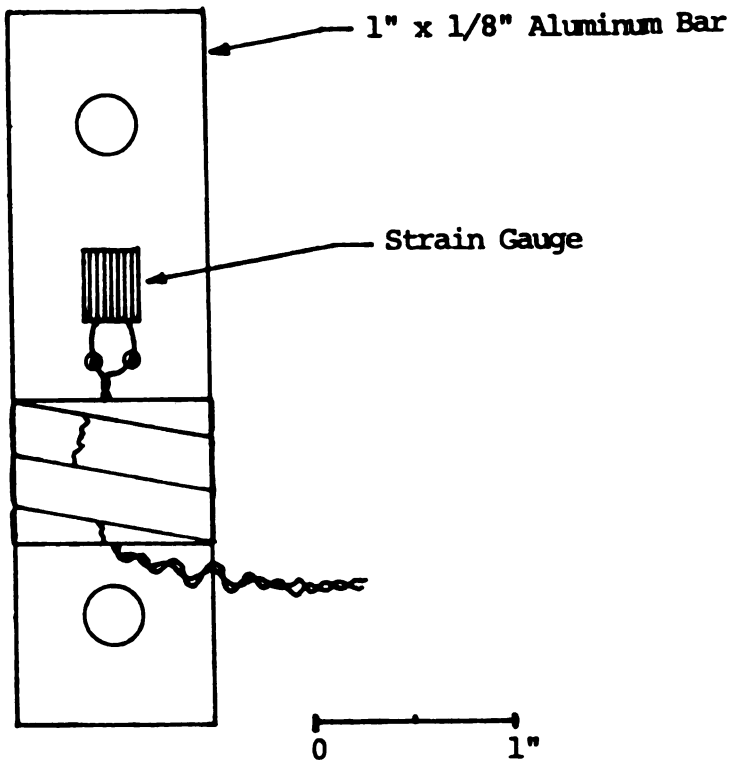


Figure 12. Strain Gauge Load Cell

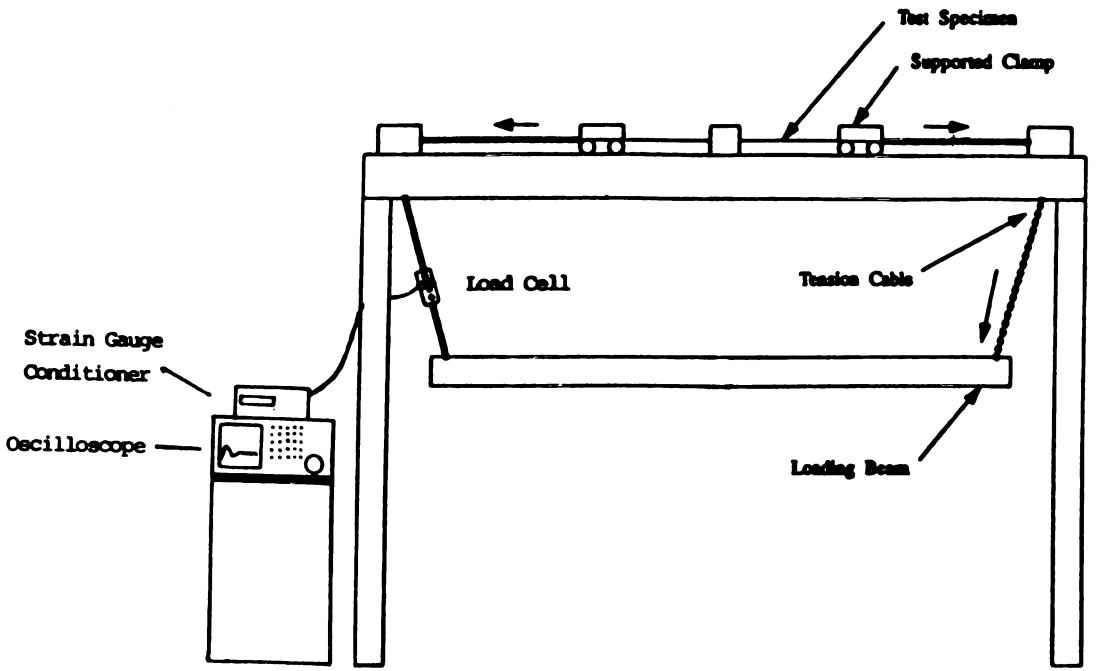


Figure 13. Uniaxial Test Fixture With Load Cell Installed

the three loadings used in the study.

The loading was shown to be a ramp of approximately 90ms. duration (nearly one-half of the information recording period used for parameter estimation), with a peak load value approximately 25% higher than the dead-weight loading of the beams, and with a significant relaxation (Figure 14)[5].

The effects of these deviations from the assumed load vs. time curve on the estimated values for the material constants are worth noting more for their implications in the method than for this particular study. Since the deflection of the material used in this study occurs over such a short period of time and since the K_1 value thus dominates the material model, the error in the K_1 estimate may be corrected directly as a function of the peak loading value. In the case of an experiment with deformation occurring over a more substantial length of time, the K_2 and K_3 variables become more significant, and the error induced in these coefficients may become more significant as well. It may be that for an extended load-time relationship the "ramping" and relaxation would constitute a fairly small component of the overall time history of the material; the major cause for concern would be, once again, error in the K1 estimate caused by the peak loading being larger than the

---

[5]It should be noted that the curves shown are smoothed significantly from the actual data which showed significant electronic noise as well as resonance data from the tensile fixture cables.

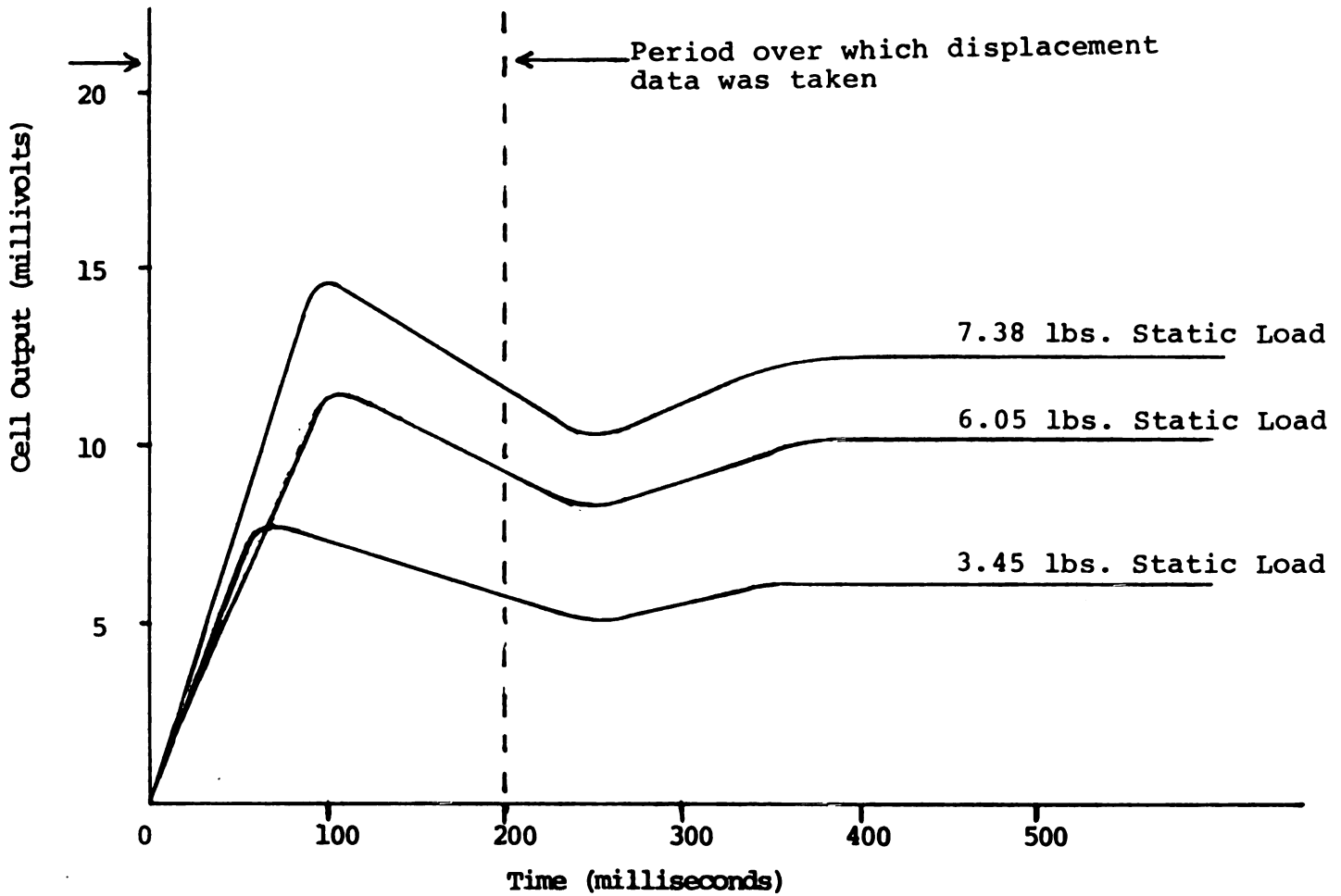


Figure 14. Load vs. Time in Tensile Testing Fixture

static level on the beam.

A solution to the problems caused by the real-world loading history compared with the "perfect" Heaviside step-function would be to incorporate matching variable loading conditions in the finite-element models used as objective functions in the parameter- estimation method and as a modelling method for the strained film. It is unclear, however, at what point the deviation from the assumptions necessary for the development of the material model will begin to affect the accuracy of the analysis. A more practical solution might be to redevelop the material model using a ramp loading function, a combination of ramp and Dirac spike functions, or some other (more accommodating) model rather than a step loading function.

6.3 Strain Within the Specimen

Finite element simulations of the test specimen were constructed and run to ascertain the state of strain in the film, particularly within the region tested for permeation and morphology changes (Figures 15 and 16). The values returned by the material parameter estimation experiments were used as the material constants in previously described VISCO2.FOR software.

The model returned values which were then plotted to give an estimate of the state of strain and change of free volume occurring within the material at the time of measurement. This model provides a good "map" of the area of interest in terms of strain and change of volume, and the change of volume and thinning during the three loading regimes is plotted in Figures 17 to 22.

As the figures clearly illustrate, there is an increase in the free volume of the material in the region of the specimen where the permeation testing took place.

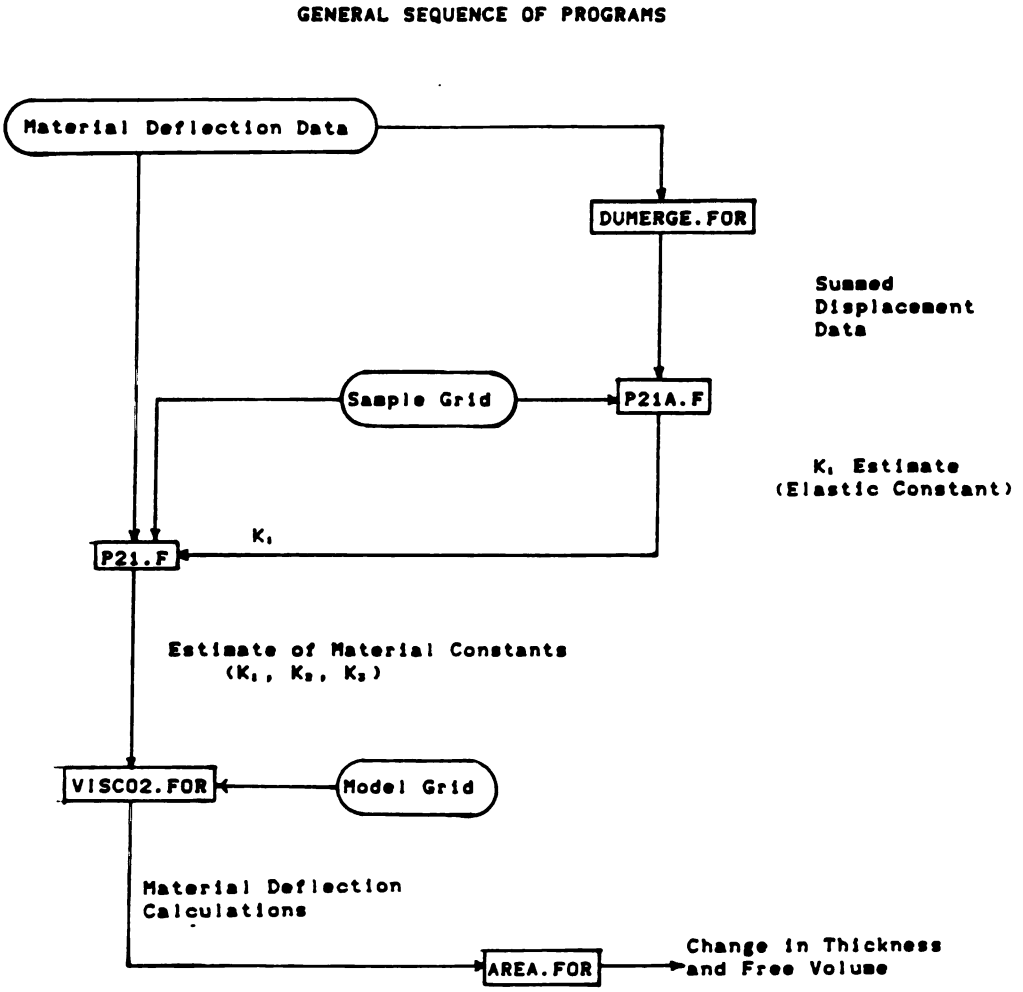


Figure 15. General Sequence of Programs Used in Parameter Estimation and Material Response Calculations

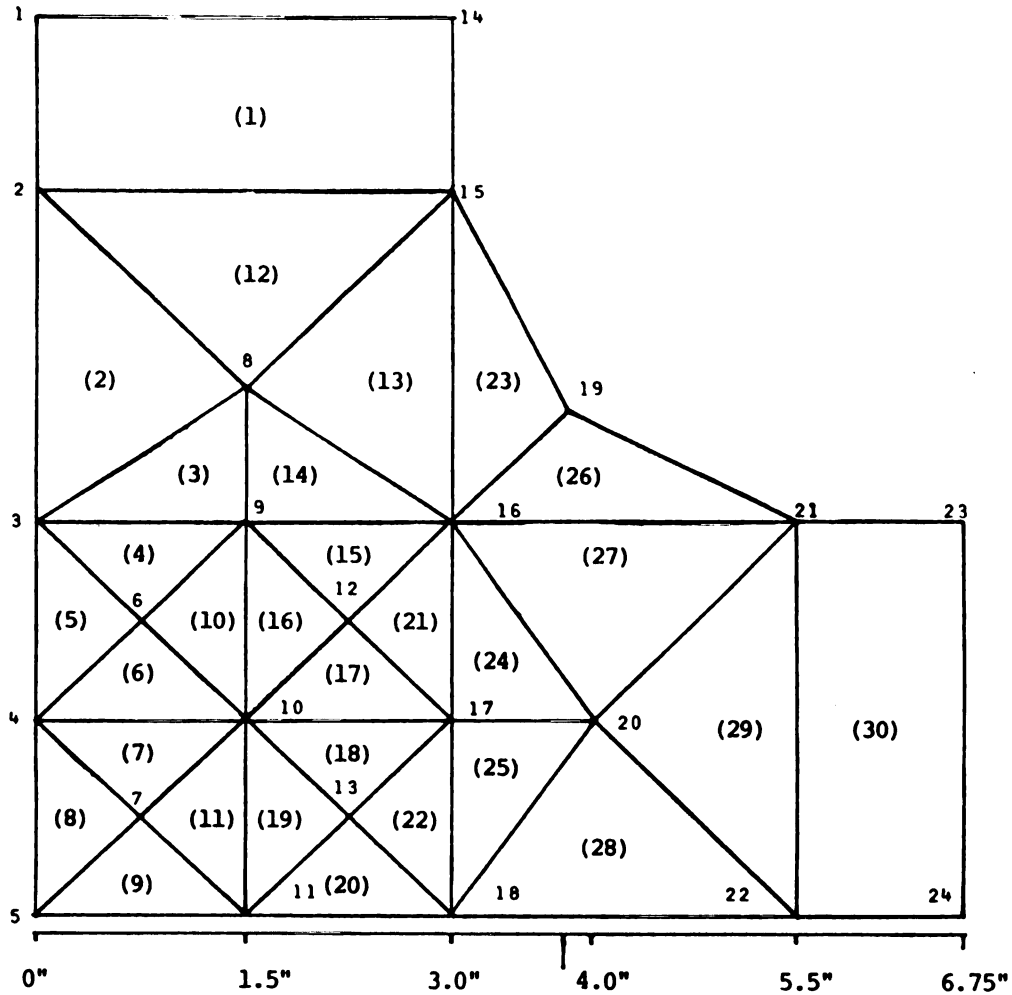


Figure 16. Finite Element Grid Used in Calculating Thinning and Changes in Free Volume.

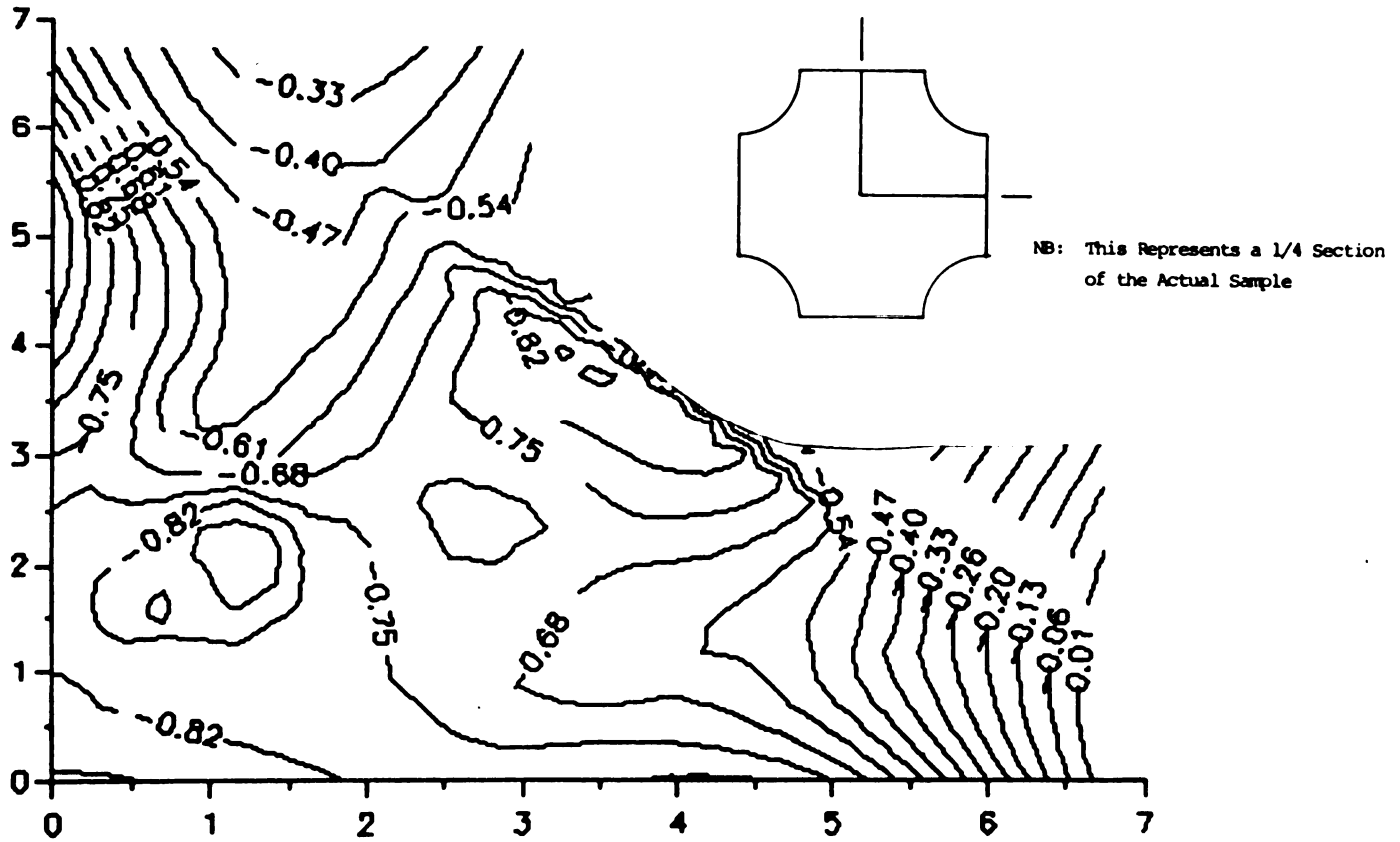


Figure 17. Thinning Calculations for Film at 100% Loading.
(All values are in percent.)

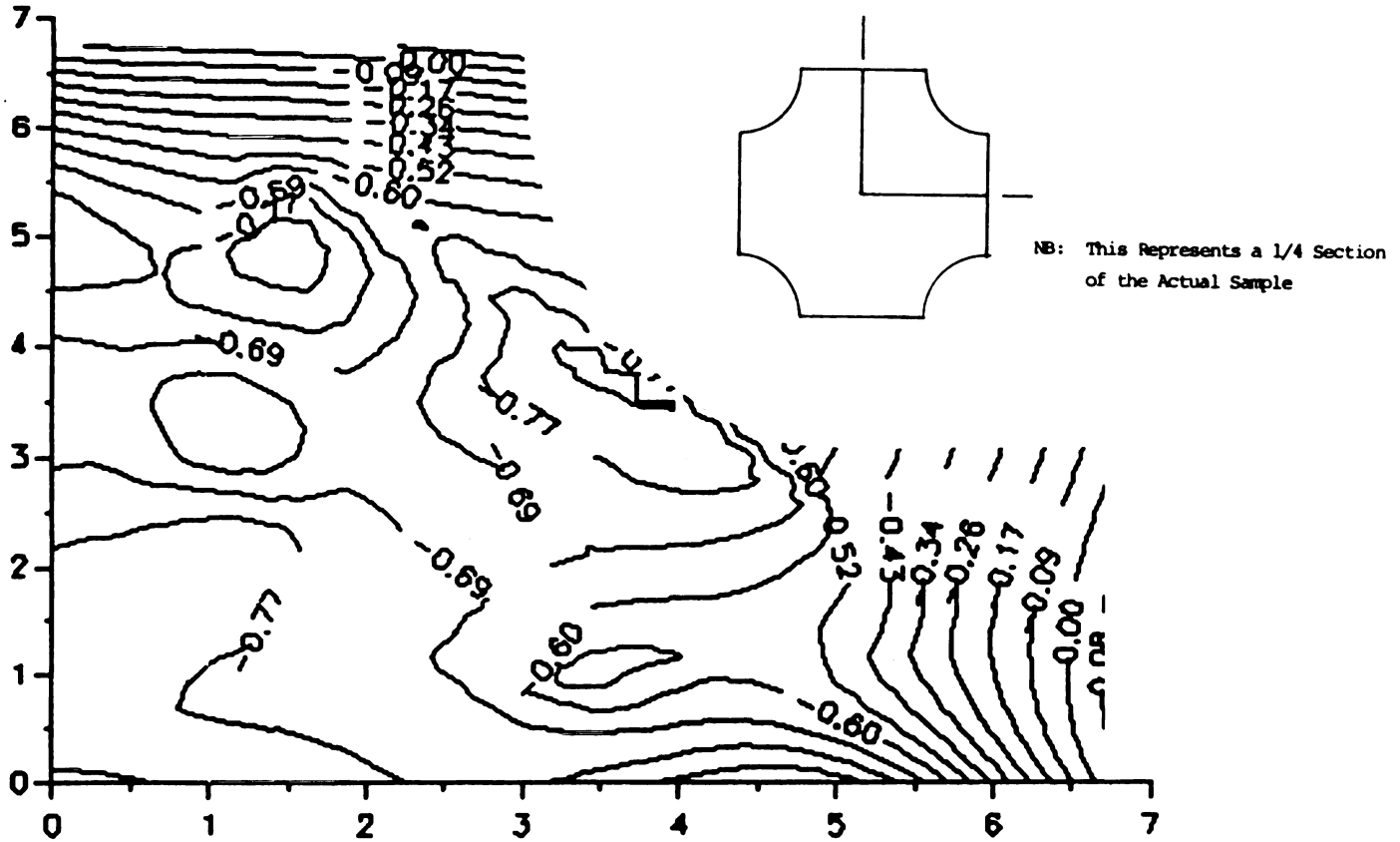


Figure 18. Thinning Calculations for Film at 82% Loading.
(All values are in percent.)

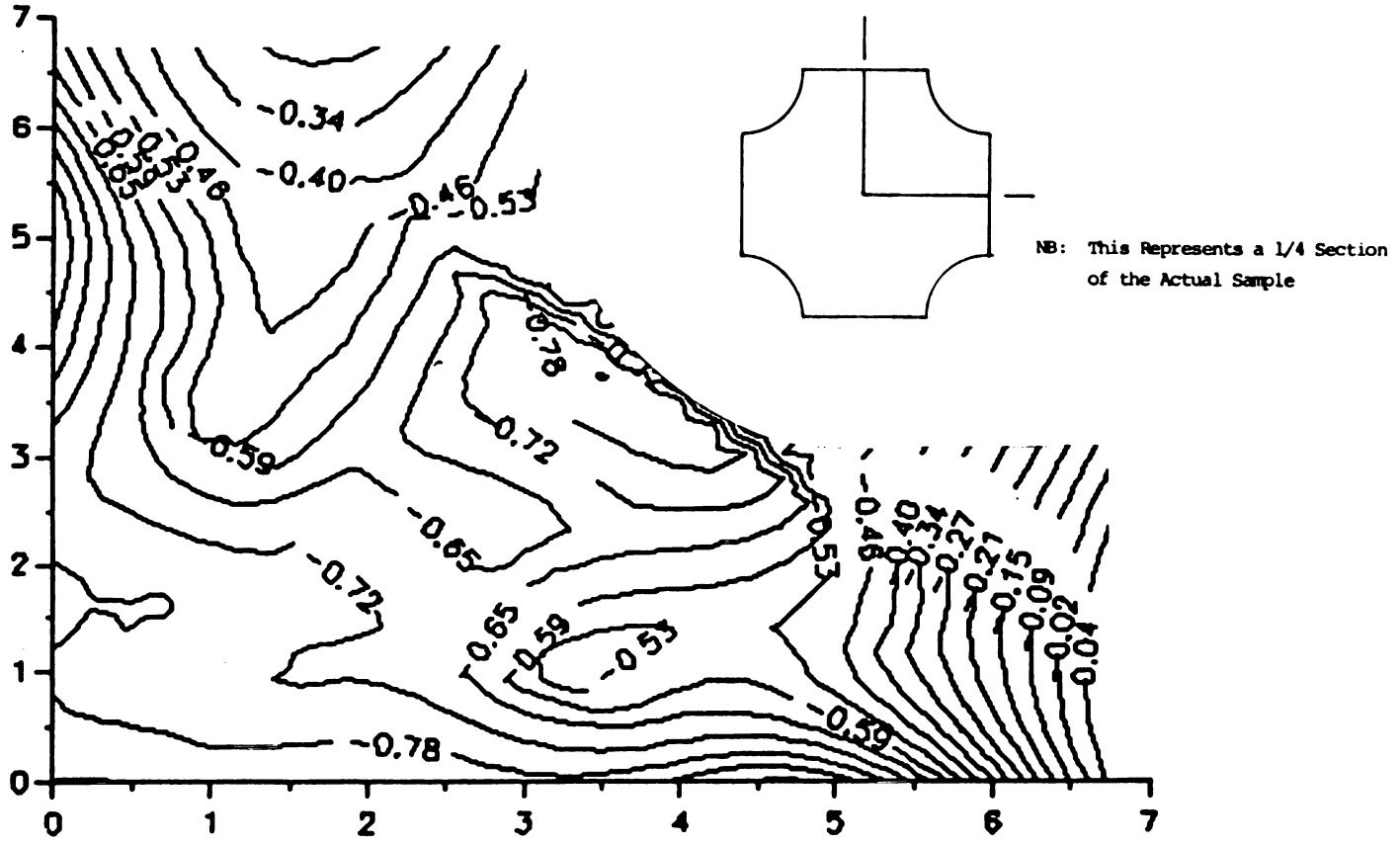


Figure 19. Thinning Calculations for Film at 47% Loading.
(All values are in percent.)

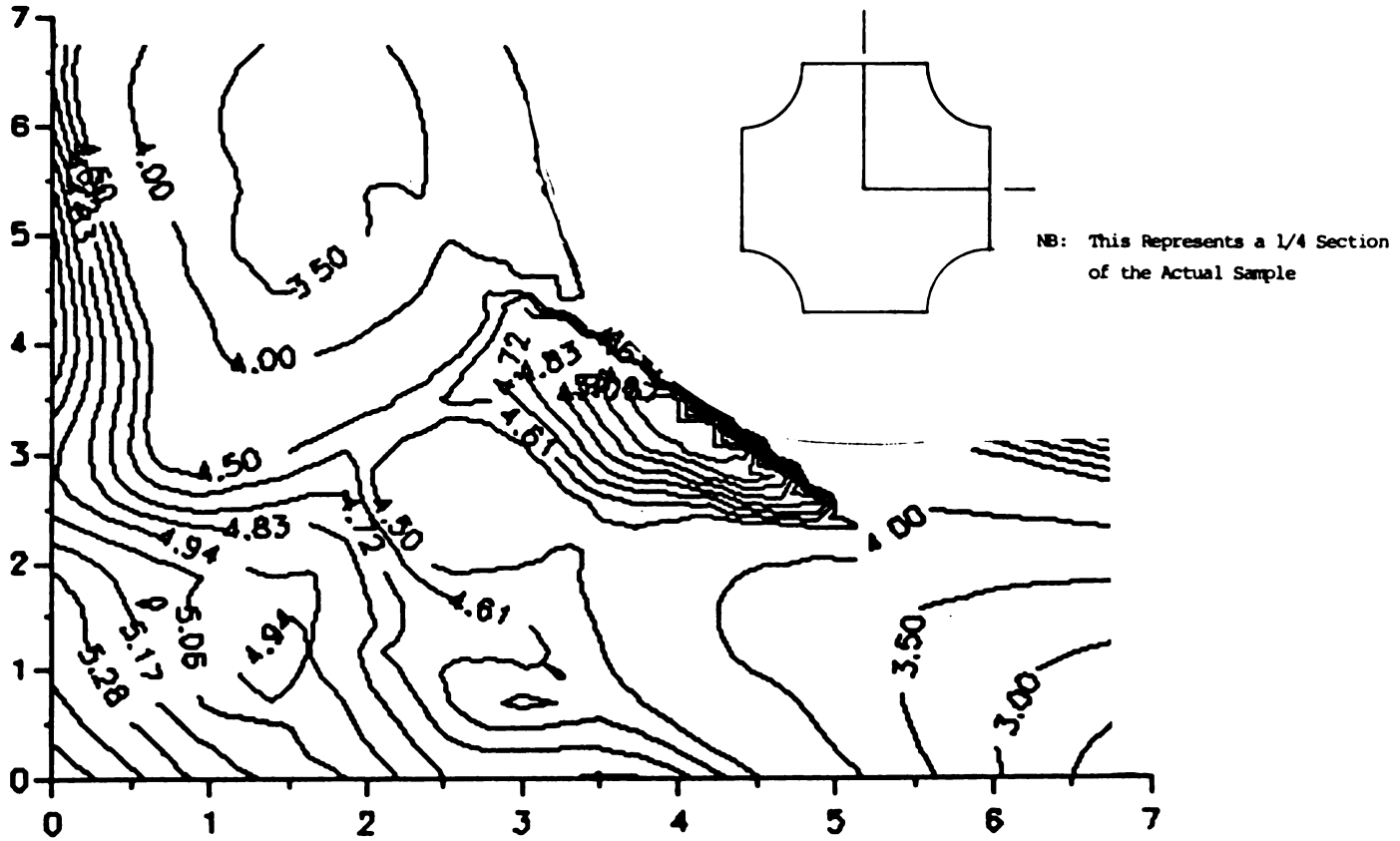


Figure 20. Calculated Change in Free Volume at 100% Loading.
(All values are in percent.)

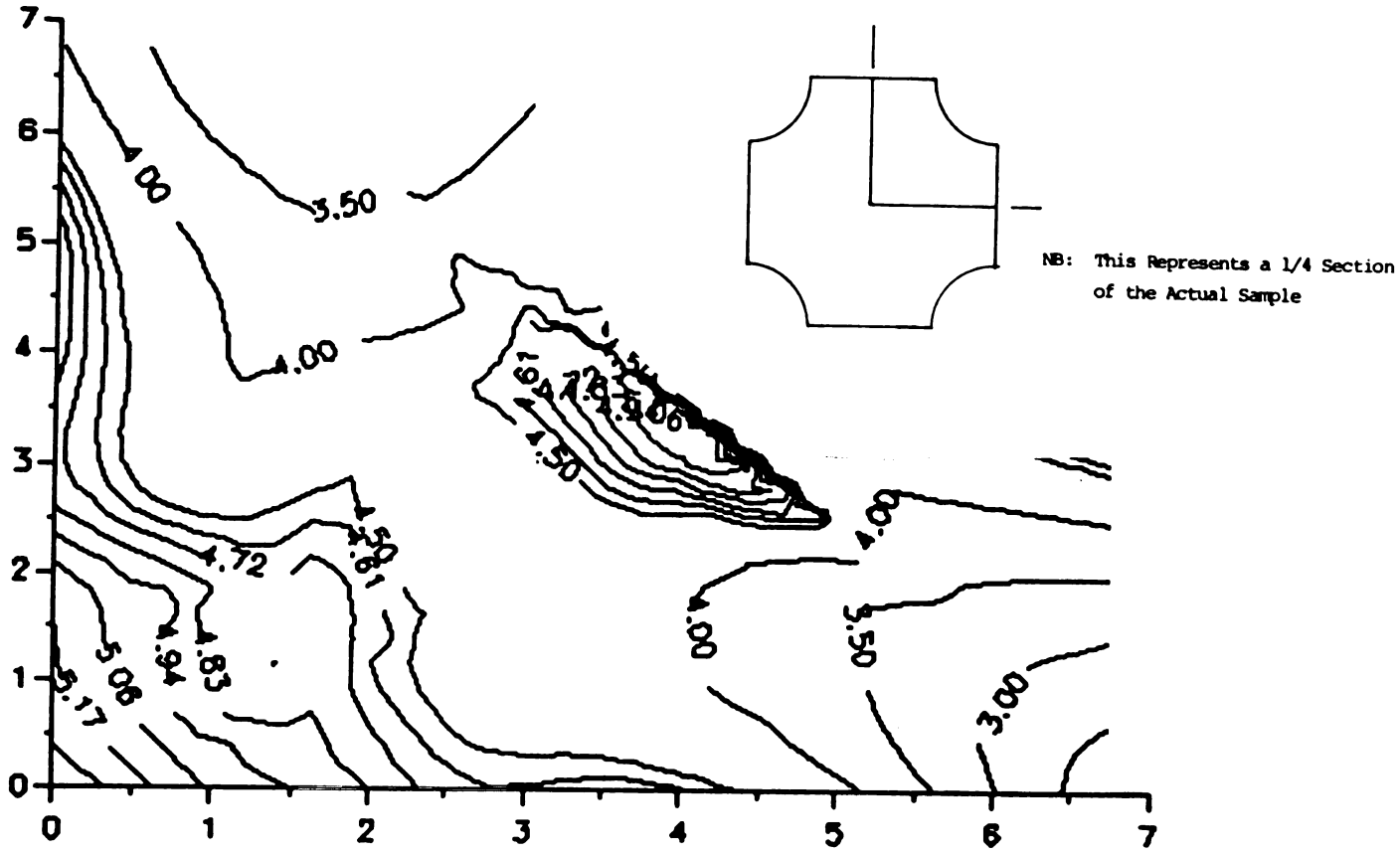


Figure 21. Calculated Change in Free Volume at 82% Loading.
(All values are in percent.)

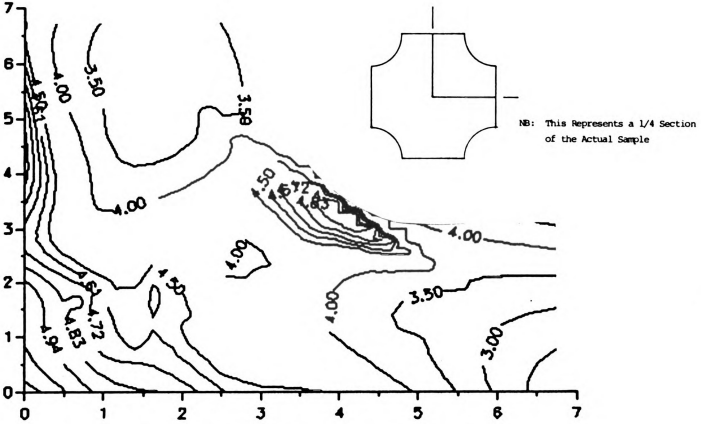


Figure 22. Calculated Change in Free Volume at 47% Loading.
(All values are in percent.)

6.4 Permeation Changes in the Material

The permeation rate of three different samples of the material were measured at three different strain levels each. The relatively small degree of strain occurring in the sample suggests that there would be a marked increase in the rate of permeation, and this was shown to be the case. Additionally the increase was not linear (Figure 23) but the increase was monotonic with increasing strain.

These findings are of considerable interest, since the permeation rate of a material apparently can increase by at least 20% due to small loads in what might be considered the "elastic" range of the material. Although this increase is ostensibly limited by the onset of fibrillar orientation in the material, there is still the potential for this phenomenon to be of interest in the engineering of polymeric material structures, either in the construction of a safety factor where barrier properties are the foremost concern, or in the manipulation of a polymeric film to the desired permeation specifications.

The curvlinear nature of the increase in permeation suggests that the nature of the mechanism by which the permeation increases in the polymer is either extremely sensitive to changes in free volume, or that there is some

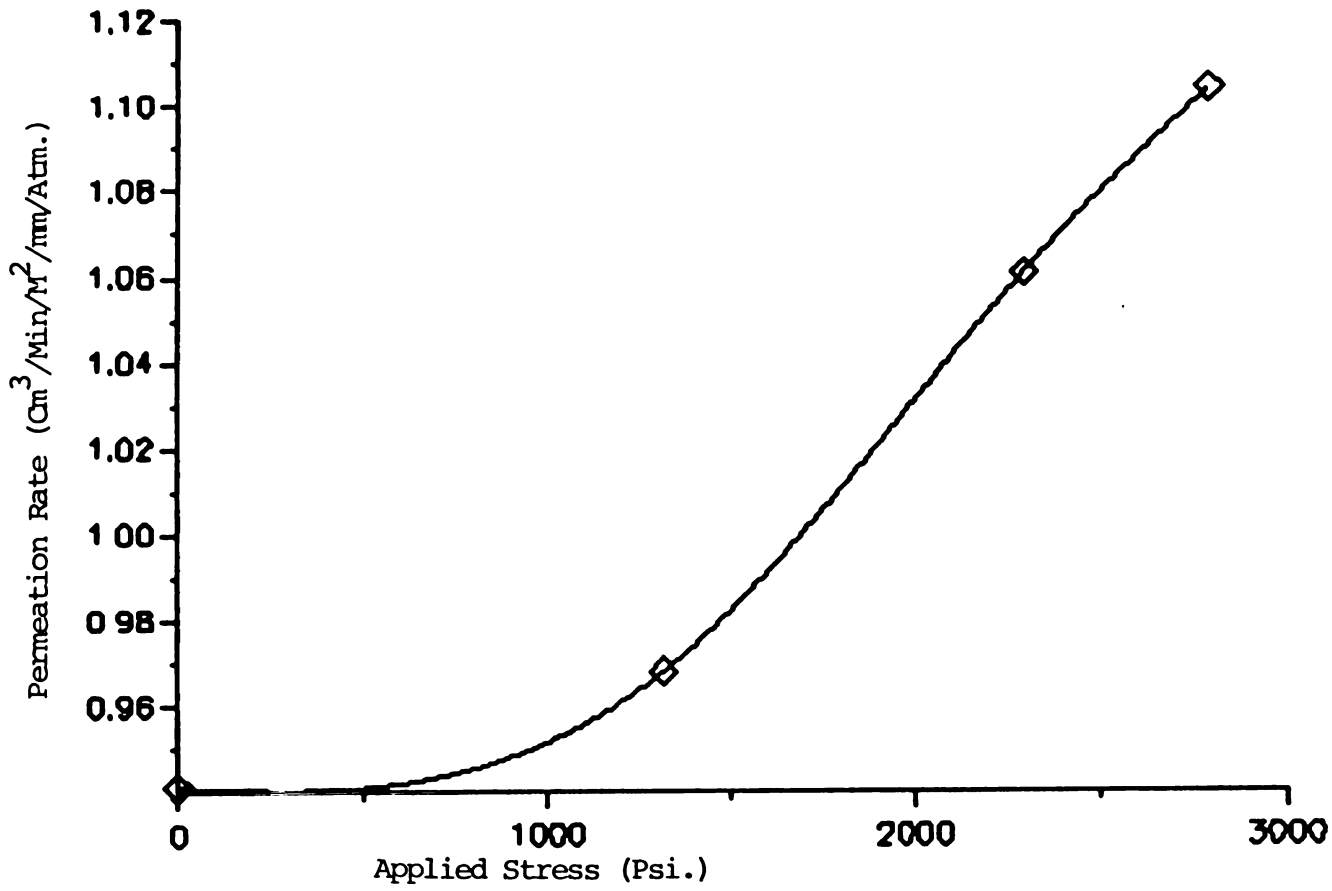


Figure 23. Graph of Permeation Rate Versus Stress Applied to Sample.

threshold level within the material above which the material has a much higher level of permeation. In the latter case, the permeation increase within the material tested would vary as the threshold-exceeded area increases, giving an approximately second-order permeation curve with a biaxially loaded sample.

6.5 Small Angle Light Scattering Analysis

The Small Angle Light Scattering (SALS) method used to test the film for changes in orientation showed that the material had a distribution of spherulites on the order of 5μm which were oriented in the "machine direction[6]" of the film. This low level of orientation is to be expected due to the stresses applied to all films during production. Unfortunately, even the largest strain applied to the material failed to show any appreciable difference in orientation direction or magnitude (as shown by the direction and degree of extension of the "flare" in the pattern in the photographs) between the strained and unstrained specimens in either the H_f (polarizers crossed) or H_v (polarizers parallel) configurations (Figures 25 to 28). This was expected from the small-strain model which suggests that the changes which occur in the material are due solely to free-volume changes in the structure in the polymer rather than the formation of morphological artifacts which would significantly alter the optical activity of the material.

---

[6]Machine direction refers to the direction in which the material is rolled up on a spool or otherwise transported through processing machinery. Almost all materials show some degree of structural orientation along the machine direction unless the process has been designed to eliminate these.

Figure 24. SALS H_v Pattern Before Applied Load

Figure 25. SALS H_v Pattern After Applied Load

Figure 26. SALS H_f Pattern Before Applied Load

Figure 27. SALS H_f Pattern After Applied Load

6.6 Scanning Electron Microscope Evaluation

The electron photomicrographs of the surface features of unstrained and maximally strained film showed no significant changes in the surface features of the film between the strained and unstrained state (Figures 29 and 30). Again, this is to be expected from a phenomenon which does not alter the morphological nature of the material. The apparent difference in surface texture is largely due to the difficulty in focusing the SEM on an almost featureless surface.

An interesting observation to be made from these photomicrographs is the existance of what appear to be "pores" in the material, although these do not change with the strain in the material and may or may not be artifacts of the replicating process.

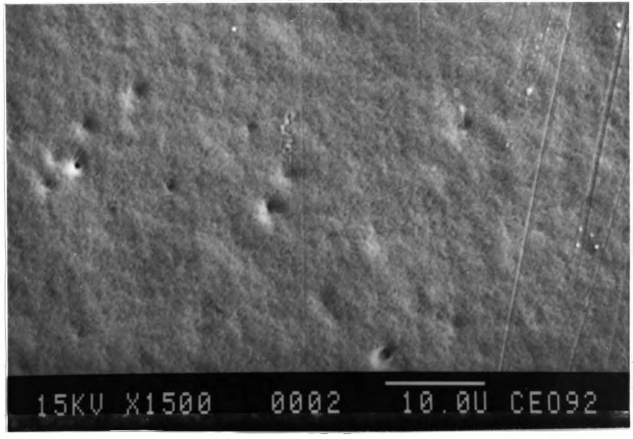


Figure 28. Photomicrograph of film before deformation.

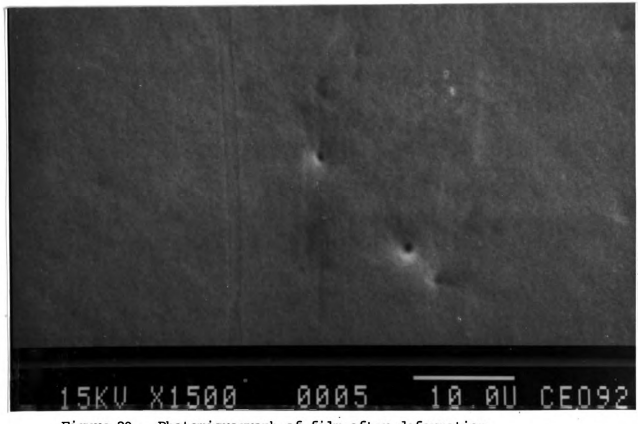


Figure 29. Photomicrograph of film after deformation.

7.0 SUMMARY AND CONCLUSIONS

7.1 Summary

The preceeding study developed a numerical model of the mechanics of deformable solids to estimate the thinning and free-volume changes that occur as a polymer film sample is stretched under a "stepped" loading regime. A tensile testing fixture was developed to load the film to various degrees of tensile stress, and to measure the movement of index marks applied to the test samples for the elucidation of material property constant estimates for use in the numerical models. The tensile fixture was also used to hold the stressed film for measurement of permeation and changes in optical activity once the loading was applied to a sample.

A permeation cell was modified for use with the abovementioned tensile tester to measure the CO_2 permeation rate at the center portion of the film sample. Further testing of the center region of the sample was accomplished using a Small Angle Light Scattering apparatus developed specifically for the study, and by casting film surface replicas for analysis by Scanning Electron Microscope.

With these tests and methods, it has been shown that the linkage between applied force and changes in permeation can

be quantified, with a much better understanding of the specific changes that are occurring in the sample.

7.2 Conclusions

It is clear that a large change of permeation is possible with relatively small dimensional changes, as predicted in the literature and shown in this study. Further, the changes may be correlated to a close estimate of the state of strain, thinning and volume change of the material at the time of permeation although the exact microstructural mechanism by which these changes occur is not elucidated by the analytical methods used here.

The utility of this type of analysis is immediately apparent; it is now possible to predict the changes in permeation occuring within a polymer film structure under mechanical loading without construction of a prototype and without testing beyond that which is necessary to ascertain a few simple material properties.

Extension of the fundamental principles underlying this study--the construction of a quantitative linkage between mechanical input and barrier property changes--should allow more rapid, and therefore more cost-effective, design and development of materials, production processes and structures using these materials.

8.0 RECOMMENDATIONS

The study described here has proven the feasibility of the linkage of several types of studies to produce a clearer picture of the circumstances surrounding the changes in permeation in strained material. To be a more practical engineering tool several improvements are in order:

The collection methodology with which the material-property data was obtained must be improved. The method has the potential to be quite accurate, but it is hampered by the inaccuracy and error-producing tedium of visual measurements and the low resolution of the video system used. A high-speed imaging and digitization system would be very useful.

A better understanding of the changes in strained polymer structure is necessary. Since gross structural changes are not occurring within the polymer, a more sensitive measure of the changes occurring (such as wide-angle X-ray scattering) may provide additional information.

A more robust tensile strain fixture is necessary if high-speed deformations are to be studied since the one used in this study exhibited a great deal of secondary motion when activated.

A load-deformation time history needs to be recorded to give a more accurate picture of the load response of the

film, and perhaps a more complex material model is in order to accomodate the less than ideal load-time history produced by the equipment used.

APPENDIX A

Program Listings

This appendix contains listings for the following main programs and (limited) supporting documentation:

VISCO1.FOR

VISCO2.FOR

P21.FOR

P21a.FOR

DUMERGE.FOR

File Input Format for VISCO1.FOR and VISCO2.FOR

Additional File Input Format for P21.FOR and P21a.FOR

Table 3. VISCO1.FOR

```
      PROGRAM VISCO1.FOR

C                 DECLARE COMMON STATEMENTS
      DOUBLE PRECISION K(8,8)

      DOUBLE PRECISION INVRS(200,200)
      DOUBLE PRECISION KINV(200,200)

      COMMON/NUM/NUMELS
      COMMON/NNBLOCK/NUMNODES

      DOUBLE PRECISION R(200)
      COMMON/RBLOCK/R

      DOUBLE PRECISION COEFF(6)
      COMMON/COEFFBLOCK/COEFF

      COMMON/IDIMBLOCK/IDIM

      COMMON /NTS/NUMBEROFTIMESTEPS

      DOUBLE PRECISION TIMSTART, TIMINCR

      DOUBLE PRECISION NODX(200),NODY(200)
      COMMON/NODE/NODX,NODY

      INTEGER EI(325),EJ(325),EK(325),EM(325)
      COMMON/ELNODES/EI,EJ,EK,EM

      DOUBLE PRECISION MASTERK(200,200)
      COMMON/BIGK/MASTERK

      DOUBLE PRECISION A(3,3)
      COMMON/ABLOCK/A

      INTEGER NUMSPYK
      COMMON/NS/NUMSTART,NUMSTOP

      INTEGER IBDY(200)
      COMMON/IBINDX/IBDY

      DOUBLE PRECISION BVAL(200)
      COMMON/BOUNDVAL/BVAL

      INTEGER NUMBDY
      COMMON/BDY/NUMBDY

      INTEGER IDPLUS1
      COMMON/IDP/IDPLUS1

      DOUBLE PRECISION KPAST(200,200)
      COMMON/KPASTBLOCK/KPAST
```

Table 3 (cont'd)

```
      DOUBLE PRECISION DELU(200)

      INTEGER IROW(200),JCOL(200),JORD(200)
      COMMON/IJJ/IROW,JCOL,JORD

      DOUBLE PRECISION KZERO(200,200)
      COMMON/KZ/KZERO

      DOUBLE PRECISION Y(200)
      COMMON/WYE/Y

      DOUBLE PRECISION FINALK(200,200),LASTR(200)
      COMMON/ENDO/FINALK,LASTR

      SAVE

  100 FORMAT(A)


      IF (NFE .EQ. 1) THEN

      WRITE (*,100)'                 *********   PARAVISCO 1    ************  '

      ENDIF

C ____________________________________________________________________________
C ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
C                            OPEN FILES
C
C  This block of commands opens, labels, and numbers the appropriate files for
C  use by VISCO1 with the exception of the series of files needed for [K(t)]
C  storage, as those are created as needed.

      OPEN (3, FILE='GENERAL_DATA', STATUS= 'OLD')

      REWIND 3

      OPEN (9, FILE='BOUNDARIES', STATUS= 'OLD')

      REWIND 9

c ____________________________________________________________________________
C ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
C                        READ IN INITAL DATA
C This reads in some of the necessary parameters to operate some of
c the arrays used in this program.

      READ (3,*)NUMELS,NumberOfNodes
        NUMNODES=NumberOfNodes*2
          NODCOUNT=NUMNODES
        NUMN=NUMNODES
      READ (3,*)(COEFF(I),I=1,6)
      READ (3,*)NUMBEROFTIMESTEPS
      READ (3,*)NUMSTART,NUMSTOP
      READ (3,*)TIMSTART,TIMINCR
```

```
      CALL MAKEARRAYS(IT)

130   FORMAT (I2,15X,D12.6)

      IF ((NFE .EQ. 1) .OR. (NPRNT .GE. 1)) THEN

      WRITE (*,100) '          COEFFICIENT        VALUE'
      DO J=1,6
      WRITE (*,130)J,COEFF(J)
      ENDDO

      ENDIF

C     Read in the known boundary conditions
C     from the 'boundaries.dat file:

      IF (NFE .EQ. 1) THEN
      WRITE (*,100)'  '
      WRITE (*,100) '  NUMBER OF KNOWN DISPLACEMENT VALUES:'
      ENDIF

      READ (9,*) NUMBDY

      IF (NFE .EQ. 1) THEN
      WRITE (*,*) NUMBDY
      WRITE (*,100)'  '
      WRITE (*,100) ' DIRECTION INDICATOR:  X=1   Y=0'
      WRITE (*,100) '        NODE       DIRECTION            VALUE'
      ENDIF

      DO I=1,NUMBDY
      READ (9,*) IBNDX, IDIR, BVAL(I)

      IF (NFE .EQ. 1) THEN
      WRITE (*,*) IBNDX,IDIR,BVAL(I)
      WRITE (*,100)'  '
      ENDIF

C  IDIR:  X=1   Y=0  FOR 2-D PROBLEMS

      IBDY(I)=(IBNDX*2)-IDIR
      ENDDO
      CLOSE (9)

C *  Once arrays are ready, construct the series of [K(t)] values.
C    for all of the timesteps in the problem.

      IF (NFE .EQ. 1) THEN
      WRITE (*,100)'  STORING [K] MATRICES FOR '
      ENDIF

      DO 5,NT=0,NUMBEROFTIMESTEPS

      IF (NFE .EQ. 1) THEN
      WRITE (*,101)' TIMESTEP=',NT
      ENDIF

101   FORMAT (A,I2)


        CALL MAKEA(NT,TIMINCR,TIMSTART)

C                            I       O
        CALL MAKESHAPES( MASTERK,NUMNODES)

C                            O     O       O
        CALL STOREMASTERK(NT,MASTERK,NUMNODES)
```

```
5       CONTINUE

C *  Solve the time-dependant problem, once all of the [K] values are
C    ready and stored.
        NNPLUS1=NUMNODES+1
        IDIM=NUMNODES-NUMBDY
        IDPLUS1=IDIM+1
        CALL SOLUTION(NUMNODES,NNPLUS1,KZERO,IDIM,IDPLUS1,FINALK,
     C  LASTR,KINV)

        WRITE (*,100) ' SOLUTION COMPLETED!!!!!'
        WRITE (*,100) '  '

        CLOSE (3)


        END
C ____________________________________________________________________
```

```
C ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
        SUBROUTINE MAKEARRAYS(IT)

C This subroutine sets up the indexed array of node and element values used
C by the [K] generation loops.

        INTEGER NODNO,IELNO

        COMMON/NUM/NUMELS

        DOUBLE PRECISION NODX(200),NODY(200)
        COMMON/NODE/NODX,NODY

        INTEGER EI(325),EJ(325),EK(325),EM(325)
        COMMON/ELNODES/EI,EJ,EK,EM


        OPEN (4, FILE='ELEMENT_DATA', STATUS= 'OLD')
        REWIND 4

C  * Create an array of indexed x & y values associated with each node

100     FORMAT (A)

        IF (NFE .EQ. 1) THEN
        WRITE(*,100) '        NODE           X                        Y'
        ENDIF

20      CONTINUE
          READ (4,*) NODNO
           IF (NODNO .EQ. -1) GO TO 23
          READ (4,*) NODX(NODNO), NODY(NODNO)

        IF (NFE .EQ. 1) THEN
          WRITE (*,*) NODNO,NODX(NODNO),NODY(NODNO)
        ENDIF

         GO TO 20



23      IF (NFE .EQ. 1) THEN

        WRITE (*,100) '  '
        WRITE (*,100) '      ELEMENT           I           J           K
     C       M'
        ENDIF

24      CONTINUE
* Produce an index of nodal values associated with each element

          READ (4,*) IELNO
           IF (IELNO .EQ. -1) GO TO 25
          READ (4,*) EI(IELNO), EJ(IELNO), EK(IELNO), EM(IELNO)

        IF (NFE .EQ. 1) THEN
        WRITE (*,*) IELNO,EI(IELNO),EJ(IELNO),EK(IELNO),EM(IELNO)
        ENDIF

        GO TO 24
25      CONTINUE

        RETURN
        END

C ___________________________________________________________________________
```

```
C ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
C
        SUBROUTINE MAKEA(NT,TIMINCR,TIMSTART)

        DOUBLE PRECISION TIMEVAL,TIMINCR,TIMSTART
        DOUBLE PRECISION A(3,3)
        DOUBLE PRECISION COEFF(6)
        DOUBLE PRECISION G1,G2,B,C,D

        COMMON/COEFFBLOCK/COEFF
        COMMON/ABLOCK/A


100     FORMAT (A)
101     FORMAT (A,I2)

C  Calculate the value of the timestep in seconds:

        TIMEVAL=(NT*TIMINCR)+TIMSTART
C       WRITE (*,*) TIMEVAL
C       WRITE (*,100) '  '


C       GO TO 30

C  This Subroutine computes the values for E and MU at some time-step NT.  It
C  utilizes a 3-parameter exponential decay model for the elastic modulus and
C  Poisson's ratio.  The COEFF array contains the necessary coefficients.

        EM=COEFF(1)+COEFF(2)*DEXP(-1.0D0*(COEFF(3)*TIMEVAL))
        PR=COEFF(4)+COEFF(5)*DEXP(-1.0D0*(COEFF(6)*TIMEVAL))

C  This part takes the E and MU values for the current time value
C  and returns the [A] matrix for that time value.


40      B=EM/(1.0D0-(PR**2.0D0))

        A(1,1)=B
        A(2,2)=B
        A(3,3)=B*(1.0D0-PR)/2.0D0
        A(1,2)=PR*B
        A(2,1)=A(2,1)
        A(1,3)=0.D0
        A(2,3)=0.D0
        A(3,1)=0.D0
        A(3,2)=0.D0

C       WRITE (*,100) '    [A]  MATRIX VALUES'

C       DO IROW=1,3
C         DO JCOL=1,3
C           WRITE (*,*) IROW,JCOL,A(IROW,JCOL)
C         ENDDO
C       ENDDO

50      CONTINUE

        RETURN
        END

C ________________________________________________________________________
```

```
C ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

        SUBROUTINE MAKESHAPES(MASTERK,NUMNODES)

        DOUBLE PRECISION MASTERK(NUMNODES,NUMNODES)
        DOUBLE PRECISION K(8,8)

        COMMON/NUM/NUMELS

        INTEGER EI(325),EJ(325),EK(325),EM(325)
        COMMON/ELNODES/EI,EJ,EK,EM

100     FORMAT (A)
        IEIGHT=8
C -This Subroutine through each of the elements and (using the proper
c  subroutine) develops an elemental [k] matrix to be added into the [K] via
c  the MERGEK subroutine.


C * Put the two together and route to appropriate calculation of [k] for
C   each element then add the value for that element into the global [K]

C   But first, the MASTERK must be cleared from the last time-step.

        DO II=1,NUMNODES
          DO JJ=1,NUMNODES
           MASTERK(II,JJ)=0.0D0
          ENDDO
        ENDDO

C   Proceed to assemble next MASTERK

        DO 30,IElement=1,NUMELS
          IF (EM(IElement).EQ.0) THEN
C                                   O     I
              CALL TRIANGLELEM(IElement,K)
              GO TO 27
          ENDIF
C                                 O    I
          CALL SQUARELEM(IELEMENT,K)

 27     CONTINUE

C                          O    O
         CALL MERGEK(K,IElement,MASTERK,NUMNODES)

C  Note that MERGEK merges the element's [K] value into the COMMON MASTERK()
C  and does not return any value.

30      CONTINUE


        RETURN
        END

C ______________________________________________________________________
```

```
C ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
C                            I        O
      SUBROUTINE TRIANGLELEM(IElement,K)

      DOUBLE PRECISION K(8,8)

      DOUBLE PRECISION NODX(200),NODY(200)
      COMMON/NODE/NODX,NODY

      INTEGER EI(325),EJ(325),EK(325),EM(325)
      COMMON/ELNODES/EI,EJ,EK,EM

      DOUBLE PRECISION A(3,3)
      COMMON/ABLOCK/A

      DOUBLE PRECISION XI,XJ,XK,XM,YI,YJ,YK,YM,BI,BJ,BK,CI,CJ,CK

      DOUBLE PRECISION X(3),Y(3),B(3,6),C(6,3),AR2,SUM

C  This subroutine uses the brute-force calculations in TRIANGLECALC
C  to produce a [k] for the selected element.

100   FORMAT (A)

C  ********
      TH=1.0D0
C  ********

      XI=NODX(EI(IElement))
      XJ=NODX(EJ(IElement))
      XK=NODX(EK(IElement))
      YI=NODY(EI(IElement))
      YJ=NODY(EJ(IElement))
      YK=NODY(EK(IElement))

      X(1)=XI
      X(2)=XJ
      X(3)=XK
      Y(1)=YI
      Y(2)=YJ
      Y(3)=YK

C ************************************************************************
C *                                                                      *
C *  What follows is from "Applied Finite Element Analysis"              *
C *  Larry J. Segerlind, J Wiley & Sons, 1984                            *
C *  P. 347                                                              *
C ************************************************************************

C  CLEAN HOUSE:

      DO I=1,8
      DO J=1,8
        K(I,J)=0.0D0
      ENDDO
      ENDDO


      DO I=1,3
        DO J=1,6
          B(I,J)=0.0D0
          C(J,I)=0.0D0
        ENDDO
      ENDDO

C  GENERATE THE {B} MATRIX
```

```
      B(1,1)=Y(2)-Y(3)
      B(1,3)=Y(3)-Y(1)
      B(1,5)=Y(1)-Y(2)
      B(2,2)=X(3)-X(2)
      B(2,4)=X(1)-X(3)
      B(2,6)=X(2)-X(1)
      B(3,1)=B(2,2)
      B(3,2)=B(1,1)
      B(3,3)=B(2,4)
      B(3,4)=B(1,3)
      B(3,5)=B(2,6)
      B(3,6)=B(1,5)

      AR2=X(2)*Y(3)+X(3)*Y(1)+X(1)*Y(2)-X(2)*Y(1)-X(3)*Y(2)-X(1)*Y(3)

C  MATRIX MULTIPLCATION TO OBTAIN C = {BT}[A]

      DO I=1,6
      DO J=1,3
          C(I,J)=0.0D0
          DO L=1,3
            C(I,J)=C(I,J)+B(L,I)*A(L,J)
          ENDDO
      ENDDO
      ENDDO
C MATRIX MULTIPLICATION TO OBTAIN [K] WHERE
C   [K]= {BT}[A]{B} = {C}{B}

      DO 27 I=1,6
      DO 27 J=1,6
        SUM=0.0D0
        DO 28 L=1,3
 28       SUM=SUM+C(I,L)*B(L,J)
        K(I,J)=SUM*TH/(2.0D0*AR2)
 27   CONTINUE

C  RETURN [K]

      RETURN
      END

C ____________________________________________________________
```

```
C ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
C                          I        O
      SUBROUTINE SQUARELEM(IElement,k)
      DOUBLE PRECISION k(8,8),C(6)

      DOUBLE PRECISION NODX(200),NODY(200)
      COMMON/NODE/NODX,NODY

      DOUBLE PRECISION A(3,3)
      COMMON/ABLOCK/A

      DOUBLE PRECISION AA,B

      INTEGER EI(325),EJ(325),EK(325),EM(325)
      COMMON/ELNODES/EI,EJ,EK,EM

      DOUBLE PRECISION XI,XJ,XK,XM,YI,YJ,YK,YM

C  This subroutine uses the brute-force calculations in SQUARECALC
C  to produce a [k] for the selected element.

100   FORMAT (A)


      XI=NODX(EI(IElement))
      XJ=NODX(EJ(IElement))
      XM=NODX(EM(IElement))
      YI=NODY(EI(IElement))
      YJ=NODY(EJ(IElement))
      YM=NODY(EM(IElement))

      AA=0.5D0*DSQRT(((XM-XI)**2.0D0)+((YM-YI)**2.0D0))
      B=0.5D0*DSQRT(((XJ-XI)**2.0D0)+((YJ-YI)**2.0D0))

      C(1)=(A(1,1)*AA)/(6.0D0*B)
      C(2)=(A(1,1)*B)/(6.0D0*AA)
      C(3)=A(1,2)/4.0D0
      C(4)=A(3,3)/4.0D0
      C(5)=(A(3,3)*AA)/(6.0D0*B)
      C(6)=(A(3,3)*B)/(6.0D0*AA)

C                         O I
      CALL RECTANGLECALC (C,k)

      RETURN
      END

C ____________________________________________________________________________
```

```
C ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

        SUBROUTINE MERGEK(K,IElement,MASTERK,NUMNODES)

        DOUBLE PRECISION MASTERK(NUMNODES,NUMNODES)

        COMMON/NUM/NUMELS
        DOUBLE PRECISION K(8,8)
        INTEGER SK(8)

        INTEGER EI(325),EJ(325),EK(325),EM(325)
        COMMON/ELNODES/EI,EJ,EK,EM

100     FORMAT(A)

C -This Subroutine adds the [K] for some element into the global [K]
C  for some time-step.
C -MasterK is Global [K] for a given time step.
C -IEL Contains the node-list for the element.
C -NEL Number of elements.

        SK(1)=2*EI(IELEMENT)-1
        SK(2)=2*EI(IELEMENT)
        SK(3)=2*EJ(IELEMENT)-1
        SK(4)=2*EJ(IELEMENT)
        SK(5)=2*EK(IELEMENT)-1
        SK(6)=2*EK(IELEMENT)

C       This skips the EM for the triangular element to avoid
C       SK(7) and SK(8) = -1 and MERGEs the SQUARE element.

        IF (EM(IELEMENT).NE.0) THEN

        SK(7)=2*EM(IELEMENT)-1
        SK(8)=2*EM(IELEMENT)

        DO 15,I=1,8
          DO 10,J=1,8
            MASTERK(SK(I),SK(J))=MASTERK(SK(I),SK(J))+K(I,J)
10        CONTINUE
15      CONTINUE

        ELSE
C  This is the MERGE routine for the TRIANGULAR element occurrs.

20      DO 35,I=1,6
          DO 30,J=1,6
            MASTERK(SK(I),SK(J))=MASTERK(SK(I),SK(J))+K(I,J)
30        CONTINUE
35      CONTINUE

        ENDIF

        RETURN
        END

C _____________________________________________________________________________
```

```
C ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^


        SUBROUTINE RECTANGLECALC(C,k)
        DOUBLE PRECISION C(6),K(8,8)

100     FORMAT (A)


C  This subroutine returns a brute force solution to the calculation of the
C  [K] matrix for the RECTANGULAR element.  Ugly but fast.

        K(1,1)=2.0D0*(C(1)+C(6))
        K(1,2)=C(3)+C(4)
        K(1,3)=-2.0D0*C(1)+C(6)
        K(1,4)=C(3)-C(4)
        K(1,5)=-1.0D0*(C(1)+C(6))
        K(1,6)=-1.0D0*(C(3)+C(4))
        K(1,7)=C(1)-2.0D0*C(6)
        K(1,8)=-1.0D0*C(3)+C(4)

        K(2,1)=K(1,2)
        K(2,2)=2.0D0*(C(2)+C(5))
        K(2,3)=-1.0D0*C(3)+C(4)
        K(2,4)=C(2)-2.0D0*C(5)
        K(2,5)=-1.0D0*(C(3)+C(4))
        K(2,6)=-1.0D0*(C(2)+C(5))
        K(2,7)=C(3)-C(4)
        K(2,8)=-2.0D0*C(2)+C(5)

        K(3,1)=K(1,3)
        K(3,2)=K(2,3)
        K(3,3)=2.0D0*(C(1)+C(6))
        K(3,4)=-1.0D0*(C(3)+C(4))
        K(3,5)=C(1)-2.0D0*C(6)
        K(3,6)=C(3)-C(4)
        K(3,7)=-1.0D0*(C(1)+C(6))
        K(3,8)=C(3)+C(4)

        K(4,1)=K(1,4)
        K(4,2)=K(2,4)
        K(4,3)=K(3,4)
        K(4,4)=2.0D0*(C(2)+C(5))
        K(4,5)=-1.0D0*C(3)+C(4)
        K(4,6)=-2.0D0*C(2)+C(5)
        K(4,7)=C(3)+C(4)
        K(4,8)=-1.0D0*(C(2)+C(5))

        K(5,1)=K(1,5)
        K(5,2)=K(2,5)
        K(5,3)=K(3,5)
        K(5,4)=K(4,5)
        K(5,5)=2.0D0*(C(1)+C(6))
        K(5,6)=C(3)+C(4)
        K(5,7)=-2.0D0*C(1)+C(6)
        K(5,8)=C(3)-C(4)

        K(6,1)=K(1,6)
        K(6,2)=K(2,6)
        K(6,3)=K(3,6)
        K(6,4)=K(4,6)
        K(6,5)=K(5,6)
        K(6,6)=2.0D0*(C(2)+C(5))
        K(6,7)=-1.0D0*C(3)+C(4)
        K(6,8)=C(2)-2.0D0*C(5)

        K(7,1)=K(1,7)
```

```
      K(7,2)=K(2,7)
      K(7,3)=K(3,7)
      K(7,4)=K(4,7)
      K(7,5)=K(5,7)
      K(7,6)=K(6,7)
      K(7,7)=2.0D0*(C(1)+C(6))
      K(7,8)=-1.0D0*(C(3)+C(4))

      K(8,1)=K(1,8)
      K(8,2)=K(2,8)
      K(8,3)=K(3,8)
      K(8,4)=K(4,8)
      K(8,5)=K(5,8)
      K(8,6)=K(6,8)
      K(8,7)=K(7,8)
      K(8,8)=2.0D0*C(2)+2.0D0*C(5)

      RETURN
      END
C -----------------------------------------------------------------------
```

```
C ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
        SUBROUTINE STOREMASTERK(NT,MASTERK,NUMNODES)


C       STORE the values for [K(t)] in a unique file, once the values have been
C       calculated.

        DOUBLE PRECISION MASTERK(NUMNODES,NUMNODES)
        COMMON/NUM/NUMELS
        COMMON/RBLOCK/R
        CHARACTER*15 FILENAME

100     FORMAT (A)

        WRITE (FILENAME,45)'KNUMBER',NT
45      FORMAT(A,I2)

        OPEN (10,FILE=FILENAME, STATUS='NEW')

        DO 55,I=1,NUMNODES
          DO 50,J=1,NUMNODES
            WRITE (10,*)MASTERK(I,J)
50        CONTINUE
55      CONTINUE

        CLOSE(10)

        RETURN
        END
C _______________________________________________________________________________
```

```
C ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

        SUBROUTINE SOLUTION (NUMNODES,NNPLUS1,KZERO,IDIM,IDPLUS1,FINALK,
    C   LASTR,KINV)

        DOUBLE PRECISION KINTER(200,200)

        DOUBLE PRECISION KZERO(NUMNODES,NNPLUS1)
        DOUBLE PRECISION FINALK(IDIM,IDPLUS1)
        INTEGER NRC

        COMMON/NUM/NUMELS
        COMMON/RBLOCK/R
        COMMON/BDY/NUMBDY

        COMMON /NTS/NUMBEROFTIMESTEPS

        DOUBLE PRECISION R(200),FINALR(200)
        DOUBLE PRECISION DELU(200)
        DOUBLE PRECISION LASTR(200)
        DOUBLE PRECISION KINV(IDIM,IDIM),VECTOR(200)

        DOUBLE PRECISION Y(200)

100     FORMAT (A)

C       This subroutine retrieves the appropriate
C       values for [K(t)], [/\U] et. and steps through the appropriate sets of
C       solutions. But first the [K(0)] values must be retieved.

        OPEN (12,FILE='KNUMBER0', STATUS='OLD')

        DO I=1,NUMNODES
          DO J=1,NUMNODES
           READ (12,*) KZERO(I,J)
          ENDDO
        ENDDO


        DO NT=1,NUMBEROFTIMESTEPS

C  Retrieve the current {R} vector's value

C                      0     0
        CALL FINDR(NT,NUMNODES)

C  NOTE:  Because of a glitch in VMS-FORTRAN {R} is passed via
C          a common statement.
C  Subtract the stresses from the previous timesteps

C                              I      0      0       0     0       0
        CALL SUBRESIDUAL(FINALR,NT,KINTER)


C  Account for the known boundary conditions

C                      0        0        0      0       0      I       I
       CALL BYVAL(NUMBDY,NUMNODES,KZERO,FINALR,IDIM,LASTR,FINALK,
    C IDPLUS1,0)

C  NOTE: FINALK is the augmented matrix containing the remaining
C  simultaneous equations to be solved.

C  Attempt a solution:

C       WRITE (*,100) ' SIMULT  CALLED, DEL-{U} VALUES FOLLOW'
```

```
      CALL SIMULT(IDIM,FINALK,DELU,1.0D-25,1,IDPLUS1,Y)

C  Store the results, AFTER re-inserting known boundary conditions.

      CALL STOREDELU(DELU,NT,NUMNODES)

C  Print the results

C     CALL PRINTDELU(        )

      ENDDO
      RETURN
      END

C ____________________________________________________________________
```

```
C ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

        SUBROUTINE FINDR(NT,NUMNODES)
        DOUBLE PRECISION R(200)
        COMMON/NUM/NUMELS
        COMMON/NS/NUMSTART,NUMSTOP
        COMMON/RBLOCK/R

        CHARACTER*15 FILENAME
C  This subroutine develops the value of {R} for timestep number NT (actual
C  time value of TIMEVAL(NT) seconds) and readies it for the subtraction of the
C  residual stress values.

100     FORMAT (A)

        IF (NT.GE.NUMSTART .AND. NT.LE.NUMSTOP) THEN
          NN=1
        ELSE
          NN=0
        ENDIF

C  This applies a stepped input between NUMSTART and NUMSTOP time intervals
C  which requires the availability of RNUMBER1 and RNUMBER0 files.  A Dirac
C  spike has the same value for NUMSTART and NUMSTOP.  Other input shapes
C  may be created by modifying the values of the function  and letting
C  NN=NT during the relevant time-frame.

        WRITE (FILENAME,50)'RNUMBER',NN
50      FORMAT (A,I2)

        OPEN (20,FILE=FILENAME, STATUS='OLD')
        REWIND 20

        DO I=1,NUMNODES
          READ (20,*)R(I)
        ENDDO

C       WRITE (*,100)'  '
C       WRITE (*,50) '            {R} VECTOR FOR TIMESTEP',NT
C       WRITE (*,100) '           NODE              X                   Y'
C       WRITE (*,100)'  '

        DO ID=1,NUMNODES,2
        JD=ID+1
        NNUM=JD/2
C       WRITE (*,*) NNUM,R(ID),R(JD)
        ENDDO

        CLOSE(20)

        RETURN
        END

C ______________________________________________________________________________
```

```
C ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
C                                O    I    I      I    I     I
        SUBROUTINE SUBRESIDUAL(FINALR,NT,KINTER)

C       This subroutine subtracts the residual force, from previous timesteps
C       from the current {R} value.

        COMMON /IDIMBLOCK/IDIM
        COMMON/NNBLOCK/NUMNODES
        COMMON/NUM/NUMELS
        COMMON /NTS/NUMBEROFTIMESTEPS
        COMMON/IBINDX/IBDY
        COMMON/BDY/NUMBDY
        COMMON/RBLOCK/R
        COMMON/KPASTBLOCK/KPAST

        INTEGER IBDY(200),NUMNODES

        DOUBLE PRECISION DU(200),DUMMY(200)
        DOUBLE PRECISION R(200),RESID(200),FINALR(200)
        DOUBLE PRECISION RINTER(200),DUINTER(200)
        DOUBLE PRECISION KPAST(200,200)
        DOUBLE PRECISION KINTER(IDIM,IDIM)
        DOUBLE PRECISION SUM


        CHARACTER*15 FILEDELU
        CHARACTER*15 FILEKNUM


 101    FORMAT (A,I2)
 100    FORMAT (A)


C       WRITE (*,101) ' TIMESTEP NUMBER',NT

        IEND=NT-1
        DO 80,I=0,IEND


C       WRITE (*,101)' *************>  I=',I
C       WRITE (*,100)'           -[K]      *     {DU}'
        WRITE (FILEDELU,70) 'DELUFILE',I
        J=NT-I
        WRITE (FILEKNUM,75) 'KNUMBER',J
C       WRITE (*,*) J,I

70      FORMAT(A,I2)
75      FORMAT(A,I2)

        OPEN (12,FILE=FILEDELU, STATUS='OLD')
        OPEN (15,FILE=FILEKNUM, STATUS='OLD')

        REWIND 12
        REWIND 15

        DO L=1,NUMNODES
          DO M=1,NUMNODES
            READ (15,*)KPAST(L,M)
          ENDDO
        ENDDO

        READ (12,*)(DU(KK),KK=1,NUMNODES)

        CLOSE (12)
        CLOSE (15)
```

```
C  DIAGNOSTIC OF FILE-READ KPAST AND DU

C       WRITE (*,100) '  OUTPUT OF READ VALUES OF [KPAST] '
C       WRITE (*,100) '  ID        JD                KPAST (ID,JD)'

C       DO ID=1,NUMNODES
C         DO JD=1,NUMNODES
C           WRITE (*,*) ID,JD,KPAST(ID,JD)
C         ENDDO
C       ENDDO


C       DO ID=1,NUMBDY
C       WRITE (*,101) '  BOUNDARY AT ROW:',IBDY(ID)
C       ENDDO


C       WRITE (*,100) '             ID        DU(ID)                    AS READ'
C       DO ID=1,NUMNODES
C         WRITE (*,*) ID,DU(ID)
C       ENDDO


C   Collapse the [KPAST] and {DU} matrices according to the boundary
C   conditions associated with the {DU}'s timestep index.

C   Do {DU} first, producing {DUINTER}:

        IFLAG=0
        DO II=1,NUMNODES
          IF (II .EQ. IBDY(IFLAG+1)) THEN
C         WRITE (*,101) '  SKIPPING DU ROW',II
            IFLAG=IFLAG+1
          ELSE
            DUINTER(II-IFLAG)=DU(II)
C           WRITE (*,101) ' COPYING DU ROW',II
C           WRITE (*,101) '  TO DUINTER ROW',II-IFLAG
C           WRITE (*,*) DU(II)
          ENDIF
        ENDDO


C       WRITE (*,100) '           ID        DUINTER(ID)'
C
C       DO ID=1,IDIM
C         WRITE (*,*) ID,DUINTER(ID)
C       ENDDO


C  Then [KPAST] producing [KINTER].
C  This is a nested sort similar to the BYVAL() subroutine.
C  Rows first:


        IFLAG=0
        DO II=1,NUMNODES
          IF (II .EQ. IBDY(IFLAG+1)) THEN
            IFLAG=IFLAG+1
          ELSE

C  Then columns:
C       WRITE (*,100) '  MYSTERY GLITCH, NUMNODES IS'
C       WRITE (*,*) NUMNODES
C       WRITE (*,100) JJ, JFLAG, IBDY(JFLAG+1)
          JFLAG=0
          DO JJ=1,NUMNODES
```

```
C           WRITE (*,*) JJ,JFLAG,IBDY(JFLAG+1)

            IF (JJ .EQ. IBDY(JFLAG+1)) THEN
              JFLAG=JFLAG+1
            ELSE
              KINTER(II-IFLAG,JJ-JFLAG)=KPAST(II,JJ)
            ENDIF
          ENDDO

        ENDIF

        ENDDO

C  DIAGNOSTIC OF COLLAPSED KPAST AND DU MATRICES (KINTER AND DUINTER)

C       WRITE (*,100) '  OUTPUT OF VALUES OF [KINTER] AND (DUINTER)'
C       WRITE (*,100) '           ID            JD        KINTER (ID,JD)'

C       DO ID=1,IDIM
C         DO JD=1,IDIM
C           WRITE (*,*) ID,JD,KINTER(ID,JD)
C         ENDDO
C       ENDDO

C       WRITE (*,100) '           ID          DUINTER(ID)'

C       DO ID=1,IDIM
C           WRITE (*,*) ID,DUINTER(ID)
C       ENDDO


C  Multiply [KINTER] by (DUINTER) to produce a "compressed" (RINTER)

        CALL SQUARBYCOL(IDIM,KINTER,DUINTER,RINTER)

C  DIAGNOSTIC OF RINTER
C       WRITE (*,100) '  AFTER SQUARBYCOL:'
C       WRITE (*,100) '            ID     RINTER(ID)'
C       DO ID=1,IDIM
C         WRITE (*,*) ID,RINTER(ID)
C       ENDDO


C  "Unpack" the (RINTER) values into the appropriate rows of (RESID)

        IFLAG=0

        DO II=1,NUMNODES

          IF (II .EQ. IBDY(IFLAG+1)) THEN

C  If this is a "deleted" row, regenerate the (RESID(II)) value from the
C  appropriate row and column of [KPAST] and (DU)

C       WRITE (*,100) '  REGENERATION CHECK OF KPAST VALUES'
C       WRITE (*,100) '  ID        JD                KPAST (ID,JD)'

C       DO ID=1,NUMNODES
C         DO JD=1,NUMNODES
C           WRITE (*,*) ID,JD,KPAST(ID,JD)
C         ENDDO
C       ENDDO

            SUM=0.0D0
C             WRITE (*,100) ' IBDY (IFLAG+1)  KK   SUM   KPAST    DU'
              DO KK=1,NUMNODES
                SUM=SUM+KPAST(IBDY(IFLAG+1),KK)*DU(KK)
```

```
C             WRITE (*,*) IBDY(IFLAG+1),KK,SUM,KPAST(IBDY(IFLAG+1),KK),DU(KK)
              ENDDO

            RESID(II)=SUM

C       WRITE (*,101) ' REGENERATED R VALUE FOR R:',II
C       WRITE (*,*) RESID(II)

            IFLAG=IFLAG+1

          ELSE

C  If not, copy the value from (RINTER) and subtract the necessary
C  coefficients (which are carried through from the solution of the
C  system of compressed equations).

          RESID(II)=RINTER(II-IFLAG)

          DO LL=1,NUMBDY
            RESID(II)=RESID(II)+KPAST(II,IBDY(LL))*DU(IBDY(LL))
          ENDDO

C  Finally, subtract the residual terms generated for this series of
C  arrays from the original force vector (R)

        ENDIF
        ENDDO

C       WRITE (*,100) '  (R) VECTOR JUST BEFORE SUBTRACTION'

C       DO ID=1,NUMNODES
C         WRITE (*,*) ID,R(ID)
C       ENDDO

C       WRITE (*,100) '              N     R(N)        RESID(N)'
C
CC
        DO N=1,NUMNODES
          R(N)=R(N)-RESID(N)
C         WRITE (*,*) N, R(N), RESID(N)
        ENDDO

 80     CONTINUE


C Copy whatever is left into (FINALR) for return from the subroutine.

        DO N=1,NUMNODES
          FINALR(N)=R(N)
        ENDDO


        RETURN
        END
C ____________________________________________________________________
```

```
C ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

        SUBROUTINE STOREDELU(DELU,NT,NUMNODES)
        DOUBLE PRECISION DELU(200)
        CHARACTER*15 FILEDELU
C  This subroutine stores the calculated values of (/\U) in individual files

        COMMON/NUM/NUMELS

        INTEGER IBDY(200)
        COMMON/IBINDX/IBDY

        DOUBLE PRECISION BVAL(200)
        COMMON/BOUNDVAL/BVAL

        DOUBLE PRECISION DUMMY(200)
        INTEGER NODEHALF,NNUM
C  Reinsert known boundary values in the (DELU) array.
C  NOTE the (DUMMY) vector is the "unpacked" solution
C  vector which also contains the known boundary conditions
C  at the particular time-step.

        IFLAG=0

        DO II=1,NUMNODES
          IF (II.EQ.IBDY(IFLAG+1)) THEN
            DUMMY(II)=BVAL(IFLAG+1)
            IFLAG=IFLAG+1
          ELSE
            DUMMY(II)=DELU(II-IFLAG)
          ENDIF
        ENDDO

C  Truncate very small values to avoid underflow errors:

        DO KK=1,NUMNODES
            IF (DABS(DUMMY(KK)) .LE. 1.0D-24) THEN
              DUMMY(KK)=0.0D0
            ENDIF
        ENDDO


        IF (NT.NE.0) THEN
C         WRITE (*,100) ' STOREDELU RUN '
100       FORMAT (A)

          WRITE (FILEDELU,85)'PARADELUFILE',NT
85        FORMAT(A,I2)
          OPEN(22,FILE=FILEDELU,STATUS='NEW')

C Write to appropriate file.

          DO I=1,NUMNODES
            WRITE(22,*)DUMMY(I)
          ENDDO

          CLOSE(22)

135     FORMAT (I2,D14.6,D14.6)
        IF ((NFE .EQ. 1) .OR. (NPRNT .GE. 2)) THEN

C  Screen output for instant gratification.
        WRITE (*,100)'  '
        WRITE (*,85) '  DISPLACEMENT VALUES FOR TIMESTEP',NT
        WRITE (*,100) '  VALUES LESS THAN 1.0D-25 ARE STORED AS 0.0D0'
        WRITE (*,100) '         NODE          DX                  DY'
```

```
      DO ID=1,NUMNODES,2
        JD=ID+1
        NNUM=(JD/2)
        WRITE (*,*) NNUM,DUMMY(ID),DUMMY(JD)
      ENDDO
      ENDIF

      ENDIF
C       WRITE(*,100)' **'
      RETURN
      END

C ____________________________________________________________________
```

```
C ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

        SUBROUTINE BYVAL(NUMBDY,NUMNODES,MASTERK,
     C  FINALR,IDIM,LASTR,FINALK,IDPLUS1,LSIG)

        DOUBLE PRECISION MASTERK(NUMNODES,NUMNODES)
        DOUBLE PRECISION FINALK(IDIM,IDPLUS1)


        INTEGER IBDY(200)
        COMMON/IBINDX/IBDY
        DOUBLE PRECISION BVAL(200)
        COMMON/BOUNDVAL/BVAL

        DOUBLE PRECISION FINALR(200)
        DOUBLE PRECISION LASTR(200)
        DOUBLE PRECISION TEMP

C       This subroutine takes the appropriate boundary conditions from the
C       IBDY (indexed list of ROWS which have known boundary conditions)
C       and BVAL (the values associated with the known boundary conditions)
C       and substitute the proper values into the MASTERK array at each
C       time step.  Note that the boundary conditions are not time-
C       dependent in this version.

100     FORMAT (A)

C  ZERO OUT THE APPROPRIATE ROWS IN [MASTERK] AND {FINALR}

        DO I=1,NUMBDY
        DO J=1,NUMNODES
          MASTERK(IBDY(I),J)=0.0D0
        ENDDO
          FINALR(IBDY(I))=0.0D0
        ENDDO


C SUBTRACT THE APPROPRIATE VALUES FROM THE {FINALR} VECTOR AND ZERO OUT THE
C APPROPRIATE COLUMNS IN [MASTERK]


        DO I=1,NUMBDY
        DO J=1,NUMNODES
          FINALR(J)=FINALR(J)-MASTERK(J,IBDY(I))*BVAL(I)
          MASTERK(J,IBDY(I))=0.0D0
        ENDDO
        ENDDO

C  NOW, RESTRUCTURE THE ARRAY SO THAT THE "ZERO" ROWS AND COLUMNS ARE
C  ELIMINATED, AND THE FINAL ARRAYS OF [K] AND {R} ARE PROPERLY DI-
C  MENSIONED.

C  FIRST, {R}:

        IFLAG=0
C       WRITE (*,100) '           II,      IFLAG'
        DO II=1,NUMNODES
          IF (II.EQ.IBDY(IFLAG+1)) THEN
            IFLAG=IFLAG+1
          ELSE
            LASTR(II-IFLAG)=FINALR(II)
          ENDIF
        ENDDO

c       WRITE (*,100) '  I,  FINALR(I),  LASTR(I)'
c       DO I=1,IDIM
c         WRITE(*,*) I,FINALR(I),LASTR(I)
```

```
c       ENDDO

C ***************************************************

C  THEN [MASTERK]:
C    FIRST ROWS...


        IFLAG=0
        DO II=1,NUMNODES
          IF (II.EQ.IBDY(IFLAG+1)) THEN
            IFLAG=IFLAG+1
          ELSE

C ***NESTED COLUMN SORT**********************

        JFLAG=0
        DO JJ=1,NUMNODES
          IF (JJ.EQ.IBDY(JFLAG+1)) THEN
            JFLAG=JFLAG+1
          ELSE
            FINALK(II-IFLAG,JJ-JFLAG)=MASTERK(II,JJ)
          ENDIF
        ENDDO

C*************************************

          ENDIF
        ENDDO


C   ...AND FINALLY AUGMENT [FINALK] WITH (LASTR) FOR RETURN AND SOLUTION.

        DO JJ=1,IDIM
          FINALK(JJ,IDPLUS1)=LASTR(JJ)
        ENDDO

        RETURN
        END


C ____________________________________________________
```

```
C ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

      SUBROUTINE SIMULT(N,A,X,EPS,INDIC,NRC,Y)

      DOUBLE PRECISION Y(200),A(N,NRC),X(200)
      DOUBLE PRECISION EPS,PIVOT,DETER,AIJCK,SIMUL

      INTEGER INDIC,NRC,N
      INTEGER IROW(200),JCOL(200),JORD(200)

C  FROM "APPLIED NUMERICAL METHODS", B.CARNAHAN, H.A.LUTHER,
C        J.O. WILKES.  J.WILEY & SONS,NEW YORK, 1969
C        CHAPTER 5, p.276
C  ****************** DIRECTORY OF VARIABLES ******************
C       N        NUMBER OF ROWS IN A
C       A        AUGMENTED MATRIX OF COEFFICIENTS
C       X         VECTOR OF SOLUTIONS
C       EPS       MINIMUM ALLOWABLE MAGNITUDE FOR A PIVOT ELEMENT
C       INDIC     COMPUTATIONAL SWITCH FOR SOLUTION TYPE
C                 (+1 FOR NO INVERSE RETURNED)
C       NRC       COLUMN DIMENSIONS FOR THE MATRIX [A] (N+1)
C  ***********************************************************

 100  FORMAT (A)

      M=1
      NPLUSM=NRC

C  BEGIN ELIMINATION PROCEDURE

      DETER=1.0D0
      DO 9 K=1,N

C  CHECK FOR A TOO-SMALL PIVOT ELEMENT

      IF (DABS(A(K,K)).GT.EPS) GO TO 5
      WRITE (*,100) ' PIVOT TOO SMALL...I QUIT!'

C  NORMALIZE THE PIVOT ROW

 5    KP1=K+1
      DO 6 J=KP1,NPLUSM
 6    A(K,J)=A(K,J)/A(K,K)
      A(K,K)=1.0D0

C  ELIMINATE THE K(TH) COLUMN ELEMENTS EXCEPT FOR THE PIVOT

      DO 9 I=1,N
      IF (I.EQ.K .OR. A(I,K).EQ.0.0D0) GO TO 9
      DO 8 J=KP1,NPLUSM
 8    A(I,J)=A(I,J)-A(I,K)*A(K,J)
      A(I,K)=0.0D0
 9    CONTINUE

C  WRITE THE SOLUTION VECTOR INTO (X) FOR RETURN

      DO II=1,N
      X(II)=A(II,NRC)
      ENDDO

C  THAT'S ALL FOLKS...

      RETURN
      END
```

```
      SUBROUTINE SQUARBYCOL(ISPEC,SQUARE,COL,RESULT)

C  Multiplies [SQUARE] by {COL} and produces {RESULT}.
C  Note that there is no safety check on dimensions here.

      DOUBLE PRECISION SQUARE(ISPEC,ISPEC)
      DOUBLE PRECISION COL(ISPEC),RESULT(ISPEC)

100   FORMAT (A)

C     VODOO FORTRAN!!!!!!!!!!!!!

      C=ISPEC

C     DO ID=1,ISPEC
C       A=COL(ID)
C       DO JD=1,ISPEC
C        B=SQUARE(ID,JD)
C       ENDDO
C     ENDDO

      A=COL(1)
      B=SQUARE(1,1)


C  Clean House

      DO I=1,ISPEC
        RESULT(I)=0.0D0
      ENDDO

C  Do the nasty

      DO IROW=1,ISPEC
        DO ICOL=1,ISPEC
C        IF (DABS(SQUARE(IROW,ICOL)*COL(ICOL)) .LE. 1.0D-26) GO TO 20
           RESULT(IROW)=RESULT(IROW)+SQUARE(IROW,ICOL)*COL(ICOL)
 20       CONTINUE
        ENDDO
      ENDDO

      RETURN
      END
```

Table 4. VISCO2.FOR

PROGRAM VISCO2.FOR

Note that the format has been compressed to fit the margin requirements for publication.

```
C      ==========================================================
C    //\                        VISCO 2                  C   // \                        V1.7
                  c   //    \  COPYRIGHT          Scott Morris
C    //             \                                Michigan  State  University,  1989-1992    C
==========================================================C      THIS SOFTWARE IS NOT FOR USE IN
APPLICATIONS OTHER THAN   C       RESEARCH AND EDUCATION/DEMONSTRATION.   IT MAY BE FREELY C
DISTRIBUTED ON THE CONDITION THAT THE ENTIRE PROGRAM IS C    DISTRIBUTED WITHOUT CHANGE AND WITHOUT
CHARGE.
C ==========================================================================
C ==========================================================================
C
C  Variable Directory (UNSORTED):
C  TIMVAL(n)     The value in seconds of timestep n.
C  NUMELS        The number of elements.
C  NumberOfNodes  The number of actual nodes in model.
C  NUMNODES      The number of nodes * 2 (used as counter for loops in 2D
C                model throughout the program)
C  COEFF         The coefficients of the equation describing the time-
C                decay of the physical constants.
C  NODX(n)
C  NODY(n)       The x,y spatial coordinates of node (n).
C  EI(n)
C  EJ(n)
C  EK(n)
C  EM(n)         The nodes i,j,k,m asociated with element (n).
C  MASTERK       The [K] produced for a given timestep.
C  A             The A matrix used in the construction of [K].
C  IFLAG         Signals the type of (r) values to use.
C  NUMSTART      The timestep number at which a STEP input starts
C  NUMSTOP       The timestep number at which a STEP input stops
C  TH            Material thickness for 2-D problem
C  R(t)          Force vector at timestep t
C  NUMBEROFTIMESTEPS
C  FINALK
C  LASTR
C  KZERO
C  KPAST         Residual force vector


C ___________________________________________________________________________
C ************************ EFFLUVIA ************************************
C  print/queue=eb270_ps <filename> if the C.Itoh printer is not working
C    "    "   =eb162_imagen <filename> for HD printout
C  show queue eb......    gives printer stack-up.
C  FORTRAN/LIST <FILENAME>
C  LINK <FILENAME>
C  LINK <FILENAME>.obj,IMSL/LIB
C  To dump output to a file:
C    DEFINE SYS$OUTPUT <FILENAME>
C    DEASSIGN SYS$OUTPUT
C  To MERGE two files:
C  COPY <file1>,<file2> <destinationfile>/LOG

C ==========================================================================
```

Table 4 (Cont'd).

```
C  Note that the I/O convention applies to each subroutine as an independant
C  entity.  From this, in a subroutine, values come in (I) and others are
C  returned (O).
C  Subroutines which call another subroutine:  When the call is made, values
C  are sent out (O) and others return to the caller (I).  This is similar
C  to double-entry bookkeeping in that there should be a match between I and O
C  designations throughout the program.
C ==============================================================================
C ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
C                    DECLARE COMMON STATEMENTS

      INCLUDE ' KSTORE.FOR'

      DOUBLE PRECISION K(8,8)

      COMMON/NUM/NUMELS
      COMMON/NNBLOCK/NUMNODES

      DOUBLE PRECISION R(200)
      COMMON/RBLOCK/R

      DOUBLE PRECISION COEFF(6)
      COMMON/COEFFBLOCK/COEFF

      DOUBLE PRECISION TH
      COMMON/THICKBLOCK/TH

      COMMON/IDIMBLOCK/IDIM

      COMMON /NTS/NUMBEROFTIMESTEPS

      DOUBLE PRECISION TIMSTART, TIMINCR

      DOUBLE PRECISION NODX(200),NODY(200)
      COMMON/NODE/NODX,NODY

      INTEGER EI(325),EJ(325),EK(325),EM(325)
      COMMON/ELNODES/EI,EJ,EK,EM

      DOUBLE PRECISION MASTERK(200,200)
      COMMON/BIGK/MASTERK

      DOUBLE PRECISION A(3,3)
      COMMON/ABLOCK/A

      INTEGER NUMSPYK
      COMMON/NS/NUMSTART,NUMSTOP

      INTEGER IBDY(200)
      COMMON/IBINDX/IBDY

      DOUBLE PRECISION BVAL(200)
      COMMON/BOUNDVAL/BVAL

      INTEGER NUMBDY
      COMMON/BDY/NUMBDY

      INTEGER IDPLUS1
      COMMON/IDP/IDPLUS1
```

```
      DOUBLE PRECISION KPAST(200,200)
      COMMON/KPASTBLOCK/KPAST

      INTEGER IROW(200),JCOL(200),JORD(200)
      COMMON/IJJ/IROW,JCOL,JORD

      DOUBLE PRECISION KZERO(200,200)
      COMMON/KZ/KZERO

      DOUBLE PRECISION Y(200)
      COMMON/WYE/Y

      DOUBLE PRECISION FINALK(200,200),LASTR(200)
      COMMON/ENDO/FINALK,LASTR

      INTEGER IBANDWIDTH
      COMMON/IBW/IBANDWIDTH

      INTEGER ISTORFLAG
      COMMON/ISF/ISTORFLAG


      SAVE


  100 FORMAT(A)

      WRITE (*,100)'                 *********   VISCO 2    ***********  '
      WRITE (*,100)'                              V1.7 '
      WRITE (*,100)'                            SCOTT MORRIS '
      WRITE (*,100)'                  Michigan State University 1991    '
      WRITE (*,100)'                 ==================================='
C ____________________________________________________________________________
C ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
C                            OPEN FILES
C
C  This block of commands opens, labels, and numbers the appropriate files for
C  use by VISCO1 with the exception of the series of files needed for [K(t)]
C  storage, as those are created as needed.

      OPEN (3, FILE='GENERAL_DATA', STATUS= 'OLD')

      REWIND 3

      OPEN (9, FILE='BOUNDARIES', STATUS= 'OLD')

      REWIND 9

c ____________________________________________________________________________
C ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
C                        READ IN INITAL DATA
C This reads in some of the necessary parameters to operate some of
c the arrays used in this program.

      READ (3,*)NUMELS,NumberOfNodes
        NUMNODES=NumberOfNodes*2
        NUMN=NUMNODES
      READ (3,*)(COEFF(I),I=1,6)
      READ (3,*)NUMBEROFTIMESTEPS
      READ (3,*)NUMSTART,NUMSTOP
      READ (3,*)TIMSTART,TIMINCR
      READ (3,*)ISTORFLAG
      CALL MAKEARRAYS


C ***************************************
C   CONSTANT THICKNESS TERM:
```

```
      TH=1.0D0
C ***************************************

      WRITE (*,100) '   COEFF 1 --> 6'
      DO J=1,6
      WRITE (*,*)J,COEFF(J)
      ENDDO

      CLOSE (3)

C  Read in the known boundary conditions from the 'boundaries.dat file
      WRITE (*,100)'  '
      WRITE (*,100) '  NUMBER OF KNOWN DISPLACEMENT VALUES:'
      READ (9,*) NUMBDY
      WRITE (*,*) NUMBDY
      WRITE (*,100)'  '
      WRITE (*,100) ' DIRECTION INDICATOR:  X=1   Y=0'
      WRITE (*,100) '        NODE       DIRECTION            VALUE'

      DO I=1,NUMBDY
      READ (9,*) IBNDX, IDIR, BVAL(I)
      WRITE (*,*) IBNDX,IDIR,BVAL(I)
      WRITE (*,100)'  '

C  IDIR:  X=1   Y=0  FOR 2-D PROBLEMS

      IBDY(I)=(IBNDX*2)-IDIR
      ENDDO
      CLOSE (9)

C *  Once arrays are ready, construct the series of [K(t)] values.
C    for all of the timesteps in the problem.

      WRITE (*,100)'  STORING [K] MATRICES FOR '

      DO 5,NT=0,NUMBEROFTIMESTEPS

        CALL MAKEA(NT,TIMINCR,TIMSTART)

C                            I       O
        CALL MAKESHAPES( MASTERK,NUMNODES)

C                            O     O       O
        CALL STOREMASTERK(NT,MASTERK,NUMNODES)


5     CONTINUE

C *  Solve the time-dependant problem, once all of the [K] values are
C    ready and stored.
      NNPLUS1=NUMNODES+1
      IDIM=NUMNODES-NUMBDY
      IDPLUS1=IDIM+1
      CALL SOLUTION(NUMNODES,NNPLUS1,KZERO,IDIM,IDPLUS1,FINALK,
     C LASTR)
      WRITE (*,100) ' SOLUTION COMPLETED!!!!!'

      STOP
      END
C ____________________________________________________________________
```

```
C ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
        SUBROUTINE MAKEARRAYS

C This subroutine sets up the indexed array of node and element values used
C by the [K] generation loops.

        INTEGER NODNO,IELNO

        COMMON/NUM/NUMELS

        DOUBLE PRECISION NODX(200),NODY(200)
        COMMON/NODE/NODX,NODY

        INTEGER EI(325),EJ(325),EK(325),EM(325)
        COMMON/ELNODES/EI,EJ,EK,EM

        INTEGER IBANDWIDTH
        COMMON/IBW/IBANDWIDTH

        INTEGER ISUB(4)

        OPEN (4, FILE='ELEMENT_DATA', STATUS= 'OLD')
        REWIND 4

C  * Create an array of indexed x & y values associated with each node

100     FORMAT (A)
110     FORMAT (A,I2)


        WRITE(*,100) '        NODE           X                        Y'
20      CONTINUE
          READ (4,*) NODNO
           IF (NODNO .EQ. -1) GO TO 23
          READ (4,*) NODX(NODNO), NODY(NODNO)
          WRITE (*,*) NODNO,NODX(NODNO),NODY(NODNO)
         GO TO 20



23      WRITE (*,100) '  '
        WRITE (*,100) '       ELEMENT          I           J           K
     C       M'

24      CONTINUE
* Produce an index of nodal values associated with each element

          READ (4,*) IELNO
           IF (IELNO .EQ. -1) GO TO 25
          READ (4,*) EI(IELNO), EJ(IELNO), EK(IELNO), EM(IELNO)

c  BANDWIDTH CALCULATION

        ISUB(1)=EI(IELNO)
        ISUB(2)=EJ(IELNO)
        ISUB(3)=EK(IELNO)
        ISUB(4)=EM(IELNO)

C  IS IT A TRIANGULAR ELEMENT?

        IF (EM(IELNO).EQ.0) THEN
          KOUNT=3
        ELSE
          KOUNT=4
        ENDIF

C  FIND THE LARGEST AND SMALLEST NODE NUMBER IN ELEMENT IELNO
```

```
      IMAX=0
      IMIN=0

      DO JJ=1,KOUNT
        IF (ISUB(JJ).GT.IMAX) IMAX=ISUB(JJ)
      ENDDO

      IMIN=IMAX

      DO JJ=1,KOUNT
        IF (ISUB(JJ).LT.IMIN) IMIN=ISUB(JJ)
      ENDDO

C  CALCULATE ELEMENTAL BANDWIDTH

      INTBW=((IMAX-IMIN)+1)*2

C  IS IT THE LARGEST IN THE GRID??

      IF (INTBW.GT.IBANDWIDTH)  IBANDWIDTH=INTBW

C  END OF BANDWIDTH CALCULATION

      WRITE (*,*) IELNO,EI(IELNO),EJ(IELNO),EK(IELNO),EM(IELNO)

      GO TO 24
25    CONTINUE

      WRITE (*,110) '  BANDWIDTH=',IBANDWIDTH

      RETURN
      END

C ____________________________________________________________
```

```
C ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
C
      SUBROUTINE MAKEA(NT,TIMINCR,TIMSTART)

      DOUBLE PRECISION TIMEVAL,TIMINCR,TIMSTART
      DOUBLE PRECISION A(3,3)
      DOUBLE PRECISION COEFF(6)
      DOUBLE PRECISION G1,G2,B,C,D

      COMMON/COEFFBLOCK/COEFF
      COMMON/ABLOCK/A


100   FORMAT (A)
101   FORMAT (A,I2)
102   FORMAT (A,I2,D12.4,A)
C  Calculate the value of the timestep in seconds:

      TIMEVAL=(NT*TIMINCR)+TIMSTART

      WRITE (*,102) ' TIMESTEP',NT,TIMEVAL,' Sec'

C ***************************************************************************

C  This Subroutine computes the values for E and MU at some time-step NT.  It
C  utilizes a 3-parameter exponential decay model for the elastic modulus and
C  Poisson's ratio.  The COEFF array contains the necessary coefficients.

      ELM=COEFF(1)+COEFF(2)*DEXP(-1.0D0*(COEFF(3)*TIMEVAL))
      write (*,*) ELM
      PR=COEFF(4)+COEFF(5)*DEXP(-1.0D0*(COEFF(6)*TIMEVAL))

C  Note that most models hold MU constant, so that COEFF5 & COEFF6 are
C  usually 0.

C ***************************************************************************

C  This part takes the E and MU values for the current time value
C  and returns the [A] matrix for that time value.


40    B=ELM/(1.0D0-(PR**2.0D0))

     A(1,1)=B
      A(2,2)=B
      A(3,3)=B*(1.0D0-PR)/2.0D0
      A(1,2)=PR*B
      A(2,1)=A(2,1)
      A(1,3)=0.D0
     A(2,3)=0.D0
      A(3,1)=0.D0
      A(3,2)=0.D0

C     WRITE (*,100) '    [A]  MATRIX VALUES'

C     DO IROW=1,3
C       DO JCOL=1,3
C        WRITE (*,*) IROW,JCOL,A(IROW,JCOL)
C       ENDDO
C     ENDDO

50    CONTINUE

      RETURN
      END

C ____________________________________________________________________
```

```
C ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^


        SUBROUTINE MAKESHAPES(MASTERK,NUMNODES)

        DOUBLE PRECISION MASTERK(NUMNODES,NUMNODES)
        DOUBLE PRECISION K(8,8)

        COMMON/NUM/NUMELS

        INTEGER EI(325),EJ(325),EK(325),EM(325)
        COMMON/ELNODES/EI,EJ,EK,EM

100     FORMAT (A)
        IEIGHT=8
C -This Subroutine through each of the elements and (using the proper
c  subroutine) develops an elemental [k] matrix to be added into the [K] via
c  the MERGEK subroutine.


C * Put the two together and route to appropriate calculation of [k] for
C   each element then add the value for that element into the global [K]

C   But first, the MASTERK must be cleared from the last time-step.

        DO II=1,NUMNODES
          DO JJ=1,NUMNODES
           MASTERK(II,JJ)=0.0D0
          ENDDO
        ENDDO

C  ***************************************************************************

C   Proceed to assemble next MASTERK

        DO 30,IElement=1,NUMELS
          IF (EM(IElement).EQ.0) THEN
C                                  O     I
                CALL TRIANGLELEM(IElement,K)
               GO TO 27
          ENDIF
C                             O    I
          CALL SQUARELEM(IELEMENT,K)

 27     CONTINUE

C  ************  MERGES [Ke] into [K(t)]  ************************

C                     O    O
          CALL MERGEK(K,IElement,MASTERK,NUMNODES)

C  Note that MERGEK merges the element's [K] value into the COMMON MASTERK()
C  and does not return any value.

30      CONTINUE


        RETURN
        END

C ____________________________________________________________________________
```

```
C ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
C                            I        O
      SUBROUTINE TRIANGLELEM(IElement,K)

      DOUBLE PRECISION K(8,8)

      DOUBLE PRECISION NODX(200),NODY(200)
      COMMON/NODE/NODX,NODY

      INTEGER EI(325),EJ(325),EK(325),EM(325)
      COMMON/ELNODES/EI,EJ,EK,EM

      DOUBLE PRECISION TH
      COMMON/THICKBLOCK/TH

      DOUBLE PRECISION A(3,3)
      COMMON/ABLOCK/A

      DOUBLE PRECISION XI,XJ,XK,XM,YI,YJ,YK,YM,BI,BJ,BK,CI,CJ,CK

      DOUBLE PRECISION X(3),Y(3),B(3,6),C(6,3),AR2,SUM

C  This subroutine uses the brute-force calculations in TRIANGLECALC
C  to produce a [k] for the selected element.

100   FORMAT (A)

c     WRITE (*,100) ' DUMP IN TRIANGLELEM'
c     WRITE (*,*) IELEMENT, EI(IELEMENT), EJ(IELEMENT), EK(IELEMENT)


      XI=NODX(EI(IElement))
      XJ=NODX(EJ(IElement))
      XK=NODX(EK(IElement))
      YI=NODY(EI(IElement))
      YJ=NODY(EJ(IElement))
      YK=NODY(EK(IElement))

      X(1)=XI
      X(2)=XJ
      X(3)=XK
      Y(1)=YI
      Y(2)=YJ
      Y(3)=YK

C ******************************************************************
C *                                                                *
C *  What follows is from "Applied Finite Element Analysis"        *
C *  Larry J. Segerlind, J Wiley & Sons, 1984                      *
C *  P. 347                                                        *
C ******************************************************************

C  CLEAN HOUSE:

      DO I=1,8
      DO J=1,8
        K(I,J)=0.0D0
      ENDDO
      ENDDO


      DO I=1,3
        DO J=1,6
          B(I,J)=0.0D0
          C(J,I)=0.0D0
        ENDDO
      ENDDO
```

```
C  GENERATE THE {B} MATRIX

        B(1,1)=Y(2)-Y(3)
        B(1,3)=Y(3)-Y(1)
        B(1,5)=Y(1)-Y(2)
        B(2,2)=X(3)-X(2)
        B(2,4)=X(1)-X(3)
        B(2,6)=X(2)-X(1)
        B(3,1)=B(2,2)
        B(3,2)=B(1,1)
        B(3,3)=B(2,4)
        B(3,4)=B(1,3)
        B(3,5)=B(2,6)
        B(3,6)=B(1,5)

        AR2=X(2)*Y(3)+X(3)*Y(1)+X(1)*Y(2)-X(2)*Y(1)-X(3)*Y(2)-X(1)*Y(3)

C  MATRIX MULTIPLCATION TO OBTAIN C = {BT}[A]

        DO I=1,6
        DO J=1,3
            C(I,J)=0.0D0
            DO L=1,3
              C(I,J)=C(I,J)+B(L,I)*A(L,J)
            ENDDO
        ENDDO
        ENDDO
C MATRIX MULTIPLICATION TO OBTAIN [K] WHERE
C   [K]= {BT}[A]{B} = {C}{B}

        DO 27 I=1,6
        DO 27 J=1,6
          SUM=0.0D0
          DO 28 L=1,3
 28         SUM=SUM+C(I,L)*B(L,J)
          K(I,J)=SUM*TH/(2.0D0*AR2)
 27     CONTINUE

C  RETURN [K]

        RETURN
        END

C ____________________________________________________________
```

```
C ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
C                                  I        O
      SUBROUTINE SQUARELEM(IElement,k)
      DOUBLE PRECISION k(8,8),C(6)

      DOUBLE PRECISION NODX(200),NODY(200)
      COMMON/NODE/NODX,NODY

      DOUBLE PRECISION A(3,3)
      COMMON/ABLOCK/A

      DOUBLE PRECISION AA,B

      DOUBLE PRECISION TH
      COMMON/THICKBLOCK/TH

      INTEGER EI(325),EJ(325),EK(325),EM(325)
      COMMON/ELNODES/EI,EJ,EK,EM

      DOUBLE PRECISION XI,XJ,XK,XM,YI,YJ,YK,YM

C  This subroutine uses the brute-force calculations in SQUARECALC
C  to produce a [k] for the selected element.

100   FORMAT (A)



      XI=NODX(EI(IElement))
      XJ=NODX(EJ(IElement))
      XM=NODX(EM(IElement))
      YI=NODY(EI(IElement))
      YJ=NODY(EJ(IElement))
      YM=NODY(EM(IElement))

      AA=0.5D0*DSQRT(((XM-XI)**2.0D0)+((YM-YI)**2.0D0))
      B=0.5D0*DSQRT(((XJ-XI)**2.0D0)+((YJ-YI)**2.0D0))

      C(1)=TH*((A(1,1)*AA)/(6.0D0*B))
      C(2)=TH*((A(1,1)*B)/(6.0D0*AA))
      C(3)=TH*(A(1,2)/4.0D0)
      C(4)=TH*(A(3,3)/4.0D0)
      C(5)=TH*((A(3,3)*AA)/(6.0D0*B))
      C(6)=TH*((A(3,3)*B)/(6.0D0*AA))

C                         O I
      CALL RECTANGLECALC (C,k)

      RETURN
      END

C ____________________________________________________________________________
```

```
C ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

        SUBROUTINE MERGEK(K,IElement,MASTERK,NUMNODES)

        DOUBLE PRECISION MASTERK(NUMNODES,NUMNODES)

        COMMON/NUM/NUMELS
        DOUBLE PRECISION K(8,8)
        INTEGER SK(8)

        INTEGER EI(325),EJ(325),EK(325),EM(325)
        COMMON/ELNODES/EI,EJ,EK,EM

100     FORMAT(A)

C -This Subroutine adds the [K] for some element into the global [K]
C  for some time-step.
C -MasterK is Global [K] for a given time step.
C -IEL Contains the node-list for the element.
C -NEL Number of elements.

        SK(1)=2*EI(IELEMENT)-1
        SK(2)=2*EI(IELEMENT)
        SK(3)=2*EJ(IELEMENT)-1
        SK(4)=2*EJ(IELEMENT)
        SK(5)=2*EK(IELEMENT)-1
        SK(6)=2*EK(IELEMENT)

C       This skips the EM for the triangular element to avoid
C       SK(7) and SK(8) = -1 and MERGEs the SQUARE element.

        IF (EM(IELEMENT).NE.0) THEN

          SK(7)=2*EM(IELEMENT)-1
          SK(8)=2*EM(IELEMENT)

C  ***************  Merge into  [K(t)]  ****************************

          DO 15,I=1,8
            DO 10,J=1,8
              MASTERK(SK(I),SK(J))=MASTERK(SK(I),SK(J))+K(I,J)
10          CONTINUE
15        CONTINUE

        ELSE
C  This is the MERGE routine for the TRIANGULAR element occurrs.

20        DO 35,I=1,6
            DO 30,J=1,6
              MASTERK(SK(I),SK(J))=MASTERK(SK(I),SK(J))+K(I,J)
30          CONTINUE
35        CONTINUE

        ENDIF

        RETURN
        END

C ___________________________________________________________________________
```

```
C ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^


        SUBROUTINE RECTANGLECALC(C,k)
        DOUBLE PRECISION C(6),K(8,8)


100     FORMAT (A)

C  INCORPORATE SOME KIND OF THICKNESS TERM HERE!!!!
C  This subroutine returns a brute force solution to the calculation of the
C  [K] matrix for the RECTANGULAR element.  Ugly but fast.

        K(1,1)=2.0D0*(C(1)+C(6))
        K(1,2)=C(3)+C(4)
        K(1,3)=-2.0D0*C(1)+C(6)
        K(1,4)=C(3)-C(4)
        K(1,5)=-1.0D0*(C(1)+C(6))
        K(1,6)=-1.0D0*(C(3)+C(4))
        K(1,7)=C(1)-2.0D0*C(6)
        K(1,8)=-1.0D0*C(3)+C(4)

        K(2,1)=K(1,2)
        K(2,2)=2.0D0*(C(2)+C(5))
        K(2,3)=-1.0D0*C(3)+C(4)
        K(2,4)=C(2)-2.0D0*C(5)
        K(2,5)=-1.0D0*(C(3)+C(4))
        K(2,6)=-1.0D0*(C(2)+C(5))
        K(2,7)=C(3)-C(4)
        K(2,8)=-2.0D0*C(2)+C(5)

        K(3,1)=K(1,3)
        K(3,2)=K(2,3)
        K(3,3)=2.0D0*(C(1)+C(6))
        K(3,4)=-1.0D0*(C(3)+C(4))
        K(3,5)=C(1)-2.0D0*C(6)
        K(3,6)=C(3)-C(4)
        K(3,7)=-1.0D0*(C(1)+C(6))
        K(3,8)=C(3)+C(4)

        K(4,1)=K(1,4)
        K(4,2)=K(2,4)
        K(4,3)=K(3,4)
        K(4,4)=2.0D0*(C(2)+C(5))
        K(4,5)=-1.0D0*C(3)+C(4)
        K(4,6)=-2.0D0*C(2)+C(5)
        K(4,7)=C(3)+C(4)
        K(4,8)=-1.0D0*(C(2)+C(5))

        K(5,1)=K(1,5)
        K(5,2)=K(2,5)
        K(5,3)=K(3,5)
        K(5,4)=K(4,5)
        K(5,5)=2.0D0*(C(1)+C(6))
        K(5,6)=C(3)+C(4)
        K(5,7)=-2.0D0*C(1)+C(6)
        K(5,8)=C(3)-C(4)

        K(6,1)=K(1,6)
        K(6,2)=K(2,6)
        K(6,3)=K(3,6)
        K(6,4)=K(4,6)
        K(6,5)=K(5,6)
        K(6,6)=2.0D0*(C(2)+C(5))
        K(6,7)=-1.0D0*C(3)+C(4)
        K(6,8)=C(2)-2.0D0*C(5)
```

```
      K(7,1)=K(1,7)
      K(7,2)=K(2,7)
      K(7,3)=K(3,7)
      K(7,4)=K(4,7)
      K(7,5)=K(5,7)
      K(7,6)=K(6,7)
      K(7,7)=2.0D0*(C(1)+C(6))
      K(7,8)=-1.0D0*(C(3)+C(4))

      K(8,1)=K(1,8)
      K(8,2)=K(2,8)
      K(8,3)=K(3,8)
      K(8,4)=K(4,8)
      K(8,5)=K(5,8)
      K(8,6)=K(6,8)
      K(8,7)=K(7,8)
      K(8,8)=2.0D0*C(2)+2.0D0*C(5)


C     >THICKNESS FACTOR HERE!!!!!!!!!!!!!!!!!!


      RETURN
      END
C ----------------------------------------------------------------------
```

```
C ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
      SUBROUTINE STOREMASTERK(NT,MASTERK,NUMNODES)

C     STORE the values for [K(t)] in a unique file, once the values have been
C     calculated.

      INCLUDE ' KSTORE.FOR'

      DOUBLE PRECISION MASTERK(NUMNODES,NUMNODES)

      COMMON/NUM/NUMELS

      INTEGER IBANDWIDTH
      COMMON/IBW/IBANDWIDTH

      CHARACTER*15 FILENAME

C  Note that this only stores the diagonal and upper values of the [K]
C  matrix.  IF IT AIN'T SYMMETRICAL IT'S GONNA BE!!

100   FORMAT (A)

C     WRITE (*,100)'  TEST DUMP BEFORE STORAGE IN STOREMASTERK'

C     DO IT=1,NUMNODES
C       DO JT=1,NUMNODES
C        WRITE (*,150) IT,JT,MASTERK(IT,JT)
C       ENDDO
C     ENDDO

150   format (I3,i3,e12.4)

C     ************* Column Matrix Storage Conversion ******

      ICOUNTER=1

      DO 55,IROW=1,NUMNODES

        IF ((IROW+IBANDWIDTH).GE.NUMNODES) THEN
          MAXCOL=NUMNODES
        ELSE
          MAXCOL=IROW+IBANDWIDTH-1
        ENDIF

        DO 50,ICOL=IROW,MAXCOL
C     WRITE (*,*) NT, ICOUNTER, IROW, ICOL
      STOREDK(NT,ICOUNTER)=MASTERK(IROW,ICOL)
      ICOUNTER=ICOUNTER+1


50      CONTINUE
55    CONTINUE

C ******* END OF COLUMN-MATRIX STORAGE ROUTINE ********


      RETURN
      END
C ________________________________________________________________________________
```

```
C ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

        SUBROUTINE SOLUTION (NUMNODES,NNPLUS1,KZERO,IDIM,IDPLUS1,FINALK,
     C  LASTR)

        INCLUDE ' KSTORE.FOR'

        DOUBLE PRECISION KINTER(200,200)

        DOUBLE PRECISION KZERO(NUMNODES,NNPLUS1)
        DOUBLE PRECISION FINALK(IDIM,IDPLUS1)
        INTEGER NRC

        COMMON/NUM/NUMELS
        COMMON/RBLOCK/R
        COMMON/BDY/NUMBDY
        COMMON/IBW/IBANDWIDTH


        COMMON /NTS/NUMBEROFTIMESTEPS

        DOUBLE PRECISION R(200),FINALR(200)
        DOUBLE PRECISION DELU(200)
        DOUBLE PRECISION LASTR(200)

        DOUBLE PRECISION Y(200)

100     FORMAT (A)

C       This subroutine retrieves the appropriate
C       values for [R(t)], [dU] etc. and steps through the appropriate sets of
C       solutions. But first the [K(0)] values must be retieved.

        NT=0


C       **************************************************************
C       This unpacks the [K] matrix from the appropriate column
C       of the STOREDK(TIME,COLUMN) array.

        WRITE (*,*) NUMNODES, IBANDWIDTH

        ICOUNTER=1

        DO IROW=1,NUMNODES

          IF ((IROW+IBANDWIDTH).GE.NUMNODES) THEN
            MAXCOL=NUMNODES
          ELSE
            MAXCOL=IROW+IBANDWIDTH-1
          ENDIF

C       WRITE (*,100) '   MAXCOL'
C       WRITE (*,*) MAXCOL

          DO ICOL=IROW,MAXCOL
C       WRITE (*,100) '   **********'

            KZERO(IROW,ICOL)=STOREDK(NT,ICOUNTER)
           KZERO(ICOL,IROW)=KZERO(IROW,ICOL)
            ICOUNTER=ICOUNTER+1

C       WRITE (*,100) IROW,ICOL,KZERO(IROW,ICOL)

          ENDDO
        ENDDO
```

```
C .********** END OF UNPACKING ROUTINE ***************

150     format (I3,I3,e12.4)
C       write (*,100) '  test dump after retrieval routine in SOLUTION'

C       DO II=1,NUMNODES
C         DO JJ=1,NUMNODES
C           WRITE (*,150) II,JJ,KZERO(II,JJ)
C         ENDDO
C       ENDDO

C       ****************************************************

C  Retrieve the {R} vector's value


C  NOTE:  Because of a glitch in VMS-FORTRAN {R} is passed via
C         a common statement.

        DO NT=1,NUMBEROFTIMESTEPS

C                   O    O
        CALL FINDR(NT,NUMNODES)

C  Subtract the stresses from the previous timesteps

C                          I     O     O     O    O      O
        CALL SUBRESIDUAL(FINALR,NT,KINTER)


C  Account for the known boundary conditions

C                   O      O      O     O     O      I     I
        CALL BYVAL(NUMBDY,NUMNODES,KZERO,FINALR,IDIM,LASTR,FINALK,
     C IDPLUS1,O)

C  NOTE: FINALK is the augmented matrix containing the remaining
C  simultaneous equations to be solved.

C  Attempt a solution:

C       WRITE (*,100) ' SIMULT  CALLED, DEL-{U} VALUES FOLLOW'

        CALL SIMULT(IDIM,FINALK,DELU,1.0D-25,1,IDPLUS1,Y)

C  Store the results, AFTER re-inserting known boundary conditions.

        CALL STOREDELU(DELU,NT,NUMNODES)

C  Print the results

C       CALL PRINTDELU(        )

        ENDDO
        RETURN
        END


C ____________________________________________________________________
```

```
C ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

      SUBROUTINE FINDR(NT,NUMNODES)
      DOUBLE PRECISION R(200)
      DOUBLE PRECISION VAL
      INTEGER INDEX,IDIR

      COMMON/NUM/NUMELS
      COMMON/NS/NUMSTART,NUMSTOP
      COMMON/RBLOCK/R

      CHARACTER*15 FILENAME
C  This subroutine develops the value of {R} for timestep number NT (actual
C  time value of TIMEVAL(NT) seconds) and readies it for the subtraction of the
C  residual stress values.

100   FORMAT (A)

      IF (NT.GE.NUMSTART .AND. NT.LE.NUMSTOP) THEN
        NN=1
      ELSE
        NN=0
      ENDIF

C  This applies a stepped input between NUMSTART and NUMSTOP time intervals
C  which requires the availability of RNUMBER1 and RNUMBER0 files.  A Dirac
C  spike has the same value for NUMSTART and NUMSTOP.  Other input shapes
C  may be created by modifying the values of the function  and letting
C  NN=NT during the relevant time-frame.

      WRITE (FILENAME,50)'RNUMBER',NN
50    FORMAT (A,I2)

      OPEN (20,FILE=FILENAME, STATUS='OLD')
      REWIND 20

c     WRITE (*,100) '   FILENAME:'
c     WRITE(*,100) FILENAME

C  Set the whole vector to 0.000

      DO I=1,NUMNODES
        R(I)=0.000
      ENDDO

C  Read the non-zero values of R from the appropriate file
C  Remember: x=1, y=0

      READ (20,*)NUMRVAL

      WRITE (*,100)

      DO J=1,NUMRVAL
        READ(20,*) INDEX,IDIR,VAL
        NEWINDEX=(INDEX*2)-IDIR
        R(NEWINDEX)=VAL
c       WRITE (*,*) INDEX, IDIR, VAL, NEWINDEX

      ENDDO

      IF (NT .LE. 2) THEN
        WRITE (*,100)'  '
        WRITE (*,50) '          {R} VECTOR FOR TIMESTEP',NT
        WRITE (*,100) '       NODE              X                  Y'
        WRITE (*,100)'  '
      ENDIF
```

```
      DO ID=1,NUMNODES,2
      JD=ID+1
      NNUM=JD/2
        IF (NT .LE. 2) THEN
          WRITE (*,*) NNUM,R(ID),R(JD)
        ENDIF
      ENDDO

      CLOSE(20)

      RETURN
      END
C ____________________________________________________________________
```

```
C ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
C                            O   I   I
      SUBROUTINE SUBRESIDUAL(FINALR,NT,KINTER)

C     This subroutine subtracts the residual force, from previous timesteps
C     from the current {R} value.

      INCLUDE ' KSTORE.FOR'

      COMMON /IDIMBLOCK/IDIM
      COMMON/NNBLOCK/NUMNODES
      COMMON/NUM/NUMELS
      COMMON /NTS/NUMBEROFTIMESTEPS
      COMMON/IBINDX/IBDY
      COMMON/BDY/NUMBDY
      COMMON/RBLOCK/R
      COMMON/KPASTBLOCK/KPAST
      COMMON/IBW/IBANDWIDTH

      INTEGER IBANDWIDTH
      INTEGER IBDY(200),NUMNODES

      DOUBLE PRECISION DU(200),DUMMY(200)
      DOUBLE PRECISION R(200),RESID(200),FINALR(200)
      DOUBLE PRECISION RINTER(200),DUINTER(200)
      DOUBLE PRECISION KPAST(200,200)
      DOUBLE PRECISION KINTER(IDIM,IDIM)
      DOUBLE PRECISION SUM


      CHARACTER*15 FILEDELU
      CHARACTER*15 FILEKNUM


 101  FORMAT (A,I2)
 100  FORMAT (A)

      WRITE (*,100) ' IDIM IN SUBRESIDUAL'
      WRITE (*,*) IDIM

      WRITE (*,101) ' TIMESTEP NUMBER',NT

      NTPLUS1=NT+1

      IEND=NT-1

c     ************************  Retrieval Loop **************

      DO 80,I=0,IEND

      J=NT-I

C     *****  This extracts [K(t)] from the appropriate column of
C     the STOREDK(TIME, Element) array.


      ICOUNTER=1
      DO IROW=1,NUMNODES

        IF ((IROW+IBANDWIDTH).GE.NUMNODES) THEN
          MAXCOL=NUMNODES
        ELSE
          MAXCOL=IROW+IBANDWIDTH-1
        ENDIF

        DO ICOL=IROW,MAXCOL
```

```
            KPAST(ICOL,IROW)=STOREDK(J,ICOUNTER)
            KPAST(IROW,ICOL)=KPAST(ICOL,IROW)
            ICOUNTER=ICOUNTER+1

          ENDDO
        ENDDO

C       ****************  End of [K(t)] Extraction **********

150     format (I3,i3,e12.4)
C       write (*,100) '  test dump after retrieval routine IN subresidual'

C       DO II=1,NUMNODES
C         DO JJ=1,NUMNODES
C           WRITE (*,150) II,JJ,kpast(II,JJ)
C         ENDDO
C       ENDDO


C       ******  {dU(t)} Extraction  ************************
C       WRITE (*,100) ' I   KK    DELUSTORED(I,KK)'
        DO KK=1,NUMNODES
          DU(KK)=DELUSTORED (I,KK)
C         WRITE (*,*) I,KK,DELUSTORED(I,KK)
        ENDDO

C       ******   End of extraction  ***************


C   Collapse the [KPAST] and {DU} matrices according to the boundary
C   conditions associated with the {DU}'s timestep index.

C   Do {DU} first, producing {DUINTER}:

        IFLAG=0
        DO II=1,NUMNODES
          IF (II .EQ. IBDY(IFLAG+1)) THEN
            IFLAG=IFLAG+1
          ELSE
            DUINTER(II-IFLAG)=DU(II)
          ENDIF
        ENDDO


C  Then [KPAST] producing [KINTER].
C  This is a nested sort similar to the BYVAL() subroutine.
C  Rows first:


        IFLAG=0
        DO II=1,NUMNODES
          IF (II .EQ. IBDY(IFLAG+1)) THEN
            IFLAG=IFLAG+1
          ELSE

C  Then columns:

          JFLAG=0
          DO JJ=1,NUMNODES
            IF (JJ .EQ. IBDY(JFLAG+1)) THEN
              JFLAG=JFLAG+1
            ELSE
C       WRITE (*,100) '  II        IFLAG        JJ         JFLAG'
C       WRITE (*,*) II,IFLAG,JJ,JFLAG
              KINTER(II-IFLAG,JJ-JFLAG)=KPAST(II,JJ)
            ENDIF
          ENDDO
```

```
        ENDIF

        ENDDO

c       WRITE (*,100) '  BEFORE SQBYCOL'
c       WRITE (*,101) ' IDIM = ',IDIM
c       DO JJ=1,IDIM
c        WRITE (*,*) JJ,DUINTER(JJ)
c
c        DO KK=1,IDIM
c
c         WRITE (*,*) JJ,KK,KINTER(JJ,KK)
c        ENDDO
c       ENDDO


C  Multiply [KINTER] by {DUINTER} to produce a "compressed" {RINTER}

        CALL SQUARBYCOL(IDIM,KINTER,DUINTER,RINTER)

c       WRITE (*,100) ' AFTER SQAREBYCOL CALL  -COMPRESSED-'
c       DO NN=1,NUMNODES
c       WRITE (*,*) NN,RINTER(NN)
c       ENDDO


C  "Unpack" the {RINTER} values into the appropriate rows of {RESID}

        IFLAG=0

        DO II=1,NUMNODES

          IF (II .EQ. IBDY(IFLAG+1)) THEN

C  If this is a "deleted" row, regenerate the {RESID(II)} value from the
C  appropriate row and column of [KPAST] and {DU}


            SUM=0.0D0
              DO KK=1,NUMNODES
                IF ( DU(KK) .LE. 1.0D-25) DU(KK)=0.0D0
                SUM=SUM+KPAST(IBDY(IFLAG+1),KK)*DU(KK)
              ENDDO

            RESID(II)=SUM

            IFLAG=IFLAG+1

          ELSE

C  If not, copy the value from {RINTER} and subtract the necessary
C  coefficients (which are carried through from the solution of the
C  system of compressed equations).

          RESID(II)=RINTER(II-IFLAG)

          DO LL=1,NUMBDY
            RESID(II)=RESID(II)+KPAST(II,IBDY(LL))*DU(IBDY(LL))
          ENDDO

C  Finally, subtract the residual terms generated for this series of
C  arrays from the original force vector {R}

        ENDIF
        ENDDO
c       write (*,100) '  in subresidual'
c       write (*,100) ' n     r(n)    resid(n)'
```

```
      DO N=1,NUMNODES
c       WRITE (*,*) N,R(N),RESID(N)
        R(N)=R(N)-RESID(N)
      ENDDO

 80   CONTINUE


C Copy whatever is left into (FINALR) for return from the subroutine.

C     write (*,100) '  at end of sub resid. '
C     write (*,100) ' n     r(n)    finalr(n)'
C     DO N=1,NUMNODES
C       write (*,*) N,R(N),FINALR(N)
C     ENDDO

      DO N=1,NUMNODES
        FINALR(N)=R(N)
      ENDDO

      RETURN
      END
C ____________________________________________________________________
```

```
C ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

        SUBROUTINE STOREDELU(DELU,NT,NUMNODES)

        INCLUDE ' KSTORE.FOR'

        DOUBLE PRECISION DELU(200)
        CHARACTER*15 FILEDELU
C  This subroutine stores the calculated values of {/\U} in individual files

        COMMON/NUM/NUMELS

        INTEGER IBDY(200)
        COMMON/IBINDX/IBDY

        DOUBLE PRECISION BVAL(200)
        COMMON/BOUNDVAL/BVAL

        INTEGER ISTORFLAG
        COMMON/ISF/ISTORFLAG

        DOUBLE PRECISION DUMMY(200)
        INTEGER NODEHALF,NNUM
C  Reinsert known boundary values in the {DELU} array.
C  NOTE the {DUMMY} vector is the "unpacked" solution
C  vector which also contains the known boundary conditions
C  at the particular time-step.

        IFLAG=0

        DO II=1,NUMNODES
          IF (II.EQ.IBDY(IFLAG+1)) THEN
            DUMMY(II)=BVAL(IFLAG+1)
            IFLAG=IFLAG+1
          ELSE
            DUMMY(II)=DELU(II-IFLAG)
          ENDIF
        ENDDO

C  Truncate very small values to avoid underflow errors:

        DO KK=1,NUMNODES
            IF (DABS(DUMMY(KK)) .LE. 1.0D-15) THEN
              DUMMY(KK)=0.0D0
            ENDIF
        ENDDO

C  ********************FOR ALL NON-ZERO TIMESTEPS******************

       IF (NT.NE.0) THEN


C         WRITE (*,100) ' STOREDELU RUN '
100       FORMAT (A)


C   Write to appropriate array column.

          DO I=1,NUMNODES
            DELUSTORED(NT,I) = DUMMY(I)
          ENDDO


C  Screen output for instant gratification.

85      FORMAT (A,I2)
```

```
      WRITE (*,100)'  '
      WRITE (*,85) '  DISPLACEMENT VALUES FOR TIMESTEP',NT
      WRITE (*,100) '  VALUES LESS THAN 1.0D-15 ARE STORED AS 0.0D0'
      WRITE (*,100) '          NODE          DX                 DY'

      DO ID=1,NUMNODES,2
        JD=ID+1
        NNUM=(JD/2)
        WRITE (*,*) NNUM,DUMMY(ID),DUMMY(JD)
      ENDDO

C  This stores the {dU} vectors to disk for use with parameter
C  estimation software.  Note that there are two format statements
C  since some compilers won't add in the leading 0 to the appended
C  filename call.

86      FORMAT (A,I1)
        IF (ISTORFLAG .EQ. 1) THEN
          IF (NT .LT. 10) THEN
            WRITE (FILEDELU,86)'DELUFILE',NT
           ELSE
            WRITE (FILEDELU,85)'DELUFILE',NT
          ENDIF

          OPEN(22,FILE=FILEDELU,STATUS='NEW')

          DO I=1,NUMNODES
              WRITE(22,*)DUMMY(I)
          ENDDO

        ENDIF

      ENDIF
C  ***********************************************************

        WRITE(*,100)' **'
        RETURN
      END

C ____________________________________________________________
```

```
C ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

        SUBROUTINE BYVAL(NUMBDY,NUMNODES,MASTERK,
     C  FINALR,IDIM,LASTR,FINALK,IDPLUS1,LSIG)

        DOUBLE PRECISION MASTERK(NUMNODES,NUMNODES)
        DOUBLE PRECISION FINALK(IDIM,IDPLUS1)


        INTEGER IBDY(200)
        COMMON/IBINDX/IBDY
        DOUBLE PRECISION BVAL(200)
        COMMON/BOUNDVAL/BVAL

        DOUBLE PRECISION FINALR(200)
        DOUBLE PRECISION LASTR(200)
        DOUBLE PRECISION TEMP

C       This subroutine takes the appropriate boundary conditions from the
C       IBDY (indexed list of ROWS which have known boundary conditions)
C       and BVAL (the values associated with the known boundary conditions)
C       and substitute the proper values into the MASTERK array at each
C       time step.  Note that the boundary conditions are NOT time-
C       dependent in this version.

100     FORMAT (A)

C  ZERO OUT THE APPROPRIATE ROWS IN [MASTERK] AND {FINALR}

        DO I=1,NUMBDY
        DO J=1,NUMNODES
          MASTERK(IBDY(I),J)=0.0D0
        ENDDO
          FINALR(IBDY(I))=0.0D0
        ENDDO


C SUBTRACT THE APPROPRIATE VALUES FROM THE {FINALR} VECTOR AND ZERO OUT THE
C APPROPRIATE COLUMNS IN [MASTERK]


        DO I=1,NUMBDY
        DO J=1,NUMNODES
          FINALR(J)=FINALR(J)-MASTERK(J,IBDY(I))*BVAL(I)
          MASTERK(J,IBDY(I))=0.0D0
        ENDDO
       ENDDO

C  NOW, RESTRUCTURE THE ARRAY SO THAT THE "ZERO" ROWS AND COLUMNS ARE
C  ELIMINATED, AND THE FINAL ARRAYS OF [K] AND {R} ARE PROPERLY DI-
C  MENSIONED.

C  FIRST, {R}:

        IFLAG=0
        DO II=1,NUMNODES
          IF (II.EQ.IBDY(IFLAG+1)) THEN
            IFLAG=IFLAG+1
          ELSE
            LASTR(II-IFLAG)=FINALR(II)
          ENDIF
        ENDDO

C ****************************************************

C  THEN [MASTERK]:
C    FIRST ROWS...
```

```
      IFLAG=0
      DO II=1,NUMNODES
        IF (II.EQ.IBDY(IFLAG+1)) THEN
          IFLAG=IFLAG+1
        ELSE

C ***NESTED COLUMN SORT***********************

      JFLAG=0
      DO JJ=1,NUMNODES
        IF (JJ.EQ.IBDY(JFLAG+1)) THEN
          JFLAG=JFLAG+1
        ELSE
          FINALK(II-IFLAG,JJ-JFLAG)=MASTERK(II,JJ)
        ENDIF
      ENDDO

C*************************************

        ENDIF
      ENDDO


C   ...AND FINALLY AUGMENT [FINALK] WITH {LASTR} FOR RETURN AND SOLUTION.

      DO JJ=1,IDIM
        FINALK(JJ,IDPLUS1)=LASTR(JJ)
      ENDDO

      RETURN
      END


C ____________________________________________________
```

```
C ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

      SUBROUTINE SIMULT(N,A,X,EPS,INDIC,NRC,Y)

      DOUBLE PRECISION Y(200),A(N,NRC),X(200)
      DOUBLE PRECISION EPS,PIVOT,DETER,AIJCK,SIMUL

      INTEGER INDIC,NRC,N
      INTEGER IROW(200),JCOL(200),JORD(200)

C FROM "APPLIED NUMERICAL METHODS", B.CARNAHAN, H.A.LUTHER,
C      J.O. WILKES.  J.WILEY & SONS,NEW YORK, 1969
C      CHAPTER 5, p.276
C ***************** DIRECTORY OF VARIABLES *****************
C      N      NUMBER OF ROWS IN A
C      A      AUGMENTED MATRIX OF COEFFICIENTS
C      X       VECTOR OF SOLUTIONS
C      EPS     MINIMUM ALLOWABLE MAGNITUDE FOR A PIVOT ELEMENT
C      INDIC   COMPUTATIONAL SWITCH FOR SOLUTION TYPE
C              (+1 FOR NO INVERSE RETURNED)
C      NRC     COLUMN DIMENSIONS FOR THE MATRIX [A] (N+1)
C *************************************************************

 100  FORMAT (A)

      M=1
      NPLUSM=NRC

C BEGIN ELIMINATION PROCEDURE

      DETER=1.0D0
      DO 9 K=1,N

C CHECK FOR A TOO-SMALL PIVOT ELEMENT
C     WRITE (*,*) N,K,A(K,K),EPS
      IF (DABS(A(K,K)).GT.EPS) GO TO 5
      WRITE (*,100) ' PIVOT TOO SMALL...I QUIT!'
      WRITE (*,100) ' ERROR TRAPPED IN SUBROUTINE -SIMULT-'
      STOP

C NORMALIZE THE PIVOT ROW

 5    KP1=K+1
      DO 6 J=KP1,NPLUSM
 6    A(K,J)=A(K,J)/A(K,K)
      A(K,K)=1.0D0

C ELIMINATE THE K(TH) COLUMN ELEMENTS EXCEPT FOR THE PIVOT

      DO 9 I=1,N
      IF (I.EQ.K .OR. A(I,K).EQ.0.0D0) GO TO 9
      DO 8 J=KP1,NPLUSM
 8    A(I,J)=A(I,J)-A(I,K)*A(K,J)
      A(I,K)=0.0D0
 9    CONTINUE

C WRITE THE SOLUTION VECTOR INTO {X} FOR RETURN

      DO II=1,N
      X(II)=A(II,NRC)
      ENDDO

C THAT'S ALL FOLKS...

      RETURN
      END
```

```
C ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

        SUBROUTINE SQUARBYCOL(ISPEC,SQUARE,COL,RESULT)

C  Multiplies [SQUARE] by {COL} and produces {RESULT}.
C  Note that there is no safety check on dimensions here.

        DOUBLE PRECISION SQUARE(ISPEC,ISPEC)
        DOUBLE PRECISION COL(ISPEC),RESULT(ISPEC)

 100    FORMAT (A)

C       VODOO FORTRAN!!!!!!!!!!!!!
C       This stuff gets around a sunspot factor in VMS FORTRAN

        C=ISPEC
        A=COL(1)
        B=SQUARE(1,1)


C  Clean House

        DO I=1,ISPEC
          RESULT(I)=0.0D0
        ENDDO

C  Do the nasty

        DO IROW=1,ISPEC
          DO ICOL=1,ISPEC
         IF (DABS(SQUARE(IROW,ICOL)*COL(ICOL)) .LE. 1.0D-20) GO TO 20
             RESULT(IROW)=RESULT(IROW)+SQUARE(IROW,ICOL)*COL(ICOL)
 20        CONTINUE
          ENDDO
        ENDDO

        RETURN
        END
C-----------------------------------------------------------------------
```

Table 5. P21.FOR

PROGRAM P21.FOR

```
c     C PROGRAM FOR 'SMART' GLOBAL OPTIMIZATION
c     C INITALIZE VARIABLES:

C  parincl.for here

        DOUBLE PRECISION  CL(50), CU(50), FF(90), PPL(20)
        DOUBLE PRECISION  RR(90, 30), WPEN(50), XC(30), XX(90, 30)
        DOUBLE PRECISION  XXOLD(30)

        DOUBLE PRECISION ALPHAP, BETA, DELTA
        DOUBLE PRECISION Z, ZXC

        INTEGER IC, ICM, ICMI, IEV1
        INTEGER IEV2, IEV3, IIS, IOPT, IGAMMA
        INTEGER IT, ITMAX, IZ, IZRQ
        INTEGER IZXC, JC, JI, JJ
        INTEGER JZ, KOUNT,KK1, NALT, NC
        INTEGER NCMPLX, NFE, NINPS, NLEG
        INTEGER NRUN, NS, NSEG, NSTK
        INTEGER NUMTIMSTEPS, NV, NPRNT

        COMMON/AAA/ALPHAP, BETA, DELTA, IGAMMA
        COMMON/BBB/IC, ICM, ICMI, IEV1
        COMMON/CCC/IEV2, IEV3, IIS, IOPT
        COMMON/DDD/IT, ITMAX, IZ, IZRQ
        COMMON/EEE/IZXC, JC, JI, JJ
        COMMON/FFF/JZ, KOUNT,KK1, NALT, NC
        COMMON/GGG/NCMPLX, NFE, NINPS, NLEG
        COMMON/HHH/NRUN, NS, NSEG, NSTK
        COMMON/III/NV, Z, ZXC, NPRNT, NUMTIMSTEPS

        COMMON/JJJ/CL, CU, FF, PPL, RR, WPEN, XC, XX, XXOLD

        DOUBLE PRECISION TH
        COMMON/THICKBLOCK/TH


       INTEGER IARG

        NFE=0

C  ***********************************
C  CONSTANT THICKNESS TERM:
        TH=1.0D0
C  ***********************************

12     NCMPLX = 1


100    FORMAT (A)
110    FORMAT (A,I2)
115     FORMAT (I2,D12.3,D12.3,D12.3)
        WRITE (*,100)  '    PARAM2 '
        WRITE (*,100)  '    Parameter Estimation Program '
```

Table 5 (Cont'd).

```
        WRITE (*,100)  '    Copyright 1991'
        WRITE (*,100)  '    Scott Morris '
        WRITE (*,100)  '    Michigan State University '

15000  CONTINUE

15001  IOPT = 1
       NRUN = 1
       ICMI = 0
       NALT = 0
       NSTK = 0

C     C WPEN(I) IS WEIGHT GIVEN TO PENALTY I IN FUNCTION


15028 KOUNT = 0


C  ++++++++++++++++++++++++++++++++++

C  READ IN THE PARAMETRIC DATA

C  ++++++++++++++++++++++++++++++++++

        OPEN (14,FILE='par_est',STATUS='old')
        REWIND 14


        READ (14,*) NC,NS,NV,ITMAX,BETA,IGAMMA

        READ (14,*) NUMPAR
        READ (14,*) NUMTIMSTEPS
        READ (14,*) NPRNT
          WRITE (*,100) ' Parameter indeces and limits:'
          WRITE (*,100) ' NI       CL(I)       CU(I)      XX(1,NI)'
          write (*,110) ' numpar=', numpar
        DO I=1,NUMPAR
          write (*,110) ' I=', I
          READ (14,*) NI,CL(NI),CU(NI),XX(1,NI)
          WRITE (*,115) NI,CL(NI),CU(NI),XX(1,NI)
        END DO


C  ----------------------------------------
        DELTA=1.0D-4
C  ----------------------------------------
C       NOTE THAT DELTA VALUE IS FIXED
C  ----------------------------------------
        CLOSE (14)
C  ****************************************************
      CALL LINE15050
C  ****************************************************

15200 CONTINUE

      IC = NC - NS
```

```
      IARG=1199999

        do mm=1,10
          dump=ran(iarg)
        enddo


15400 DO IZ = 2,NV
15410   DO JZ = 1,NS
        write (*,100) ' line 15410'
15420     RR(IZ, JZ) = ran(iarg)
        write (*,100) ' line 15420'
        END DO
15430  continue
        END DO
        write (*,100) ' line 15430'


15450 WRITE (*,100)' PARAMETER VALUES FOR THIS RUN'
      WRITE (*,100)'            NS        NC        NV        ITMAX'
15460 WRITE (*,*)NS,NC,NV,ITMAX
      WRITE (*,100)'   ALPHAP      BETA      IGAMMA      DELTA'

120   FORMAT (D12.2,3X,D12.2,I7,D12.2)

15470 WRITE (*,120) ALPHAP, BETA, IGAMMA, DELTA
      WRITE (*,*) ' '
      WRITE (*,100)' Parameter Estimation Routine'
      WRITE (*,100)' Subroutine trace:'
      WRITE (*,100)' '


15520 GO TO 15535


C     * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C      * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

C     CALL MAIN SUBROUTINE

15535   CALL LINE15700

C     * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C      * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *


15540 IF ((IT - ITMAX) .LE. 0) THEN
       GO TO 15550
      ELSE
       GO TO 15660
      END IF

15550   DO JZ = 1,NS
          XX(IEV1,JZ) = XC(JZ)
        END DO

15552 IIS = IEV1


C  ****************************************

      CALL FUNC

C  ****************************************

      WRITE (*,100) ' '
```

```
15553 WRITE (*,100) '  VALUE OF THE FUNCTION AT THE CENTROID='
      WRITE (*,*) FF(IEV1)

      WRITE (*,100) ' '
15555 WRITE (*,100) '  COODINATES OF THE CENTROID'
15556 DO JZ = 1,NS
      WRITE (*,100) '          INDEX                VALUE'
15557 WRITE (*,*) JZ, XC(JZ)
      END DO

      WRITE (*,100) ' '
15559 WRITE (*,100) ' BEST NON-CENTROID VALUE OF THE FUNCTION ='
      WRITE (*,*) FF(IEV2)


      WRITE (*,100) ' '
15560 WRITE (*,100) '  BEST NON-CENTROID X VALUES'
15590 DO JZ = 1,NS
      WRITE (*,100) '                  INDEX            VALUE'
15600 WRITE (*,*) JZ,XX(IEV2, JZ)
15620 continue
        END DO

15630 WRITE (*,175)' NUMBER OF ITERATIONS=',IT
175   FORMAT (A,I3)
      WRITE (*,175)' FUNCTION EVALUATIONS=',NFE

15650 IF (FF(IEV1) .GE. FF(IEV2)) THEN

        DO JZ = 1,NS
        XX(1,JZ) = XC(JZ)
        END DO

        GOTO 15691
      ELSE

        DO JZ = 1,NS
        XX(1,JZ) = XX(IEV2,JZ)
        END DO

        GOTO 15691
      END IF

15660 WRITE (*,175) ' THE NUMBER OF ITERATIONS HAS EXCEEDED',ITMAX
15665 WRITE (*,100) ' PROGRAM TERMINATED PREMATURELY.  Evaluations:'
      WRITE (*,*) NFE

15690 IF (IT .GE. 0) THEN
        IEV3 = 1

        DO ICM = 2,NV
          IF ((FF(IEV3) - FF(ICM)) .LE. 0.0) THEN
            IEV3 = ICM
          END IF
        END DO

        WRITE (*,100)' THE BEST FUNCTION VALUE YET IS'
        WRITE (*,*) FF(IEV3)
        WRITE (*,100) ' Parameters associated with this best point:'

        DO JC = 1,NS
          WRITE (*,100)'  IEV3        JC       XX( )'
          WRITE (*,*)IEV3,JC,XX(IEV3, JC)
        END DO

        DO JJ = 1,NS
          XX(1,JJ) = XX(IEV3,JJ)
        END DO
```

```
      END IF

15691 WRITE (*,100) '  ************************'
      WRITE (*,100) '  END OF ESTIMATION RUN'
      WRITE (*,100) '  ********************** '

99999 END

C     ****************************************************
C     XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
      SUBROUTINE LINE15700
C     XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
C     ****************************************************
C     MAIN SUBROUTINE STARTS HERE


C  parincl.for here

        DOUBLE PRECISION  CL(50), CU(50), FF(90), PPL(20)
        DOUBLE PRECISION  RR(90, 30), WPEN(50), XC(30), XX(90, 30)
        DOUBLE PRECISION  XXOLD(30)

        DOUBLE PRECISION ALPHAP, BETA, DELTA
        DOUBLE PRECISION Z, ZXC

        INTEGER IC, ICM, ICMI, IEV1
        INTEGER IEV2, IEV3, IIS, IOPT, IGAMMA
        INTEGER IT, ITMAX, IZ, IZRQ
        INTEGER IZXC, JC, JI, JJ
        INTEGER JZ, KOUNT,KK1, NALT, NC
        INTEGER NCMPLX, NFE, NINPS, NLEG
        INTEGER NRUN, NS, NSEG, NSTK
        INTEGER NUMTIMSTEPS, NV, NPRNT

        COMMON/AAA/ALPHAP, BETA, DELTA, IGAMMA
        COMMON/BBB/IC, ICM, ICMI, IEV1
        COMMON/CCC/IEV2, IEV3, IIS, IOPT
        COMMON/DDD/IT, ITMAX, IZ, IZRQ
        COMMON/EEE/IZXC, JC, JI, JJ
        COMMON/FFF/JZ, KOUNT,KK1, NALT, NC
        COMMON/GGG/NCMPLX, NFE, NINPS, NLEG
        COMMON/HHH/NRUN, NS, NSEG, NSTK
        COMMON/III/NV, Z, ZXC, NPRNT, NUMTIMSTEPS

        COMMON/JJJ/CL, CU, FF, PPL, RR, WPEN, XC, XX, XXOLD

100   FORMAT (A)
        WRITE (*,100) ' (15700)'
15700 CONTINUE

15715 KSTK = 0
      NFE = 0
15730 IT = 1
15740 KODE = 0
15750 IF ((NC - NS) .LE. 0) THEN
        GO TO 15770
      ELSE
        GO TO 15760
      END IF

15760 KODE = 1

15770  CONTINUE

15780 DO II = 2,NV
15790 DO IV = 1,NS
```

```
15800 XX(II,IV) = 0.000
      END DO
      END DO

15820 DO II = 2,NV
15830   DO IV = 1,NS
15840   IIS = II
15850 CONTINUE
15860   XX(II, IV) = CL(IV) + RR(II, IV) * (CU(IV) - CL(IV))
15870 continue
        END DO

15880 KK1 = II
15890 CONTINUE

C  ***************************************************

C     CHECK CONSTRAINTS
15895 CALL LINE16700

C  ***************************************************

15900 IF ((II - 2) .LE. 0) THEN
        GO TO 15910
      ELSE
        GO TO 15970
      END IF

15910 IF (IPRINT .NE. 0) THEN
        GO TO 15920
      ELSE
        GO TO 15990
      END IF

15920 WRITE (*,100) 'COORDINATES OF INITIAL COMPLEX'
15940 IO = 1

15945   DO IV = 1,NS
15950     WRITE (*,100) XX(IO, IV)
15955   continue
        END DO

15970 IF (IPRINT .NE. 0) THEN
        GO TO 15975
      ELSE
        GO TO 15990
      END IF

      WRITE (*,100) ' II      IV      XX(II,IV)'
15975   DO IV = 1,NS
15980     WRITE (*,*) II, IV, XX(II, IV)
15985   continue
        END DO
15990   continue
        END DO


16000 KK1 = NV

16010 DO IIS = 1,NV

16020 CONTINUE
16025   NFE = NFE + 1

C     ******************************
        CALL FUNC
C     ******************************
```

```
16030 continue
        END DO

16040 KOUNT = 1
16050 IA = 0
16060 IF (IPRINT .NE. 0) THEN
        GO TO 16070
      ELSE
        GO TO 16110
      END IF

16070 WRITE (*,100) 'VALUES OF THE FUNCTION'
      WRITE (*,100) ' IV    FF(IV)'
16080 DO IV = 1,NV
16090   WRITE (*,*) IV,FF(IV)
16100 continue
        END DO
16110 IEV1 = 1

16120 DO ICM = 2,NV
16130  IF ((FF(IEV1) - FF(ICM)) .LE. 0.0D0) THEN
        GO TO 16150
       ELSE
        GO TO 16140
       END IF

16140  IEV1 = ICM
16150 continue
        END DO

16160 IEV2 = 1

16170 DO ICM = 2,NV
16180   IF ((FF(IEV2) - FF(ICM)) .LE. 0.0) THEN
          GO TO 16190
        ELSE
          GO TO 16200
        END IF

16190   IEV2 = ICM
16200 continue
        END DO

16210 IF ((FF(IEV2)-(FF(IEV1)+BETA)) .LT. 0.0) THEN
       GO TO 16240
      ELSE
       GO TO 16220
      END IF

16220 KOUNT = 1
16230 GOTO 16260

16240 KOUNT = KOUNT + 1


16250 IF ((KOUNT - IGAMMA) .LT. 0) THEN
        GO TO 16260
      ELSE
        GO TO 16600
      END IF

16260 CONTINUE

C  ******************************

C  COMPUTE CENTROID

16265 CALL LINE17000
```

```
C  ******************************

16267 IF (IT .LE. (2 * NV)) THEN
        ALPHA = 1.6
       ELSEIF (((2 * NV) .LT. IT) .AND. (2 * NV) .LE. (4 * NV)) THEN
        ALPHA = 1.3
       ELSEIF ((4 * NV) .LT. IT) THEN
        ALPHA = 1
      END IF
16268 IF (KOUNT .GT. 0) THEN ALPHA = .8
16269 IF (IALPH .EQ. 0) THEN ALPHA = ALPHAP

16271 IF (NSTK .LT. 1) GO TO 16275

16272 DO JJ = 1,NS
16273   XXOLD(JJ) = XX(IEV1, JJ)
      END DO

16274 KSTK2 = 0
16275 DO JJ = 1,NS
16280 XX(IEV1, JJ) = (1.0D0 + ALPHA) * (XC(JJ)) - ALPHA * (XX(IEV1, JJ))
16285 continue
        END DO
16290 IIS = IEV1

16300 CONTINUE

C     CHECK CONSTRAINTS*******************

16305 CALL LINE16700

C     ************************************

16310 CONTINUE
16315 NFE = NFE + 1

C     COMPUTE FUNCTION********************

        CALL FUNC

C     ************************************

16320 IEV2 = 1


16330 DO ICM = 2,NV
16340 IF ((FF(IEV2) - FF(ICM)) .LE. 0.0) THEN
        GO TO 16360
      ELSE
        GO TO 16350
      END IF

16350 IEV2 = ICM
16360 continue
        END DO

16370 IF ((IEV2 - IEV1) .NE. 0) THEN
        GO TO 16450
      ELSE
        GO TO 16380
      END IF

16380 KSTK = KSTK + 1
110   FORMAT (A,I2)

        IF (KSTK .GE. 8) THEN
          WRITE (*,100) ' HELP! I'M STUCK!  -ERROR TRAP AT 16380-'
          WRITE (*,100) ' KSTK >6'
```

```
          STOP
        ENDIF

        NSTK = NSTK + 1
16381 IF ((NSTK .LT. 3) .AND. (KSTK .GE. 6)) THEN
        WRITE (*,100) ' JUMPING OUT AT LINE 16381'
        GO TO 16600
      END IF


16382  IF ((NSTK .GE. 3) .AND. (KSTK .GE. 6) .AND. (KOUNT .LT. 2)) THEN
         KSTK2 = KSTK2 + 1

         IF (KSTK2 .GE. 2) THEN
          WRITE (*,100)'  JUMPING OUT AT LINE 16382'
          GO TO 16600
         END IF

         IEV3 = 1

         DO ICM = 2,NV
           IF ((FF(IEV3) - FF(ICM)) .LE. 0.0) THEN
             IEV3 = ICM
           END IF
         END DO

         DO JJ = 1,NS
           XC(JJ) = XX(IEV3, JJ)
         WRITE (*,*) XC(JJ)
           XX(IEV1, JJ) = XXOLD(JJ)
         END DO
         KSTK = 0
         WRITE (*,100) ' REPLACING CENTROID BY BEST POINT'
         GOTO 16275
      END IF

16385 DO JJ = 1,NS
16390 XX(IEV1,JJ) = ((XX(IEV1, JJ) + XC(JJ)) / 2.0D0)
16400 continue
        END DO

16410 IIS = IEV1
16420 CONTINUE

C  ********************************

C      CHECK CONSTRAINTS
16425 CALL LINE16700

C  ********************************

16430 CONTINUE
16435 NFE = NFE + 1

C  **********************************

        CALL FUNC

C  **********************************

16440 GOTO 16320
16450 KSTK = 0
16460 IF (IPRINT .NE. 0) THEN
        GO TO 16470
      ELSE
        GO TO 16580
      END IF
180   FORMAT (A,I3)
```

```
16470 WRITE (*,180) '  ITERATION NUMBER',IT
16490 WRITE (*,100) '  COORDINATES OF CORRECTED POINT'

16500 DO JC = 1,NS
16510 WRITE (*,100) ' XX(      )='
      WRITE (*,*) IEV1,JC,XX(IEV1,JC)

      END DO
16520 WRITE (*,100) ' VALUES OF THE FUNCTION'

16525 DO IIS = 1,NV
16530 WRITE (*,100) ' FF( )='
      WRITE (*,*) IIS,FF(IIS)

      END DO

16540 WRITE (*,100) ' COORDINATES OF THE CENTROID'

16550 DO JC = 1,NS
16560 WRITE (*,100) ' XC(   )='
      WRITE (*,*) JC,XC(JC)

      END DO

16580 IT = IT + 1
16590 IF ((IT - ITMAX) .LE. 0) THEN
        GO TO 16110
      ELSE
        CONTINUE
      END IF

16600 END


C     16700 16700 16700 16700 16700 16700 16700 16700 16700 16700
C     ****************************************************************
      SUBROUTINE LINE16700
C     ****************************************************************


C  parincl.for here

        DOUBLE PRECISION  CL(50), CU(50), FF(90), PPL(20)
        DOUBLE PRECISION  RR(90, 30), WPEN(50), XC(30), XX(90, 30)
        DOUBLE PRECISION  XXOLD(30)

        DOUBLE PRECISION ALPHAP, BETA, DELTA
        DOUBLE PRECISION Z, ZXC

        INTEGER IC, ICM, ICMI, IEV1
        INTEGER IEV2, IEV3, IIS, IOPT, IGAMMA
        INTEGER IT, ITMAX, IZ, IZRQ
        INTEGER IZXC, JC, JI, JJ
        INTEGER JZ, KOUNT,KK1, NALT, NC
        INTEGER NCMPLX, NFE, NINPS, NLEG
        INTEGER NRUN, NS, NSEG, NSTK
        INTEGER NUMTIMSTEPS, NV, NPRNT

        COMMON/AAA/ALPHAP, BETA, DELTA, IGAMMA
        COMMON/BBB/IC, ICM, ICMI, IEV1
        COMMON/CCC/IEV2, IEV3, IIS, IOPT
        COMMON/DDD/IT, ITMAX, IZ, IZRQ
        COMMON/EEE/IZXC, JC, JI, JJ
        COMMON/FFF/JZ, KOUNT,KK1, NALT, NC
        COMMON/GGG/NCMPLX, NFE, NINPS, NLEG
        COMMON/HHH/NRUN, NS, NSEG, NSTK
        COMMON/III/NV, Z, ZXC, NPRNT, NUMTIMSTEPS
```

```
        COMMON/JJJ/CL, CU, FF, PPL, RR, WPEN, XC, XX, XXOLD

100   FORMAT (A)
        WRITE (*,100) ' (16700)'

16700 CONTINUE
16720 KT = 0

16740 DO IV = 1,NS
16750 IF ((XX(IIS, IV) - CL(IV)) .LE. 0.0) THEN
        GO TO 16760
      ELSE
        GO TO 16790
      END IF

16760 XX(IIS, IV) = CL(IV) + DELTA
16780 GOTO 16810
16790 IF ((CU(IV) - XX(IIS, IV)) .LE. 0.0) THEN
        GO TO 16800
      ELSE
        GO TO 16810
      END IF

16800 XX(IIS, IV) = CU(IV) - DELTA
16810 continue
        END DO

16820 IF (KODE .LE. 0) THEN
        GO TO 16960
      ELSE
        GO TO 16830
      END IF

16830 NN = NS + 1
16840 DO IV = NN,NC
16850 CONTINUE

C  ******************************

C     CALL CONSTRAINT SUBROUTINE

16855  CONTINUE
        WRITE (*,100) '  CONTRAINT ACCESS FAILED.  PARAM/16855'
C      CALL CONSTR

C  ******************************

16860 IF ((XX(IIS, IV) - CL(IV)) .LT. 0.0) THEN
        GO TO 16880
      ELSE
        GO TO 16870
      END IF

16870 IF ((CU(IV) - XX(IIS, IV)) .LT. 0.0) THEN
        GO TO 16880
      ELSE
        GO TO 16940
      END IF

16880 IEV1 = IIS
16890 KT = 1
16900 CONTINUE

C  ************************************

C     COMPUTE CENTROID

16905 CALL LINE17000
```

```
C  ************************************

16910 DO JJ = 1,NS
16920 XX(IIS, JJ) = ((XX(IIS, JJ) + XC(JJ)) / 2.0D0)
16930  continue
        END DO
16940  continue
        END DO
16950 IF (KT .LE. 0) THEN
        GO TO 16960
      ELSE
        GO TO 16720
      END IF

16960 END


C     ***************************************************
      SUBROUTINE LINE17000
C     ****************************************************
C     SUBROUTINE TO COMUPUTE CENTROID


C  parincl.for here

        DOUBLE PRECISION  CL(50), CU(50), FF(90), PPL(20)
        DOUBLE PRECISION  RR(90, 30), WPEN(50), XC(30), XX(90, 30)
        DOUBLE PRECISION  XXOLD(30)

        DOUBLE PRECISION ALPHAP, BETA, DELTA
        DOUBLE PRECISION Z, ZXC

        INTEGER IC, ICM, ICMI, IEV1
        INTEGER IEV2, IEV3, IIS, IOPT, IGAMMA
        INTEGER IT, ITMAX, IZ, IZRQ
        INTEGER IZXC, JC, JI, JJ
        INTEGER JZ, KOUNT,KK1, NALT, NC
        INTEGER NCMPLX, NFE, NINPS, NLEG
        INTEGER NRUN, NS, NSEG, NSTK
        INTEGER NUMTIMSTEPS, NV, NPRNT

        COMMON/AAA/ALPHAP, BETA, DELTA, IGAMMA
        COMMON/BBB/IC, ICM, ICMI, IEV1
        COMMON/CCC/IEV2, IEV3, IIS, IOPT
        COMMON/DDD/IT, ITMAX, IZ, IZRQ
        COMMON/EEE/IZXC, JC, JI, JJ
        COMMON/FFF/JZ, KOUNT,KK1, NALT, NC
        COMMON/GGG/NCMPLX, NFE, NINPS, NLEG
        COMMON/HHH/NRUN, NS, NSEG, NSTK
        COMMON/III/NV, Z, ZXC, NPRNT, NUMTIMSTEPS

        COMMON/JJJ/CL, CU, FF, PPL, RR, WPEN, XC, XX, XXOLD
100     FORMAT(A)

        WRITE (*,100) ' (17000)  Centroid Computation'
17000 CONTINUE

17020 DO IV = 1,NS
17030   XC(IV) = 0.0D0
17040   DO IL = 1,KK1
17050    XC(IV) = XC(IV) + XX(IL, IV)
        END DO
17060   RK = KK1
17070   XC(IV) = (XC(IV) - XX(IEV1, IV)) / (RK - 1.0D0)
      END DO

17080   END
```

```
C       ************************************************************
        SUBROUTINE FUNC
C       ************************************************************


C  parincl.for here

         DOUBLE PRECISION  CL(50), CU(50), FF(90), PPL(20)
         DOUBLE PRECISION  RR(90, 30), WPEN(50), XC(30), XX(90, 30)
         DOUBLE PRECISION  XXOLD(30)

         DOUBLE PRECISION ALPHAP, BETA, DELTA
         DOUBLE PRECISION Z, ZXC

         INTEGER IC, ICM, ICMI, IEV1
         INTEGER IEV2, IEV3, IIS, IOPT, IGAMMA
         INTEGER IT, ITMAX, IZ, IZRQ
         INTEGER IZXC, JC, JI, JJ
         INTEGER JZ, KOUNT,KK1, NALT, NC
         INTEGER NCMPLX, NFE, NINPS, NLEG
         INTEGER NRUN, NS, NSEG, NSTK
         INTEGER NUMTIMSTEPS, NV, NPRNT

         COMMON/AAA/ALPHAP, BETA, DELTA, IGAMMA
         COMMON/BBB/IC, ICM, ICMI, IEV1
         COMMON/CCC/IEV2, IEV3, IIS, IOPT
         COMMON/DDD/IT, ITMAX, IZ, IZRQ
         COMMON/EEE/IZXC, JC, JI, JJ
         COMMON/FFF/JZ, KOUNT,KK1, NALT, NC
         COMMON/GGG/NCMPLX, NFE, NINPS, NLEG
         COMMON/HHH/NRUN, NS, NSEG, NSTK
         COMMON/III/NV, Z, ZXC, NPRNT, NUMTIMSTEPS

         COMMON/JJJ/CL, CU, FF, PPL, RR, WPEN, XC, XX, XXOLD

C        KSTORE.FOR here

C                                  // THIS GIVES A 0->#  OF TIMESTEPS ARRAY
         DOUBLE PRECISION STOREDK(0:50,10000)
         COMMON /KSTOR/ STOREDK

         DOUBLE PRECISION DELUSTORED(0:50,500)
         COMMON /DUSTORE/ DELUSTORED


         COMMON /NTS/ NUMBEROFTIMESTEPS

         DOUBLE PRECISION DIFF
         DOUBLE PRECISION DATAVALUE (50,500)
         INTEGER NODCOUNT
         CHARACTER*15 CURRENTFILE


100     FORMAT(A)
110     FORMAT(A,I3)
111     FORMAT(A,I1)

        WRITE (*,100) ' *'
        WRITE (*,100) ' **'
        WRITE (*,100) ' ***'
        WRITE (*,110) '     ====== Function Evaluation ======',NFE
        WRITE (*,100) ' '

18020   CONTINUE


C  ****************************************
```

```
      CALL PARAVISCO2 (NODCOUNT)

C  ***************************************************************
C  *  Note that NODCOUNT is returned into the subroutine from *
C  *  PARAVISCO to be used in the function evaluation.        *
C  ***************************************************************

C  Read the array of "real world" data into DATAVALUE(TIMESTEP,INDEX)

c       WRITE (*,100) '  IN FUNC ; NUMBEROFTIMESTEPS,NODCOUNT'
c       WRITE (*,*) NUMBEROFTIMESTEPS,NODCOUNT


        IF (NFE .EQ. 1)  THEN
          DO NT=1,NUMBEROFTIMESTEPS
            IF (NT .LT. 10) THEN
              WRITE (CURRENTFILE,111)'datafile',NT
              OPEN (16,FILE=CURRENTFILE, STATUS='old')
            ELSE
              WRITE (CURRENTFILE,110)'datafile',NT
              OPEN (16,FILE=CURRENTFILE, STATUS='old')
            ENDIF

            REWIND 16

            DO I=1,NODCOUNT
              READ (16,*) DATAVALUE(NT,I)
            END DO

            WRITE (*,100) ' LOADED DATA FROM FILE'
            WRITE (*,100) CURRENTFILE
            CLOSE (16)
          END DO
        ENDIF

        DIFF=0.0D0


C   Note that DATAVALUE is the historical value for the x or y displacement
C   from the DELUFILE, and the program takes DELU(n) directly from the
C   PARAVISCO2 version of VISCO1.for.

C       WRITE(*,100) '  Data Comparison:'

C       WRITE (*,100) ' NT  I  DATAVALUE(NT,I)    DELUSTORED(NT,I)'

        DO NT=1,NUMBEROFTIMESTEPS
        WRITE (*,100) '   '
          DO I=1,NODCOUNT


C      THIS SKIPS THE TRIVIAL VALUES OF "REAL" DATA

C       write (*,*)  NT,I,DATAVALUE(NT,I),DELUSTORED(NT,I)


            IF (DATAVALUE(NT,I) .GE. 1.0D-8) THEN
             DIFF=DIFF+(-1.0D2*DABS((DATAVALUE(NT,I)
     C      -DELUSTORED(NT,I))/DATAVALUE(NT,I)))
            ENDIF
          END DO
        END DO

        FF(IIS)=DIFF

130     FORMAT (A,I2,A,D8.2,A)

        WRITE (*,130) '  DIFF',IIS,' =',DIFF,' %'
```

```
      CLOSE (6)

     END

C     ********************************************************
      SUBROUTINE  LINE15050
C     ********************************************************


C  parincl.for here

      DOUBLE PRECISION  CL(50), CU(50), FF(90), PPL(20)
      DOUBLE PRECISION  RR(90, 30), WPEN(50), XC(30), XX(90, 30)
      DOUBLE PRECISION  XXOLD(30)

      DOUBLE PRECISION ALPHAP, BETA, DELTA
      DOUBLE PRECISION Z, ZXC

      INTEGER IC, ICM, ICMI, IEV1
      INTEGER IEV2, IEV3, IIS, IOPT, IGAMMA
      INTEGER IT, ITMAX, IZ, IZRQ
      INTEGER IZXC, JC, JI, JJ
      INTEGER JZ, KOUNT,KK1, NALT, NC
      INTEGER NCMPLX, NFE, NINPS, NLEG
      INTEGER NRUN, NS, NSEG, NSTK
      INTEGER NUMTIMSTEPS, NV, NPRNT

      COMMON/AAA/ALPHAP, BETA, DELTA, IGAMMA
      COMMON/BBB/IC, ICM, ICMI, IEV1
      COMMON/CCC/IEV2, IEV3, IIS, IOPT
      COMMON/DDD/IT, ITMAX, IZ, IZRQ
      COMMON/EEE/IZXC, JC, JI, JJ
      COMMON/FFF/JZ, KOUNT,KK1, NALT, NC
      COMMON/GGG/NCMPLX, NFE, NINPS, NLEG
      COMMON/HHH/NRUN, NS, NSEG, NSTK
      COMMON/III/NV, Z, ZXC, NPRNT, NUMTIMSTEPS

      COMMON/JJJ/CL, CU, FF, PPL, RR, WPEN, XC, XX, XXOLD


100   FORMAT (A)
110   FORMAT (A,I3)
        WRITE (*,110) ' NUMBER OF VERTICES=',NV
15055 IF (NV .LE. NS) THEN
        WRITE (*,100) '  ERROR!-!-!-!  NV MUST EXCEED NS!!!'
        STOP
      ENDIF

15060 WRITE (*,110)' ITMAX=',ITMAX

15065 WRITE (*,110)' IPRINT=',IPRINT

15070 IF ((IPRINT .LT. 0) .OR. (IPRINT.GT.1)) THEN
        WRITE (*,110) ' IPRINT must be either 0 or 1. It is',IPRINT
        WRITE (*,100) ' This is a non-fatal error.'
      ENDIF

15072 IPEN = 1
      ALPHAP = 1.3
      IALPH = 1


15100  CONTINUE
15105      IALPH=1

15107 IF (IALPH .EQ. 1) WRITE(*,100) ' VARIABLE ALPHA'
```

```
15109 IF (IALPH .EQ. 0) THEN
        WRITE (*,100) ' STATIC ALPHA VALUES'
      ELSE
        IF (IALPH .NE. 1) WRITE (*,100) ' ALPHA PROBLEM AT LINE 15105'
      END IF

15120 IGAMMA = 4

99999 END


C  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
C  ######################################################################
C  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


        SUBROUTINE PARAVISCO2(NODCOUNT)


C  parincl.for here

        DOUBLE PRECISION  CL(50), CU(50), FF(90), PPL(20)
        DOUBLE PRECISION  RR(90, 30), WPEN(50), XC(30), XX(90, 30)
        DOUBLE PRECISION  XXOLD(30)

        DOUBLE PRECISION ALPHAP, BETA, DELTA
        DOUBLE PRECISION Z, ZXC

        INTEGER IC, ICM, ICMI, IEV1
        INTEGER IEV2, IEV3, IIS, IOPT, IGAMMA
        INTEGER IT, ITMAX, IZ, IZRQ
        INTEGER IZXC, JC, JI, JJ
        INTEGER JZ, KOUNT,KK1, NALT, NC
        INTEGER NCMPLX, NFE, NINPS, NLEG
        INTEGER NRUN, NS, NSEG, NSTK
        INTEGER NUMTIMSTEPS, NV, NPRNT

        COMMON/AAA/ALPHAP, BETA, DELTA, IGAMMA
        COMMON/BBB/IC, ICM, ICMI, IEV1
        COMMON/CCC/IEV2, IEV3, IIS, IOPT
        COMMON/DDD/IT, ITMAX, IZ, IZRQ
        COMMON/EEE/IZXC, JC, JI, JJ
        COMMON/FFF/JZ, KOUNT,KK1, NALT, NC
        COMMON/GGG/NCMPLX, NFE, NINPS, NLEG
        COMMON/HHH/NRUN, NS, NSEG, NSTK
        COMMON/III/NV, Z, ZXC, NPRNT, NUMTIMSTEPS

        COMMON/JJJ/CL, CU, FF, PPL, RR, WPEN, XC, XX, XXOLD

C =========================================================================


C                    DECLARE COMMON STATEMENTS

        DOUBLE PRECISION K(8,8)

        DOUBLE PRECISION INVRS(200,200)
        DOUBLE PRECISION KINV(200,200)

        COMMON/NUM/NUMELS
        COMMON/NNBLOCK/NUMNODES

        DOUBLE PRECISION R(200)
        COMMON/RBLOCK/R

        DOUBLE PRECISION COEFF(6)
        COMMON/COEFFBLOCK/COEFF
```

```
      COMMON/IDIMBLOCK/IDIM


      COMMON /NTS/NUMBEROFTIMESTEPS

      DOUBLE PRECISION TIMSTART, TIMINCR

      DOUBLE PRECISION NODX(200),NODY(200)
      COMMON/NODE/NODX,NODY

      INTEGER EI(325),EJ(325),EK(325),EM(325)
      COMMON/ELNODES/EI,EJ,EK,EM

      DOUBLE PRECISION MASTERK(200,200)
      COMMON/BIGK/MASTERK

      DOUBLE PRECISION A(3,3)
      COMMON/ABLOCK/A

      INTEGER NUMSPYK
      COMMON/NS/NUMSTART,NUMSTOP

      INTEGER IBDY(200)
      COMMON/IBINDX/IBDY

      DOUBLE PRECISION BVAL(200)
      COMMON/BOUNDVAL/BVAL

      INTEGER NUMBDY
      COMMON/BDY/NUMBDY

      INTEGER IDPLUS1
      COMMON/IDP/IDPLUS1

      DOUBLE PRECISION KPAST(200,200)
      COMMON/KPASTBLOCK/KPAST

      DOUBLE PRECISION DELU(200)


      INTEGER IROW(200),JCOL(200),JORD(200)
      COMMON/IJJ/IROW,JCOL,JORD

      DOUBLE PRECISION KZERO(200,200)
      COMMON/KZ/KZERO

      DOUBLE PRECISION Y(200)
      COMMON/WYE/Y

      DOUBLE PRECISION FINALK(200,200),LASTR(200)
      COMMON/ENDO/FINALK,LASTR

      SAVE

100   FORMAT(A)


      IF (NFE .EQ. 1) THEN

        WRITE (*,100)'     *********   PARAVISCO 2    ***********  '
        WRITE (*,100)' '
        WRITE (*,100)'       General Model Parameters for the'
        WRITE (*,100)'              Function Evaluation'

      ENDIF

C ____________________________________________________________________
```

```
C ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
C                              OPEN FILES
C
C  This block of commands opens, labels, and numbers the appropriate files for
C  use by VISCO1 with the exception of the series of files needed for [K(t)]
C  storage, as those are created as needed.

        OPEN (13, FILE='general_data', STATUS= 'old')

        REWIND 13

        OPEN (19, FILE='boundaries', STATUS= 'old')

        REWIND 19

c _______________________________________________________________________
C ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
C                         READ IN INITAL DATA
C This reads in some of the necessary parameters to operate some of
c the arrays used in this program.

        READ (13,*)NUMELS,NumberOfNodes
          NUMNODES=NumberOfNodes*2
            NODCOUNT=NUMNODES
          NUMN=NUMNODES
        READ (13,*)(COEFF(I),I=1,6)
        READ (13,*)NUMBEROFTIMESTEPS
        READ (13,*)NUMSTART,NUMSTOP
        READ (13,*)TIMSTART,TIMINCR

        CLOSE (13)

C  ************************************************************************
C  HERE'S THE SPLICE TO GET THE XX(IIS,n) VALUES INTO THE PROGRAM!!


        COEFF(2)=XX(IIS,1)
        COEFF(3)=XX(IIS,2)

        WRITE (*,100) ' COEFF 1          2            3'
        WRITE (*,*) COEFF(1), COEFF(2),COEFF(3)


C************************** END OF SPLICE  ******************************

        CALL MAKEARRAYS
101     FORMAT (A,I2)
130     FORMAT (10X,I2,13X,D12.3)
140     FORMAT (A40,I3)
150     FORMAT (10X,I3,10X,I3,10X,D12.6)

        IF ((NFE .EQ. 1) .OR. (NPRNT .GE. 1)) THEN

         WRITE (*,100) '          COEFFICIENT        VALUE'
         DO J=1,6
          WRITE (*,130)J,COEFF(J)
         END DO

        ENDIF

C       Read in the known boundary conditions
C       from the 'boundaries.dat file:
C       IDIR:  X=1   Y=0  FOR 2-D PROBLEMS

        READ (19,*) NUMBDY

        IF (NFE .EQ. 1) THEN
         WRITE (*,100)'  '
```

```
         WRITE (*,140) '  NUMBER OF KNOWN DISPLACEMENT VALUES:',NUMBDY
         WRITE (*,100)'  '
         WRITE (*,100) ' DIRECTION INDICATOR:  X=1   Y=0'
         WRITE (*,100) '        NODE       DIRECTION            VALUE'
        ENDIF

141     FORMAT (8X,I3,8X,I2,10X,D12.3)

        DO I=1,NUMBDY
         READ (19,*) IBNDX, IDIR, BVAL(I)
         IF (NFE .EQ. 1) WRITE (*,141) IBNDX,IDIR,BVAL(I)
         IBDY(I)=(IBNDX*2)-IDIR
        END DO

        CLOSE (19)

         WRITE (*,100)'  '
C *  Once arrays are ready, construct the series of [K(t)] values.
C    for all of the timesteps in the problem.
         IF (NFE .EQ. 1) WRITE (*,100)'  STORING [K] MATRICES FOR '

        DO 5,NT=0,NUMBEROFTIMESTEPS

         IF (NFE .EQ. 1) THEN
          WRITE (*,101)'      TIMESTEP #',NT
          WRITE (*,100)'  '
         ENDIF

          CALL MAKEA(NT,TIMINCR,TIMSTART)

C                              I       O
          CALL MAKESHAPES( MASTERK,NUMNODES)

C                              O     O       O
          CALL STOREMASTERK(NT,MASTERK,NUMNODES)


5       CONTINUE

C *  Solve the time-dependant problem, once all of the [K] values are
C    ready and stored.

        NNPLUS1=NUMNODES+1
        IDIM=NUMNODES-NUMBDY
        IDPLUS1=IDIM+1

        CALL SOLUTION(NUMNODES,NNPLUS1,KZERO,IDIM,IDPLUS1,FINALK,
     C  LASTR)

        END

C

******Note that the rest of the program is the same as VISCO2
```

Table 6. P21a.FOR

PROGRAM P21a.FOR

Note that for P21a.FOR the following modifications are made to the code:

```
C  ++++++++++++++++++++++++++++++++++

C  READ IN THE PARAMETRIC DATA

C  ++++++++++++++++++++++++++++++++++

        OPEN (14,FILE='Elastic',STATUS='old')
        REWIND 14

        READ (14,*) NV,ITMAX,BETA,IGAMMA


C  ****************************************************************
C  The number of parameters is fixed at 1 for the elastic problem.

        NUMPAR=1
        NC=1
        NS=1
        NUMTIMESTEPS=1

C  ****************************************************************
```

The other modification necessary is:

```
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
C                        READ IN INITAL DATA
C This reads in some of the necessary parameters to operate some of
c the arrays used in this program.

        READ (13,*)NUMELS,NumberOfNodes
          NUMNODES=NumberOfNodes*2
            NODCOUNT=NUMNODES
          NUMN=NUMNODES
        READ (13,*)(COEFF(I),I=1,6)
        READ (13,*)NUMBEROFTIMESTEPS
        NUMBEROFTIMESTEPS=1
        READ (13,*)NUMSTART,NUMSTOP
        READ (13,*)TIMSTART,TIMINCR

        CLOSE (13)

C  ******************************************************************
C  HERE'S THE SPLICE TO GET THE XX(IIS,n) VALUES INTO THE PROGRAM!!


        COEFF(1)=XX(IIS,1)
        COEFF(2)=0.0D0
        COEFF(3)=0.0D0

C************************** END OF SPLICE
```

Table 7. DUMERGE.FOR

```
      PROGRAM DUMERGE

      INTEGER IDUM(10)
      INTEGER NUMELS,NUMNODES,NTS

      DOUBLE PRECISION DDATA(10)
      DOUBLE PRECISION RDATA

      DOUBLE PRECISION SUMDATA(2000)

      CHARACTER*15 FILEA,FILEB

      OPEN (20, FILE='GENERAL_DATA', STATUS= 'OLD')
      REWIND (20)
        READ (20,*) NUMELS,NUMBEROFNODES
         NUMNODES=NUMBEROFNODES*2
        READ (20,*)(DDATA(K),K=1,6)
        READ (20,*)NTS
      CLOSE (20)

50    FORMAT(A)
51    FORMAT(A,I1)
52    FORMAT(A,I2)


C    CLEAN HOUSE

      DO KL=1,NUMNODES
        SUMDATA(KL)=0.0D0
      ENDDO

      DO II=1,NTS

        IF (II.LT.10) THEN
          WRITE (*,51) ' FILE NUMBER ',II
          WRITE (FILEA,51) 'DELUFILE',II
        ELSE
          WRITE (*,52) ' FILE NUMBER ',II
          WRITE (FILEA,52) 'DELUFILE',II
        ENDIF

        OPEN (22, FILE=FILEA, STATUS='OLD')

        DO  LL=1,NUMNODES
```

Table 7 (Cont'd).

```
    READ (22,*) RDATA
    SUMDATA(LL)=SUMDATA(LL)+RDATA
  ENDDO
ENDDO

  OPEN (24, FILE='SUMDELU',  STATUS='NEW')

    DO JJ=1,NUMNODES
     WRITE (24,*) SUMDATA(JJ)
    ENDDO
  WRITE (*,50) ' COPIED'
  CLOSE (22)
  CLOSE (24)

WRITE (*,50) '  ALL DONE CREATING FILE SUMDELU'
WRITE (*,52) '  FILES SUMMED:',NTS

END
```

Table 8. Documentation for programs.

The general form for the data and element files is:

****** For the <General_Data.dat> data file:**

```
NUMELS  NUMNODES
C1 C2 C3 C4 C5 C6
NUMBEROFTIMESTEPS
NUMSTART  NUMSTOP
TIMSTART  TIMINCR
```

```
****   For the <Element_Data.dat> Data File:

       1 NODX(1)  NODY(1)  <-----These are the x,y coordinates for the
       2 NODX(2)  NODY(2)        nodes.
       .   .        .
       .   .        .
       n NODX(n)  NODY(n)

       1 EI(1) EJ(1) EK(1) EM(1)  <--- These are the nodes which make up
       2 EI(2) EJ(2) EK(2) EM(2)        each element.  For a triangular
       .  .     .     .     .          element, the EM( ) value should be
       .  .     .     .     .          0 (Integer).
       n EI(n) EJ(n) EK(n) EM(n)
```

```
****   For the <Boundaries.dat> data file:

       NUMBDY

       IBNDX   IDIR    BVAL(IBNDX) \
       IBNDX   IDIR    BVAL(IBNDX)  |===> NUMBDY number of times
       IBNDX   IDIR    BVAL(IBNDX) /


       IBNDX   The node at which the boundary condition is known
                >>>>>>>THESE MUST BE IN SEQUENTIAL ORDER!!!<<<<<

       IDIR    Direction of Displacement:

                                        X=1
                                        Y=0

       BVAL(IBNDX)  Displacement Value

       Note that the index system obviates the need for sequential
       ordering of the bounday conditions  EXCEPT that due to a bug
       in the program, the "1" IDIR **MUST** precede the "0" IDIR for any
       node which has two known boundary conditions.
```

```
****   For the <rnumber_.dat>  file:

       ** Remember X = 1  and Y = 0

       NUMRVAL
```

Table 8 (Cont'd).

INDEX	IDIR	VAL

INDEX Node number
IDIR Direction code (see above)
VAL Force value

The File PAR_EST.DAT
required for using P21.F is:

```
NC  NS  NV  ITMAX  BETA  IGAMMA

NUMPAR

NUMTIMSTEPS

NPRNT

NI1  CL(NI1)  CU(NI1)  XX(1,NI1)
NI2  CL(NI2)  CU(NI2)  XX(1,NI2)
 .     .        .        .
 .     .        .        .
 .     .        .        .
 .     .        .        .
```

Where

```
NPRNT   Print level during function evaluation
      0 - Prints nothing after first evaluation
      1 - Prints new coefficients after first run
         2 -  (1) plus displacement values

NUMPAR   Number of parameters to be estimated
NC       Number of Constraints
NS       Number of Search Variables
NV       Number of Vertices
ITMAX    Maximum Number of Iterations
BETA     Model Parameter (Convergence Criteria)
IGAMMA   Model Parameter

NIn         Numerical Index for Parameter n
CL(NIn)     Lower Constraint for Parameter n
CU(NIn)     Upper Constraint for Parameter n
XX(1,NIn)   Best Guesstimate of Parameter n
```

The File Elastic.DAT
required for usingg p21a.F is:

NV ITMAX BETA IGAMMA

NPRNT

NI1 CL(NI1) CU(NI1) XX(1,NI1)

Where

NPRNT Print level during function evaluation
0 - Prints nothing after first evaluation
1 - Prints new coefficients after first run
2 - (1) plus displacement values

NV Number of Vertices
ITMAX Maximum Number of Iterations
BETA Model Parameter (Convergence Criteria)
IGAMMA Model Parameter

NIn Numerical Index for Parameter n (n=1 for all elastic data)
CL(NIn) Lower Constraint for Parameter n
CU(NIn) Upper Constraint for Parameter n
XX(1,NIn) Best Guesstimate of Parameter n

APPENDIX B

Material Deflection Data

Note that the symmetry of the sample was exploited to simplify the parameter estimation process. The 1/4 grid (Figure 31) deflection points are listed below.

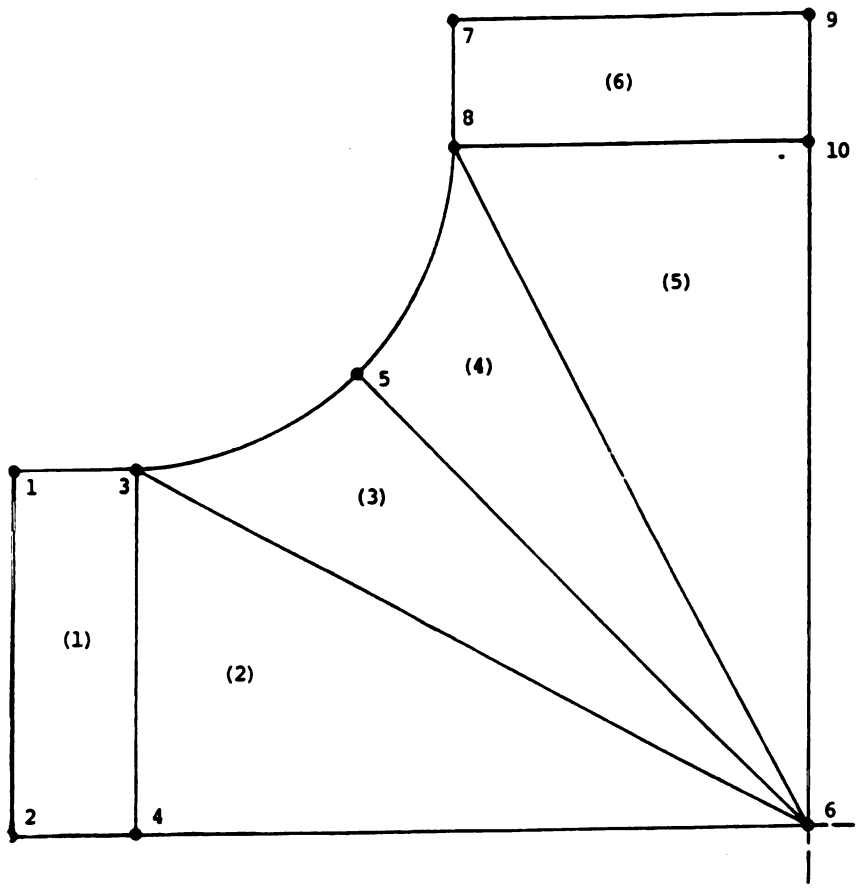


Figure 30. Finite Element Grid Used in Parameter Estimation Evaluation Model.

Table 9.

Change in Position of Indeces at 100% Loading.

Point:	xy:	Timestep 0	1	2	3	4	5	6
1	x	0	0.0656	0.1517	0.1862	0.2687	0.3173	0.3563
1	y	0	0.0000	0.0000	0.0000	0.0000	0.0000	0
2	x	0	0.0862	0.1243	0.2197	0.2829	0.3129	0.3563
2	y	0	0.0000	0.0000	0.0000	0.0000	0.0000	0
3	x	0	0.0796	0.0606	0.0967	0.1435	0.1895	0.1813
3	y	0	0.0409	0.0944	0.0947	0.1159	0.1432	0.1375
4	x	0	0.0437	0.0799	0.1113	0.1631	0.1638	0.1813
4	y	0	0.0000	0.0000	0.0000	0.0000	0.0000	0
5	x	0	0.0596	0.1020	0.1267	0.1837	0.1861	0.2013
5	y	0	0.0569	0.1100	0.1393	0.1919	0.2452	0.2516
6	x	0	0.0000	0.0000	0.0000	0.0000	0.0000	0
6	y	0	0.0000	0.0000	0.0000	0.0000	0.0000	0
7	x	0	0.0000	0.0000	0.0000	0.0000	0.0000	0
7	y	0	0.0701	0.1521	0.2013	0.2791	0.3210	0.3563
8	x	0	0.0243	0.0875	0.0902	0.1260	0.1187	0.1375
8	y	0	0.0570	0.0778	0.1167	0.1608	0.1959	0.1813
9	x	0	0.0000	0.0000	0.0000	0.0000	0.0000	0
9	y	0	0.0952	0.1558	0.2273	0.2412	0.3108	0.3563
10	x	0	0.0000	0.0000	0.0000	0.0000	0.0000	0
10	y	0	0.0730	0.0750	0.1033	0.1592	0.1917	0.1813

Position of Index Mark at 100% Loading.

Values in Inches.

Table 10.

Change in Position of Indeces at 82% Loading

Point:	xy	Timestep 0	1	2	3	4	5	6
1	x	0	0.0734	0.1393	0.1852	0.2199	0.2699	0.2873
1	y	0	0.0000	0.0000	0.0000	0.0000	0.0000	0
2	x	0	0.0922	0.1006	0.1500	0.2010	0.2467	0.2873
2	y	0	0.0000	0.0000	0.0000	0.0000	0.0000	0
3	x	0	0.0536	0.0792	0.1145	0.1302	0.1708	0.1632
3	y	0	0.0543	0.0655	0.0610	0.1075	0.0980	0.1021
4	x	0	0.0627	0.0813	0.0889	0.1497	0.1782	0.1632
4	y	0	0.0000	0.0000	0.0000	0.0000	0.0000	0
5	x	0	0.0426	0.0804	0.0976	0.1243	0.1672	0.1425
5	y	0	0.0811	0.0866	0.0982	0.1654	0.1750	0.1936
6	x	0	0.0000	0.0000	0.0000	0.0000	0.0000	0
6	y	0	0.0000	0.0000	0.0000	0.0000	0.0000	0
7	x	0	0.0000	0.0000	0.0000	0.0000	0.0000	0
7	y	0	0.0664	0.1441	0.1825	0.2301	0.2753	0.2873
8	x	0	0.0614	0.0503	0.0971	0.0692	0.1311	0.1021
8	y	0	0.0735	0.0924	0.1271	0.1366	0.1540	0.1632
9	x	0	0.0000	0.0000	0.0000	0.0000	0.0000	0
9	y	0	0.0518	0.1141	0.1773	0.1965	0.2508	0.2873
10	x	0	0.0000	0.0000	0.0000	0.0000	0.0000	0
10	y	0	0.0652	0.0852	0.1090	0.1526	0.1541	0.1632

Postion of Index Marks at 82% Loading.

All Values in Inches.

Table 11.

Change in Position of Indeces at 47% Loading.

Point:	xy:	Timestep 0	1	2	3	4	5	6
1	x	0	0.0533	0.1094	0.1036	0.1220	0.1513	0.1793
1	y	0	0.0000	0.0000	0.0000	0.0000	0.0000	0
2	x	0	0.0400	0.0620	0.0976	0.1415	0.1764	0.1793
2	y	0	0.0000	0.0000	0.0000	0.0000	0.0000	0
3	x	0	0.0514	0.0729	0.0879	0.0694	0.1220	0.1031
3	y	0	0.0439	0.0309	0.0839	0.0638	0.1049	0.0731
4	x	0	0.0199	0.0767	0.0628	0.0994	0.0975	0.1031
4	y	0	0.0000	0.0000	0.0000	0.0000	0.0000	0
5	x	0	0.0253	0.0582	0.0540	0.1065	0.1175	0.0989
5	y	0	0.0566	0.0505	0.0711	0.0887	0.1002	0.1077
6	x	0	0.0000	0.0000	0.0000	0.0000	0.0000	0
6	y	0	0.0000	0.0000	0.0000	0.0000	0.0000	0
7	x	0	0.0000	0.0000	0.0000	0.0000	0.0000	0
7	y	0	0.0740	0.1044	0.1224	0.1585	0.1966	0.1793
8	x	0	0.0502	0.0492	0.0844	0.0958	0.0903	0.0731
8	y	0	0.0339	0.0463	0.0904	0.1071	0.1020	0.1031
9	x	0	0.0000	0.0000	0.0000	0.0000	0.0000	0
9	y	0	0.0340	0.1073	0.1119	0.1299	0.1570	0.1793
10	x	0	0.0000	0.0000	0.0000	0.0000	0.0000	0
10	y	0	0.0233	0.0622	0.0773	0.1146	0.1190	0.1031

Position of Index Marks at 47% Loading.

All Values in Inches.

APPENDIX C

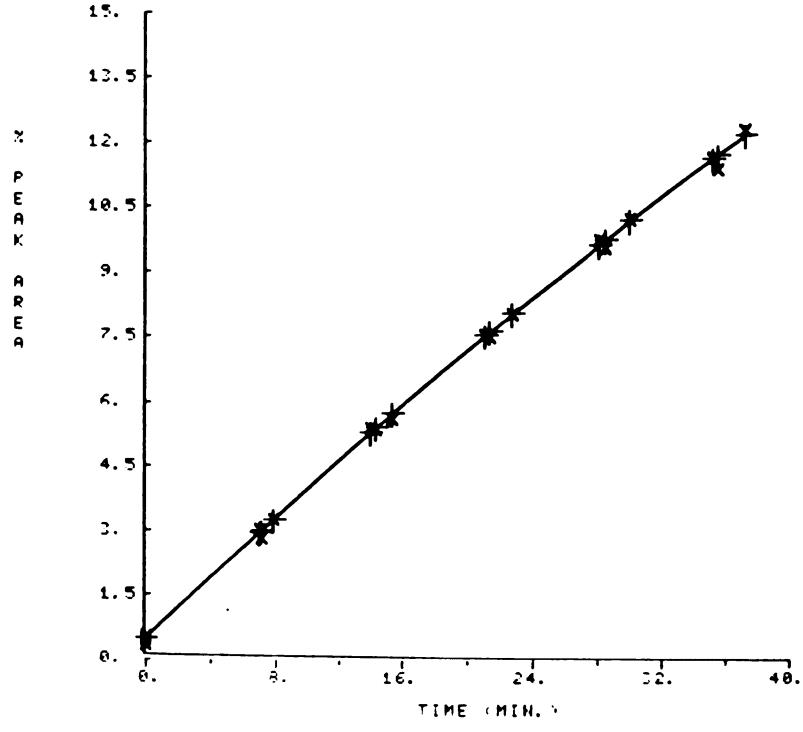
Permeation Data

Table 12. Summary of Permeation Data at Various Loading Levels.

Loading:	0 0	2794 1930	2303 1583	1323 Psi. 906 N/cm²	
Percent of Maximum Load:	0%	100%	82%	47%	
Quadratic Coefficients:					
A:	-0.00162	-0.00133	-0.00160	-0.00151	
B:	0.33586	0.36331	0.35602	0.34411	
C:	0.35710	0.45182	0.55308	0.45553	
Correlation Coefficent:	0.9965	0.9973	0.9947	0.9885	
Coefficient of Determination:	0.9931	0.9970	0.9947	0.9885	
Standard Time:	20.4	18.3	18.7	19.7	min.
dC/dt at Standard Time:	0.2681	0.3146	0.2962	0.2842	%/min.
Permeation Rate:[1]	0.9411	1.1045	1.0396	0.9977	
Change:	0	+17.36%	+10.47%	+6.02%	

[1] $Cm^3/min/m^2/mm/Atm$

08/11/92



COEFFICIENTS OF LEAST SQUARES FIT TO A QUADRATIC EQUATION

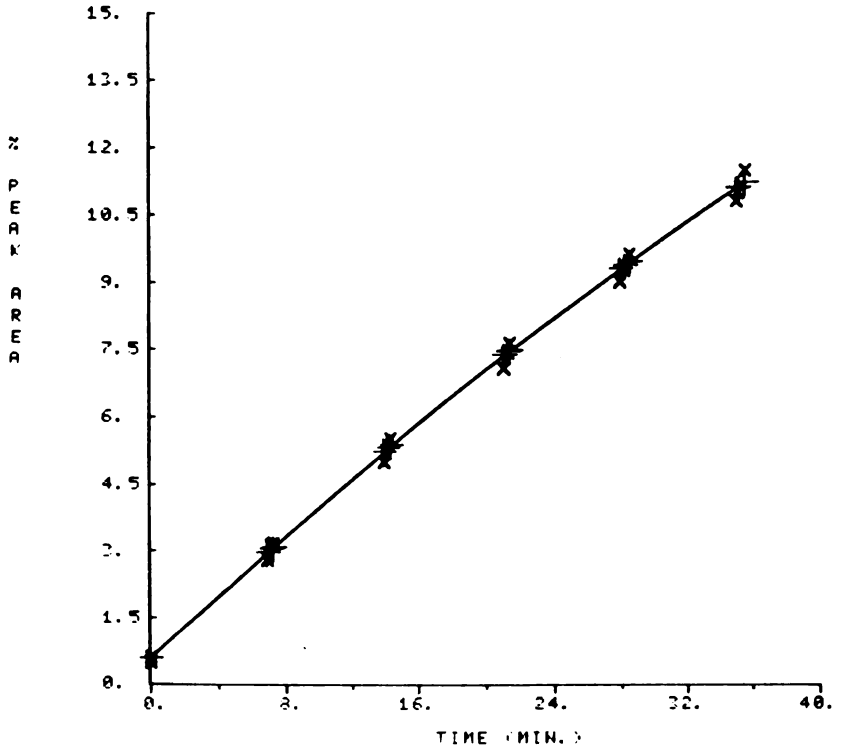
KA= -.0013309 KB= 0.2633142 KC= 0.4518184

CORRELATION COEFFICIENT OF X-Y PAIRS = 0.9985133

COEFFICIENT OF DETERMINATION = 0.9970288

Figure 31. Change of CO_2 Concentration in Permeation Cell Versus Time at 100% Loading.

08/11/92



COEFFICIENTS OF LEAST SQUARES FIT TO A QUADRATIC EQUATION

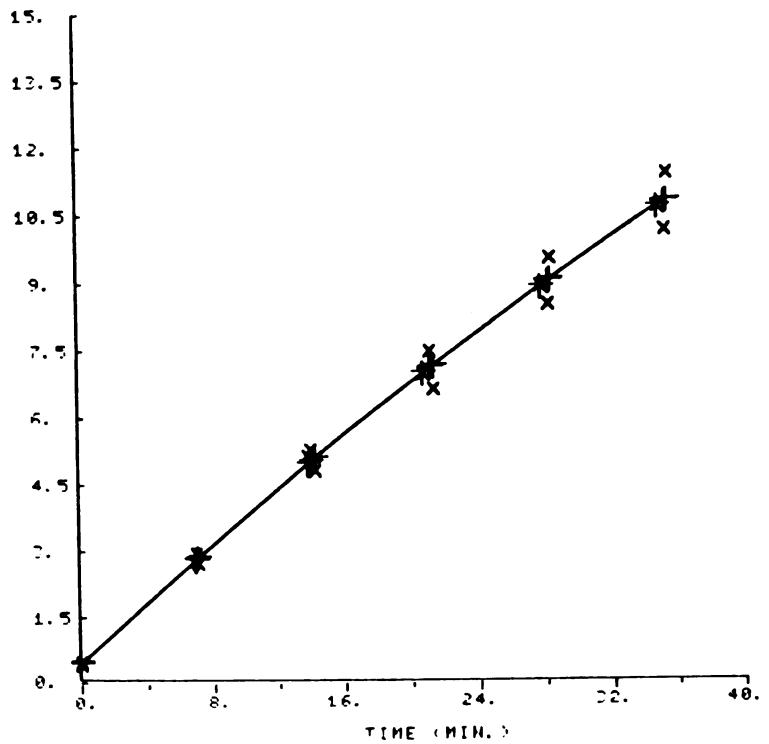
KA= -.0016022 KB= 0.3560284 KC= 0.5530775

CORRELATION COEFFICIENT OF X-Y PAIRS = 0.9973468

COEFFICIENT OF DETERMINATION = 0.9947007

Figure 32. Change of CO_2 Concentration in Permeation Cell Versus Time at 82% Loading.

08/13/92



COEFFICIENTS OF LEAST SQUARES FIT TO A QUADRATIC EQUATION

KA= -.0015185 KB= 0.344115 KC= 0.4555308

CORRELATION COEFFICIENT OF X-Y PAIRS = 0.9942745

COEFFICIENT OF DETERMINATION = 0.9885818

Figure 33. Change of CO_2 Concentration in Permeation Cell Versus Time at 47% Loading.

BIBLIOGRAPHY

Allcock, H.R. and Lampe, F.W. 1981. Contemporary Polymer Chemistry. Prentiss-Hall: Englewood Cliffs, NJ. p. 436-445.

Augl, J.M. and Land, D.J. 1985. A Numerical Approach for Obtaining Nonlinear Viscosity Parameters of Polymeric Materials and Composites. J. App. Polymer Sci. 30:4203.

Boresi A. P. and Sidebottom O. M. 1985. Advanced Mechanics of Materials. John Wiley & Sons: New York. p.644-647.

Box, M. J., Davies, D., and Swann, W.H. 1969. Non-Linear Optimization Techniques. ICI Monograph No.5. Oliver & Boyd: Edinburgh UK.

Burke, P.E. and Weatherly, G.C. 1987. Uniaxial Roll-Drawing of Isotactic Polypropylene Sheet. Polymer Engineering and Science 27:518.

Chen, T. W. 1991. Superposition Principle in Small Angle Light Scattering at High Frequency. J. App. Physics. 70(2):1031.

Christensen, R. M. 1967. Theory of Viscoelasticity. Academic Press: New York. p.1-32.

Cook, R. D. and Young, W. C. 1984. Advanced Mechanics of Materials. Macmillan Pub. Co.: New York. p.508.

DeBaerdemaker, J. G. 1975. Experimental and Numerical Techniques Related to the Stress Analysis of Apples Under Static Loads. Ph.D. Dissertation, Michigan State University.

El-Hibri, M.J. and Paul, D.R. 1985. Effects of Uniaxial Drawing and Heat-Treatment on Gas Sorption and Transport in PVC. J. App. Polymer Sci. 30:3649.

El-Hibri, M.J. and Paul, D.R. 1986. Gas Transport in Poly(vinylidene Fluoride): Effects of Uniaxial Drawing and Processing Temperature. J. App. Polymer Sci. 31:2533.

Ferry, J.D. 1980. Viscoelastic Properties of Polymers, 3d ed. J. Wiley & Sons: New York. p.130-167.

Flugge, W. 1967. Viscoelasticity. Blaisdell Pub. Co.: Waltham MA. p.1-21.

Gere, R. and Timoshenko, B. 1984. Mechanics of Materials. PWS Engineering: Boston p.304.

Hashimoto, T. et al 1976. Superstructure of High-Density Polyethylene Film Crystallized From Stressed Polymer Melts as Observed by Small-Angle Light Scattering. Polymer 17:1075.

Hashimoto, T. et al 1976. Deformation Mechanisms of Hard Elastic Polyethylene Films. Polymer 17:1063.

Hendra, P.J., Taylor, M.A., and Willis, H.A. 1985. Plastic Deformation in Linear Polyethylene. Polymer. 26:1501.

Howells et al. 1991. Enhanced Effects with Scanning Force Microscopy. J. App. Physics 69(10):7330.

Jang, B.Z. et al., 1985. Crazing in Polyethylene. Polymer Engineering and Science 25(2):98-101.

Jehle, U. and Mlejvek, H.P. 1990. Application and Implementation of Approximate Explicit Models in Optimum Structural Design. Computer methods in Applied Mechanics and Engineering 83(100):33.

Klomparens, K.L. et al. 1986. Procedures of Transmission on Scanning Electron Microscopy for Biological and Medical Science. Ladd Research Industries: Burlington VT. p.112-114

Kuester, J.L. and Mize, J.H. 1973. Optimization Techniques With Fortran. McGraw-Hill: New York.

Lyon, R.E. and Farris, R.J. 1984. Thermodynamic Behavior of Solid Polymers in Uniaxial Deformation. Polymer Engineering and Science. 24:908.

Manetsch, T.J. 1989. Towards Efficient Global Optimization in Large Dynamic Systems-The Adaptive Complex Method. Department of Systems Science, Michigan State University, East Lansing MI, 48826.

Morris, S.A. and Lee, J. 1987. The Effects of Tensile Strain on the Performance of High-Barrier Retortable Pouch Material. J. Packaging Technology. 1:194.

NASA MSC-21157. Keeping Blood Platelets Healthy During Storage. Reprints available from NASA STI Facility, Manager TU Division, POB 8757, Baltimore MD 21240-9985.

O'Brian, K.C., Koros,W.J., and Husk, G.R. 1987. Influence of Casting and Curing Conditions on Gas Sorption and Transport in Polyimide Films. Polymer Engineering and Science. 27:211.

Pan, S.J., Tang, H.I., Hiltner, A., Baer, E. 1987. Biaxial Orientation of Polypropylene by Hydrostatic Solid State Extrusion Part II: Morphology and Properties. Polymer Engineering and Science. 27:869.

Paulsen, C.W. 1992. Product Design on a P.C.. Sound & Vibration, January. p 58.

Paulos, J.P. and Thomas, E.L. 1980. Effect of Postdrawing on the Permeability of Gases in Blown Polyethylene Film. J. App. Polymer Sci. 25:15.

Reiss et al. 1991. Scanning Tunnelling Microscopy on Rough Surfaces--Quantitative Image Analysis. J. App. Physics. 70(1):523.

Rhodes, M.B. and Stein, R.S. 1969. Scattering of Light from Assemblies of Oriented Rods. J. Polymer Science: Part A-2. 7:1539.

Roulin-Maloney A. C. ed. 1991. Fractography and Failure Mechanisms of Polymers and Composites. Elsevier Applied Scientific Pub.

Seguela, R. and Rietsch, F. 1986. Tensile Drawing Behavior of a Linear Low-Density Polyethylene: Changes in Physical and Mechanical properties. Polymer. 27:532.

Segerlind, L. J. 1984. Applied Finite Element Analysis. John Wiley & Sons: New York. p. 1-3.

Schultz, J. M. 1984. Microstructural Aspects of Failure in Semicrystalline Polymers. Polymer Engineering and Science. 24:770.

Smith, T. L. and Adam, R. E. 1981. Effect of Tensile Deformations on Gas Transport in Glassy Polymer Films. Polymer. 22:299.

Speigel, M.R. 1965. Schaum's Outline of Theory and Practice of Laplace Transforms. Schaum's Outline Series. McGraw-Hill: New York. p. 1-35.

Stein, R.S. and Rhodes, M.B. 1960. Photographic Light Scattering by Polyethylene Films. J. App. Physics 31:1873.

Stein, R.S. and Rhodes, M.B. 1961. Light Scattering Study of the Annealing of Drawn Polyethylene. J. App. Physics 32:11.

Ugrual, A. C. 1981. Stresses in Plates and Shells. McGraw-Hill: New York. p.106-139.

Williams, J.L. and Peterlin, A. 1971. Journal of Polymer Science, Polymer Physics Edition. 9:1483.

Yasuda, H. and Peterlin, A. 1974. Gas Permeability of Deformed Polyethylene Films. J. App. Polymer Sci. 18:531.