An Affirmative Action/Equal Opportunity Employer

This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor any agency thereof, nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof.

LA--11993-MS

DE91 006791

*CHARADE: A Characteristic Code
for Calculating Rate-Dependent
Shock-Wave Response*

*J. N. Johnson
D. L. Tonks*

MASTER

Los Alamos Los Alamos National Laboratory
Los Alamos, New Mexico 87545

DISTRIBUTION OF THIS DOCUMENT IS UNLIMITED

CHARADE: A CHARACTERISTIC CODE
FOR CALCULATING RATE-DEPENDENT
SHOCK-WAVE RESPONSE

by

J. N. Johnson and D. L. Tonks

ABSTRACT

In this report we apply spatially one-dimensional methods and simple shock-tracking techniques to the solution of rate-dependent material response under flat-plate-impact conditions. This method of solution eliminates potential confusion of material dissipation with artificial dissipative effects inherent in finite-difference codes, and thus lends itself to accurate calculation of elastic-plastic deformation, shock-to-detonation transition in solid explosives, and shock-induced structural phase transformation. Equations are presented for rate-dependent thermoelastic-plastic deformation (or [100] planar shock-wave propagation in materials of cubic symmetry (or higher)). Specific numerical calculations are presented for polycrystalline copper using the mechanical threshold stress model of Follansbee and Kocks with transition to dislocation drag. A listing of the CHARADE (for characteristic rate dependence) code and sample input deck are given.

I. INTRODUCTION

The detailed calculation of rate-dependent material response to plate-impact or explosive loading often requires special consideration. Finite-difference (FD) calculations usually employ an artificial viscous stress¹ to limit the maximum rate of compression. This does not mean that FD codes cannot be used to study dynamic material response, but it does mean that additional care must be exercised to ensure that artificial viscosity does not influence calculated material behavior or result in an incorrect interpretation of experimental data.

In a rate-dependent characteristics code, dissipation is provided naturally by the rate-dependent material response itself, provided special attention is given to the elastic precursor (or leading shock wave in the case of explosive initiation or polymorphic phase transition). For near-discontinuous loading conditions, the main deformational (shock) wave is preceded by a precursor with a time-dependent amplitude. The risetime of the precursor is controlled by dissipative processes with characteristic times that are much shorter than those in the main shock wave, and thus the precursor can be approximated mathematically as a discontinuity. While it is recognized that true discontinuous changes in stress, density, particle velocity, etc. do not exist in nature, it is acceptable and to our advantage to treat them as such when risetimes are orders of magnitude less than can be resolved by available experimental techniques.

The amplitude of the elastic precursor (or leading shock) is controlled by the shock-change equation, which involves the rate of material dissipation and stress gradient (or particle acceleration)

evaluated at the precursor front. Derivation of the shock-change equation and the conditions under which the leading wave can be treated as a mathematical discontinuity are presented in detail in Section V and Appendix A.

The general thermoelastic-plastic response of solids is represented by the analysis of Wallace.^{2,3} This involves knowledge of second- and third-order adiabatic elastic moduli, isotropic Grüneisen coefficient, and specific heat at constant volume. Expressions are presented for the case of shock compression in the [100] direction of cubic crystals. The expressions reduce to the results originally presented by Wallace for initially isotropic solids.

For the purpose of specific illustration, the material model that is used in calculations presented here is the Mechanical Threshold Stress (MTS) model of rate-dependent plastic flow with transition to dislocation drag^{4,5} in copper.

II. CONSERVATION LAWS AND GENERAL MATERIAL RATE DEPENDENCE

For one-dimensional deformation in terms of Eulerian spatial coordinate x , the conservation laws can be written, in the absence of thermal conduction, as (see reference 6 and Appendix A)

$$\dot{\rho} + \rho(\partial u / \partial x)_t = 0 \quad , \quad (1)$$

$$\rho \dot{u} + (\partial \sigma / \partial x)_t = 0 \quad , \quad (2)$$

$$\rho \dot{e} - \sigma \dot{\rho} / \rho = 0 \quad , \quad (3)$$

where ρ is the material density, u is the x component of particle velocity, σ is the longitudinal component of stress (positive in compression), and e

is the internal energy per unit mass. A dot over a variable indicates differentiation following a material particle:

$$(\dot{}) = d/dt = (\partial/\partial t)_x + u(\partial/\partial x)_t . \quad (4)$$

The material response is expressed in the form given by Wallace²:

$$\dot{\tau}_{ij} - B_{ijkl}\dot{\epsilon}_{kl} = - (B_{ijkl} + \gamma_{ij}\tau_{kl})\dot{\epsilon}_{kl}^P , \quad (5)$$

$$\dot{T} + T\gamma_{ij}\dot{\epsilon}_{ij} = (T\gamma_{ij} + \tau_{ij}/\rho C_\eta)\dot{\epsilon}_{ij}^P , \quad (6)$$

where τ_{ij} is the general stress tensor (positive in tension), $\dot{\epsilon}_{kl}$ is the strain-rate tensor

$$\dot{\epsilon}_{kl} = (1/2)(\partial u_k/\partial x_l + \partial u_l/\partial x_k) , \quad (7)$$

B_{ijkl} are the adiabatic stress-strain coefficients related to the elastic moduli and the current stress state, γ_{ij} is the Grüneisen tensor, T is the temperature, and C_η is the specific heat at constant configuration.

For plane-wave propagation in the x_1 direction, $\sigma = -\tau_{11}$ and $x = x_1$. In addition to the longitudinal compressive stress component σ , we calculate the shear stress τ defined in general as

$$\tau^2 = (3/8)s_{ij}s_{ij} , \quad (8)$$

where s_{ij} is the deviatoric stress tensor whose components are defined by

$$s_{ij} = r_{ij} - (r_{kk}/3)\delta_{ij} . \quad (9)$$

In the case of plane-wave propagation in the x_1 direction

$$s_{11} = - (4/3)r , \quad (10)$$

$$s_{22} = s_{33} = (2/3)r , \quad (11)$$

and hence the transverse stress components are given by

$$r_{22} = r_{33} = r_{11} + 2r . \quad (12)$$

The plastic strain-rate components are expressed in terms of the scalar function $\dot{\psi}$:

$$\dot{\epsilon}_{11}^p = -\dot{\psi} , \quad (13)$$

$$\dot{\epsilon}_{22}^p = \dot{\epsilon}_{33}^p = \dot{\psi}/2 , \quad (14)$$

with the result that the plastic strain gives no contribution to the volume strain, i.e.,

$$\dot{\epsilon}_{kk}^p = 0 . \quad (15)$$

In the special case of cubic symmetry (or higher) and plane-wave propagation in the $x = x_1$ direction, the adiabatic stress-strain coefficients B_{ijkl} have the following symmetry (in Voigt notation)

$$\left[\begin{array}{ccc|ccc} B_{11} & B_{12} & B_{12} & & & \\ B_{21} & B_{22} & B_{23} & & 0 & \\ B_{21} & B_{23} & B_{22} & & & \\ \hline & & & B_{44} & 0 & 0 \\ & & & 0 & B_{66} & 0 \\ & & & 0 & 0 & B_{66} \end{array} \right] \quad (16)$$

In this case Eqs. (5) and (6) become (with $\gamma_{ij} = \gamma\delta_{ij}$)

$$\dot{\sigma} - B_{11}(\dot{\rho}/\rho) = - (B_{11} - B_{12} - 2\gamma\tau)\dot{\psi} \quad , \quad (17)$$

$$\dot{\tau} - \frac{1}{2}(B_{11} - B_{21})(\dot{\rho}/\rho) = - \frac{1}{4}(2B_{11} - 2B_{21} - 2B_{12} + B_{22} + B_{23})\dot{\psi} \quad , \quad (18)$$

$$\dot{T} - \gamma T(\dot{\rho}/\rho) = (2\tau/\rho C_{\eta})\dot{\psi} \quad . \quad (19)$$

The characteristic solution is obtained most conveniently in terms of a Lagrangian spatial coordinate X that moves along with the material. The relationship between Eulerian and Lagrangian coordinates is formally expressed as

$$x = x(X, t) \quad , \quad (20)$$

where

$$u = \partial_t x = (\partial x / \partial t)_X \quad , \quad (21)$$

$$\rho_0/\rho = \partial_X x = (\partial x / \partial X)_t \quad , \quad (22)$$

$$(\partial X / \partial t)_x = -\rho u / \rho_0 \quad , \quad (23)$$

$$(\partial X / \partial x)_t = \rho / \rho_0 \quad . \quad (24)$$

The transformation between the Eulerian and Lagrangian descriptions is expressed as (where φ is a general function of x , t or X , t)

$$\partial_t \varphi = (\partial \varphi / \partial t)_x + u(\partial \varphi / \partial x)_t = \dot{\varphi} , \quad (25)$$

$$\partial_X \varphi = (\rho_0 / \rho)(\partial \varphi / \partial x)_t , \quad (26)$$

$$(\partial \varphi / \partial t)_x = \partial_t \varphi - (\rho u / \rho_0) \partial_X \varphi , \quad (27)$$

$$(\partial \varphi / \partial x)_t = (\rho / \rho_0) \partial_X \varphi . \quad (28)$$

In Lagrangian coordinates, Eqs. (1)-(3) become

$$\rho_0 \partial_t v - \partial_X u = 0 , \quad (29)$$

$$\rho_0 \partial_t u + \partial_X \sigma = 0 , \quad (30)$$

$$\partial_t e + \sigma \partial_t v = 0 , \quad (31)$$

where the specific volume v replaces the density as one of the dependent variables

$$v = \frac{1}{\rho} . \quad (32)$$

Equations (17)-(19) become

$$\partial_t \sigma + (\rho c)^2 \partial_t v = F , \quad (33)$$

$$\begin{aligned} \partial_t r + \frac{1}{2}(B_{11} - B_{21}) \rho \partial_t v = \\ - \frac{1}{4}(2B_{11} - 2B_{21} - 2B_{12} + B_{22} + B_{23}) \partial_t \psi , \end{aligned} \quad (34)$$

$$\partial_t T + \gamma T \rho \partial_t v = (2r / \rho C_\eta) \partial_t \psi , \quad (35)$$

where

$$\rho c^2 = B_{11} \quad , \quad (36)$$

and

$$F = -(B_{11} - B_{12} - 2\gamma\tau)\partial_t\psi \quad . \quad (37)$$

Here c is the Eulerian longitudinal sound speed and F is called the rate function, or relaxation function in the case of elastic-plastic flow. If Eq. (33) is combined with Eq. (29) we obtain

$$\partial_t\sigma + \rho_0 C_L^2 \partial_X u = F \quad , \quad (38)$$

where

$$C_L = \rho c / \rho_0 \quad (39)$$

is the Lagrangian longitudinal sound speed.

The Lagrangian characteristic equations are obtained from Eqs. (30) and (38): Eq. (30) is multiplied by C_L and alternatively added and subtracted from Eq. (38) to give

$$[\partial_t\sigma \pm C_L \partial_X \sigma] \pm \rho_0 C_L [\partial_t u \pm C_L \partial_X u] = F \quad . \quad (40)$$

This means that

$$d\sigma \pm \rho_0 C_L du = F dt \text{ on } dX/dt = \pm C_L \quad (41)$$

on $C^\pm$ characteristic paths.

Fig. 1. Characteristic paths and particle path ($X = X'$) in X, t plane.

If values of σ , u pairs are known at two different points, as shown in Figure 1, new values of σ and u can be obtained at the intersection of the C^+ and C^- characteristics coming from the two initial points L and R :

$$C^+: (\sigma - \sigma_L) + \rho_0 C_{LL}(u - u_L) = F_L dt_L, \quad dX/dt = C_{LL}, \quad (42)$$

$$C^-: (\sigma - \sigma_R) - \rho_0 C_{LR}(u - u_R) = F_R dt_R, \quad dX/dt = -C_{LR}, \quad (43)$$

where F_L and C_{LL} are evaluated at some point along C^+ and F_R and C_{LR} are evaluated at some point along C^- . Simultaneous solution of Eqs. (42) and (43) give

$$\sigma = \frac{C_{LR} \left[\sigma_L + \rho_0 C_{LL} u_L + F_L dt_L \right] + C_{LL} \left[\sigma_R - \rho_0 C_{LR} u_R + F_R dt_R \right]}{C_{LR} + C_{LL}}, \quad (44)$$

$$u = \frac{\left[\sigma_L + \rho_0 C_{LL} u_L + F_L dt_L \right] - \left[\sigma_R - \rho_0 C_{LR} u_R + F_R dt_R \right]}{\rho_0 \left[C_{LR} + C_{LL} \right]}, \quad (45)$$

at the Lagrangian position X' and time t' (Fig. 1).

The specific volume and specific internal energy, can be integrated along the $X = X'$ particle path from point I to the intersection of the C^+ , C^- characteristic paths (Fig. 1). The equations to be integrated are obtained from Eqs. (31) and (33):

$$\partial_t v = (F - \partial_t \sigma) / (\rho c)^2, \quad (46)$$

$$\partial_t e = -\sigma \partial_t v. \quad (47)$$

To obtain the shear stress τ and temperature T , we have to specify the $\partial_t \psi$ - $\dot{\psi}$ function. In general the scalar plastic strain rate $\dot{\psi}$ depends on τ , T , v , e , ψ , and other internal-state variables in the material constitutive description. Let these internal-state variables be designated by ψ_i ($i = 1, 2, 3, \dots, N$) and assume their evolution is governed by the first-order set of ordinary differential equations:

$$\partial_t \psi_i = \dot{\psi}_i(\tau, T, v, e, \psi, \{\psi_j\}) \quad (48)$$

where $\{\psi_j\}$ indicates the set of all ψ_j ($j = 1, 2, 3, \dots, N$).

III. STRESS-STRAIN COEFFICIENTS

Integration of Eqs. (17)-(19) requires knowledge of the stress-strain

coefficients B_{11} , B_{12} , B_{21} , B_{22} , and B_{23} .

The relationships between the stress-strain coefficients B_{ijkl} and the adiabatic elastic constants C_{ijkl} are given by Wallace.^{2,3,7} The adiabatic (constant entropy S) elastic constants are the expansion coefficients for the internal energy function

$$U(\underline{x}, S) = U(\underline{X}, \eta_{ij}, S), \quad (49)$$

where $\underline{X}$ represents an arbitrary initial configuration, $\underline{x}$ represents the current configuration, and η_{ij} is given by

$$\eta_{ij} = \frac{1}{2} \left(\frac{\partial x_k}{\partial X_i} \frac{\partial x_k}{\partial X_j} - \delta_{ij} \right). \quad (50)$$

Thus

$$\begin{aligned} \rho_1 U(\underline{X}, \eta_{ij}, S) &= \rho_1 U(\underline{X}, 0, S) \\ &+ C_{ij} \eta_{ij} + \frac{1}{2} C_{ijkl} \eta_{ij} \eta_{kl} + \frac{1}{6} C_{ijklmn} \eta_{ij} \eta_{kl} \eta_{mn} + \dots, \end{aligned} \quad (51)$$

where ρ_1 is the density in the arbitrary initial state $\underline{X}$. In reference 7 (Eq. 6.10) it is shown that

$$B_{ijkl} = C_{ijkl} + \frac{1}{2} (\tau_{ik} \delta_{jl} + \tau_{il} \delta_{jk} + \tau_{jk} \delta_{il} + \tau_{jl} \delta_{ik} - 2\tau_{ij} \delta_{kl}). \quad (52)$$

The elastic moduli C_{ijkl} refer to a general anisotropic state of elastic strain $\epsilon_{ij}^e = \epsilon_{ij} - \epsilon_{ij}^p$. We are interested in expressing these moduli in terms of those in an arbitrary initial isotropic configuration. That is,

we carry out an adiabatic elastic strain from the isotropic state (V, S) to the state $(V + \epsilon_{ij}^e, S)$. The adiabatic strain expansion of the second-order elastic constants is given by Eq. (15.5) of reference 2:

$$C_{ijkl} = \bar{C}_{ijkl} + \left[\bar{C}_{ijkm} \epsilon_{lm}^e + \bar{C}_{ijml} \epsilon_{km}^e + \bar{C}_{imkl} \epsilon_{jm}^e + \bar{C}_{mjkl} \epsilon_{im}^e - \bar{C}_{ijk} \epsilon_{mm}^e + \bar{C}_{ijk2mn} \epsilon_{mn}^e \right] \quad (53)$$

Elastic constants in the initial isotropic state are denoted by overhead bars. Here we take the initial state to be at zero pressure. In this state the material is assumed to have cubic or higher symmetry. In Wallace's² application to uniaxial compression, the initial state was assumed to be isotropic. For cubic symmetry the independent nonzero elastic constants are (in Voigt notation)

$$\text{2nd order: } \bar{C}_{11}, \bar{C}_{12}, \bar{C}_{44}$$

$$\text{3rd order: } \bar{C}_{111}, \bar{C}_{112}, \bar{C}_{123}, \bar{C}_{144}, \bar{C}_{155}, \bar{C}_{456}$$

with symmetry relations given by Brugger⁸ in Tables II (symmetry C) and III (symmetry CI).

For planar shock compression in the x_1 direction, we have two independent elastic strain components: ϵ_{11}^e and $\epsilon_{22}^e = \epsilon_{33}^e$. After a great deal of algebra, we finally obtain expressions for the pertinent stress-strain coefficients:

$$B_{11} = C_{11} + r_{11} - \bar{C}_{11} + (4\bar{C}_{11} + \bar{C}_{111})\epsilon_{11}^e + (2\bar{C}_{12} - 2\bar{C}_{11} + 2\bar{C}_{112})\epsilon_{22}^e, \quad (54)$$

$$B_{12} = C_{12} - r_{11} - \bar{C}_{12} + (\bar{C}_{12} - \bar{C}_{11} + \bar{C}_{112})\epsilon_{11}^e + (\bar{C}_{112} + \bar{C}_{123} - 2\bar{C}_{12})\epsilon_{22}^e, \quad (55)$$

$$B_{21} = C_{12} - r_{22} - \bar{C}_{12} + \bar{C}_{112}\epsilon_{11}^e + (\bar{C}_{112} + \bar{C}_{123} - \bar{C}_{11} - \bar{C}_{12})\epsilon_{22}^e, \quad (56)$$

$$B_{22} = C_{22} + r_{22} - \bar{C}_{11} + (\bar{C}_{112} - \bar{C}_{11} + \bar{C}_{12})\epsilon_{11}^e + (3\bar{C}_{11} + \bar{C}_{111} + \bar{C}_{112} + \bar{C}_{12})\epsilon_{22}^e, \quad (57)$$

$$B_{23} = C_{23} - r_{22} - \bar{C}_{12} + (\bar{C}_{123} - 2\bar{C}_{12})\epsilon_{11}^e + (\bar{C}_{12} + 2\bar{C}_{112} - \bar{C}_{11})\epsilon_{22}^e, \quad (58)$$

where we have used the relationships (correct to first order in the elastic strains)

$$r_{11} = \bar{C}_{11}\epsilon_{11}^e + 2\bar{C}_{12}\epsilon_{22}^e, \quad (59)$$

$$r_{22} = \bar{C}_{12}\epsilon_{11}^e + (\bar{C}_{11} + \bar{C}_{12})\epsilon_{22}^e. \quad (60)$$

The dependent strain variables that are used in our calculation are not ϵ_{11}^e and ϵ_{22}^e , but rather the total strain ϵ_{11} and the plastic strain ψ . With the relationships

$$\epsilon_{11}^e = \epsilon_{11} + \psi, \quad (61)$$

$$\epsilon_{22}^e = -\psi/2, \quad (62)$$

Eqs. (54)-(58) become

$$\begin{aligned} B_{11} = & \bar{C}_{11} + (4\bar{C}_{11} + \bar{C}_{111})\epsilon_{11} \\ & + (5\bar{C}_{11} - \bar{C}_{12} + \bar{C}_{111} - \bar{C}_{112})\psi, \end{aligned} \quad (63)$$

$$\begin{aligned} B_{12} = & \bar{C}_{12} - (\bar{C}_{11} - \bar{C}_{12} - \bar{C}_{112})\epsilon_{11} \\ & - (\bar{C}_{11} - 2\bar{C}_{12} - \frac{1}{2}\bar{C}_{112} + \frac{1}{2}\bar{C}_{123})\psi, \end{aligned} \quad (64)$$

$$B_{21} = \bar{C}_{12} + \bar{C}_{112}\epsilon_{11} + \frac{1}{2}(\bar{C}_{11} + \bar{C}_{12} + \bar{C}_{112} - \bar{C}_{123})\psi, \quad (65)$$

$$\begin{aligned} B_{22} = & \bar{C}_{11} - (\bar{C}_{11} - \bar{C}_{12} - \bar{C}_{112})\epsilon_{11} \\ & - \frac{1}{2}(5\bar{C}_{11} - \bar{C}_{12} + \bar{C}_{111} - \bar{C}_{112})\psi, \end{aligned} \quad (66)$$

$$\begin{aligned} B_{23} = & \bar{C}_{12} - (2\bar{C}_{12} - \bar{C}_{123})\epsilon_{11} \\ & + \frac{1}{2}(\bar{C}_{11} - 5\bar{C}_{12} - 2\bar{C}_{112} + 2\bar{C}_{123})\psi. \end{aligned} \quad (67)$$

These results, which hold for materials of initial cubic symmetry or higher, reduce to the results given by Wallace^{2,3} for initially isotropic solids, which develop tetragonal symmetry under conditions of biaxial elastic strain. The relationships for isotropic elastic moduli are as follows:

$$\bar{C}_{11} = \lambda + 2\mu, \quad (68)$$

$$\bar{C}_{12} = \lambda, \quad (69)$$

$$\bar{C}_{44} = \mu, \quad (70)$$

$$\bar{C}_{111} = 4\xi + 2\zeta \quad , \quad (71)$$

$$\bar{C}_{112} = 2\zeta \quad , \quad (72)$$

$$\bar{C}_{123} = \nu - 2\xi + 2\zeta \quad , \quad (73)$$

where λ and μ define second-order moduli and ξ , ζ , and ν define third-order moduli according to standard convention.⁷ Additional identities (for initially isotropic material), which are not used in the one-dimensional calculations presented here, are given in the interest of completeness:

$$\bar{C}_{144} = \frac{1}{2}(\bar{C}_{112} - \bar{C}_{123}) \quad , \quad (74)$$

$$\bar{C}_{155} = \frac{1}{4}(\bar{C}_{111} - \bar{C}_{112}) \quad , \quad (75)$$

$$\bar{C}_{456} = \frac{1}{8}(\bar{C}_{111} - 3\bar{C}_{112} + 2\bar{C}_{123}) \quad . \quad (76)$$

For many materials we do have a complete set of second- and third-order elastic constants. For other materials, data are often incomplete or inconsistent. In these cases it is useful to express the isotropic third-order elastic constants in terms of adiabatic second-order elastic constants and their isothermal pressure derivatives⁹:

$$\nu = \frac{3}{2}[-3B(\partial B/\partial P)_T - 6\zeta] \quad , \quad (77)$$

$$\xi = \frac{1}{3}[-3B(\partial \mu/\partial P)_T - 3B - \mu + \frac{1}{2}\nu] \quad , \quad (78)$$

where

$$B = \lambda + 2\mu/3 \quad , \quad (79)$$

$$\zeta = \frac{1}{2} \bar{C}_{112} \quad (80)$$

In the case of polycrystalline copper λ , μ (and hence B), $(\partial B/\partial P)_T$, and $(\partial \mu/\partial P)_T$ are measured. However, there do not exist consistent, reproducible values of ζ . Fortunately, for uniaxial-strain compression, the computational results presented here are not sensitive to ζ , and it is possible to choose a nominal value without compromising the analysis. Values of mechanical and thermodynamic constants for copper are given in Table I.

TABLE I. Material Parameters for Copper.

Symbol	Value	Description	Reference
<u>Equation of State</u>			
ρ_o	8.94 g/cm ³	initial density	10
C_p	0.09213 $\frac{\text{cal}}{\text{gK}}$	specific heat	11
α	$1.65 \times 10^{-5} \text{ K}^{-1}$	thermal expansion coeff.	12
γ	1.99	Grüneisen coefficient	13
B	1.372 Mbar	bulk modulus	14
μ	0.4432 Mbar	shear modulus	15
$(\partial B / \partial P)_T$	5.28	isothermal derivative	14
$(\partial \mu / \partial P)_T$	1.30	isothermal derivative	15
ζ	-5.5 Mbar	third-order elastic modulus	16
<u>MTS Model</u>			
$\dot{\epsilon}_o$	$10 \mu\text{s}^{-1}$	strain-rate factor	4
ν_o	$10^5 \mu\text{s}^{-1}$	frequency factor	5
g_o	1.6	thermal activation coeff.	4
b	$2.55 \times 10^{-8} \text{ cm}$	The Burgers vector	4
$\hat{\tau}_a$	20 MPa	initial MTS	4
MB_o/b	$0.01686 \frac{\text{Mbar } \mu\text{s}}{\text{cm}}$	viscous drag parameter	5
C_s	0.217 cm/ μs	shear wave speed	5
$\hat{\tau}_{so}$	450 MPa	MTS parameter (Eq. 87)	4
A	0.312	MTS parameter (Eq. 87)	4
$\dot{\epsilon}_{so}$	$6.2 \times 10^4 \mu\text{s}^{-1}$	MTS parameter (Eq. 87)	4

IV. THE MECHANICAL THRESHOLD STRESS MODEL

Follansbee and Kocks⁴ have described a model representing the rate-dependent plastic deformation of copper for strain rates up to approximately $2 \times 10^4 \text{ s}^{-1}$. Follansbee⁵ has shown how this model can be modified to include transition to dislocation drag at higher strain rates. The principal variable of this analysis is the mechanical threshold stress (MTS), the flow stress at 0 K. The MTS defines the internal microstructural state of the material. Its evolution is strain path dependent. With dislocation drag included, the MTS model can be written as follows (in the present notation):

$$\partial_t \psi = \dot{\epsilon}_0 [\exp(\Delta G/kT) + \Lambda \nu_0 / V(\tau)]^{-1} , \quad (81)$$

$$\Delta G = (g_0 \mu b^3) \left[1 - \left(\frac{\tau - \hat{\tau}_a}{\hat{\tau} - \hat{\tau}_a} \right)^{2/3} \right] , \quad (82)$$

$$V(\tau) = (b/MB_0) 2\tau \left[1 + \left(\frac{2\tau}{C_s MB_0/b} \right)^2 \right]^{-1/2} , \quad (83)$$

$$\partial_t \hat{\tau} = (\theta_0/2) \partial_t \psi \left[1 - \frac{\tanh(2Z)}{\tanh(2)} \right] , \quad (84)$$

$$Z = \frac{\hat{\tau} - \hat{\tau}_a}{\hat{\tau}_s - \hat{\tau}_a} , \quad (85)$$

$$\theta_0 \text{ (Mbar)} = 0.02556 + 1.2 \times 10^{-4} \ln \dot{\epsilon} (\mu\text{s}^{-1}) + 0.34 \dot{\epsilon} (\mu\text{s}^{-1}) , \quad (86)$$

$$\ln (\hat{\tau}_s / \hat{\tau}_{so}) = \frac{kT}{A\mu b^3} \ln (\partial_t \psi / \dot{\epsilon}_{so}) , \quad (87)$$

where $\hat{\tau}$ is the mechanical threshold stress, $V(\cdot)$ is the dislocation velocity based on a viscous drag model (with B_0 equal to the linear drag coefficient), and C_s is the limiting dislocation velocity. The term M represents the Taylor factor and b is the magnitude of the Burgers vector. These quantities enter only through the combination MB_0/b , which has units of stress divided by velocity. These and remaining terms are material constants whose values are listed in Table I. In adapting the MTS (Follansbee's notation) model^{4,5} to conditions of uniaxial strain we have made the identifications that

$$\sigma(\text{MTS}) \rightarrow 2\tau, \quad (88)$$

$$\dot{\epsilon}(\text{MTS}) \rightarrow \partial_t \psi, \quad (89)$$

in order to avoid confusion between longitudinal and shear stress components, as well as between total and plastic strain rates.

The quantity Λ which appears in Eq. (81) is the average distance between obstacles, which is unknown and must be determined by comparison with experimental shock-wave data.

V. THE SHOCK-CHANGE EQUATION

1. The CHARADE Form

As part of the characteristic solution used in CHARADE, we treat the leading wave as a discontinuity in stress, particle velocity, specific volume, and internal energy. The leading wave in the case of shock initiation is a shock to a point on the unreacted Hugoniot, and in the case of a phase transition is a shock in the initial phase. The discussion

presented here is in terms of elastic-plastic response.

There are numerous ways to express the rate of change of variables along the shock path and there are several approximations to exact relationships: both in Eulerian (spatial) and Lagrangian (material) coordinates. This has led to some confusion, and hence we would like to have a record of these different expressions and approximations. We give here a general derivation of the shock-change equation in one dimension and list its many forms.

The shock-change equation is the relationship between derivatives of quantities in terms of x and t (or X and t) and derivatives of variables following the shock front, which moves with speed U into undisturbed material at rest.

In Eulerian coordinates x and t , the mass and momentum conservation laws and material constitutive equation can be written as [see Eqs. (1), (2) and (33)]:

$$\dot{\rho} + \rho(\partial u / \partial x)_t = 0 \quad , \quad (90)$$

$$\rho \dot{u} + (\partial \sigma / \partial x)_t = 0 \quad , \quad (91)$$

$$\dot{\sigma} - c^2 \dot{\rho} = F \quad , \quad (92)$$

where a dot over a variable signifies differentiation along a particle path according to Eq. (4). If we define D_t to be the time derivative operator following the shock path $dx/dt = U(t)$, then

$$D_t = (\partial / \partial t)_x + U(\partial / \partial x)_t \quad , \quad (93)$$

$$(\dot{}) = D_t - (U-u)(\partial/\partial x)_t \quad (94)$$

The jump conditions across a discontinuity moving into undisturbed material at rest are

$$\sigma = \rho_o U u \quad , \quad (95)$$

$$\rho(U-u) = \rho_o U \quad . \quad (96)$$

We define two quantities specifying the elastic shock response (the subscript H refers to conditions along the elastic Hugoniot curve):

$$Z_H = (d\sigma/du)_H \quad , \quad (97)$$

$$C_H^2 = (d\sigma/d\rho)_H \quad , \quad (98)$$

which are related according to

$$Z_H = \frac{2\rho_o U}{1 + (\rho_o U/\rho C_H)^2} \quad (99)$$

This allows us to write, for example,

$$D_t \sigma = Z_H D_t u \quad , \quad (100)$$

$$D_t \sigma = C_H^2 D_t \rho \quad . \quad (101)$$

The quantities Z_H and C_H define the differential properties of the elastic Hugoniot. At low stress amplitudes $Z_H \cong \rho_0 c_0$ and $C_H \cong c_0$, where c_0 is the adiabatic longitudinal elastic sound speed at $\rho = \rho_0$.

With Eq. (96) and the definition of D_t , Eqs. (90)-(91) can be written

$$\rho_0 U \dot{\rho} + \rho^2 (D_t u - \dot{u}) = 0 \quad , \quad (102)$$

$$\rho_0 U \dot{u} + D_t \sigma - \dot{\sigma} = 0 \quad , \quad (103)$$

which are valid only at the shock front. One form of the shock-change equation is obtained from Eqs. (92), (100), (102), and (103):

$$D_t \sigma = \frac{F + [(\rho c / \rho_0 U)^2 - 1] \rho_0 U \dot{u}}{1 + (\rho c / \rho_0 U)^2 (\rho_0 U / Z_H)} \quad . \quad (104)$$

This is expressed in terms of the particle acceleration immediately behind the shock front. From Eqs. (25), (30), and (39) we can express Eq. (104) in terms of the Lagrangian stress gradient $\partial_X \sigma$ and the Lagrangian longitudinal sound speed C_L :

$$D_t \sigma = \frac{F - [(C_L / U)^2 - 1] U \partial_X \sigma}{1 + (C_L / U)^2 (\rho_0 U / Z_H)} \quad . \quad (105)$$

Equation (105) is the form that is used in CHARADE.

Equation (104) is identical to the result of Ahrens and Duvall¹⁷ [their Eq. (26)] when Eqs. (96) and (99) used:

$$D_x \sigma = \frac{-\rho_0 [(U-u)^2 - c^2] \dot{u} - (U-u)^2 (-F)/U}{(U-u)^2 (1 + c^2/2C_H^2) + c^2/2}, \quad (106)$$

noting that the relaxation function of Ahrens and Duvall is the negative of the one used here, and that the derivative with respect to position along the shock path is $D_x = U^{-1} D_t$.

2. Other Forms of the Shock-Change Equation

For the purposes of this report, we could stop here and say nothing more about the shock-change equation. Because of different approximations used by various authors, confusion arises as to which expressions are exact and which are approximations. For this reason we want to say a little more about the shock-change equation.

Equations (104)-(106) are exact inasmuch as the elastic wave can be treated as a mathematical discontinuity. Possible errors introduced by this approximation depend on the actual risetime of the elastic wave, which is unknown. A brief discussion and analysis of this problem is given in Appendix A.

If we accept the assumption that the elastic wave can be treated to good approximation as a mathematical discontinuity, then the stress decay at the elastic wavefront is given by Eq. (104) in terms of the material-dependent and amplitude-dependent wavespeeds: c (the isentropic longitudinal elastic wavespeed), U (the finite-amplitude elastic shock velocity), and C_H [Eq. (98), the Hugoniot sound speed as defined by Ahrens and Duvall]. In general, all three wave velocities are different. We know, for example, that

$$u + c > U \quad (107)$$

for normal materials whose elastic wave "shocks up". Hence

$$c > U - u = \rho_0 U / \rho , \quad (108)$$

and

$$(\rho c / \rho_0 U) = C_L / U > 1 \quad (109)$$

at the elastic wavefront.

The isentropic sound speed c differs in principle from the Hugoniot sound speed C_H because of the entropy increase on the Hugoniot:

$$C_H > c . \quad (110)$$

For elastic compression the entropy increase is small and can come only from effects of thermal conduction. Thus, to a very good approximation at low compressions in most materials,

$$C_H \cong c . \quad (111)$$

This is the approximation used by Clifton¹⁸ and Partom.¹⁹ When Eq. (111) applies,

$$\rho_0 U / Z_H \cong \frac{1}{2} [1 + (\rho_0 U / \rho c)^2] , \quad (112)$$

and Eq. (104) becomes

$$D_t \sigma \approx \frac{F + [(\rho c / \rho_0 U)^2 - 1] \rho_0 U \dot{u}}{\frac{1}{2} + \frac{1}{2} (\rho c / \rho_0 U)^2} . \quad (113)$$

An additional approximation can be used in the case when $\rho c \approx \rho_0 U$ and the acceleration behind the elastic shock front is small, so that

$$F \gg [(\rho c / \rho_0 U)^2 - 1] \rho_0 U \dot{u} . \quad (114)$$

In this case

$$D_t \sigma \approx F/2 , \quad (115)$$

which can be a very useful approximation in estimating decay rates at low-stress amplitudes, or for nearly linear material behavior such that

$$\rho c = \rho_0 U . \quad (116)$$

Equation (116) holds, for example, in materials which obey the isentropic elastic relationship

$$\sigma = C_{11} (1 - \rho_0 / \rho) , \quad (117)$$

where C_{11} is the (constant) isentropic longitudinal modulus.

Equation (114) was derived with the decay rate dependent upon F and $\dot{u}$. We could just as well have derived expressions for $D_t \sigma$ in terms of F and $\dot{\rho}$, or F and $\dot{\sigma}$, or other combinations. Two of these expressions are listed below (in exact form):

$$D_t \sigma = \frac{F + [(\rho c / \rho_o U)^2 - 1](\rho_o U / \rho)^2 \dot{\rho}}{1 + \rho_o U / Z_H} \quad (118)$$

$$D_t \sigma = \frac{F + [(\rho c / \rho_o U)^2 - 1] \dot{\sigma}}{(\rho c / \rho_o U)^2 (1 + \rho_o U / Z_H)} \quad (119)$$

The time rates of change of the shock amplitude can also be obtained for other variables by differentiating the jump conditions, Eqs. (95) and (96), along a shock path to obtain

$$D_t u = (1/Z_H) D_t \sigma \quad (120)$$

$$D_t \rho = (\rho / \rho_o U)^2 (2 \rho_o U / Z_H - 1) D_t \sigma \quad (121)$$

$$\rho_o u D_t U = (1 - \rho_o U / Z_H) D_t \sigma \quad (122)$$

A summary of the shock-change equations is presented in Tables II, III, and IV with the exact relationships given in Table II, the $c = C_H$ approximation in Table III, and the $\rho c = \rho_o U$ approximation in Table IV.

TABLE II. Shock-Change Equations (Exact).

$$D_t \sigma = \frac{F + [(\rho c / \rho_0 U)^2 - 1] \rho_0 U \dot{u}}{1 + (\rho c / \rho_0 U)^2 (\rho_0 U / Z_H)} = \frac{F - [(\rho c / \rho_0 U)^2 - 1] (\rho_0^2 U^2 / \rho) (\partial u / \partial x)_t}{1 + \rho_0 U / Z_H}$$

$$D_t \rho = \frac{F + [(\rho c / \rho_0 U)^2 - 1] (\rho_0 U / \rho)^2 \dot{\rho}}{1 + \rho_0 U / Z_H} = \frac{F - [(\rho c / \rho_0 U)^2 - 1] (\rho_0 U / \rho)^3 (\partial \rho / \partial x)_t}{3 \rho_0 U / Z_H + (\rho c / \rho_0 U)^2 (1 - 2 \rho_0 U / Z_H)}$$

$$D_t \sigma = \frac{F + [(\rho c / \rho_0 U)^2 - 1] \dot{\sigma}}{(\rho c / \rho_0 U)^2 (1 + \rho_0 U / Z_H)} = \frac{F - [(\rho c / \rho_0 U)^2 - 1] (\rho_0 U / \rho) (\partial \sigma / \partial x)_t}{1 + (\rho c / \rho_0 U)^2 (\rho_0 U / Z_H)}$$

$$D_t u = (1 / Z_H) D_t \sigma$$

$$D_t \rho = (1 / C_H^2) D_t \sigma$$

$$\rho_0 u D_t U = (1 - \rho_0 U / Z_H) D_t \sigma$$

$$Z_H = \frac{2 \rho_0 U}{1 + (\rho_0 U / \rho C_H)^2}$$

To change from the Eulerian to Lagrangian description:

$$\dot{u} = \partial_t u, (\partial u / \partial x)_t = (\rho / \rho_0) \partial_X u, \text{ etc.}$$

$$C_L = \rho c / \rho_0$$

TABLE III. Shock-Change Equations ($C_H = c$ Approximation).

$$D_t \sigma = \frac{F + [(\rho c / \rho_0 U)^2 - 1] \rho_0 U \dot{u}}{\frac{3}{2} + \frac{1}{2}(\rho c / \rho_0 U)^2} = \frac{F - [(\rho c / \rho_0 U)^2 - 1] (\rho_0^2 U^2 / \rho) (\partial u / \partial x)_t}{\frac{3}{2} + \frac{1}{2}(\rho_0 U / \rho c)^2}$$

$$D_t \rho = \frac{F + [(\rho c / \rho_0 U)^2 - 1] (\rho_0 U / \rho)^2 \dot{\rho}}{\frac{3}{2} + \frac{1}{2}(\rho_0 U / \rho c)^2} = \frac{F - [(\rho c / \rho_0 U)^2 - 1] (\rho_0 U / \rho)^3 (\partial \rho / \partial x)_t}{\frac{1}{2} + \frac{3}{2}(\rho_0 U / \rho c)^2}$$

$$D_t \sigma = \frac{F + [(\rho c / \rho_0 U)^2 - 1] \dot{\sigma}}{\frac{1}{2} + \frac{3}{2}(\rho c / \rho_0 U)^2} = \frac{F - [(\rho c / \rho_0 U)^2 - 1] (\rho_0 U / \rho) (\partial \sigma / \partial x)_t}{\frac{3}{2} + \frac{1}{2}(\rho c / \rho_0 U)^2}$$

$$D_t u = (1/Z_H) D_t \sigma$$

$$D_t \rho = (1/c^2) D_t \sigma$$

$$\rho_0 u D_t U = \frac{1}{2} [1 - (\rho_0 U / \rho c)^2] D_t \sigma$$

$$Z_H = \frac{2 \rho_0 U}{1 + (\rho_0 U / \rho c)^2}$$

To change from the Eulerian to Lagrangian description:

$$\dot{u} = \partial_t u, (\partial u / \partial x)_t = (\rho / \rho_0) \partial_X u, \text{ etc.}$$

$$C_L = \rho c / \rho_0$$

TABLE IV. Shock-Change Equations ($\rho c = \rho_o U$ Approximation).

$$D_t \sigma = F/2$$

$$D_t u = (1/\rho_o U) F/2$$

$$D_t \rho = (\rho/\rho_o U)^2 F/2$$

$$\rho_o u D_t U = 0$$

$$Z_H = \rho_o U = \rho_o C_L$$

VI. NUMERICAL CHARACTERISTIC SOLUTIONS

Here we present the basic equations and some special cases for numerical integration of the characteristic equations in the order in which they appear in CHARADE. At present CHARADE considers a single sample material backed by a window material (or free surface) and is loaded by a time-dependent particle velocity at the left boundary: Figure 2.

Fig. 2. CHARADE geometry.

1. Shock Front

At $t = t_1$ the elastic shock front is at $X = X_a$ in computational cell ia as shown in Figure 3. The longitudinal stress amplitude and particle velocity at $t = t_1$ and $X = X_a$ are σ_a and u_a . The stresses at $i = ia$ and $i = ia-1$ ($t = t_1$) are $\sigma(ia)$ and $\sigma(ia-1)$. Hence, an approximation to $\partial_X \sigma$ near the shock front (at $t = t_1$) is

$$[\partial_X \sigma]_a \approx \frac{\sigma_a - [\sigma(ia) + \sigma(ia-1)]/2}{X_a - [X(ia) - dX/2]}.$$

The shock velocity at point a is $U = \sigma_a / \rho_0 u_a$. The Lagrangian sound speed C_L is also evaluated at point a . For plane shock propagation ($\alpha = 0$) the time rate of change of longitudinal stress along the shock path S in Figure 3 is given by Eq. (105):

$$[D_t \sigma]_a \approx \frac{F_a - [(C_L/U)^2 - 1]U[\partial_X \sigma]_a}{1 + (C_L/U)^2(\rho_o U/Z_{Ha})},$$

where F_a (the rate function) and Z_{Ha} (the differential elastic Hugoniot impedance) are evaluated at point a.

From this information new values of shock position, stress, and particle velocity are given to first-order by

$$x_b = x_a + Udt,$$

$$\sigma_b = \sigma_a + [D_t \sigma]_a dt,$$

$$u_b = \sigma_b / \rho_o U.$$

Fig. 3. Advance shock front.

The remainder of the flow variables at shock point b are obtained from subroutine HUGO. When the shock front jumps a cell boundary, $ib = ia + 1$. Otherwise, $ib = ia$.

When the elastic shock front reaches the interface between the sample and the window (it is assumed that the window impedance is less than that of the sample) or free surface, a rarefaction wave is reflected back into the sample. In this case we integrate down an isentrope for the sample and up the Hugoniot for the window to find the interaction state at the point of reflection. The shock is then continued on into the window (if present). If the shock is reflected from a free surface, the shock ceases to exist and all further flow calculations are continuous.

2. Left Boundary

Calculations at the left boundary must distinguish between the three cases shown in Figure 4.

(a) C^- intersects the shock

In this case we must interpolate between a and b (on the shock path S) to obtain flow variables at point R (to the right of $i = 1$). The fractional distance from a to b is

$$f_R = t^*/dt ,$$

here t^* is obtained from simultaneous solution of the pair of linear equations for C^- and S as shown in Figure 4(a):

$$t^* = \frac{X_o - X_a + C_{LR} dt}{U + C_{LR}} .$$

Fig. 4. Characteristic solutions for $i = 1$. The slope of the C^- characteristic is $-1/C_{LR}$ (computed at $i = 1$ and $t = t_1$). The slope of the shock path (S) is $1/U$ (computed at point a): (a) C^- intersects shock: (b) C^- does not intersect shock (shock in first cell); (c) shock not in first cell.

Hence

$$\bar{v}_R = (1 - f_R)\bar{v}_a + f_R\bar{v}_b,$$

$$dt_R = dt - t^*,$$

where $\bar{v}_R$ is a general variable and dt_R is C^- time increment ($t_2 - t_1 - t^*$).

(b) C^- does not intersect shock (shock in first cell)

When the C^- characteristic does not intersect the shock, but point a lies in the first cell, we must interpolate between $i = 1$ and point a (along $t = t_1$) to obtain flow variables at point R: Figure 4(b). The fractional distance from $i = 1$ to a is

$$f_R = C_{LR} dt / (X_a - X_o) ,$$

and

$$\bar{V}_R = (1 - f_R) \bar{V}_1 + f_R \bar{V}_a ,$$

$$dt_R = dt .$$

(c) Shock not in first cell

In this case we simply interpolate between points 1 and 2 as shown in Figure 4(c):

$$f_R = C_{LR} dt / dX_1 ,$$

$$\bar{V}_R = (1 - f_R) \bar{V}_1 + f_R \bar{V}_2 ,$$

$$dt_R = dt .$$

Since u is specified along the $i = 1$ boundary in all three cases, the characteristic equations we want to solve are

$$u = u_o(t) ,$$

$$(\sigma - \sigma_R) - \rho_o C_{LR} (u - u_R) = F_R dt_R ,$$

where F_R is the rate function evaluated from interpolated variables at point R: Figure 4. Hence the stress and particle velocity at $i = 1$ and $t = t_2$ are given by

$$\sigma = \sigma_R + \rho_o C_{LR} [u_o(t_2) - u_R] + F(v_R) dt_R ,$$

$$u = u_o(t_2) .$$

3. Cells Near the Shock Front

Because of the proximity of the shock front to cells $i = ib$ and $i = ib - 1$, special care must be taken to find correct interpolated values for flow variables and time steps for the C^+ and C^- characteristics.

Early in the shock front calculation a quantity called $ijump$ is calculated: $ijump = 1$ if we have crossed a computational cell; otherwise $ijump = 0$. Figure 5 shows three cases that must be considered for characteristic solution of σ and u at $i = ib$ and $t = t_1 + dt$: (a) $ijump = 0$ and C^- does not intersect shock; (b) $ijump = 0$ and C^- does intersect shock; (c) $ijump = 1$ (in this case C^- must intersect the shock, and shock point a influences interpolated values for C^+ characteristic).

Fig. 5. Characteristic solutions for $i = ib$. Here t^* is the time difference between C^- , S intersection and t_1 : t^* is positive for cases (b) and (c), but negative for case (a).

We first calculate C_L to the left and to the right of $i - ib$ (ib is the final index of the shock after advance to time $t_1 + dt$; this is also called $ishok$ in the main program). Let C_{LL} and C_{LR} be these two quantities. When the material to the left and to the right of $i - ib$ is the same, $C_{LL} = C_{LR}$; they differ when $i - ib$ is the boundary between the sample and the window. We then calculate a test quantity t^* :

$$t^* = dt - (X_b - X_i)/(C_{LR} + U) ,$$

where U is the shock speed at a . When $t^* > 0$, the C^- characteristic intersects S for $t > t_1$. The equation for t^* is obtained from simultaneous solution for the C^- and S paths shown in Figure 5:

$$C^- : X - X_i = - C_{LR}(t - dt) ,$$

$$S : X - X_b = U(t - dt) .$$

Once the starting points (and time steps) to the left and the right of $i - ib$ are determined, we use Eqs. (44) and (45) to solve for σ and u at $i - ib$ and $t = t_1 + dt$:

$$\sigma = \frac{C_{LR}(\sigma_L + \rho_o C_{LL} u_L + F_L dt_L) + C_{LL}(\sigma_R - \rho_o C_{LR} u_R + F_R dt_R)}{C_{LR} + C_{LL}} ,$$

$$u = \frac{(\sigma_L + \rho_o C_{LL} u_L + F_L dt_L) - (\sigma_R - \rho_o C_{LR} u_R + F_R dt_R)}{\rho_o (C_{LR} + C_{LL})} .$$

(a) $t^* < 0$; C^- does not intersect shock front

To the left of $i = ib$, the fractional distance from ib to $ib - 1$ is given by

$$f_L = C_{LL} dt/dX_L ,$$

and hence interpolated variables for C^+ are

$$\bar{V}_L = (1 - f_L)\bar{V}_{ib} + f_L\bar{V}_{ib-1} .$$

To the right of $i = ib$, the fractional distance from ib to point a is

$$f_R = C_{LR} dt/(X_a - X_i) ,$$

and hence

$$\bar{V}_R = (1 - f_R)\bar{V}_{ib} + f_R\bar{V}_a ,$$

$$dt_L = dt_R = dt .$$

(b) $t^* > 0$; C^- intersects shock ($X_a > X_i$)

In this case f_L , $\bar{V}_L$, and dt_L are the same as for case (a); however, to the right of $i = ib$ the interpolation relationships are

$$f_R = t^*/dt ,$$

$$\bar{V}_R = (1 - f_R)\bar{V}_a + f_R\bar{V}_b ,$$

$$dt_R = dt - t^* .$$

(c) $t^* > 0$; C^- intersects shock ($X_a < X_1$)

In this case we must interpolate between $ib - 1$ and point a for the C^+ characteristic to the left of $i = ib$. The fractional distance from point a to $ib - 1$ is

$$f_L = (C_{LL}dt - X_i + X_a)/(dX_L - X_i + X_a) ,$$

$$\bar{v}_L = (1 - f_L)\bar{v}_a + f_L\bar{v}_{ib-1} ,$$

$$dt_L = dt .$$

Equations for interpolation on the right remain as in case (b).

If, for $i = ib$ we had $ijump = 1$, this means that some special consideration must be given to $ib-1$ also: interpolations to the right of $ib-1$ will involve point a on the shock path S . In this case we simply set $i = ib-1$ and $ijump = 0$ and go back through the logic of this section. Correct calculations for σ and u at $ib-1$ are then performed.

4. Normal Interior Point Calculation

The fractional distance from i to $i-1$ on the left is given by

$$f_L = C_{LL}dt/dX_L ,$$

and hence

$$\bar{V}_L = (1 - f_L)\bar{V}_i + f_L\bar{V}_{i-1} ,$$

where C_{LL} is the Lagrangian sound speed to the left evaluated at the start of the time step. The fractional distance from i to $i + 1$ on the right is given by

$$f_R = C_{LR}dt/dx_R ,$$

and hence

$$\bar{V}_R = (1 - f_R)\bar{V}_i + f_R\bar{V}_{i+1} .$$

Longitudinal stress and particle velocity are obtained by means of Eqs. (44) and (45) with

$$dt_L = dt_R = dt ,$$

$$\sigma = \frac{C_{LR}(\sigma_L + \rho_o C_{LL}u_L + F_L dt) + C_{LL}(\sigma_R - \rho_o C_{LR}u_R + F_R dt)}{C_{LR} + C_{LL}} ,$$

$$u = \frac{(\sigma_L + \rho_o C_{LL}u_L + F_L dt) - (\sigma_R - \rho_o C_{LR}u_R + F_R dt)}{\rho_o (C_{LR} + C_{LL})} .$$

5. Integration Along Particle Paths

Once σ and u have been obtained at time t_2 at some particular Lagrangian point (index = i), we then integrate all remaining variables along this particle path (index = i) by calling the XPATH subroutine. Here

we specify the index, the time step dt, all "old" variables (last time step), and the "new" values of σ and u . XPATH returns to the main program with all other new variables.

Within the subroutine XPATH the rate of change of volume and internal energy are calculated from

$$\partial_t v = (F - \partial_t \sigma) / (\rho_o C_L)^2 ,$$

$$\partial_t e = - \sigma \partial_t v .$$

In the present version of CHARADE the explosive reaction rate is set to zero. For materials other than CUB100 and ISOTRP a call is made to EOSMIX to return additional variables.

If we are in material identified as CUB100 or ISOTRP, we then calculate the stress-strains coefficients B_{ij} (Voigt notation), plastic strain rate, temperature rate, and shear stress rate:

$$\partial_t \psi = - F / (B_{11} - B_{12} - 2\gamma\tau) ,$$

$$\partial_t T = - \gamma T \rho \partial_t v + (2\tau / \rho C_\eta) \partial_t \psi ,$$

$$\partial_t \tau = - \frac{1}{2} (B_{11} - B_{21}) \partial_t v - \frac{1}{4} (2B_{11} - 2B_{21} - 2B_{12} + B_{22} + B_{23}) \partial_t \psi .$$

The function F is evaluated in the subroutine FRATE. For the MTS model described here

$$F = - \frac{(B_{11} - B_{12} - 2\gamma r)\dot{\epsilon}_0}{\exp(\Delta G/kT) + \Lambda\nu_0/V(r)} ,$$

where ΔG depends on r and $\hat{r}$ according to Eq. (82) and $V(r)$ is the dislocation drag velocity given by Eq. (83). For materials other than CUB100 and ISOTRP, $F = 0$.

The one remaining evolutionary equation integrated in subroutine XPATH (for materials designated CUB100 or ISOTRP and which obey the MTS description) is the mechanical threshold stress itself:

$$\partial_t \hat{r} = (\theta_0/2) \left[1 - \frac{\tanh(2Z)}{\tanh(2)} \right] \partial_t \psi ,$$

$$Z = \frac{\hat{r} - \hat{r}_a}{\hat{r}_s - \hat{r}_a} ,$$

$$\theta_0 \text{ (Mbar)} = 0.02556 + 1.2 \times 10^{-4} \ln(\partial_t \psi) + 0.34 \partial_t \psi ,$$

(for $\partial_t \psi$ in units of μs^{-1})

$$\hat{r}_s = \hat{r}_{so} \exp \left[\frac{kT}{A\mu b^3} \ln(\partial_t \psi / \dot{\epsilon}_{so}) \right] ,$$

with MTS parameters given in Table I.

VII. SAMPLE PROBLEMS

Experimental data of Swegle and Grady¹³ and Warnes and Tonks²⁰ on the maximum strain rates ($\dot{\rho}/\rho$), average risetimes, and maximum accelerations for 30-, 61-, and 100-kbar shock waves in copper are given in Table V.

TABLE V. Shock-Wave Data for Copper.

Stress (kbar)	Strain Rate (μs^{-1})	Risetime (μs)	Acceleration ($\text{cm}/\mu\text{s}^2$)
30	0.35	0.057	0.2
61	11	0.0035	5
100	120(est.)	0.0005 (est.)	50 (est.)

We will use the data of Table V in conjunction with the CHARADE code to illustrate some properties of the MTS model (with transition to dislocation drag) as given in reference 5.

We first consider a copper sample shock loaded to 30 kbar and examine the calculated material response in the 2.5 mm adjacent to the impact surface. The only remaining unspecified MTS variable is the average separation between obstacles, Λ . This quantity is chosen to give a maximum acceleration of $0.2 \text{ cm}/\mu\text{s}^2$ in agreement with experimental data (Table V). This gives a value of $\Lambda = 0.5 \mu\text{m}$, which is quite reasonable: if dislocation interactions are responsible for these obstacles, this value would correspond to an immobile dislocation density of $(1/\Lambda^2) = 4 \times 10^8 \text{ cm}^{-2}$. Results of these calculations are shown in Figure 6. The input deck for this calculation is given in Appendix B.

The computational zone size is $1.67 \times 10^{-3} \text{ cm}$ and variables are monitored at cell indices 21, 41, 61, 81, and 101 as shown in Figure 6. The longitudinal stress component σ is shown in Figure 6(a). The shear stress τ divided by $\hat{\tau}$ (the mechanical threshold stress) is shown in Figure 6(b): a value of $(\tau/\hat{\tau}) > 1$ indicates transition to drag-dominated plastic flow. The ratio $\tau/\hat{\tau}$ is the same as $\sigma/\hat{\sigma}$ (Follansbee's notation, ref. 5).

We note from Figure 6(b) that the passage of a plastic shock wave drives the material state into the drag regime.

Figure 6(c) shows the evolution of the mechanical threshold stress $2\hat{\tau} = \hat{\sigma}$ (Follansbee's notation). We note here some enhanced hardening near the impact surface. Figure 6(d) simply demonstrates that a value $\Lambda = 0.5 \mu\text{m}$ results in a maximum acceleration of $\sim 0.20 \text{ cm}/\mu\text{s}^2$ in the plastic wave.

Figure 7 shows "snapshots" (variables as functions of position for fixed times) for the same calculation. The time between the five individual plots is approximately $0.098 \mu\text{s}$. In Figure 7(d) plastic strain replaces maximum acceleration.

Figure 8 shows the material response based on constant $\Lambda = 0.5 \mu\text{m}$ for a 61-kbar shock wave in copper. We note from Figure 8(d) that the maximum acceleration increases from $0.2 \text{ cm}/\mu\text{s}^2$ (30-kbar shock) to $\sim 0.3 \text{ cm}/\mu\text{s}^2$ (61-kbar shock), but does not come close to the experimental value of $5 \text{ cm}/\mu\text{s}^2$. This is a shortcoming of the dislocation drag description. "Snapshots" of these calculations are shown in Figure 9.

The reason for the disagreement between theory and experiment comes from the nearly linear dependence of $V(\tau)/\Lambda v_0$ on shear stress τ . Swegle and Grady¹³ and others²⁰⁻²³ have shown that approximately quadratic dependence of plastic strain rate on applied shear stress is required to represent the strong dependence of risetime on shock amplitude.

A second set of calculations was performed with the assumption that Λ decreased with increasing dislocation density, or equivalently with increasing mechanical threshold stress.⁵ The relationship between Λ and $2\hat{\tau} = \hat{\sigma}$ (Follansbee's notation) is

$$\Lambda \cong \mu b / \hat{\sigma} .$$

When the above expression is used in place of constant Λ , we must adjust the frequency factor ν_0 to give a peak acceleration of $0.2 \text{ cm}/\mu\text{s}^2$ for a 30-kbar shock wave in copper: $b\nu_0 = 0.01 \text{ cm}/\mu\text{s}$. The results are shown in Figure 10. Without making any changes in the model or model parameters, we then calculate the response to a 61-kbar shock wave: Figure 11. Figure 11(d) shows that the maximum acceleration had been increased to $\sim 0.8 \text{ cm}/\mu\text{s}^2$, but still remains much less than the experimentally measured value of $5 \text{ cm}/\mu\text{s}^2$. This observation combined with the extremely slowly decaying elastic wave, shown in Figures 11(a)-(b), indicate that we still do not have the correct description of the dislocation drag contribution to high-strain-rate plastic deformation in copper.

These computations suggest that the stress dependence of the plastic constitutive behavior is stronger than represented by Eqs. (81)-(87). This could be in part due to the neglect of a stress dependence in the constant $\dot{\epsilon}_0$, arising from the stress dependence of the mobile dislocation density. It is possible that the inclusion of a first or second power stress dependence of the mobile dislocation density, which would lead to the same stress dependence in $\dot{\epsilon}_0$, would give shock-wave acceleration dependence on peak pressure that is in agreement with the data. Alternatively, a purely phenomenological analysis in the drag controlled regime can be combined with the MTS model in the thermally activated regime to provide an adequate description of shock waves in copper (Tonks and Johnson, ref. 21).

Fig. 6. (a) Longitudinal stress, (b) $\sigma/\hat{\sigma}$ (Follansbee's notation), (c) $\hat{\sigma}$, and (d) $\dot{u}$ as functions of time for a 30-kbar shock wave in copper: $\Lambda = 0.5 \mu\text{m}$.

Fig. 7. (a) Longitudinal stress, (b) $\sigma/\hat{\sigma}$ (Follansbee's notation), (c) $\hat{\sigma}$, and (d) plastic strain ψ as functions of position for a 30-kbar shock wave in copper: $\Lambda = 0.5 \mu\text{m}$.

Fig. 8. (a) Longitudinal stress, (b) $\sigma/\hat{\sigma}$ (Follansbee's notation), (c) $\hat{\sigma}$, and (d) $\dot{u}$ as functions of time for a 61-kbar shock wave in copper: $\Lambda = 0.5 \mu\text{m}$.

Fig. 9. (a) Longitudinal stress, (b) $\sigma/\hat{\sigma}$ (Follansbee's notation), (c) $\hat{\sigma}$, and (d) plastic strain ψ as functions of position for a 61-kbar shock wave in copper: $\Lambda = 0.5 \mu\text{m}$.

Fig. 10. (a) Longitudinal stress, (b) $\sigma/\hat{\sigma}$ (Follansbee's notation), (c) $\hat{\sigma}$, and (d) $\dot{u}$ as functions of time for a 30-kbar shock wave in copper: $\Lambda = \mu b/\hat{\sigma}$, $b\nu_0 = 0.01 \text{ cm}/\mu s$.

Fig. 11. (a) Longitudinal stress, (b) $\sigma/\hat{\sigma}$ (Follansbee's notation), (c) $\hat{\sigma}$, and (d) $\dot{u}$ as functions of time for a 61-kbar shock wave in copper: $\Lambda = \mu b/\hat{\sigma}$, $b\nu_0 = 0.01 \text{ cm}/\mu s$.

VIII. CONCLUSIONS

This report presents some background and details regarding the characteristic method and its application to plate-impact problems for rate-dependent material response. The allowable material constitutive behavior is in the form

$$\dot{\sigma} - c^2 \dot{\rho} = F(\rho, (\psi_j)) \quad ,$$

$$\dot{\psi}_i = A_i \dot{\rho} + G_i(\rho, (\psi_j)), \quad i = 1, 2, 3, \dots, N$$

where σ is the longitudinal stress component in the direction of wave propagation, c is the longitudinal Eulerian sound speed, ρ is the material density, and ψ_i is one of N internal-state variables (including shear stress and temperature) whose evolutionary laws are known.

Details are given for elastic-plastic materials with initial cubic symmetry (or higher) obeying the Mechanical Threshold Stress (MTS) model with transition to dislocation drag. Several calculations were performed simply to illustrate the usefulness of the CHARADE code for investigating the dynamic effects of material models. While we obtained two negative results, no attempt was made to "improve" the models; that was not the purpose of the present investigation.

CHARADE has already been used for other applications, notably calculation of explosive initiation.²⁴ In the near future we plan to extend this version of CHARADE to be able to handle subtle details of quasi-elastic unloading from a shocked state produced by symmetric impact, and problems related to the pulse-duration hardening effect.²⁵

An additional option that we want to include at a later time is that of cylindrical and spherical flow. This was omitted in the present version in the interest of timeliness.

IX. ACKNOWLEDGEMENTS

The authors wish to acknowledge Charles A. Forest for his contributions of the EOSMIX subroutine and the PIX-related subroutines for producing graphics, and Paul S. Follansbee for assessment of MTS response calculations.

APPENDIX A:

THE JUMP CONDITIONS FOR NON-STEADY, NEAR-DISCONTINUOUS SHOCKS

A question sometimes arises regarding the correct expressions for conservation of mass, momentum, and energy across nonsteady, nearly discontinuous shocks. A common derivation of the "jump conditions" involves either the assumption of steady propagation (that is, constant propagation velocity without change of shape in an Eulerian coordinate system) or that the wave is an ideal discontinuity.

Mathematical discontinuities in real physical systems are impossible. Therefore the question of how well the assumption of discontinuous flow approximates a particular physical situation is of some practical interest.

We write the basic conservation laws in two ways: the first is the integral form as applied to a finite material volume V (moving with the material and bounded by the surface S) and the second is the corresponding set of partial differential operations which hold at each point in space and time where the flow is continuous. Thus⁶

$$d/dt \int_V \rho dV = 0 \quad , \quad (A-1)$$

$$d/dt \int_V \rho u_i dV = \oint_S r_{ji} n_j dS + \int_V \rho B_i dV \quad , \quad (A-2)$$

$$d/dt \int_V \rho (e + u^2/2) dV = \oint_S u_i r_{ji} n_j dS - \oint_S J_i n_i dS + \int_V \rho u_i B_i dV \quad , \quad (A-3)$$

and

$$\partial \rho / \partial t + \partial / \partial x_i (\rho u_i) = 0 \quad , \quad (A-4)$$

$$\rho du_i/dt = \partial \tau_{ji}/\partial x_j + \rho B_i, \quad (A-5)$$

$$\rho d/dt(e + u^2/2) = \partial/\partial x_j(u_i \tau_{ji}) - \partial J_i/\partial x_i + \rho u_i B_i, \quad (A-6)$$

where ρ = material density, τ_{ji} is a component of the stress tensor (positive in tension), B_i is the body force per unit mass, u_i is the material velocity as measured in the (stationary) laboratory coordinate system, e is the internal energy per unit mass, J_i is a component of non-mechanical energy (heat) flux, and x_i are the Cartesian spatial coordinates fixed in the laboratory (stationary) reference frame. All variables are functions of x_i and t (time): $\partial/\partial t$ is taken at constant x_i , $\partial/\partial x_i$ is taken at constant x_j ($j \neq i$) and t , and $d/dt = \partial/\partial t + u_i \partial/\partial x_i$. When transformations are made to other coordinate systems, care will be exercised to define precisely which variables are held constant when taking a partial derivative. When d/dt is applied to a function of time alone, it represents the usual, simple time derivative. A dot over a variable also indicates a simple time derivative.

Under normal conditions the body forces are often ignored, although if one wanted to be thoroughly rigorous, terms involving B_i would be kept and it would make a difference whether or not a shock-wave experiment was conducted parallel or perpendicular to the earth's gravitational field. It is generally found that the magnitude of this effect is negligibly small in comparison with other forces involved in the shock process. Similar considerations apply to the heat flux term J_i in regions of continuous variation. Only in regions of extreme temperature gradient does the heat flux term become significant. In fact, the heat flux may actually control the shock thickness in the absence of other dissipative

effects and, within the shock front itself, cannot always be neglected.

Terms involving B_i and J_i are carried along in the following development to see how they enter the analysis in a formal way. In any particular application these terms would be kept or omitted depending on their magnitude in comparison to other additive quantities appearing in the equations.

The discussion in this report is restricted to one spatial dimension: $x = x_1$ with the lone non-zero material velocity component being $u = u_1$. The volume V is contained between two Lagrangian coordinates X_b and X_a : $X_b < X_a$. The Eulerian position and material velocity coordinates of the one dimensional boundaries to the volume integrals in Eqs. (A-1) - (A-3) are

$$a(t) = x(X_a, t); u(a(t), t) = (\partial x / \partial t)_{X_a} = \dot{a}(t) \quad , \quad (A-7)$$

$$b(t) = x(X_b, t); u(b(t), t) = (\partial x / \partial t)_{X_b} = \dot{b}(t) \quad . \quad (A-8)$$

The compressive normal stress $\sigma = -\tau_{11}$ is also used and the subscript 1 is dropped from all vector quantities. In this case Eqs. (A-1) - (A-6) become

$$d/dt \int_{b(t)}^{a(t)} \rho dx = 0 \quad , \quad (A-9)$$

$$d/dt \int_{b(t)}^{a(t)} \rho u dx = \sigma(b(t), t) - \sigma(a(t), t) + \int_{b(t)}^{a(t)} \rho B dx \quad , \quad (A-10)$$

$$d/dt \int_{b(t)}^{a(t)} \rho (e + u^2/2) dx = \dot{b}(t) \sigma(b(t), t) - \dot{a}(t) \sigma(a(t), t)$$

$$+ J(b(t), t) - J(a(t), t) + \int_{b(t)}^{a(t)} \rho u B dx, \quad (A-11)$$

and

$$\partial \rho / \partial t + \partial / \partial x (\rho u) = 0, \quad (A-12)$$

$$\rho du/dt = - \partial \sigma / \partial x + \rho B, \quad (A-13)$$

$$\rho d/dt (e + u^2/2) = - \partial (\sigma u) / \partial x - \partial J / \partial x + \rho u B. \quad (A-14)$$

The variables ρ , u , σ , e , J , and B in Eqs. (A-9) - (A-14) are functions of x and t , whether explicitly stated or not.

The time derivatives of the integrals on the left-hand sides of Eqs. (A-9) - (A-11) are all of the form

$$d/dt \int_{b(t)}^{a(t)} \psi(x, t) dx,$$

which, in regions of continuous variation of ψ , can be expressed as

$$d/dt \int_{b(t)}^{a(t)} \psi(x, t) dx = \int_{b(t)}^{a(t)} (\partial \psi / \partial t) dx + \dot{a}(t) \psi(a(t), t) - \dot{b}(t) \psi(b(t), t). \quad (A-15)$$

Application of Eq. (A-15) to Eqs. (A-9)-(A-11) when the region $b(t) < x < a(t)$ contains a moving singular surface leads to the usual jump conditions.²⁶

We examine the jump conditions from another viewpoint by starting with the differential equations of motion for continuous flow in one spatial dimension, Eqs. (A-12)-(A-14), and make a transformation to the non-steady (wave propagation) coordinate system defined by

$$x' = x - \xi(t) , \quad (A-16)$$

$$t' = t , \quad (A-17)$$

where $\dot{\xi}(t) = U(t)$ is the wave velocity. The inverse transformation is

$$x = x' + \xi(t') , \quad (A-18)$$

$$t = t' . \quad (A-19)$$

Let f be one of the flow variables, which can be thought of as either a function of x, t or x', t' :

$$df = (\partial f / \partial x')_t dx' + (\partial f / \partial t')_x dt' , \quad (A-20)$$

from which we find

$$(\partial f / \partial x)_t = (\partial f / \partial x')_{t'} , \quad (A-21)$$

$$(\partial f / \partial t)_x = (\partial f / \partial t')_{x'} - \dot{\xi}(t) (\partial f / \partial x')_{t'} . \quad (A-22)$$

These transformations are applied to Eqs. (A-12)-(A-14) written in conservative form:

$$\partial \rho / \partial t + \partial \rho u / \partial x = 0 , \quad (A-23)$$

$$\partial \rho u / \partial t + (\partial / \partial x)(\sigma + \rho u^2) = \rho B , \quad (A-24)$$

$$(\partial / \partial t)[\rho(e + u^2/2)] + (\partial / \partial x)[\rho u(e + \sigma/\rho + u^2/2) + J] = \rho u B , \quad (A-25)$$

and we replace u by $u' + \dot{\xi}(t)$, where u' is the material velocity as measured in the moving coordinate system. For example, if material ahead of the shock in the laboratory coordinate system is stationary and $u = 0$, then $u' = -\dot{\xi}(t)$; in the moving coordinate system material is seen streaming in the negative direction with a speed equal to the shock velocity.

Equation (A-23) becomes (it is clear from the context which variables are being held constant)

$$(\partial/\partial t' - \dot{\xi}(t)\partial/\partial x')\rho + (\partial/\partial x')[\rho(u' + \dot{\xi}(t))] = 0, \quad (A-26)$$

or

$$\partial\rho/\partial t' + \partial/\partial x'(\rho u') = 0. \quad (A-27)$$

Applying the same transformation to Eq. (A-24) we find

$$\begin{aligned} &(\partial/\partial t' - \dot{\xi}(t)\partial/\partial x')[\rho u' + \rho\dot{\xi}(t)] + \\ &(\partial/\partial x')(\sigma + \rho[u' + \dot{\xi}(t)]^2) = \rho B, \end{aligned} \quad (A-28)$$

or

$$\partial(\rho u')/\partial t' + (\partial/\partial x')(\sigma + \rho u'^2) = \rho[B - \ddot{\xi}(t)]. \quad (A-29)$$

In a similar manner the energy equation, Eq. (A-25), becomes

$$\begin{aligned} &(\partial/\partial t')[\rho(e + u'^2/2)] + (\partial/\partial x')[\rho u'(e + \sigma/\rho + u'^2/2) + J] \\ &= \rho u'[B - \ddot{\xi}(t)]. \end{aligned} \quad (A-30)$$

In the x', t' coordinate system, large gradients result in unbounded values of terms involving $(\partial f/\partial x')_{t'}$. The time derivatives $(\partial f/\partial t')_{x'}$,

are all bounded. This is not true in the x, t coordinate system as shown by Eqs. (A-21) and (A-22) in which the ill-behaved term, $(\partial f / \partial x')_t$, influences both $(\partial f / \partial x)_t$ and $(\partial f / \partial t)_x$. Equations (A-27), (A-29), and (A-30) are all of the form

$$(\partial / \partial t')_x, \rho \phi + (\partial / \partial x')_t, q = R(x', t') \quad . \quad (A-31)$$

Let the mathematical discontinuity be located at $x' = 0$ and integrate Eq. (A-31) across the discontinuity from $x' = x'_b$ (behind) to $x' = x'_a$ (ahead) such that $x'_b < 0 < x'_a$ (at constant t'):

$$\int_{x'_b}^{x'_a} (\partial / \partial t')_x, \rho \phi \, dx' + \int_{x'_b}^{x'_a} dq = \int_{x'_b}^{x'_a} R(x', t') \, dx' \quad . \quad (A-32)$$

It has been already pointed out that the term involving $(\partial / \partial t')_x$, is bounded. If the term involving $R(x', t')$ is also bounded, that is, if $B-\ddot{\xi}(t)$ in Eqs. (A-29) and (A-30) is bounded (no infinite body forces or infinite shock accelerations), then the first and last integrals in Eq. (A-32) vanish as $x'_b \rightarrow 0$ from below and $x'_a \rightarrow 0$ from above. This gives

$$q_b - q_a = 0 \quad (A-33)$$

as the jump condition across a non-steady mathematical discontinuity in the flow variables. Referring to Eqs. (A-27), (A-29), and (A-30) and the relationship $u' = u - D(t)$, where $D(t) = \dot{\xi}(t)$, we find:

$$\begin{array}{ll} \text{mass} & q = \rho u', \\ \text{momentum} & q = \sigma + \rho u'^2, \end{array}$$

energy $q = \rho u' (e + \sigma/\rho + u'^2/2) + J.$

These substitutions into Eq. (A-33) give the usual results (for a mathematical discontinuity):

$$\rho_b u'_b = \rho_a u'_a, \quad (A-34)$$

$$\sigma_b - \sigma_a = \rho_a u'^2_a - \rho_b u'^2_b = \rho_a u'_a (u'_a - u'_b), \quad (A-35)$$

$$\rho_a u'_a (e_b - e_a + [u'^2_b - u'^2_a]/2) + u'_b \sigma_b - u'_a \sigma_a = J_a - J_b \quad (A-36)$$

As expected, the non-steady (accelerating or decelerating) nature of a mathematically discontinuous shock has no influence on the jump conditions. However, when a shock acquires a finite duration wave structure (i.e., risetime) this is not true. For waves of finite structure, departure from the ideal jump conditions depends on integrals of the form

$$\int_{x'_b}^{x'_a} (\partial \rho / \partial t')_{x'} dx',$$

$$\int_{x'_b}^{x'_a} \left\{ (\partial / \partial t')_{x'} \rho u' + \rho [\ddot{\xi}(t) - B] \right\} dx', \text{ and}$$

$$\int_{x'_i}^{x'_a} \left\{ (\partial / \partial t')_{x'} \left[\rho (e + u'^2/2) \right] + \rho u' [\ddot{\xi}(t) - B] \right\} dx'.$$

The magnitude of this effect is small for shock thicknesses less

than $\sim 1\mu\text{m}$ and reasonable values of $\ddot{\xi}(t) - \ddot{U}(t)$, but this involves the question of how well discontinuous flow approximates the shock process in a particular real physical situation. Each case must be investigated individually.

APPENDIX B:

CHARADE INDEXING, VARIABLE DEFINITION, AND INPUT

(A) INDEXING

The left boundary (located at $i=1$ and $x=x_0$) has a time-dependent particle velocity applied according to the function subroutine UZERO. The right boundary (located at $i=ibnd$ and $x=xbnd$) defines the right limit of the material being studied. Cells between $i=ibnd$ ($x=xbnd$) and $i=imax$ ($x=xmax$) define the window material (when used). The logical variable FRESURF determines whether a window is present (FRESURF=.false. when window is present, FRESURF=.true. when window is absent). Here $ibnd$ and $imax$ determine the spatial increments $dx1$ (sample material between $i=1$ and $i=ibnd$) and $dx2$ (window material between $i=ibnd$ and $i=imax$):

$$dx1 = (xbnd - x0) / (ibnd - 1)$$
$$dx2 = (xmax - xbnd) / (imax - ibnd)$$

(B) DEFINITION OF VARIABLES

Twenty-one variables are read in via namelist. These are defined as follows:

x_0 - left boundary position. In general a non-zero value should be used. This will guard against division by zero when cylindrical and spherical options are eventually included.

u_0 - initial left boundary velocity

α - 0. (planar geometry)
1. (cylindrical geometry)
2. (spherical geometry)
Currently only planar geometry is coded for use.

t_{max} - maximum problem time (i.e., microseconds)

$courant$ - factor in Courant stability condition -
 $dt = courant * dx / cmin$. Currently a value of 0.5 is used throughout.

$xbnd$ - see INDEX figure (above)

$ibnd$ - see INDEX figure (above)

$xmax$ - see INDEX figure (above)

$imax$ - see INDEX figure (above)

$iprint$ - print cycle for hardcopy output file OUTCHA

$ida(mat)$ - eos identification for initial phase of
mat = 1 (target) or mat = 2 (window)
(1) isotropic solid (6hisotrp)
(2) 100 cubic crystal (6hcubl00)
(3) EOSMIX hom1 (4hhom1)
(4) EOSMIX hom2 (4hhom2)
(5) EOSMIX bkW (3hbkW)
(6) EOSMIX jw1 (3hjw1)

*NOTE: for (1) and (2), only a single phase is used.
For (3)-(6), a second phase may be present.

$idb(mat)$ - eos identification for second phase of
mat = 1 (sample) or mat = 2 (window)
see (3)-(6) above for choices

$eosa(i,mat)$ - eos constants for initial phase of
mat = 1 (sample) or mat = 2 (window)
see DEFINITION OF EOS CONSTANTS below ($i=1,25$)

$eosb(i,mat)$ - eos constants for second phase of
mat = 1 (sample) or mat = 2 (window)
for use with EOSMIX subroutine only ($i=1,25$)

rc(1) - rate constants for material model (sample).
 see MTS MATERIAL MODEL below for specific assignment
 (i=1,12)
 for i=13-15, rc(i) is used for other control
 ngage - number of imaginary Lagrangian gages for output
 igage(i) - index locations of gages (i=1,ngage)
 fresurf - logical variable defining presence of free surface
 (.true.) or window (.false.)
 dxflyer - thickness of flying plate (symmetric impact)
 for dxflyer > 99., constant left vel. is used
 c0 - zero pressure sound speed for flying plate
 (symmetric impact)
 s - shock velocity / particle velocity slope for
 flying plate (symmetric impact)

Twenty variables (index=n) are stored at each of the mesh points
 (dimensioned for 2000, index=i):

v(n,i) - general mesh variable (t = time)
 v2(n,i) - general mesh variable (new t = time + dt)
 va(n) - shock-front mesh variable (t = time)
 vakeep(n) - temporary storage for for va(n)
 vb(n) - shock-front mesh variable (new t = time + dt)
 vl(n) - interpolated variable, left
 vr(n) - interpolated variable, right
 vbnd(n) - bndry variable (to right of i=ibnd, t = time)
 v2bnd(n) - bndry variable (to right of i=ibnd, t = time + dt)
 vshnd(n) - shock-front mesh variable to right of bndry

 variable (1) - specific volume
 variable (2) - internal energy per unit mass
 variable (3) - mass fraction unreacted he - when used
 variable (4) - longitudinal stress - or pressure for fluid
 variable (5) - mean bulk temperature
 variable (6) - particle velocity
 variable (7) - dp/dv, EOSMIX
 variable (8) - dp/de, EOSMIX
 variable (9) - dp/dw, EOSMIX
 variable (10) - lagrangean sound speed - c1
 variable (11) - dp/du on hugoniot
 variable (12) - dT/de, EOSMIX
 variable (13) - dT/dw, EOSMIX
 variable (14) - shear stress - 0.5*(sig11 - sig22)
 variable (15) - not used
 variable (16) - not used
 variable (17) - mechanical threshold stress - sigma-hat
 variable (18) - psi
 variable (19) - not used
 variable (20) - not used

(C) DEFINITION OF RATE CONSTANTS rc(i,m)
 MTS MATERIAL MODEL (WITH VISCOUS DRAG)

m - material number (= 1 for sample)

rc(1,m) - psidot-zero
 rc(2,m) - mb0/b
 rc(3,m) - delta-g/k
 rc(4,m) - limiting dislocation velocity (shear-wave speed)
 rc(5,m) - sigma-hat-a
 rc(6,m) - first constant in expression for theta-0
 rc(7,m) - 2 nd constant in expression for theta-0
 rc(8,m) - 3 rd constant in expression for theta-0
 rc(9,m) - sigma-hat-s0
 rc(10,m) - epsilon-dot-s0
 rc(11,m) - k/(a*mu*b**3) units of 1/K
 rc(12,m) - lambda*nu0

the following are independent of the MTS model:

rc(13,m) - control on 2nd-order xpath integration
 rc(14,m) - time duration for lft bndry vel.
 rc(15,m) - time lft bndry (i=1) reaches zero vel.

(D) DEFINITION FOR EOS CONSTANTS

***** (1) For an isotropic solid (id=isotrp)

```

c11 = lambda + 2.*mu
c12 = lambda
c44 = mu = 0.5*(c11 - c12)
c111 = 2.*zeta + 4.*xi
c112 = 2.*zeta
c123 = 2.*zeta - 2.*xi + nu
c144 = 0.5*(c112 - c123)
c155 = 0.25*(c111 - c112)
c456 = 0.125*(c111 - 3.*c112 + 2.*c123)

eosa( 1,1) = c11           (outp)
eosa( 2,1) = c12, lambda  (inp and outp)
eosa( 3,1) = c44, mu      (inp and outp)
eosa( 4,1) = c111         (outp)
eosa( 5,1) = c112, 2.*zeta (inp and outp)
eosa( 6,1) = dbdt, dB/dT  (inp)
              = c123      (outp)
eosa( 7,1) = dmudt, dmu/dT (inp)
              = c144      (outp)
eosa( 8,1) = bsp, dB/dP   (inp)
              = c155      (outp)
eosa( 9,1) = mup, dmu/dP  (inp)
              = c456      (outp)
eosa(10,1) = not used
eosa(11,1) = gamma (Gruneisen coefficient)
eosa(12,1) = cp (mbar*cm**3/gm*K)
eosa(13,1) = v0 (cm**3/gm)
eosa(14,1) = alpha (linear expansion coefficient)
eosa(15,1) = tempa (temp at which bsp and mup are meas.)

```

***** (2) For cubic symmetry and wave propagation in (100) direction
second and third-order elastic moduli at zero pressure
are read in directly (id=cub100)

```

eosa( 1,1) = c11
eosa( 2,1) = c12
eosa( 3,1) = c44
eosa( 4,1) = c111
eosa( 5,1) = c112
eosa( 6,1) = c123
eosa( 7,1) = c144
eosa( 8,1) = c155
eosa( 9,1) = c456

```

***** (3) For EOSMIX (window material)

eosa(i,mat) and eosb(i,mat) for
mat = 1 (target) or mat = 2 (window)
and index i=1,25 listed below

```

hom solid parameters, sin style (id=4hhom1)
(1) s1      (2) s      (3) vsw      (4) c1
(5) i       (6) f      (7) g        (8) h
(9) v0      (10) j      (11) gamma   (12) cv
(13) t0     (14) alpha  (15) spa     (16) usp
(17) plap   (18) p0    (19) 2/3 y0  (20) mu
(21) vmax   (22) spall p (23) minv    (24) vmin
(25)

```

```

hom solid parameters, eos only (id=4hhom2)
(1) s1      (2) s      (3) vsw      (4) c1
(5) i       (6) f      (7) g        (8) h
(9) v0      (10) j      (11) gamma   (12) cv
(13) t0     (14) alpha  (15) t0      (16) p0
(17) vmmin  (18) vmmax

```

ln(th) = f + lnvs*(g + lnvs*(h + lnvs*(i + lnvs*j)))

```

hom bkw gas parameters (id=3hbkw)
(1) a       (2) b       (3)          (4) d
(5) e       (6) k       (7) k      (8) m
(9) n       (10) o      (11) q     (12) r
(13) s      (14) t      (15) u     (16) cv
(17) z      (18) vmin   (19) vmax

```

vg polynomial domain = (vgmin,vgmax) interval
ln(pi) = a + lnvg*(b + lnvg*(c + lnvg*(d + lnvg*e)))
ln(ez) = k + lnpi*(k + lnpi*(m + lnpi*(n + lnpi*o)))

```

ei = ez - z
ln(ti) = q + lnvg*(r + lnvg*(s + lnvg*(t + lnvg*u)))
-1/beta = r + lnvg*(2s + lnvg*(3t + lnvg*4u))

jwl parameters      (id=3hjwt)
(1) a                (2) b                (3)                (4) r1
(5) r2               (6) w                (7) e0               (8) rho
(9) tz               (10) cv              (11) vmin

```

(E) OTHER MAJOR VARIABLES (i refers to spatial index -
n refers to variable number)

```

dxl(i)  - dx to left (index i)
dxr(i)  - dx to right (index i)
xlag(i) - Lagrangian position (index i)
ts(i)   - shock arrival time (index i)
vlavg(n) - average vl(n) over dt: 2nd order
vravg(n) - average vr(n) over dt: 2nd order
matl(i) - material number to left (index i)
matr(i) - material number to right (index i)
          material number = 1 for sample
          material number = 2 for window
c1l(i)  - Lagrangian wave speed to left (index i)
c1r(i)  - Lagrangian wave speed to right (index i)
zl      - impedance to the left
zr      - impedance to the right

time    - current time
dt      - time increment

xa      - initial shock position (t = time)
ishoka  - initial shock index (t = time)
mata    - material containing shock (t = time)
xakeep  - temporary storage for xa

xb      - final shock position (t = time + dt)
ishokb  - final shock index (t = time + dt)
matb    - material containing shock (t = time + dt)

dsdx    - longitudinal stress gradient
us      - shock velocity
matjump - 0; shock does not change material
          1; shock jumps from mata to matb
ijump   - 0; shock does not change index
          1; shock index changes by one

```

```

1 c$#####
2 c$ CHARADE INPUT DECK FOR SAMPLE PROBLEM
3 c$ SHOWN IN FIGURES 6 AND 7
4 c$#####
5 c$
6 c$ * * * read label0 (5a8)
7 c$ * * * label for post-processing pix graphics output
8 30kbar mts 1*nu0=5
9 c$ * * * read label (10a8)
10 c$ * * * identification label for graphs
11 30kbar mts 1*nu0=5
12 c$ * * * read ipub (a8)
13 c$ logical variable for post-processing pix graphics output:
14 c$ ipub=nopixpub (no pix graphics)
15 c$ * * * ipub=pixpub (pix graphics)
16 nopixpub
17 p$datain x0=1.0, u0=0.0085, alpha=0.0, tmax=0.050,
18 xbnd=2.0, ibnd=601, xmax=2.3, imax=801,
19 iprint=1, nplate = 10,
20 courant=0.5,
21 fresurf = .true., dxflyer = 1.0e+06,
22 nage=5, igage(1)=21,41,61,81,101,
23 ida(1)=6hisotrp,
24 c$ * * * input for copper is given in the following order:
25 c$ c0=bulk sound speed, slope-us-up slope
26 c$ 1=not used, 2=lamdba, 3=mu, 4=not used, 5=c112,
27 c$ 6=dB/dT, 7=dmu/dT, 8=dB/dP, 9=dmu/dP, 10=not used,
28 c$ * * * 11=gamma, 12=cp(mbar*cm**3/gm K), 13=v0, 14=alpha, 15=tempa
29 c0=0.394, slope=1.49,
30 eos(1,1) = 0., 1.077, 0.4432, 0., -11.,
31 -0.2227e-03, -0.2e-03, 5.28, 1.30, 0.,
32 1.99, 3.857e-06, 0.1119, 16.5e-06, 300.,
33 c$ * * * rate constants: 1=psi-dot-zero, 2=MB0/b, 3=delta-G/k,
34 c$ 4=shear wave speed, 5=sigma-a-hat,
35 c$ 6,7,8=constants in expression for theta=0, 9=sigma-hat-s0,
36 c$ 10=epsilon-dot-s0, 11=k/A*mu*b**3, 12=lamdba*nu0,
37 c$ 13=2nd order control, 14=time (t1) duration for left bndry,
38 c$ * * * 15=time (t2) left bndry vel.=0 (t1 & t2 calc. int.)
39 rc(1,1)= 00., 1.686e-02, 8.107e+04, 0.217, 0.0004,
40 0.02556, 1.2e-04, 0.34, 0.009, 6.2e+04,
41 6.325e-05, 5., 0.5, 2*0,
42 rc(1,2)=1.0, 14*0,
43 ida(2)=4hhom2,
44 c$ * * * eos for plexiglas (if used as window)
45 c$ * * * when fresurf = .true., this eos is ignored
46 eos(1,2)= 0.243, 0.00, 3*0,
47 +5.29380243506e+00, -4.24950371368e+00, -1.55055576332e+01,
48 -3.08638075572e+01, -1.46708193739e+01,
49 2.157, 0.35, 0.8475, 1.0e-04, 300.0, 0.00,
50 0.50, 1.00 $

```

APPENDIX C:
CHARADE LISTING

```

1 *select printlog=none
2 *file name=cha
3   program cha( i, tty, outcha, pixout, tape1=i, tape3=tty,
4     1 tape4=outcha )
5
6 c      #####
7 c      #   this particular version is called chamts      #
8 c      #   and is used explicitly to calculate the shock   #
9 c      #   response of copper using the rate model of      #
10 c     #   P. S. Follansbee and U. F. Kocks                #
11 c     #   2nd order solution at normal interior points    #
12 c     #   input file to be used is npchamts               #
13 c     #####
14
15 c* * * CHARADE
16 c* * * a characteristic code for
17 c* * * (1) elastic-plastic deformation
18 c* * * (2) shock-wave initiation of condensed explosives
19 c* * * (3) phase transformations
20 c* * * in
21 c* * * planar      (alpha = 0)
22 c* * * cylindrical (alpha = 1)
23 c* * * and spherical (alpha = 2)      geometry
24 c* * * only planar geometry is programmed in this version of CHARADE
25 c* * *
26 c* * * by J. N. Johnson and D. L. Tonks
27 c* * * Los Alamos National Laboratory
28 c* * * Begun:      October 1986
29 c* * * Current version: January 1990
30 c* * * logical bstop, left, fresurf, noshok, pub
31 c* * * integer wkday, cdate
32 c* * *
33 c* * * character tlabel*80, ylabel*80, title*80, title1*40, title2*40
34 c* * *      ,xlabel*80
35 c* * *
36 c* * * dimension dxi(2000), dxr(2000),
37 c* * *      xlag(2000), ts(2000), vavg(20), vavg(20),
38 c* * *      v(20,2000), v2(20,2000), vs(20,2000),
39 c* * *      va(20), vb(20), vl(20), vr(20), vakeep(20),
40 c* * *      vbnd(20), v2bnd(20), vsbnd(20), xmm(2), ymm(2),
41 c* * *      igage(5), tgage(3500,5), ygage(3500,5,6), npts(10)
42 c* * *      ,nlistp(3), label(10), label(10), x00(2), y00(2)
43 c* * *      ,ylabel(2,6),nsnap(5),xsnap(2000,5),ysnap(2000,5,5)
44 c* * *      ,xlabel(2),tlabel(2), title(2)
45
46 c* * * definition of variables
47
48 c   variable ( 1) = specific volume
49 c   variable ( 2) = internal energy per unit mass
50 c   variable ( 3) = mass fraction unreacted he - when used
51 c   variable ( 4) = longitudinal stress - or pressure for fluid
52 c   variable ( 5) = mean bulk temperature
53 c   variable ( 6) = particle velocity
54 c   variable (10) = lagrangean sound speed - cl
55 c   variable (11) = dp/du on hugoniot
56 c   variable (14) = shear stress - 0.5*( sig11 - sig22 )
57 c   variable (15) = not used
58 c   variable (16) = not used
59 c   variable (17) = mechanical threshold stress - sigma-hat
60 c   variable (18) = psi
61 c   variable (19) = not used
62
63 c* * *
64 c* * * real nu, mupad
65 c* * * common/block1/ matr(2000),matl(2000), cll(2000), clr(2000),
66 c* * *      #      ida(2),eosa(25,2), idb(2), eosb(25,2)
67 c* * * common/block2/ rc(15,2), nv
68
69 c* * * data x00, y00 / 0., 0., 0., 0. /
70 c* * * data nlistp / 3, 4, 5 /
71
72 c* * * definition of rate constants rc(i,m)
73
74 c   rc(1,m) = psidot-zero
75 c   rc(2,m) = mb0/b
76 c   rc(3,m) = delta-g/k
77 c   rc(4,m) = limiting dislocation velocity (shear-wave speed)
78 c   rc(5,m) = sigma-hat-a
79 c   rc(6,m) = first constant in expression for theta-0
80 c   rc(7,m) = 2 nd constant in expression for theta-0

```

```

81 c rc(8,m) = 3 rd constant in expression for theta-0
82 c rc(9,m) = sigma-hat-s0
83 c rc(10,m) = epsilon-dot-s0
84 c rc(11,m) = k/(a*mu*b**3) units of 1/K
85 c rc(12,m) = lambda*nu0
86 c rc(13,m) = control on 2nd-order xpath integration
87 c rc(14,m) = time constant for left bndry
88 c rc(15,m) = max shear stress for pix plot
89
90 c* * * set up all problems with nonzero x0 (even for alpha = 0 )
91 c u0 = maximum particle velocity at left bndry
92 namelist/datain/ x0, u0, alpha, tmax, courant,
93 # xbnd, ibnd, xmax, imax, iprint,
94 # ida, eosb, idb, eosb, rc,
95 # nage, igage, fresurf, dxflyer,
96 # c0, slope
97 c* * *
98 c -----
99 call filerep
100 call namprt( 3, nlistp )
101 call today( wkday, cdate )
102
103 label0(1) = 5hcha87
104 label0(7) = cdate
105 10 read(1, '(5a8)') (label0(i), i=2,6)
106
107 read( 1, '(10a8)' ) label
108 pub = .f.
109 read( 1, '(a8)' ) ipub
110 if( ipub .eq. 6hpixpub ) pub = .t.
111
112 call pixup( lhu, label0, 5, pub, 'duplx', 1. )
113
114 c -----
115 c* * *
116 c* * * set-up and first dt
117
118 read ( 1, datain )
119 write( 4, datain )
120
121 czero = eosb(1,1)
122
123 if( ida(1) .ne. 6hisotrp ) go to 50
124
125 c * * * * for an isotropic solid
126 c c11 = lambda + 2.*mu
127 c c12 = lambda
128 c c44 = mu = 0.5*(c11 - c12)
129 c c111 = 2.*zeta + 4.*xi
130 c c112 = 2.*zeta
131 c c123 = 2.*zeta - 2.*xi + nu
132 c c144 = 0.5*(c112 - c123)
133 c c155 = 0.25*(c111 - c112)
134 c c456 = 0.125*(c111 - 3.*c112 + 2.*c123)
135 c
136 c eosb( 1,1) = c11 (outp)
137 c ( 2,1) = c12, lambda (inp and outp)
138 c ( 3,1) = c44, mu (inp and outp)
139 c ( 4,1) = c111 (outp)
140 c ( 5,1) = c112, 2.*zeta (inp and outp)
141 c ( 6,1) = dbdt, dB/dT (inp)
142 c = c123 (outp)
143 c ( 7,1) = dmudt, dmu/dT (inp)
144 c = c144 (outp)
145 c ( 8,1) = bsp, dB/dP (inp)
146 c = c155 (outp)
147 c ( 9,1) = mup, dmu/dP (inp)
148 c = c456 (outp)
149 c (10,1) = not used
150 c (11,1) = gamma
151 c (12,1) = cp (mbar*cm**3/gm*K)
152 c (13,1) = v0 (cm**3/gm)
153 c (14,1) = alpha (linear expansion coefficient)
154 c (15,1) = tempa (temp at which bsp and mup are meas.)
155
156 c * * * bsp and mup are at constant temperature
157 c * * * bspad and mupad are at constant entropy
158
159 tfac = eosb(15,1)*2.*eosb(14,1)*eosb(13,1)/eosb(12,1)
160 bspad = eosb(8,1) + eosb(6,1)*tfac

```

```

161      mupad = eosa(9,1) + eosa(7,1)*tfac
162
163      zeta = 0.5*eosa(5,1)
164      bs = eosa(2,1) + 2.*eosa(3,1)/3.
165      nu = 1.5*(-3.*bs*bspad - 6.*zeta)
166      xi = (-3.*bs*mupad - 3.*bs - eosa(3,1) + .5*nu)/3.
167
168      eosa( 1,1) = eosa(2,1) + 2.*eosa(3,1)
169 c      eosa( 2,1) = unchanged
170 c      eosa( 3,1) = unchanged
171      eosa( 4,1) = 2.*zeta + 4.*xi
172 c      eosa( 5,1) = unchanged
173      eosa( 6,1) = 2.*zeta - 2.*xi + nu
174      eosa( 7,1) = 0.5*( eosa(5,1) - eosa(6,1) )
175      eosa( 8,1) = 0.25*(eosa(4,1) - eosa(5,1) )
176      eosa( 9,1) = 0.125*( eosa(4,1) - 3.*eosa(5,1) + 2.*eosa(6,1) )
177
178      go to 60
179
180 50 continue
181
182      if( ida(1) .ne. 6hcub100 ) go to 70
183
184 c * * * * for cubic symmetry and wave propagation in (100) direction
185 c      second and third-order elastic moduli at zero pressure
186 c      are read in directly
187 c      eosa( 1,1) = c11
188 c      eosa( 2,1) = c12
189 c      eosa( 3,1) = c44
190 c      eosa( 4,1) = c111
191 c      eosa( 5,1) = c112
192 c      eosa( 6,1) = c123
193 c      eosa( 7,1) = c144
194 c      eosa( 8,1) = c155
195 c      eosa( 9,1) = c456
196
197 60 continue
198
199 c the zero pressure elastic sound speed is
200      eosb(1,1) = sqrt( eosa(13,1)*eosa(1,1) )
201 c the constant alpha defined by asay is
202      eosb(2,1) = - 0.5*(3. + eosa(4,1)/eosa(1,1) )
203 c the other constants in the equations for bij are
204      eosb(3,1) = 4.*eosa(1,1) + eosa(4,1)
205      eosb(4,1) = 5.*eosa(1,1) - eosa(2,1) + eosa(4,1) - eosa(5,1)
206      eosb(5,1) = eosa(1,1) - eosa(2,1) - eosa(5,1)
207      eosb(6,1) = eosa(1,1) - 2.*eosa(2,1) - 0.5*eosa(5,1) +
208      # 0.5*eosa(6,1)
209      eosb(7,1) = 0.5*( eosa(1,1) + eosa(2,1) + eosa(5,1) -
210      # eosa(6,1) )
211      eosb(8,1) = eosa(1,1) - eosa(2,1) - eosa(5,1)
212      eosb(9,1) = 0.5*( 5.*eosa(1,1) - eosa(2,1) + eosa(4,1) -
213      # eosa(5,1) )
214      eosb(10,1) = 2.*eosa(2,1) - eosa(6,1)
215      eosb(11,1) = 0.5*( eosa(1,1) - 5.*eosa(2,1) - 2.*eosa(5,1) +
216      # 2.*eosa(6,1) )
217      eosb(12,1) = 0.5*( eosa(1,1) - eosa(2,1) )
218      eosb(13,1) = 0.5*( 3.*eosa(1,1) + eosa(2,1) + eosa(4,1) -
219      # eosa(5,1) )
220
221      czero = eosb(1,1)
222
223      write( 4, 9030) ida(1), idb(1), ida(2), idb(2)
224
225      write( 4, 9040) (i, eosa(i,1), eosb(i,1), eosa(i,2), eosb(i,2),
226      # i = 1, 25 )
227
228 70 continue
229
230      nv = 20
231      ipr = 0
232
233      dx1 = ( xbnd - x0 )/( ibnd - 1 )
234      dx2 = ( xmax - xbnd ) / ( imax - ibnd )
235
236      do 100 i = 1,imax
237      matl(i) = 1
238      matr(i) = 1
239      dxi(i) = dxi
240      dxr(i) = dx1

```



```

241      xlag(i) = (i - 1)*dx1 + x0
242
243      if( i .lt. ibnd ) go to 90
244      if( i .eq. ibnd ) go to 80
245
246      matl(i) = 2
247      dx1(i) = dx2
248      xlag(i) = xbnd + ( i - ibnd )*dx2
249
250      80 matr(i) = 2
251      dxr(i) = dx2
252
253      90 continue
254
255      m = matl(i)
256      v( 1,i) = eosa( 13,m)
257      v( 2,i) = 0.
258      v( 3,i) = 1.
259      v( 4,i) = 0.
260      v( 5,i) = 300.
261      v( 6,i) = 0.
262      v(14,i) = 0.
263      v(15,i) = 0.
264      v(16,i) = 0.
265      v(17,i) = rc(5,m)
266      v(18,i) = 0.
267      v(19,i) = 0.
268      call clag( v(1,i), ida(m), eosa(1,m), idb(m), eosb(1,m) )
269      cll(i) = v(10,i)
270      clr(i) = v(10,i)
271
272      100 continue
273
274      cmin = clr(1)
275
276      clr(imax) = 0.
277      clbnd = cll( ibnd + 1 )
278      clr(ibnd) = clbnd
279      vbnd(1) = v(1,ibnd+1)
280      vbnd(2) = 0.
281      vbnd(3) = 1.
282      vbnd(4) = 0.
283      vbnd(5) = 300.
284      vbnd(6) = 0.
285      vbnd(14) = 0.
286      vbnd(15) = 0.
287      vbnd(16) = 0.
288      vbnd(17) = 0.
289      vbnd(18) = 0.
290      vbnd(19) = 0.
291
292      do 110 n = 1,nv
293      v2bnd(n) = vbnd(n)
294
295      do 105 i = 1,imax
296      v2(n,i) = v(n,i)
297      vs(n,i) = v(n,i)
298      105 continue
299
300      110 continue
301
302      do 130 l = 1,ngage
303      npts(l) = 0
304
305      do 120 i = 1,500
306      tgage(i,1) = 0.
307      ygage(i,1,1) = 0.
308      120 continue
309
310      130 continue
311
312      c
313      c
314      c* * * * end: set-up
315      c* * * * first dt - calculate shock advance and i-1
316
317      noshok = .false.
318      ishok = 1
319      ishoka = 1
320      ishokb = 1

```

```

321      xa = x0
322      time = 0.
323      va(6) = uzzero( time, u0 )
324      mata = 1
325      call hugo( va, ida(mata), eosa(1,mata), idb(mata), eosb(1,mata) )
326      call clag( va, ida(mata), eosa(1,mata), idb(mata), eosb(1,mata) )
327
328      cll(1) = va(10)
329      clr(1) = va(10)
330      cmax = clr(1)
331
332
333      do 140 n = 1,nv
334      v(n,1) = va(n)
335      vs(n,1) = va(n)
336 140 continue
337
338      ts(1) = 0.
339
340      us = czero
341      if( va(6) .ne. 0. ) us = eosa(13,mata)*va(4)/va(6)
342      xb = 0.5*dx1 + x0
343      time = 0.5*dx1/us
344      icycle = 1
345      dt = time
346
347  c * * * set up finite-thickness flyer constants
348      usbulk = c0 + slope*u0
349      rc(14,1) = dxflyer*( 1./usbulk + 1./cmax )
350      rc(15,1) = dxflyer*( 1./usbulk + 1./c0 )
351      if( dxflyer .gt. 99. ) rc(14,1) = 0.
352
353      write(3, 9001) usbulk, cmax, c0, va(4)
354 9001 format(5x, *usbulk = *, f8.5, * cmax = *, f8.5, * c0 = *, f8.5/
355      #      5x, *shock pressure = *, f8.5)
356
357      isnap = 0
358      tsnap = tmax/5.1
359      tsnap2 = time
360
361  c * * * special for symmetric impact
362      dsdx = 0.
363
364      con1 = ( clr(1)/us )**2
365      con2 = us*(con1 - 1.)
366      con3 = alpha*va(1)*us/(xa * eosa(13,mata) )
367
368      delta = 2.*va(14)
369
370      if( ida(1) .eq. 6hcub100 .or. ida(1) .eq. 6hisotrp ) then
371      eps = 1. - va(1)/eosa(13,mata)
372
373      denom = 1. + con1*( 1. + 1.5*eosb(2,1)*eps )/
374      #      ( 1. + 2.*eosb(2,1)*eps )
375
376      else
377
378      denom = 1. + con1*us/( us + eosa(2,mata)*va(6) )
379
380      endif
381
382      dsdtsh = ( frate(mata,va) - con2*dsdx - con3*delta )/denom
383
384      vb(4) = va(4) + dsdtsh*dt
385      vb(6) = eosa(13,mata)*vb(4)/us
386      call hugo( vb, ida(mata), eosa(1,mata), idb(mata), eosb(1,mata) )
387
388      tstar = clr(1)*dt/( us + clr(1) )
389      fr = tstar/dt
390      omfr = 1. - fr
391      dtr = dt - tstar
392
393      do 200 n = 1,nv
394      vr(n) = omfr*va(n) + fr*vb(n)
395 200 continue
396
397      zr = clr(1)/eosa(13,mata)
398
399      v2(6,1) = uzzero( time, u0 )
400      v2(4,1) = vr(4) + zr*( v2(6,1) - vr(6) ) + irate( mata, vr )*dtr

```

```

401
402      i = 1
403      left = .true.
404      call xpath( 1., i, left, v(1,i), v2(1,i), dt )
405
406      ib = ishokb
407
408      go to 6000
409
410 c* * * * end: first dt
411 c* * *
412 c* * * * return point for time step (1000)
413
414
415      1000 continue
416
417 c          START OF NEW TIME CYCLE * * * * *
418
419 c* * *
420 c* * * * find minimum dt and then advance the time step
421 c* * *
422
423 c* * * * calculate and store lagrangean sound speeds at all
424 c* * * * mesh points. calculated from clag and stored as cll(i)
425 c* * * * for cl to left of i and clr(i) for cl to right of i
426 c* * * * with one additional variable clbnd for the sound speed
427 c* * * * immediately to the right of the boundary between
428 c* * * * mat = 1 and mat = 2
429
430      do 1010 i = 1, ishok
431      mat = matl(i)
432      call clag( v(1,i), ida(mat), eosa(1,mat), idb(mat), eosb(1,mat) )
433      cll(i) = v(10,i)
434      clr(i) = v(10,i)
435      1010 continue
436
437      clr(imax) = 0.
438      if( ishok .lt. ibnd ) go to 1020
439
440      mat = 2
441      call clag( vbnd, ida(mat), eosa(1,mat), idb(mat), eosb(1,mat) )
442      clbnd = vbnd(10)
443      clr(ibnd) = clbnd
444
445      1020 continue
446
447      dtmin = dx2/clbnd
448
449      do 1100 i = 1, ishok
450      dtest = dx1(i)/c11(i)
451      if( dtest .lt. dtmin ) dtmin = dtest
452      1100 continue
453
454      dt = 0.5*dtmin
455      time = time + dt
456
457      if( time .ge. tmax ) go to 7000
458
459      icycle = icycle + 1
460      tsnap2 = tsnap2 + dt
461
462 c* * * * advance shock front (2000)
463
464      if( noshok ) go to 8000
465
466      ishoka = ifront( x0, xa, dx1, dx2, ibnd, xbnd )
467      mata = matr( ishoka )
468
469 c* * * * calculate d(sigma r)/dx at the shock front
470
471      if( ishoka .eq. 1 ) then
472      delx = xa - x0
473      sx = v(4,1)
474      go to 2010
475
476      elseif( ishoka .eq. ibnd ) then
477      delx = xa - xbnd
478      sx = vbnd(4)
479      go to 2010
480

```

```

481     elseif( ishoka .eq. ibnd+1 ) then
482         delx = 0.5*dx2 + xa - xlag(ibnd+1)
483         sx = 0.5*( vbnd(4) + v(4,ibnd+1) )
484
485     else
486
487         delx = xa - xlag(ishoka) + 0.5*dx1(ishoka)
488         sx = 0.5*( v(4,ishoka) + v(4,ishoka-1) )
489
490     endif
491
492 2010 continue
493
494     dsdx = ( va(4) - sx )/delx
495
496 c * * * * calculate new shock speed us and final lagrangian
497 c * * * * shock position xb
498
499     us = czero
500     if( va(6) .ne. 0. ) us= eosa( 13, mata )*va(4)/va(6)
501     xb = xa + us*dt
502
503     ishokb = ifront( x0, xb, dx1, dx2, ibnd, xbnd )
504     matb = matr( ishokb )
505
506 c * * * * save xa and va(n) for later use - restore values
507 c * * * * following advance of shock front
508
509     xakeep = xa
510     do 2020 n = 1,nv
511         vakeep(n) = va(n)
512 2020 continue
513
514     matjump = matb -mata
515     ijump = ishokb - ishoka
516
517 c * * * * find initial value of ds/dt on shock. In general
518 c * * * * denominator = 1 + rho0*us*(dup/dp)*(cl/us)**2
519 c * * * * For straight-line us-up fluid we have special case
520 c * * * * denominator = 1 + (us/(us+s*up))*(cl/us)**2
521 c * * * * delta = sigma(r) - sigma(theta)
522
523     dt2 = dt
524
525 2050 continue
526     con1 = ( clr(ishoka)/us )**2
527     con2 = us*(con1 - 1.)
528     con3 = alpha*va(1)*us/(xa * eosa(13,mata) )
529
530     delta = 2.*va(14)
531
532     if( ida(1) .eq. 6hcub100 .or. ida(1) .eq. 6hisotrp ) then
533         eps = 1. - va(1)/eosa(13,mata)
534
535         denom = 1. + con1*( 1. + 1.5*eosb(2,1)*eps )/
536 #           ( 1. + 2.*eosb(2,1)*eps )
537
538     else
539
540         denom = 1. + con1*us/( us + eosa(2,mata)*va(6) )
541
542     endif
543
544     dsdtsh = ( frate(mata,va) - con2*dsdx - con3*delta )/denom
545
546     if( matjump .lt. 1 ) go to 2400
547
548 c * * * * here is where we put special-case calculations for shock
549 c * * * * jump across boundary from mata to matb
550 c * * * * when fresurf = .true. set noshok (from .false. to .true. )
551
552 c * * * * let sbndl and ubndl be values of sigma-r and u-p immediately before
553 c * * * * interaction of leading shock with interface
554
555     if( fresurf ) noshok = .true.
556
557     dt1 = dt*( xbnd - xa )/( xb - xa )
558     dt2 = dt - dt1
559
560     sbndl = va(4) + dsdtsh*dt1

```

```

561      ubnd1 = eosa(13,mata)*sbnd1/us
562      vb(5) = ubnd1
563      call hugo( vb, ida(mata), eosa(1,mata), idb(mata), eosb(1,mata) )
564
565 c * * * integrate down isentrope of mata to intersect hugoniot
566 c * * * of matb. first set up initial conditions for integration
567
568      ubnd = ubnd1
569      volbnd = vb(1)
570      rhobnd = 1./volbnd
571      ebnd = vb(2)
572      sbnd = vb(4)
573      sbnd1 = sbnd
574      bstop = .false.
575
576      call clag( vb, ida(mata), eosa(1,mata), idb(mata), eosb(1,mata) )
577      zmata = vb(10)/eosa(13,mata)
578      call hugo( vb, ida(matb), eosa(1,matb), idb(matb), eosb(1,matb) )
579      if( fresurf ) vb(4) = 0.
580      zmatb = vb(4)/vb(6)
581      du = 0.01*( sbnd - vb(4) )/( zmata + zmatb )
582
583 2100 continue
584
585      ubnd = ubnd + du
586      rhobnd = rhobnd - (du/zmata)*rhobnd**2
587      ebnd = ebnd - (sbnd - 0.5*zmata*du)*(1./rhobnd - volbnd )
588      volbnd = 1./rhobnd
589      sbnd = sbnd - zmata*du
590
591      vs(1,ishokb) = volbnd
592      vs(2,ishokb) = ebnd
593      vs(6,ishokb) = ubnd
594      call clag( vs(1,ishokb), ida(mata), eosa(1,mata), idb(mata),
595      # eosb(1,mata) )
596      zmata = vs(10,ishokb)/eosa(13,mata)
597
598      vb(6) = ubnd
599      call hugo( vb, ida(matb), eosa(1,matb), idb(matb), eosb(1,matb) )
600      zmatb = vb(11)
601      if( fresurf ) vb(4) = 0.
602      if( fresurf ) zmatb = 0.
603
604      if( bstop ) go to 2200
605      if( vb(4) .lt. sbnd ) go to 2100
606
607      bstop = .true.
608      du = - ( vb(4) - sbnd )/( zmata + zmatb )
609      go to 2100
610
611 2200 continue
612
613      vs(4,ishokb) = vb(4)
614      ts(ishokb) = time - dt2
615
616      cll(ishokb) = vs(10,ishokb)
617      clr(ishokb) = vb(10)
618
619      if( fresurf ) go to 8000
620
621 c * * * now continue solution into matb: time step = dt2
622
623      do 2210 n = 1,nv
624      vsbnd(n) = vb(n)
625      vbnd(n) = vb(n)
626      v(n,ishokb) = vs(n,ishokb)
627      va(n) = vb(n)
628 2210 continue
629
630      zbar = ( sbnd1 - vsbnd(4) )/( vsbnd(6) - ubnd1 )
631      dsdx = dsdx*(eosa(13,mata)/eosa(13,matb))*(zmata+zbar)/
632      # (zmatb+zbar)
633      us = eosa(13,matb)*vsbnd(4)/vsbnd(6)
634      xa = xbnd
635      xb = xa + us*dt2
636      ishoka = ibnd
637      matjump = -1
638      mata = matb
639
640      go to 2050

```

```

641
642 2400 continue
643
644 c * * * * advance shock front solution
645
646     vb(4) = va(4) + dsdtsh*dt2
647     vb(6) = eosa(13,matb)*vb(4)/us
648
649 c * * * * calculate the remainder of variables at new shock
650 c * * * * position - with vb (6) known
651
652     call hugo( vb, ida(matb), eosa(1,matb), idb(matb), eosb(1,matb) )
653
654     if( ijump .eq. 0 .or. matjump .lt. 0 ) go to 2600
655
656 c * * * * set interpolated values at mesh points
657
658     fs = ( xlag(ishokb) - xa )/( xb - xa )
659     omfs = 1. - fs
660
661     do 2510 n = 1,nv
662     vs(n,ishokb) = omfs*va(n) + fs*vb(n)
663     v(n,ishokb) = vs(n,ishokb)
664 2510 continue
665
666     ts( ishokb ) = time - omfs*dt
667
668 c      WACKFIX*****
669 c      WACKFIX 11/25/88. J. Wackerle noticed instabilities near the
670 c      the shock front, and found that they had something to do with
671 c      the fact that I (JNJ) forgot to update the Lagrangian sound
672 c      speed at the index crossed by the shock (ishokb). This is
673 c      remedied here.
674
675     call clag( v(1,ishokb), ida(matb), eosa(1,matb), idb(matb),
676     #         eosb(1,matb) )
677
678     cll(ishokb) = v(10,ishokb)
679     clr(ishokb) = v(10,ishokb)
680
681 c      ENDFIX*****
682
683 2600 continue
684
685 c * * * * restore xa and va(n)
686     ishoka = ibnd - 1
687     xa = xakeep
688     do 2610 n = 1,nv
689     va(n) = vakeep(n)
690 2610 continue
691
692 c * * * * end: advance shock front
693
694 c * * * * number of normal interior points = nint
695     nint = ishokb - ijump - 2
696     ishok = ishokb
697
698     go to 3000
699
700 8000 continue
701
702 c * * * * This point reached when shock arrives at free surface
703 c     i=ibnd. First time matjump=1 and interactions are calculated
704 c     at ibnd (stress=0) and ibnd-1: nint=ibnd-3.
705 c     At subsequent times matjump=0 and variables are calculated
706 c     at ibnd (stress=0) only: nint=ibnd-2.
707
708     if( matjump .eq. 0 ) go to 8500
709
710 c * * * * solution at ibnd immediately following shock arrival
711
712     ibm1 = ibnd - 1
713     xbm1 = xlag(ibm1)
714     fl = ( xa - ( xband - dt*cll(ibnd) ) )/( xa - xbm1 )
715     omfl = 1. - fl
716     zl = cll(ibnd)/eosa(13, matl(ibnd) )
717
718     do 8050 n = 1, nv
719     vl(n) = fl*v(n, ibm1) + omfl*va(n)
720 8050 continue

```

```

721
722      v2(4, ibnd) = 0.
723      v2(6, ibnd) = v1(6) + ( v1(4) + dt*frate( matl(ibnd), v1 ) )/zl
724      call xpath( 1., ibnd, left, vs(1,ibnd), v2(1,ibnd), dt2 )
725
726 c * * * obtain solution at ibnd-1 immediately following shock
727 c      arrival at the free surface. dxstar.gt.0 when c-
728 c      characteristic intersects the shock on the right.
729
730      dxstar = dt*clr(ibml) - xa + xbml
731      fl = cll(ibml)*dt/dxl(ibml)
732      fr = - dxstar/( xa - xbml )
733      dtr = dt
734      if( dxstar .le. 0. ) go to 8110
735      fr = dxstar/( xbnd - xa )
736      dtr = dt - dxstar/( us + clr(ibml) )
737 8110 continue
738      omfl = 1. - fl
739      omfr = 1. - fr
740
741      do 8150 n = 1, nv
742          vright = v(n, ibml)
743          if( dxstar .gt. 0. ) vright = vs(n, ibnd)
744          vr(n) = fr*vright + omfr*va(n)
745          vl(n) = fl*v(n, ibml-1) + omfl*v(n, ibml)
746 8150 continue
747
748      zl = cll(ibml)/eosa(13, matl(ibml) )
749      zr = clr(ibml)/eosa(13, matr(ibml) )
750      fplus = vl(4) + zl*vl(6) + dt*frate( matl(ibml), v1 )
751      fminus = vr(4) - zr*vr(6) + dtr*frate( matr(ibml), vr )
752      denoz = 1./( zl + zr )
753
754      v2(4, ibml) = denoz*(zr*fplus + zl*fminus )
755      v2(6, ibml) = denoz*( fplus - fminus )
756      call xpath( 1., ibml, left, v(1,ibml), v2(1,ibml), dt )
757
758      matjump = 0
759      nint = ibnd - 3
760      ishok = ibnd
761      ishokb = ibnd
762
763      go to 3000
764
765 8500 continue
766
767 c * * * later interactions with the free surface at i = ibnd
768
769      ibml = ibnd - 1
770      fl = dt*cll(ibnd)/dxl(ibnd)
771      omfl = 1. - fl
772      zl = cll(ibnd)/eosa(13, matl(ibnd))
773
774      do 8650 n = 1, nv
775          vl(n) = fl*v(n, ibml) + omfl*v(n, ibnd)
776 8650 continue
777
778      v2(4, ibnd) = 0.
779      v2(6, ibnd) = v1(6) + ( v1(4) + dt*frate(matl(ibnd), v1) )/zl
780      call xpath( 1., ibnd, left, v(1,ibnd), v2(1,ibnd), dt )
781
782      nint = ibnd - 2
783      ishok = ibnd
784      ishokb = ibnd
785
786
787 c* * * * left boundary calculation (3000)
788
789 3000 continue
790
791 c - - - - -
792 c      simple left boundary calculation - symmetric impact
793 c      time-dependent particle velocity u0
794 c      j n johnson Oct 1986
795 c - - - - -
796
797      if ( ishoka .ge. 2 ) go to 3200
798
799      tstar = ( x0 + clr(1)*dt - xa )/( us + clr(1) )
800

```

```

801      if( tstar .ge. 0. ) go to 3100
802 c * * * * no intersection of c- with shock
803 c * * * * interpolate between i = 1 and a
804
805      fr = clr(1)*dt/(xa-x0)
806      omfr = 1. - fr
807      dtr = dt
808
809      do 3010 n = 1,nv
810      vr(n) = omfr*v(n,1) + fr*va(n)
811 3010 continue
812
813      go to 3400
814
815 3100 continue
816 c * * * * c- intersects shock
817 c * * * * interpolate between a and b
818
819      fr = tstar/dt
820      omfr = 1. - fr
821      dtr = dt - tstar
822
823      do 3110 n = 1,nv
824      vr(n) = omfr*va(n) + fr*vb(n)
825 3110 continue
826
827      go to 3400
828
829 3200 continue
830 c * * * * shock not in first cell
831 c * * * * interpolate between 1 and 2
832
833      fr = clr(1)*dt/dx1
834      omfr = 1. - fr
835      dtr = dt
836
837      do 3210 n = 1,nv
838      vr(n) = omfr*v(n,1) + fr*v(n,2)
839 3210 continue
840
841 3400 continue
842
843      zr = clr(1)/eosa(13,matr(1))
844
845 c * * * * can put in functional time dependence of up at x=0
846
847      v2(6,1) = uzero( time, u0 )
848      v2(4,1) = vr(4) + zr*( v2(6,1) - vr(6) ) + frate(matr(1),vr)*dtr
849
850      i = 1
851      left = .true.
852      call xpath( 1., i, left, v(1,i), v2(1,i), dt )
853
854 c * * * * end: left boundary calculation
855
856      if( noshok ) go to 5000
857
858 c * * * * i = ishok calculation (4000)
859
860 4000 continue
861
862      i = ishok
863      xi = xlag(i)
864
865      if( i .eq. 1 ) go to 5000
866
867      fl = cll(i)*dt/dx1(i)
868      omfl = 1. - fl
869
870 c * * * * tstar .gt. 0 indicates c- intersection with shock
871 c * * * * ta is the time that the shock remains in mat=1 when
872 c * * * * material boundary is crossed.
873
874      tstar = dt - (xb - xi)/( clr(i) + us )
875      ta = dt - (xb - xi)/us
876
877      if( tstar .ge. 0. ) go to 4200
878 c * * * * no intersection of c- with shock
879 c * * * * interpolate between i and a
880

```



```

881      fr = clr(i)*dt/(xa - xi)
882      omfr = 1. - fr
883      dtr = dt
884
885      do 4110 n = 1,nv
886      vni = v(n,i)
887      if( i .eq. ibnd ) vni = vbnd(n)
888      vniml = v(n,i-1)
889      if( i .eq. ibnd+1 ) vniml = vbnd(n)
890      vl(n) = omfl*v(n,i) + fl*vniml
891      vr(n) = omfr*vni + fr*va(n)
892 4110 continue
893
894      go to 4400
895 4200 continue
896 c * * * c- intersects shock
897 c * * * interpolate between a and b
898
899      if( ta .gt. 0. ) then
900          fr = (tstar - ta)/(dt - ta)
901          dtr = dt - tstar
902      else
903          fr = tstar/dt
904          dtr = dt - tstar
905      endif
906
907      omfr = 1. - fr
908
909      do 4210 n = 1,nv
910          if( ta .lt. 0. ) vni = va(n)
911          if( i .eq. ibnd .and. ta .gt. 0. ) vni = vsbnd(n)
912          if( i .ne. ibnd .and. ta .gt. 0. ) vni = v(n,i)
913          vr(n) = omfr*vni + fr*vb(n)
914 4210 continue
915
916      if( ijump .gt. 0. ) go to 4300
917 c * * * interpolate between i-1 and i
918
919      do 4220 n = 1,nv
920      vniml = v(n,i-1)
921      if( i .eq. ibnd+1 ) vniml = vbnd(n)
922      vl(n) = omfl*v(n,i) + fl*vniml
923 4220 continue
924
925      go to 4400
926
927 4300 continue
928 c * * * interpolate between i-1 and a
929
930      fl = (cll(i)*dt - xi + xa)/(dxl(i) - xi + xa)
931      omfl = 1. - fl
932
933      do 4310 n = 1,nv
934      vniml = v(n,i-1)
935      if( i .eq. ibnd+1 ) vniml = vbnd(n)
936      vl(n) = omfl*va(n) + fl*vniml
937 4310 continue
938
939 4400 continue
940
941      zl = cll(i)/eosa(13,matl(i))
942      zr = clr(i)/eosa(13,matr(i))
943
944      fplus = vl(4) + zl*vl(6) + frate( matl(i), vl )*dt
945      fminus = vr(4) - zr*vr(6) + frate( matr(i), vr )*dtr
946      denoz = 1./ ( zl + zr )
947
948      v2(4,i) = denoz*(zr*fplus + zl*fminus)
949      v2(6,i) = denoz*(fplus - fminus)
950
951      dt0 = dt
952      if( (xb-xi)/us .lt. dt ) dt0 = (xb-xi)/us
953      left = .true.
954      call xpath( 1., i, left, v(1,i), v2(1,i), dt0 )
955
956      if( i .ne. ibnd ) go to 4410
957      v2bnd(4) = v2(4,i)
958      v2bnd(6) = v2(6,i)
959      left = .false.
960      call xpath( 1., i, left, vbnd, v2bnd, dt0 )

```

```

961
962 4410 continue
963
964     if( ijump .eq. 0) go to 5000
965     ishok = ishok - 1
966     ijump = ijump - 1
967     go to 4000
968
969 c* * * * end: i = ishok calculation
970
971
972 c* * * * normal interior point calculation
973
974 5000 continue
975     iend = nint + 1
976
977     do 5900 i = 2,iend
978
979         fl = cll(i)*dt/dxl(i)
980         omfl = 1. - fl
981         fr = clr(i)*dt/dxr(i)
982         omfr = 1. - fr
983
984         do 5100 n = 1,nv
985             vni = v(n,i)
986             vniml = v(n,i-1)
987             if( i .eq. ibnd ) vni = vbnd(n)
988             if( i .eq. ibnd+1 ) vniml = vbnd(n)
989             vl(n) = omfl*v(n,i) + fl*vniml
990             vr(n) = omfr*vni + fr*v(n,i+1)
991 5100 continue
992
993 c * * * * save old sound speeds at interpolated l and r points
994     m = matl(i)
995     call clag( vl, ida(m), eosa(1,m), idb(m), eosb(1,m) )
996     cll0 = vl(10)
997     m = matr(i)
998     call clag( vr, ida(m), eosa(1,m), idb(m), eosb(1,m) )
999     clr0 = vr(10)
1000
1001 c * * * * first-order step
1002 c     characteristic solution based on old sound speeds
1003 c     at point i
1004
1005     zl = cll(i)/eosa(13,matl(i))
1006     zr = clr(i)/eosa(13,matr(i))
1007
1008     fplus = vl(4) + zl*vl(6) + frate( matl(i), vl )*dt
1009     fminus = vr(4) - zr*vr(6) + frate( matr(i), vr )*dt
1010     denoz = 1./( zl + zr )
1011
1012     v2(4,i) = denoz*(zr*fplus + zl*fminus)
1013     v2(6,i) = denoz*(fplus - fminus)
1014
1015 c * * * *do not replace v2(4, ) and v2(6, ) into v(4, ) and v(6, )
1016 c * * * *until the end of the icycle
1017
1018     left = .true.
1019     call xpath( 1., i, left, v(1,i), v2(1,i), dt )
1020     m = matl(i)
1021     call clag( v2(1,i), ida(m), eosa(1,m), idb(m), eosb(1,m) )
1022     cll1 = v2(10,i)
1023     clr1 = cll1
1024
1025     if( i .ne. ibnd ) go to 5200
1026
1027     left = .false.
1028     v2bnd(4)=v2(4,i)
1029     v2bnd(6)=v2(6,i)
1030     call xpath( 1., i, left, vbnd, v2bnd, dt )
1031     m = matr(i)
1032     call clag( v2bnd, ida(m), eosa(1,m), idb(m), eosb(1,m) )
1033     clr1 = v2bnd(10)
1034
1035 5200 continue
1036
1037 c * * * * second-order step * * * * *
1038 c     cll0 and clr0 = sound speeds at t (first order)
1039 c     cll1 and clr1 = sound speeds at t + dt (first order)
1040

```

```

1041      cll2 = 0.5*(cll0 + cll1)
1042      clr2 = 0.5*(clr0 + clr1)
1043      fl  = cll2*dt/dxl(i)
1044      fr  = clr2*dt/dxr(i)
1045
1046      if( i .eq. ibnd ) then
1047
1048          omfl = 1. - fl
1049          omfr = 1. - fr
1050          deltax = dxl(i+1)
1051          deltaa = dxr(i+1)
1052          if( ishoka .eq. ibnd+1 ) deltaa = xa - xlag(ibnd+1)
1053          wt1 = deltax**2/( deltaa*(deltax + deltaa) )
1054          wt2 = deltaa/(deltax + deltaa)
1055          wt3 = deltax/(deltax + deltaa)
1056
1057          do 5300 n = 1, nv
1058              vl(n) = v(n,i-1) + 0.5*( v(n,i) - v(n,i-2) )*omfl
1059              # + 0.5*( v(n,i) - 2.*v(n,i-1) + v(n,i-2) )*omfl**2
1060
1061              van = v(n,i+2)
1062              if( ishoka .eq. ibnd+1 ) van = va(n)
1063
1064              vr(n) = v(n,i+1)
1065              # -( wt1*(van - v(n,i+1)) - wt2*(v(n,i) - v(n,i+1)) )*omfr
1066              # +( wt1*(van - v(n,i+1)) + wt3*(v(n,i) - v(n,i+1)) )*omfr**2
1067
1068              vlvag(n) = 0.5*( vl(n) + v2(n,i) )
1069              vravg(n) = 0.5*( vr(n) + v2bnd(n) )
1070          5300 continue
1071
1072      else
1073
1074          do 5400 n = 1, nv
1075              vl(n) = v(n,i) - 0.5*( v(n,i+1) - v(n,i-1) )*fl
1076              # + 0.5*( v(n,i+1) - 2.*v(n,i) + v(n,i-1) )*fl**2
1077              vr(n) = v(n,i) + 0.5*( v(n,i+1) - v(n,i-1) )*fr
1078              # + 0.5*( v(n,i+1) - 2.*v(n,i) + v(n,i-1) )*fr**2
1079              vlvag(n) = 0.5*( vl(n) + v2(n,i) )
1080              vravg(n) = 0.5*( vr(n) + v2(n,i) )
1081          5400 continue
1082
1083      endif
1084
1085      zl = cll2/eosa(13, matl(i) )
1086      zr = clr2/eosa(13, matl(i) )
1087
1088      fplus = vl(4) + zl*vl(6) + frate( matl(i), vlvag )*dt
1089      fminus = vr(4) - zr*vr(6) + frate( matr(i), vravg )*dt
1090      denoz = 1./( zl + zr )
1091
1092      v2(4,i) = denoz*( zr*fplus + zl*fminus )
1093      v2(6,i) = denoz*( fplus - fminus )
1094
1095      c * * * do not replace v2(4, ) and v2(6, ) into v(4, ) and v(6, )
1096      c * * * until the end of the icycle
1097
1098      left = .true.
1099      call xpath( 2., 1, left, v(1,i), v2(1,i), dt )
1100
1101      if( i .ne. ibnd ) go to 5900
1102      left = .false.
1103      v2bnd(4) = v2(4,i)
1104      v2bnd(6) = v2(6,i)
1105      call xpath( 2., i, left, vbnd, v2bnd, dt )
1106
1107      5900 continue
1108
1109      6000 continue
1110
1111      c * * * set up arrays for time plots (6000)
1112
1113      ib = ishokb
1114
1115      do 6100 i = 1,ngage
1116
1117          if( igage(i) .gt. ib ) go to 6090
1118          if( igage(i) .lt. ib ) go to 6050
1119
1120      c * * * here igage(i) = ib

```

```

1121
1122      if( npts(1) .ge. 2 ) go to 6050
1123
1124      ij = ishokb - ishoka
1125      if( ij .eq. 0 .and. icycle .ne. 1 ) go to 6050
1126
1127      mm = matl( ib )
1128      tgage(1,1) = ts(ib)
1129      ygage(1,1,1) = 0.0
1130      ygage(1,1,2) = 0.0
1131      ygage(1,1,3) = 0.0
1132      ygage(1,1,4) = rc(5,mm)
1133      ygage(1,1,5) = 0.0
1134      ygage(1,1,6) = 0.0
1135
1136      tgage(2,1) = ts(ib)
1137      ygage(2,1,1) = vs(6,ib)*10000.
1138      ygage(2,1,2) = vs(4,ib)*1000.
1139      ygage(2,1,3) = 2.*vs(14,ib)/vs(17,ib)
1140      ygage(2,1,4) = vs(17,ib)*1000.
1141      ygage(2,1,5) = vs(18,ib)
1142      ygage(2,1,6) = fdrag( mm, vs(1,ib) )*100.
1143      npts(1) = 2
1144
1145      6050 continue
1146
1147      npts(1) = npts(1) + 1
1148      nn = npts(1)
1149      mm = matl( igage(1) )
1150      tgage( nn, 1 ) = time
1151      ygage( nn, 1, 1 ) = v2( 6, igage(1) )*10000.
1152      ygage( nn, 1, 2 ) = v2( 4, igage(1) )*1000.
1153      ygage( nn, 1, 3 ) = 2.*v2(14, igage(1) )/v2(17, igage(1) )
1154      ygage( nn, 1, 4 ) = v2(17, igage(1) )*1000.
1155      ygage( nn, 1, 5 ) = v2(18, igage(1) )*100.
1156      ygage( nn, 1, 6 ) = fdrag( mm, v2( 1, igage(1) ) )
1157
1158      6090 continue
1159      6100 continue
1160
1161      c * * * replace all variables with new values at t+dt (6500)
1162
1163      xa = xb
1164
1165      do 6600 n = 1,nv
1166      vbnd(n) = v2bnd(n)
1167      va(n) = vb(n)
1168
1169      do 6550 i = 1,ishokb
1170      v(n,i) = v2(n,i)
1171      6550 continue
1172
1173      6600 continue
1174
1175      c * * * store variables for snapshots (fixed t plots)
1176
1177      if( tsnap2 .lt. tsnap ) go to 6800
1178      tsnap2 = 0.
1179      isnap = isnap + 1
1180      if( isnap .gt. 5 ) go to 6800
1181
1182      c * * * nsnap = no. of points for snapshot isnap
1183      c      the + 2 is for the additional points at shock front
1184      nsnap(isnap) = ib + 2
1185
1186      do 6750 i = 1,ib
1187      xsnap( i, isnap ) = xlag( i ) - x0
1188      ysnap( i, isnap, 1 ) = v(6, i)*10000.
1189      ysnap( i, isnap, 2 ) = v(4, i)*1000.
1190      ysnap( i, isnap, 3 ) = 2.*v(14,i)/v(17,i)
1191      ysnap( i, isnap, 4 ) = v(17,i)*1000.
1192      ysnap( i, isnap, 5 ) = v(18,i)*100.
1193      6750 continue
1194
1195      xsnap( ib+1, isnap ) = xb - x0
1196      ysnap( ib+1, isnap, 1 ) = vb(6)*10000.
1197      ysnap( ib+1, isnap, 2 ) = vb(4)*1000.
1198      ysnap( ib+1, isnap, 3 ) = 2.*vb(14)/vb(17)
1199      ysnap( ib+1, isnap, 4 ) = vb(17)*1000.
1200      ysnap( ib+1, isnap, 5 ) = vb(18)*100.

```

```

1201
1202      mm = matl( ib )
1203      xsnap( ib+2, isnap ) = xb - x0
1204      ysnap( ib+2, isnap, 1 ) = 0.0
1205      ysnap( ib+2, isnap, 2 ) = 0.0
1206      ysnap( ib+2, isnap, 3 ) = 0.0
1207      ysnap( ib+2, isnap, 4 ) = rc(5,mm)
1208      ysnap( ib+2, isnap, 5 ) = 0.0
1209
1210 6800 continue
1211
1212      ipr = ipr + 1
1213      if( ipr .lt. iprint ) go to 1000
1214
1215      ipr = 0
1216      write( 4, 9010) time, xb, vb(4)
1217
1218      do 6150 i = 1, ishokb
1219      write( 4, 9020) i, v(4,i), v(6,i), v(1,i), v(2,i), v(14,i),
1220      # v(17,i), v(18,i)
1221 6150 continue
1222      write( 4, 9020) ishokb, vb(4), vb(6), vb(1), vb(2), vb(14), vb(17),
1223      # vb(18)
1224
1225 c* * * return to start of time cycle
1226      go to 1000
1227
1228 c * * * gage plots (7000)
1229
1230 7000 continue
1231
1232      do 7100 l = 1, ngage
1233
1234      do 7050 n = 1, npts(l)
1235
1236 c*****calculation of acceleration*****jnj 6/27/88
1237
1238      if( n .le. 3 ) then
1239      rdt = 0.0001/( tgage(4,l) - tgage(2,l) )
1240      ygage(n,l,5) = rdt*( ygage(4,l,1) - ygage(2,l,1) )
1241
1242      else
1243
1244      if( n .eq. npts(l) ) then
1245      nml = n - 1
1246      rdt = 0.0001/( tgage(n,l) - tgage(nml,l) )
1247      ygage(n,l,5) = rdt*( ygage(n,l,1) - ygage(nml,l,1) )
1248
1249      else
1250
1251      nml = n - 1
1252      npl = n + 1
1253      rdtm = 0.0001/( tgage(n,l) - tgage(nml,l) )
1254      rdtpl = 0.0001/( tgage(npl,l) - tgage(n,l) )
1255      ygage(n,l,5) = rdtm*( ygage(n,l,1) - ygage(nml,l,1) )
1256      # + rdtpl*( ygage(npl,l,1) - ygage(n,l,1) )
1257
1258      endif
1259
1260      endif
1261
1262 7050 continue
1263
1264 7100 continue
1265
1266      write( title(1), '(5a8)' ) (label(i), i=1,5 )
1267      title(2) = " "
1268      ylabel(1,1) = "particle velocity (m/s)"
1269      ylabel(1,2) = "longitudinal stress (kbar)"
1270      ylabel(1,3) = "sigma/sigma-hat"
1271      ylabel(1,4) = "sigma-hat(kbar)"
1272      ylabel(1,5) = "acceleration (cm/musec**2)"
1273      ylabel(1,6) = "fractime in drag"
1274      tlabel(1) = "time (microseconds)"
1275
1276 c      labels for DISSPLA
1277      ylabel(2,1) = "particle velocity (m/s)$"
1278      ylabel(2,2) = "longitudinal stress (kbar)$"
1279      ylabel(2,3) = "sigma/sigma-hat$"
1280      ylabel(2,4) = "sigma-hat (kbar)$"

```

```

1281      ylabel(2,5) = "acceleration (cm/[m]s@eh.7]2@exhx])$"
1282      ylabel(2,6) = "fracture in drag$"
1283      tlabel(2) = "time ([m]s)$"
1284
1285      do 7300 k = 1,5
1286
1287          ymm(1) = 0.
1288          ymm(2) = -1.0e+99
1289
1290          xmm(1) = 0.
1291          xmm(2) = tmax
1292
1293          do 7150 ll = 1, ngage
1294              call minmax( -npts(ll), ygage(1, ll, k), ymm )
1295          7150 continue
1296
1297      c          SET 5.0 KBAR SCALE ON SIGMA-HAT
1298          if( k .eq. 4 ) ymm(2) = 5.
1299
1300          l = 1
1301          call pix( 1, tlabel, ylabel(1,k), title, npts(1),
1302              #          tgage(1,1), ygage(1,1,k), 0, 0, 0, xmm, ymm )
1303
1304          do 7200 l = 2, ngage
1305              call pixadd( npts(1), tgage(1,1), ygage(1,1,k), 0, 0, 0 )
1306          7200 continue
1307
1308          7300 continue
1309
1310      c * * * * snapshots ( plots at constant time )
1311
1312          xlabel(1) = "position (cm)"
1313          xlabel(2) = "position (cm)$"
1314          ylabel(1,5) = "plastic strain (percent)"
1315          ylabel(2,5) = "plastic strain (percent)$"
1316
1317          do 7500 k = 1,5
1318
1319              ymm(1) = 0.
1320              ymm(2) = -1.0e+99
1321
1322              xmm(1) = 0.
1323              xmm(2) = -1.0e+99
1324
1325              l = 5
1326              call minmax( -nsnap(1), xsnap(1,1), xmm )
1327              call minmax( -nsnap(1), ysnap(1,1,k), ymm )
1328              do 7350 l = 1,4
1329                  call minmax( -nsnap(1), ysnap(1,1,k), ymm )
1330              7350 continue
1331
1332      c          SET 5. KBAR SCALE ON SIGMA-HAT
1333          if( k .eq. 4 ) ymm(2) = 5.
1334          l = 5
1335
1336          call pix( 1, xlabel, ylabel(1,k), title, nsnap(1), xsnap(1,1),
1337              #          ysnap(1,1,k), 0, 0, 0, xmm, ymm )
1338          do 7400 l = 1,4
1339              call pixadd( nsnap(1), xsnap(1,1), ysnap(1,1,k), 0, 0, 0 )
1340          7400 continue
1341
1342          7500 continue
1343
1344          9010 format( 1h0/5x, "time = ", f6.3, 5x, "shock x = ", f9.6, 5x,
1345              #          "sigma-shock = ", f9.6//8x, "i", 3x, "long str", 5x,
1346              #          "part vel", 5x, "volume", 7x, "int energy", 3x, "shear str",
1347              #          4x, "sigma-hat", 4x, "psi"/ )
1348
1349          9020 format( 5x, i4, 7e13.6 )
1350
1351          9030 format( 6x, 4a15 /,
1352              #          5x, "i", 6x, "eosa(i,1)", 6x, "eosb(i,1)", 6x, "eosa(i,2)",
1353              #          6x, "eosb(i,2)"/ )
1354
1355          9040 format( 4x, i2, 4e15.8 )
1356
1357      c * * * * end of main program
1358
1359          9900 continue
1360

```

1361 call gdone
1362 stop
1363 end

```

1364 c * * * * *
1365 c   CLAG: Lagrangian wave speed
1366 c * * * * *
1367
1368       subroutine clag( y, ia, ea, ib, eb )
1369       dimension y(20), ea(25), eb(25), ierr(13)
1370
1371 c*this subroutine calculates the Lagrangian wave speed from
1372 c*   y(1) = specific volume
1373 c*   y(2) = specific internal energy
1374 c*   y(3) = mass fraction unreacted - phase denoted 1
1375 c*   y(4) = pressure
1376 c*   y(5) = equilibrium temperature
1377 c*and other internal state variables as required and contained
1378 c*in the array y(20) for index .ge. 14
1379
1380       if( ia .eq. 6hcub100 .or. ia .eq. 6hisotrp ) then
1381
1382         b11 = ea(1) + eb(3)*alog( y(1)/ea(13) )
1383         #   + eb(4)*y(18)
1384
1385 c   c1 = sqrt( rho*b11 )/rho0 for elastic-viscoplastic solid
1386       y(10) = ea(13)*sqrt( b11/y(1) )
1387
1388       else
1389
1390 c* * * *forest eos routine
1391
1392       pres = y(4)
1393       call eosmix( y, ia, ea, ib, eb, ierr )
1394
1395 c   c1 = rho*c/rho0 for elastic fluid ( c**2 = dp/drho constant s )
1396       y(10) = ( ea(13) / y(1) )*sqrt( y(10) )
1397       y(4) = pres
1398
1399       endif
1400
1401       return
1402       end

```



```

1403 c * * * * *
1404 c   HUGO calculates the remaining Hugoniot state variables from
1405 c   the particle velocity up - variable 6
1406 c   The variables are stored according to the convention of
1407 c   Charles A. Forest as shown below.
1408 c * * * * *
1409
1410       subroutine hugo( y, ia, ea, ib, eb )
1411       common/block2/ rc(15,2), nv
1412       dimension y(20), ea(25), eb(25), jerr(13)
1413
1414 c*      y(1) = specific volume
1415 c*      y(2) = specific internal energy
1416 c*      y(3) = mass fraction unreacted-phase denoted 1
1417 c*      y(4) = pressure (or longitudinal stress)
1418 c*      y(5) = equilibrium temperature
1419 c*      y(6) = particle velocity (not used by eosmix)
1420 c*      y(10) = lagrangean sound velocity
1421 c* * * additional variables as defined by j n johnson
1422 c*      y(11) = d sigma / du along hugoniot
1423 c*      y(14) = shear stress, tau (delta/2.)
1424 c*      y(15) = not used
1425 c*      y(16) = not used
1426 c*      y(17) = mechanical threshold stress, sigma-hat
1427 c*      y(18) = psi
1428 c*      y(19) = not used
1429
1430       if( ia .eq. 6hcub100 .or. ia .eq. 6hisotrp ) then
1431
1432           if( y(6) .eq. 0. ) then
1433 c* * * * * first jump (elastic shock) of zero amplitude
1434               y(1) = ea(13)
1435               y(2) = 0.
1436               y(3) = 1.
1437               y(4) = 0.
1438               y(5) = ea(15)
1439               y(6) = 0.
1440               y(10) = eb(1)
1441               y(11) = eb(1)/ea(13)
1442               y(14) = 0.
1443               y(15) = 0.
1444               y(16) = 0.
1445               y(17) = rc(5,1)
1446               y(18) = 0.
1447               y(19) = 0.
1448
1449           else
1450
1451 c   first approximation to strain for up = y(6)
1452           e = y(6)/eb(1)
1453
1454 c   Newton's method for determination of strain 1 - v/v0
1455 c   corresponding to particle velocity y(6)
1456           do 100 it = 1,10
1457               f = ( 1. + eb(2)*e )**2 - ( y(6)/eb(1) )**2
1458               fp = ( 2. + 3.*eb(2)*e )*e
1459               if( abs( f/(e*fp) ) .lt. 1.e-06 ) go to 200
1460               e = e - f/fp
1461           100 continue
1462
1463           200 y(1) = ea(13)*(1. - e)
1464               y(2) = 0.5*( y(6) )**2
1465               y(3) = 1.
1466               y(4) = ( y(6)**2 )/( ea(13)*e )
1467               y(5) = ea(15)*( ea(13)/y(1) )**ea(11)
1468
1469               b11 = ea(1) + eb(3)*alog( y(1)/ea(13) )
1470 c   c1 = sqrt( rho*b11 )/rho0 for elastic-viscoplastic solid
1471               y(10) = ea(13)*sqrt( b11/y(1) )
1472 c   position 11 contains d sigma / du on hugoniot
1473               y(11) = 2.*y(4)/y(6)/( 1. + y(4)/
1474               #   ( y(4) + 2.*ea(1)*eb(2)*e*e ) )
1475               y(14) = e*( eb(12) - e*eb(13) )
1476               y(15) = 0.
1477               y(16) = 0.
1478               y(17) = rc(5,1)
1479               y(18) = 0.
1480               y(19) = 0.
1481
1482           endif

```

```

1483
1484     else
1485
1486 c - - - - -
1487 c     special case Oct 1986 J N Johnson
1488 c     straight-line us-up Hugoniot
1489
1490     u = ea(1) + ea(2)*y(6)
1491 c - - - - -
1492
1493     y(1) = ea(13)*(1. - y(6)/u )
1494     y(2) = 0.5*( y(6) )**2
1495     y(3) = 1.
1496
1497 c * * * * forest eos routine eosmix
1498
1499     call eosmix( y, ia, ea, ib, eb, jerr )
1500
1501 c * * * * y(4) = pressure and y(5) = temperature
1502 c * * * * are set internal to eosmix
1503
1504     y(10) = ( ea(13) / y(1) )*sqrt( y(10) )
1505     y(14) = 0.
1506     y(17) = 0.
1507     y(18) = 0.
1508
1509     endif
1510
1511     return
1512     end

```

```

1513 c * * * * *
1514 c   IFRONT locates the index immediately to the left of the shock
1515 c * * * * *
1516
1517     function ifront( x0, x, d1, d2, ib, xb )
1518     ifront = 1 + int( (x - x0)/d1 )
1519     if( ifront .lt. ib ) go to 100
1520     ifront = ib + int ( ( x - xb )/d2 )
1521 100 continue
1522     return
1523     end

```

```

1524 c * * * * *
1525 c XPATH integrates v, e, w, and all remaining internal
1526 c state variables along a particle path at constant X through
1527 c a time step dt. Longitudinal stress vnew(4) is known.
1528 c order = 1. first-order integration
1529 c order = 2. second-order integration with previously determined
1530 c first-order approximation at new time step. rc(13,1)
1531 c determines explicit or implicit nature of the integration.
1532 c * * * * *
1533
1534 subroutine xpath( order, i, gauche, vold, vnew, dt )
1535 logical gauche
1536
1537 common/block1/ matr(2000),matl(2000), cll(2000), clr(2000),
1538 # ida(2), eosa(25,2), idb(2), eosb(25,2)
1539 common/block2/ rc(15,2), nv
1540 dimension vold(20), vnew(20), vavg(20), ierr(11)
1541 real lv
1542
1543 if( gauche ) then
1544     m = matl(i)
1545     recz2 = ( eosa(13,m)/cll(i) )**2
1546 else
1547     m = matr(i)
1548     recz2 = ( eosa(13,m)/clr(i) )**2
1549 endif
1550
1551 c * * * * wgt = weight toward old value (last time step)
1552 c         = 1. for first-order integration
1553 c         = 1. - rc(13,1) for second-order integration
1554 wgt = 1. - rc(13,1)*(order - 1.)
1555
1556 do 100 n = 1, nv
1557     vavg(n) = wgt*vold(n) + (1. - wgt)*vnew(n)
1558 100 continue
1559
1560 c * * * * find dv/dt and de/dt from dsigma-r/dt
1561
1562 dvdt = ( frate(m,vavg) - ( vnew(4) - vold(4) )/dt )
1563 # *recz2
1564
1565 dedt = - 0.5*( vnew(4) + vold(4) )*dvdt
1566
1567 c * * * * set reaction rate for explosive problems
1568
1569 dwdt = 0.
1570
1571 c * * * * advance v, e, and w : store as vnew
1572
1573 vnew(1) = vold(1) + dvdt*dt
1574 vnew(2) = vold(2) + dedt*dt
1575 vnew(3) = vold(3) + dwdt*dt
1576
1577 c * * * * advance remaining internal-state variables here!
1578
1579 if( ida(m) .eq. 6hcub100 .or. ida(m) .eq. 6hisotrp ) then
1580
1581     vbar = vavg(1)
1582     lv = alog( vavg(1)/eosa(13,m) )
1583     psi = vavg(18)
1584     b11 = eosa(1,m) + eosb(3,m)*lv + eosb(4,m)*psi
1585     b12 = eosa(2,m) - eosb(5,m)*lv - eosb(6,m)*psi
1586     b21 = eosa(2,m) + eosa(5,m)*lv + eosb(7,m)*psi
1587     b22 = eosa(1,m) - eosb(8,m)*lv - eosb(9,m)*psi
1588     b23 = eosa(2,m) - eosb(10,m)*lv + eosb(11,m)*psi
1589
1590     econ1 = b11 - b12 - 2.*eosa(11,m)*vold(14)
1591     econ2 = 0.5*(b11 - b21)
1592     econ3 = 0.25*( 2.*(b11 - b21 - b12) + b22 + b23 )
1593
1594     ddtpsi = -frate( m, vavg)/econ1
1595
1596     vnew(18) = vold(18) + ddtpsi*dt
1597
1598     ddttemp = -vold(5)*eosa(11,m)*dvdt/vbar
1599 # + (2.*vold(14)*vbar/eosa(12,m))*ddtpsi
1600     vnew(5) = vold(5) + ddttemp*dt
1601
1602     ddttau1 = -(econ2/vbar)*dvdt
1603     ddttau2 = -econ3*ddtpsi

```

```

1604      t1 = vold(14) + ddttau1*dt
1605      t2 = t1 + ddttau2*dt
1606      vnew(14) = t2
1607
1608      ddtps11 = abs( ddtpsi )
1609      theta0 = rc(6,m) + rc(7,m)*alog(amax1(1.e-2,ddtps11))
1610      1 + rc(8,m)*ddtps11
1611      ss = rc(9,m)*(ddtps11/rc(10,m))*((rc(11,m)*vavg(5))
1612      rr = ( vavg(17) - rc(5,m) )/( ss - rc(5,m) )
1613      dmtsde = theta0*(1. - tanh( 2.*rr ) )
1614      if( dmtsde .gt. econ2 ) dmtsde = econ2
1615      ddtmts = dmtsde*ddtps11
1616      vnew(17) = vold(17) + ddtmts*dt
1617
1618      else
1619
1620      call eosmix( vnew, ida(m), eosa(1,m), idb(m), eosb(1,m),
1621      # ierr )
1622
1623      endif
1624
1625      return
1626      end

```

```

1627 c * * * * *
1628 c FRATE defines the rate-dependent behavior
1629 c of material m = 1 in terms of the flow variables var(nv).
1630 c Rate-dependent material constants and functional forms
1631 c are for the MTS material model. If other models are used
1632 c this is where the changes go.
1633 c * * * * *
1634
1635 function frate( m, var )
1636 common/block1/ matr(2000),mat1(2000), c11(2000), clr(2000),
1637 # ida(2),eosa(25,2), idb(2), eosb(25,2)
1638 common/block2/ rc(15,2), nv
1639 dimension var(20)
1640 real lv
1641
1642 if( m .eq. 1 ) then
1643
1644 lv = alog( var(1) / eosa(13,m) )
1645 psi = var(18)
1646 b11 = eosa(1,m) + eosb(3,m)*lv + eosb(4,m)*psi
1647 b12 = eosa(2,m) - eosb(5,m)*lv - eosb(6,m)*psi
1648 econ = b11 - b12 - 2.*eosa(11,m)*var(14)
1649
1650 sign = 1.
1651 if( var(14) .lt. 0. ) sign = -1.
1652 s = 2.*abs( var(14) )
1653
1654 eps = 1.0e-12
1655 r = ( s - rc(5,m) )/( eps + var(17) - rc(5,m) )
1656 if( r .lt. 0. ) r = 0.
1657 call dummy1
1658 exparg = (1. - r**0.66667)*rc(3,m)/var(5)
1659 if( exparg .gt. 100. ) exparg=+100.
1660 if( exparg .lt. -100. ) exparg=-100.
1661
1662 c a negative value of rc(12,m) is read in if we want to
1663 c calculate with a non-constant value of lambda given by
1664 c lambda = mu*b/sigma-hat. Otherwise, constant value of
1665 c lambda*nu(zero) is used.
1666 c if rc(12,m) < 0, then rc(12,m) = - b*nu(zero)
1667 c if rc(12,m) > 0, then rc(12,m) = lambda*nu(zero)
1668
1669 afac = rc(2,m)*rc(12,m)
1670 if( rc(12,m) .lt. 0. ) afac = -afac*eosa(3,m)/(eps + var(17))
1671
1672 deno = afac*sqrt(1. + (s/(rc(2,m)*rc(4,m)))**2)
1673 deno = deno + s*exp( exparg )
1674
1675 frate = -econ*rc(1,m)*s/deno
1676 frate = sign*frate
1677
1678 else
1679
1680 frate = 0.
1681
1682 endif
1683
1684 return
1685 end

```

```

1686 c * * * * *
1687 c      UZERO specifies the particle velocity at the left boundary
1688 c      as a function of time
1689 c * * * * *
1690
1691      function uzero( t, umax )
1692      common/block2/ rc(15,2), nv
1693 c * * * rc(14,1) = time duration for left bndry velocity
1694 c * * * rc(15,1) = t2 (time at which left bndry vel goes to 0.)
1695
1696      if( rc(14,1) .eq. 0. ) then
1697
1698      uzero = umax
1699
1700      else
1701
1702      delt = rc(15,1) - rc(14,1) + 1.0e-06
1703      uzero = umax*( 1. - ( t - rc(14,1) )/delt )
1704      if( uzero .gt. umax ) uzero = umax
1705      if( uzero .lt. 0. ) uzero = 0.
1706
1707      endif
1708
1709      return
1710      end

```

```

1711 c * * * * *
1712 c EOSMIX: Equation of state of a mixture
1713 c * * * * *
1714
1715     subroutine eosmix( vv, id1, param1, id2, param2, ierr)
1716 c
1717 c     charles a forest, april 1984
1718 c
1719 c     eosmix : (e)quation (o)f (s)tate (mix)ture
1720 c             input = v (sp. vol.), e (sp. energy), w (mass fraction)
1721 c             output = p (Mbar), t (deg. k), dp/dv, dp/de, dp/dw, c**2
1722 c
1723 c     mixture options:
1724 c             single component (not mixed) or
1725 c             two component mixture in pressure-temperature equilibrium
1726 c             with v = w*v1 + (1-w)*v2 and e = w*e1 + (1-w)*e2
1727 c
1728 c     component options:
1729 c             hom solid      (id=4hhom1 or id=4hhom2)
1730 c             hom bkw gas    (id=3hbkw)
1731 c             jwl            (id=3hjwl)
1732 c
1733 c     vv(1) = total specific volume (input)
1734 c     vv(2) = total internal energy (input)
1735 c     vv(3) = mass fraction of component 1 (input)
1736 c
1737 c     vv(4) = pressure (output)
1738 c     vv(5) = temperature (output)
1739 c     vv(6) = not used (output)
1740 c     vv(7) = dp/dv, partial derivative (output)
1741 c     vv(8) = dp/de, partial derivative (output)
1742 c     vv(9) = dp/dw, partial derivative (output)
1743 c     vv(10) = c**2, c-frozen sound speed (output)
1744 c     note: (c/v)**2 = p*(dp/de) - (dp/dv)
1745 c
1746 c     id1,param1 = identification and parameter list for component 1
1747 c     id2,param2 = identification and parameter list for component 2
1748 c
1749 c     ierr = 0, normal exit (output)
1750 c     ierr = nerr,message (80 characters), error exit (output)
1751 c     nerr = 1, mixture iteration error, use last iteration values
1752 c     nerr = 2, eosid's not recognized, no calculation
1753 c
1754 c     hom solid parameters, sin style (id=4hhom1)
1755 c     (1) c (2) s (3) vsw (4) c1
1756 c     (5) sl (6) f (7) g (8) h
1757 c     (9) i (10) j (11) gamma (12) cv
1758 c     (13) v0 (14) alpha (15) spa (16) usp
1759 c     (17) t0 (18) p0 (19) 2/3 y0 (20) mu
1760 c     (21) plap (22) spall p (23) minv (24) vmin
1761 c     (25) vmax
1762 c
1763 c     hom solid parameters, eos only (id=4hhom2)
1764 c     (1) c (2) s (3) vsw (4) c1
1765 c     (5) sl (6) f (7) g (8) h
1766 c     (9) i (10) j (11) gamma (12) cv
1767 c     (13) v0 (14) alpha (15) t0 (16) p0
1768 c     (17) vmin (18) vmax
1769 c
1770 c     ln(th) = f + lnvs*(g + lnvs*(h + lnvs*(i + lnvs*j)))
1771 c
1772 c     hom bkw gas parameters (id=3hbkw)
1773 c     (1) a (2) b (3) c (4) d
1774 c     (5) e (6) k (7) k (8) m
1775 c     (9) n (10) o (11) q (12) r
1776 c     (13) s (14) t (15) u (16) cv
1777 c     (17) z (18) vmin (19) vmax
1778 c
1779 c     vg polynomial domain = (vgmin,vgmax) interval
1780 c     ln(pi) = a + lnvg*(b + lnvg*(c + lnvg*(d + lnvg*e)))
1781 c     ln(ez) = k + lnpi*(k + lnpi*(m + lnpi*(n + lnpi*o)))
1782 c     ei = ez - z
1783 c     ln(ti) = q + lnvg*(r + lnvg*(s + lnvg*(t + lnvg*u)))
1784 c     -1/beta = r + lnvg*(2s + lnvg*(3t + lnvg*4u))
1785 c
1786 c     jwl parameters (id=3hjwl)
1787 c     (1) a (2) b (3) c (4) r1
1788 c     (5) r2 (6) w (7) e0 (8) rho
1789 c     (9) tz (10) cv (11) vmin
1790 c

```



```

1791 c      tz = tcj*(vcj**w)
1792 c
1793 c      logical variables (prefix b is in honor of george boole)
1794 c      bmix = .true. , two component mixture
1795 c      bcvrq = .true. , newtons method converged to bound cvrg0
1796 c
1797 c -----
1798 c
1799 c      logical bmix, bcvrq, equal
1800 c      real lvs, lth, lvq, lpi, lez, lti
1801 c
1802 c      calling sequence arrays
1803 c      dimension vv(13), param1(25), param2(25), ierr(11)
1804 c
1805 c      arrays and variables for internal use.
1806 c      dimension vn(2), en(2), pn(2), tn(2), dpndvn(2), dpnden(2),
1807 c      1 dtndvn(2), dtnden(2), idcomp(2), kq(2), q(50),
1808 c      2 nparam(4), idopt(4), jeos(4), jgoto(3), cof(4,4)
1809 c
1810 c      equivalence (v1,vn(1)), (v2,vn(2)), (e1,en(1)), (e2,en(2)),
1811 c      1 (p1,pn(1)), (p2,pn(2)), (t1,tn(1)), (t2,tn(2)),
1812 c      2 (dp1dv1,dpndvn(1)), (dp2dv2,dpndvn(2)), (dp1de1,dpnden(1)),
1813 c      3 (dp2de2,dpnden(2)), (dt1dv1,dtndvn(1)), (dt2dv2,dtndvn(2)),
1814 c      4 (dt1de1,dtnden(1)), (dt2de2,dtnden(2))
1815 c
1816 c      data nopt,idopt / 4, 4hhom1, 4hhom2, 3hbkw, 3hjlw /
1817 c      data nparam / 25, 18, 19, 11 /
1818 c      data jeos / 1, 1, 2, 3 /
1819 c
1820 c      data rrr / 4.1858518e-05 /
1821 c      data wbnr / +0.0001 /
1822 c      data cvrg0 / +1.0e-3 /
1823 c      data itmax / 100 /
1824 c
1825 c      define logical valued arithmetic statement function
1826 c      equal(a,b,bound) = abs(a-b) .le. bound*amax1(abs(a),abs(b))
1827 c
1828 c -----
1829 c
1830 c      set error indicator to normal exit value, reset at error exits.
1831 c      ierr(1) = 0
1832 c
1833 c      count components, store parameters and set indicies
1834 c      idcomp(1) = id1
1835 c      idcomp(2) = id2
1836 c      ncomp = 0
1837 c      k = 0
1838 c      do 100 icmp=1,2
1839 c      do 90 jopt=1,nopt
1840 c
1841 c      if(idcomp(icmp) .ne. idopt(jopt)) go to 90
1842 c      ncomp = ncomp + 1
1843 c      jgoto(ncomp) = jeos(jopt)
1844 c      nq = nparam(jopt)
1845 c
1846 c      go to (30,34) icmp
1847 c      30 continue
1848 c      do 31 iq=1,nq
1849 c      31 q(k+iq) = param1(iq)
1850 c      go to 60
1851 c      34 continue
1852 c      do 35 iq=1,nq
1853 c      35 q(k+iq) = param2(iq)
1854 c      go to 60
1855 c
1856 c      60 continue
1857 c      if(idcomp(icmp) .ne. idopt(1)) go to 80
1858 c      hom1 solid , place parameters in hom2 solid positions.
1859 c      q(k+15) = q(k+17)
1860 c      q(k+16) = q(k+18)
1861 c      q(k+17) = q(k+24)
1862 c      q(k+18) = q(k+25)
1863 c      nq = 18
1864 c      80 continue
1865 c      kq(ncomp) = k
1866 c      k = k + nq
1867 c      go to 100
1868 c
1869 c      90 continue
1870 c      100 continue

```

```

1871 c      if(ncomp .eq. 0) go to 6110
1872
1873 c      set logical variables bmix, bcvrq
1874 c      bcvrq = .false.
1875 c      bmix = ncomp .eq. 2
1876
1877 c      v = vv(1)
1878 c      e = vv(2)
1879 c      if(.not.bmix) go to 390
1880 c      w = amin1( 1., amax1( 0., vv(3)) )
1881 c      omw = 1.-w
1882
1883 390 continue
1884
1885 c      set volume, energy values for each component
1886 c      do 400 m=1,ncomp
1887 c      vn(m) = v
1888 c      en(m) = e
1889 400 continue
1890
1891 c      if(.not.bmix) go to 450
1892 c      reset volumes for hom1 or hom2 mixed with jwl or bkw
1893 c      do 410 ma=1,2
1894 c      if(jgoto(ma) .ne. 1) go to 410
1895 c      mb = ma + (-1)**(ma+1)
1896 c      if(jgoto(mb) .eq. 1) go to 450
1897 c      k = kq(ma)
1898 c      vn(ma) = 0.97*amin1(q(k+13),v)
1899 c      if(omw .gt. wbnd) vn(mb) = (v - w*vn(ma))/omw
1900 410 continue
1901 450 continue
1902
1903 c      do 6000 iter=1,itmax
1904 c      main loop, iterate on (v1,e1,v2,e2)
1905
1906 c      do 4000 m=1,ncomp
1907 c      k = kq(m)
1908 c      jgo = jgoto(m)
1909 c      go to ( 1100,1200,1300) jgo
1910
1911 1100 continue
1912 c      evaluate functions for hom solid
1913 c      vn(m) = amax1( q(k+17), amin1( q(k+18), vn(m)) )
1914 c      vs = vn(m)
1915 c      es = en(m)
1916 c      rcvs = 1./(q(k+12)*rrr)
1917 c      if(vs .gt. q(k+13)) go to 1120
1918 c      solid eos, (vs less than v0)
1919 c      jj = 1
1920 c      if(vs .lt. q(k+3)) jj = 4
1921 c      dv = q(k+13) - vs
1922 c      a = (q(k+jj))/(q(k+13)-q(k+jj+1)*dv)**2
1923 c      psr = dv*a + q(k+16)
1924 c      esr = .5*(psr + q(k+16))*dv
1925 c      lvs = alog(vs)
1926 c      lth = q(k+6)+lvs*(q(k+7)+lvs*(q(k+8)+lvs*(q(k+9)+lvs*q(k+10))))
1927 c      tsr = exp(lth)
1928 c      dpsr = -a*(q(k+13)+q(k+jj+1)*dv)/(q(k+13)-q(k+jj+1)*dv)
1929 c      desr = 0.5*(dv*dpsr - psr - q(k+16))
1930 c      dtsr = (q(k+7)+lvs*(2.*q(k+8)+lvs*(3.*q(k+9)+lvs*4.*q(k+10))))
1931 c      1 *tsr/vs
1932 c      go to 1130
1933 1120 continue
1934 c      solid eos, (vs greater than v0)
1935 c      psr = 0.
1936 c      dpsr = 0.
1937 c      dtsr = 1./(3.*q(k+13)*q(k+14))
1938 c      desr = q(k+12)*rrr*dtsr
1939 c      esr = desr*(vs-q(k+13)) - q(k+16)*q(k+13)/q(k+11)
1940 c      tsr = q(k+15) + esr*rcvs
1941 1130 continue
1942 c      evaluate ps, ts, and derivatives
1943 c      pn(m) = q(k+11)*(es-esr)/vs + psr
1944 c      tn(m) = (es-esr)*rcvs + tsr
1945 c      dpnden(m) = q(k+11)/vs
1946 c      dpndvn(m) = -(es-esr+vs*desr)*q(k+11)/(vs*vs) + dpsr
1947 c      dtnden(m) = rcvs
1948 c      dtndvn(m) = -dtnden(m)*desr + dtsr
1949 c      go to 4000
1950

```

```

1951 1200 continue
1952 c evaluate functions for hom bkw gas
1953 vn(m) = amax1( q(k+18), amin1( q(k+19), vn(m)) )
1954 vg = vn(m)
1955 eg = en(m)
1956 lvg = alog(vg)
1957 lpi = q(k+1)+lvg*(q(k+2)+lvg*(q(k+3)+lvg*(q(k+4)+lvg*q(k+5))))
1958 pgi = exp(lpi)
1959 lez = q(k+6)+lpi*(q(k+7)+lpi*(q(k+8)+lpi*(q(k+9)+lpi*q(k+10))))
1960 ez = exp(lez)
1961 egi = ez - q(k+17)
1962 gng = -( q(k+12)+lvg*(2.*q(k+13)+lvg*(3.*q(k+14)+
1963 1 lvg*4.*q(k+15))) )
1964 lti = q(k+11)+lvg*(q(k+12)+lvg*(q(k+13)+lvg*(q(k+14)+
1965 1 lvg*q(k+15))))
1966 tgi = exp(lti)
1967 c ln-ln derivatives
1968 dlpi = d(ln(pi))/d(ln(vg)), dlez = d(ln(ez))/d(ln(pi))
1969 c dlti = d(ln(tl))/d(ln(vg)), dgng = d(1/beta)/d(ln(vg))
1970 c gng = (1/beta) = -d(ln(tl))/d(ln(vg))
1971 dlpi = q(k+2)+lvg*(2.*q(k+3)+lvg*(3.*q(k+4)+lvg*4.*q(k+5)))
1972 dlez = q(k+7)+lpi*(2.*q(k+8)+lpi*(3.*q(k+9)+lpi*4.*q(k+10)))
1973 dlti = -gng
1974 dgng = -(2.*q(k+13)+lvg*(6.*q(k+14)+lvg*12.*q(k+15)))
1975 c evaluate pg, tg, and derivatives
1976 pn(m) = gng*(eg - egi)/vg + pgi
1977 tn(m) = (eg - egi)/(q(k+16)*rrr) + tgi
1978 dpnden(m) = gng/vg
1979 dpndvn(m) = ((eg-egi)*(dgng-gng)+dlpi*(pgi*vg-gng*ez*dlez))
1980 1 / (vg*vg)
1981 dtnden(m) = 1./((q(k+16)*rrr)
1982 dtndvn(m) = -(dtnden(m)*dlez*dlpi*ez+tgi*dlti)/vg
1983 go to 4000
1984 c
1985 1300 continue
1986 c evaluate functions for jwl
1987 vn(m) = amax1( q(k+11)/q(k+8), vn(m) )
1988 vjwl = q(k+8)*vn(m)
1989 ejwl = q(k+8)*en(m)
1990 gng = q(k+6)
1991 cvg = q(k+8)*q(k+10)*rrr
1992 aexp = q(k+1)*exp(-q(k+4)*vjwl)
1993 bexp = q(k+2)*exp(-q(k+5)*vjwl)
1994 vmw = +1.0/(vjwl**gng)
1995 cvmw = q(k+3)*vmw
1996 pgi = aexp + bexp + cvmw/vjwl
1997 egi = (aexp/q(k+4)) + (bexp/q(k+5)) + (cvmw/gng) - q(k+7)
1998 tgi = q(k+9)*vmw
1999 c isentrope derivatives
2000 dpgi = -q(k+4)*aexp - q(k+5)*bexp - (gng+1.)*cvmw/vjwl
2001 degi = -pgi
2002 dtgi = -gng*tgi/vjwl
2003 c evaluate pg, tg, and derivatives in specific variables vg and eg
2004 pn(m) = gng*(ejwl-egi)/vjwl + pgi
2005 tn(m) = (ejwl-egi)/cvg + tgi
2006 dpnden(m) = q(k+8)*gng/vjwl
2007 dpndvn(m) = q(k+8)*( dpgi-gng*(vjwl*degi+ejwl-egi)/(vjwl**2) )
2008 dtnden(m) = q(k+8)/cvg
2009 dtndvn(m) = q(k+8)*(-degi/cvg + dtgi )
2010 go to 4000
2011 c
2012 4000 continue
2013 c
2014 if(.not.bmix) go to 6300
2015 c
2016 c two component mixture, solve for pressure-temperature equilibrium
2017 c
2018 c solve: p1(v1,e1)-p2(v2,e2) = 0, t1(v1,e1)-t2(v2,e2) = 0,
2019 c w*v1 + (1-w)*v2 - v = 0, w*e1 + (1-w)*e2 - e = 0
2020 c
2021 c newton's solution method for 4x4 system : (df/dx)(dx) = (-f0)
2022 c
2023 c ( dp1dv1 dp1de1 -dp2dv2 -dp2de2 )( dv1 ) ( p2 - p1 )
2024 c ( dt1dv1 dt1de1 -dt2dv2 -dt2de2 )( de1 ) = ( t2 - t1 )
2025 c ( w 0 1-w 0 )( dv2 ) ( v-w*v1-omw*v2 )
2026 c ( 0 w 0 1-w )( de2 ) ( e-w*e1-omw*e2 )
2027 c
2028 c form cofactor matrix ( cof )
2029 c note: inverse = transpose cofactor / determinant
2030 r1 = omw*dp1dv1 + w*dp2dv2

```

```

2031      r2 = omw*dp1del + w*dp2de2
2032      r3 = omw*dt1dvl + w*dt2dv2
2033      r4 = omw*dt1del + w*dt2de2
2034      r5 = dp2de2*dt1del - dp1del*dt2de2
2035      r6 = dp2dv2*dt1dvl - dp1dvl*dt2dv2
2036      det = r1*r4 - r2*r3
2037      if(abs(det) .lt. +1.0e-14) go to 6100
2038      rdet = 1.0/det
2039      cof(1,1) = omw*r4
2040      cof(2,1) = -omw*r2
2041      cof(3,1) = -dt2dv2*r2 + dp2dv2*r4
2042      cof(4,1) = omw*r5
2043      cof(1,2) = -omw*r3
2044      cof(2,2) = omw*r1
2045      cof(3,2) = -omw*r6
2046      cof(4,2) = dt2de2*r1 - dp2de2*r3
2047      cof(1,3) = -w*r4
2048      cof(2,3) = w*r2
2049      cof(3,3) = dp1dvl*r4 - dt1dvl*r2
2050      cof(4,3) = -w*r5
2051      cof(1,4) = w*r3
2052      cof(2,4) = -w*r1
2053      cof(3,4) = w*r6
2054      cof(4,4) = dt1del*r1 - dp1del*r3
2055      if(bcvrq) go to 6200
2056 c
2057 c      right hand side
2058      dp = p2 - p1
2059      dt = t2 - t1
2060      dv = v - w*v1 - omw*v2
2061      de = e - w*e1 - omw*e2
2062 c      solve linear system
2063      dv1 = ( cof(1,1)*dp+cof(2,1)*dt+cof(3,1)*dv+cof(4,1)*de )*rdet
2064      del = ( cof(1,2)*dp+cof(2,2)*dt+cof(3,2)*dv+cof(4,2)*de )*rdet
2065      dv2 = ( cof(1,3)*dp+cof(2,3)*dt+cof(3,3)*dv+cof(4,3)*de )*rdet
2066      de2 = ( cof(1,4)*dp+cof(2,4)*dt+cof(3,4)*dv+cof(4,4)*de )*rdet
2067 c      advance variables
2068      v1 = v1 + dv1
2069      e1 = e1 + del
2070      v2 = v2 + dv2
2071      e2 = e2 + de2
2072 c
2073      if(amax1(abs(dv1),abs(dv2)) .gt. cvrg0) go to 6000
2074      bcvrq = equal(p1,p2,cvrg0) .and. equal(t1,t2,cvrg0)
2075      6000 continue
2076 c
2077      6100 continue
2078 c      error exit for mixtures
2079      ierr(1) = 1
2080      encode( 80, 6101, ierr(2)) v,e,w
2081      6101 format(4x,16heosmix error, v=,1p,e12.4,4h e=,e12.4,4h w=,e12.4,
2082      1 16x )
2083      go to 6200
2084 c
2085      6110 continue
2086 c      error exit, no component id's recognized
2087      ierr(1) = 2
2088      encode( 80, 6111, ierr(2)) id1, id2
2089      6111 format(4x,35heosmix error, no components, id1 = ,a4,
2090      1 3x,6hid2 = ,a4,24x )
2091      return
2092 c
2093      6200 continue
2094 c      normal exit for mixtures
2095 c      partial derivatives calculated at solution point
2096 c
2097 c      partial with respect to (v)
2098 c      right hand side = ( 0, 0, 1, 0)
2099 c      dv1dv=cof(3,1)*rdet, del1dv=cof(3,2)*rdet,
2100 c      dv2dv=cof(3,3)*rdet, de2dv=cof(3,4)*rdet
2101 c      dp1dv = ( dp1dvl*cof(3,1) + dp1del*cof(3,2) )*rdet
2102 c      dp2dv = ( dp2dv2*cof(3,3) + dp2de2*cof(3,4) )*rdet
2103 c      dt1dv = ( dt1dvl*cof(3,1) + dt1del*cof(3,2) )*rdet
2104 c      dt2dv = ( dt2dv2*cof(3,3) + dt2de2*cof(3,4) )*rdet
2105 c
2106 c      partial with respect to (e)
2107 c      right hand side = ( 0, 0, 0, 1)
2108 c      dvlde=cof(4,1)*rdet, delde=cof(4,2)*rdet,
2109 c      dv2de=cof(4,3)*rdet, de2de=cof(4,4)*rdet
2110 c      dplde = ( dpldvl*cof(4,1) + dpldel*cof(4,2) )*rdet

```

```

2111      dp2de = ( dp2dv2*cof(4,3) + dp2de2*cof(4,4) ) * rdet
2112      dtlde = ( dtldv1*cof(4,1) + dtldel*cof(4,2) ) * rdet
2113      dt2de = ( dt2dv2*cof(4,3) + dt2de2*cof(4,4) ) * rdet
2114 c
2115 c      partial with respect to (w)
2116 c      right hand side = ( 0, 0, v2-v1, e2-e1 )
2117 c      dvlde = ( cof(3,1)*(v2-v1) + cof(4,1)*(e2-e1) ) * rdet
2118 c      delde = ( cof(3,2)*(v2-v1) + cof(4,2)*(e2-e1) ) * rdet
2119 c      dv2de = ( cof(3,3)*(v2-v1) + cof(4,3)*(e2-e1) ) * rdet
2120 c      de2de = ( cof(3,4)*(v2-v1) + cof(4,4)*(e2-e1) ) * rdet
2121      dv = v2 - v1
2122      de = e2 - e1
2123      dplde = ( dpldv1*(cof(3,1)*dv + cof(4,1)*de) +
2124      1      dplde1*(cof(3,2)*dv + cof(4,2)*de) ) * rdet
2125      dp2de = ( dp2dv2*(cof(3,3)*dv + cof(4,3)*de) +
2126      1      dp2de2*(cof(3,4)*dv + cof(4,4)*de) ) * rdet
2127      dtlde = ( dtldv1*(cof(3,1)*dv + cof(4,1)*de) +
2128      1      dtldel*(cof(3,2)*dv + cof(4,2)*de) ) * rdet
2129      dt2de = ( dt2dv2*(cof(3,3)*dv + cof(4,3)*de) +
2130      1      dt2de2*(cof(3,4)*dv + cof(4,4)*de) ) * rdet
2131 c
2132      vv(4) = w*p1 + omw*p2
2133      vv(5) = w*t1 + omw*t2
2134 c* * * modification 7 Oct 1986 J N Johnson
2135 c* * * vv(6) is used for particle velocity
2136 c      vv(6) = 0.
2137      vv(7) = w*dpldv + omw*dp2dv
2138      vv(8) = w*dplde + omw*dp2de
2139      vv(9) = w*dpldw + omw*dp2dw
2140      vv(10) = v*v*amax1(0., vv(4)*vv(8)-vv(7))
2141 c* * * modification 29 Oct 1986 J N Johnson
2142 c* * * vv(11) is used for (dp/du)h
2143      deltav=param1(13)-vv(1)
2144      if(deltav.eq.0.)then
2145          podv=vv(10)/(param1(13))**2
2146      else
2147          podv=vv(10)/deltav
2148      endif
2149      vv(11) = ( podv*vv(6)*vv(8)-2.*vv(7)*sqrt(podv) )
2150      # / ( podv - vv(7) )
2151 c      vv(11) = w*dtldv + omw*dt2dv
2152 c      vv(12) = w*dtlde + omw*dt2de
2153 c      vv(13) = w*dtldw + omw*dt2dw
2154      return
2155 c
2156 6300 continue
2157 c      exit for single component
2158      vv(4) = p1
2159      vv(5) = t1
2160 c* * * modification 7 Oct 1986 J N Johnson
2161 c* * * vv(6) is used for particle velocity
2162 c      vv(6) = 0.
2163      vv(7) = dpldv1
2164      vv(8) = dplde1
2165      vv(9) = 0.
2166      vv(10) = v*v*amax1(0., vv(4)*vv(8)-vv(7))
2167 c* * * modification 29 Oct 1986 J N Johnson
2168 c* * * vv(11) is used for (dp/du)h
2169      deltav=param1(13)-vv(1)
2170      if(deltav.eq.0.)then
2171          podv=vv(10)/(param1(13))**2
2172      else
2173          podv=vv(10)/deltav
2174      endif
2175      vv(11) = ( podv*vv(6)*vv(8)-2.*vv(7)*sqrt(podv) )
2176      # / ( podv - vv(7) )
2177 c      vv(11) = dtldv1
2178 c      vv(12) = dtldel
2179 c      vv(13) = 0.
2180      return
2181      end

```

```

2182      subroutine today( wkday, cdate)
2183 c      lasl program library.... q124b
2184 c      ftn version -- 8hmm/dd/yy
2185 c      chuck forest, november 1974
2186
2187      integer d, y
2188      integer wkday,cdate,days
2189      dimension month(12)
2190      dimension kk(48), days(7)
2191      data month / 3hjan, 3hfeb, 3hmar, 3hapr, 3hmay, 3hjun,
2192 a      3hjul, 3haug, 3hsep, 3hoct, 3hnov, 3hdec /
2193
2194      data days / 4hsun., 4hmon., 5htues., 4hwed.,
2195 1      6thurs., 4hfri., 4hsat. /
2196
2197      data kk / 0,3,4,0,2,5,0,3,6,1,4,6,
2198 1      2,5,5,1,3,6,1,4,0,2,5,0,
2199 2      3,6,6,2,4,0,2,5,1,3,6,1,
2200 3      4,0,0,3,5,1,3,6,2,4,0,2 /
2201
2202      call dateh(mmddyy)
2203      decode( 8, 100, mmddyy) m,d,y
2204      if(d .gt. 9) encode( 7, 101, cdate) d,month(m),y
2205      if(d .le. 9) encode( 7, 102, cdate) d,month(m),y
2206
2207      y = y+1900
2208      i = mod(y-1920,28)
2209      j = mod(i,4)*12 + m
2210      k = mod( d-1 + kk(j) + 5*(i/4) + 4, 7) + 1
2211      wkday = days(k)
2212      return
2213 100 format(i2,1x,i2,1x,i2)
2214 101 format(i2,a3,i2)
2215 102 format(1h0,1l,a3,i2)
2216      end

```

```

2217 c * * * * *
2218 c   FDRAG calculates the fractional time devoted to
2219 c   dislocation drag as compared to thermal activation
2220 c   when fdrag = 0 it is all thermal activation
2221 c   when fdrag = 1 it is all dislocation drag
2222 c * * * * *
2223
2224   function fdrag( m, var )
2225   common/block1/ matr(2000),matl(2000), cll(2000), clr(2000),
2226   #             ida(2),eosa(25,2), idb(2), eosb(25,2)
2227   common/block2/ rc(15,2), nv
2228   dimension var(20)
2229
2230   s = 2.*abs( var(14) )
2231
2232   eps = 1.0e-12
2233   r = ( s - rc(5,m) )/( eps + var(17) - rc(5,m) )
2234   if( r .lt. 0. ) r = 0.
2235   exparg = (1. - r**0.66667)*rc(3,m)/var(5)
2236   if( exparg .gt. 100. ) exparg=+100.
2237   if( exparg .lt. -100. ) exparg=-100.
2238
2239 c       a negative value of rc(12,m) is read in if we want to
2240 c       calculate with a non-constant value of lambda given by
2241 c       lambda = mu*b/sigma-hat. Otherwise, constant value of
2242 c       lambda*nu(zero) is used.
2243 c       if rc(12,m) < 0, then rc(12,m) = - b*nu(zero)
2244 c       if rc(12,m) > 0, then rc(12,m) = lambda*nu(zero)
2245
2246   afac = rc(2,m)*rc(12,m)
2247   if( rc(12,m) .lt. 0. ) afac = -afac*eosa(3,m)/(eps + var(17))
2248
2249   deno1 = afac*sqrt(1. + (s/(rc(2,m)*rc(4,m)))**2)
2250   deno2 = s*exp( exparg )
2251
2252   fdrag = deno1/( deno1 + deno2 )
2253
2254   return
2255   end

```

```
2256 c * * * * *
2257 c   DUMMY1 is a dummy subroutine used for debugging purposes
2258 c * * * * *
2259
2260     subroutine dummy1
2261     return
2262     end
```



```

2263 c * * * * *
2264 c The PIX subroutine (and all following subroutines) were developed
2265 c by Charles A. Forest (LANL, group M-9) for creating plots using
2266 c LANL-CCF library routines
2267 c * * * * *
2268
2269 subroutine pix( nty,xlabel,ylabel,title,nx1,x1,y1,nx2,x2,y2,
2270 1 x0,y0 )
2271
2272 character xlabel*80,ylabel*80,title*80, line*8,mark*8,
2273 1 line0*8,mark0*8, font*8,font0*8, patrn*40, xtype*4,ytype*4,
2274 1 blank*8, title0*80
2275
2276 logical bpub, bpub0
2277
2278 dimension title(2),x1(nx1),y1(nx1),x2(nx2),y2(nx2),x0(2),y0(2),
2279 1 xmm(2),ymm(2),label0(7),
2280 1 xlabel(2), ylabel(2)
2281
2282 common / pixcom1 / line(6), mark(6)
2283 common / pixcom2 / jline,jmark,axmin,axmax,aymin,aymax,nframe
2284
2285 data linecy / 3 /
2286 data line / 'solid','dash','dash','sdash','dotdash','dct' /
2287 data markcy / 3 /
2288 data mark / 'circle','square','diamond','star','plus','x' /
2289
2290 data jline0,jmark0,number,size,nframe / 1, 1, 50, 0.5, 0 /
2291
2292
2293 c-----
2294
2295 c graph the curve (x1,y1) as connected line with no point symbols
2296 c plot the points (x2,y2) as unconnected point symbols
2297 c nty = type number of graph as follows:
2298 c If nty=1, (x,y) plot. If nty=2, (x,log(y)) plot.
2299 c If nty=3, (log(x),y) plot. If nty=4, (log(x),log(y)) plot.
2300 c If (x0(1),x0(2)) .ne. (0,0) , then
2301 c x0(1),x0(2) = xmin0,xmax0 and y0(1),y0(2) = ymin0,ymax0
2302 c and is compared to the data min,max
2303
2304 xmm(1) = 1.0e+300
2305 xmm(2) = - 1.0e+300
2306 call minmax( nx1,x1,xmm )
2307 call minmax(-nx2,x2,xmm )
2308 ymm(1) = 1.0e+300
2309 ymm(2) = - 1.0e+300
2310 call minmax( nx1,y1,ymm )
2311 call minmax(-nx2,y2,ymm )
2312 if(x0(1).ne.0. .or. x0(2).ne.0.) then
2313 call minmax( -2,x0,xmm )
2314 end if
2315 if(y0(1).ne.0. .or. y0(2).ne.0.) then
2316 call minmax( -2,y0,ymm )
2317 end if
2318
2319 call gpage
2320
2321 c Number the frame in lower left corner
2322 nframe = nframe + 1
2323 call gtstrm
2324 call lincnt( 61 )
2325 write(ioplot,'(1x,13)') nframe
2326
2327 call hcvmrk( 'nomarker',2,'off' )
2328 call hcvis( 'noline','off' )
2329 blank = " "
2330 if(nty .eq. 1) call hxy ( xmm,ymm,2,xlabel,ylabel,blank )
2331 if(nty .eq. 2) call hxlogy( xmm,ymm,2,xlabel,ylabel,blank )
2332 if(nty .eq. 3) call hylogy( xmm,ymm,2,xlabel,ylabel,blank )
2333 if(nty .eq. 4) call hlogy( xmm,ymm,2,xlabel,ylabel,blank )
2334
2335 call htiadd( title(2), 1)
2336 call httitle( title(1) )
2337
2338 xtype = 'lin'
2339 ytype = 'lin'
2340 if(nty.eq.3 .or. nty.eq.4) xtype='log'
2341 if(nty.eq.2 .or. nty.eq.4) ytype='log'
2342 call haxscl( xmm(1),xmm(2),0.75,xtype,axmin,axmax,xstep )

```

```

2343      call haxscl( ymm(1),ymm(2),0.70,ytype,aymin,aymax,ystep )
2344
2345      if( bpub ) then
2346      write('pixout','(5hframe,i3)') mod(nframe,1000)
2347
2348      n = lastch(title(1))
2349      if(n .gt. 0) title0 = title(1)(1:n)//$"
2350      if(n .eq. 0) title0 = "$"
2351      write('pixout','("title1 ="a64 )') title0
2352
2353      n = lastch(title(2))
2354      if(n .gt. 0) title0 = title(2)(1:n)//$"
2355      if(n .eq. 0) title0 = "$"
2356      write('pixout','("title2 ="a64 )') title0
2357
2358      write('pixout','("xlabel ="a64 )') xlabel(2)
2359      write('pixout','("ylabel ="a64 )') ylabel(2)
2360
2361      write('pixout','(" $frame maxsym=30, xtype="a5", ytype="a5,
2362      1 1h, / 1p," xstep="e12.4", ystep="e12.4,1h, /" xmin="
2363      1 e12.4", xmax="e12.4", ymin="e12.4", ymax="e12.4" $")')
2364      1 3h'//xtype, 3h'//ytype, xstep, ystep,
2365      1 axmin,axmax,aymin,aymax
2366      end if
2367
2368      jline = jline0
2369      jline0 = 1
2370
2371      jmark = jmark0
2372      jmark0 = 1
2373
2374 c-----
2375
2376      entry pixadd( nx1,x1,y1,nx2,x2,y2 )
2377
2378      if(nx1 .gt. 0) then
2379 c      (add) line to (pix) graph without symbols
2380      call hcvtls( line(jline),'off' )
2381      call hcvmrk( 'nomarker',1,'off' )
2382      call hcurve( x1,y1,nx1 )
2383
2384      if( bpub ) then
2385      write('pixout','(5hcurve)')
2386      write('pixout','(" $curve line=8h"a8", mark=5hblank, fac= 0.00",
2387      1 " , nxy= " ,i4" $")') line(jline), nx1
2388      write('pixout','(1x,lp6el3.4)') (x1(i),y1(i),i=1,nx1)
2389      end if
2390
2391      jline = mod(jline,linecy) + 1
2392      end if
2393
2394 c      ++++++
2395
2396      if(nx2 .gt. 0) then
2397 c      (add) data points to (pix) graph without connecting line.
2398      call hcvtls( 'noline','off' )
2399      call hcvmrk( mark(jmark),min0(nx2,number),'off' )
2400      call hmrksz( size )
2401      call hcurve( x2,y2,nx2 )
2402
2403      if( bpub ) then
2404      write('pixout','(5hcurve)')
2405      write('pixout','(" $curve line=5hblank, mark=7h"a7", fac= 1.00",
2406      1 " , nxy= " ,i4" $")') mark(jmark), nx2
2407      write('pixout','(1x,lp6el3.4)') (x2(i),y2(i),i=1,nx2)
2408      end if
2409
2410      jmark = mod(jmark,markcy) + 1
2411      end if
2412
2413      call gmakpc
2414      return
2415
2416 c-----
2417
2418      entry pixline( line0 )
2419
2420 c
2421 c      DEFINE: line type (line0)
2422 c      for next call to PIX or ADDPIX, it cycles thereafter

```

```

2423 c      character line0*8
2424 c      data line / 'solid','ldash','sdash','dot','dotdash','dash' /
2425
2426      call cmatch( line0,6,line,jline0 )
2427      jline = jline0
2428      return
2429
2430 c-----
2431
2432      entry pixmark( mark0 )
2433
2434 c      DEFINE: point marker type (mark0)
2435 c      for next call to PIX or ADDPIX, with cycle thereafter
2436 c      character mark0*8
2437 c      data mark / 'circle','plus','square','star','diamond','x' /
2438
2439      call cmatch( mark0,6,mark,jmark0 )
2440      jmark = jmark0
2441      return
2442
2443 c-----
2444
2445      entry pixnum( number0 )
2446
2447 c      DEFINE: number of markers (number)
2448      number = number0
2449      return
2450
2451 c-----
2452
2453      entry pixsiz( size0 )
2454
2455 c      DEFINE: marker size (size)
2456      size = size0
2457      return
2458
2459 c-----
2460
2461      entry pixpub( bpub0 )
2462
2463 c      bpub0 = .t. (write to pixout file)
2464 c      bpub0 = .f. (do not write to pixout file)
2465      bpub = bpub0
2466      return
2467
2468 c-----
2469
2470      entry pixup( icl,label0,ioplot0, bpub0,font0,ffont0 )
2471
2472 c      icl      = classification (id est: lhu, lhc, lhs; see GR800, GPLOT)
2473 c      label0   = graphics output label
2474 c      ioplot0  = logical unit number for the plot metafile
2475 c      font0    = DISSPLA font style (default = 5hduplex)
2476 c      ffont0   = factor for font character size (default = 1.)
2477
2478 c      character font*8,font0*8 / real ffont,ffont0
2479
2480      data ffont, font / 1.0, "duplx" /
2481
2482      call gplot( icl,label0,60 )
2483
2484      ioplot = ioplot0
2485      call grphlun( ioplot )
2486      call grphcft( 1 )
2487
2488      call libhi
2489      call setflsh
2490
2491      if(font0(1:1) .ne. " ") font = font0
2492
2493      if(ffont0 .ne. 0.) ffont = ffont0
2494
2495      bpub = bpub0
2496
2497      if( bpub ) then
2498      write('pixout','(7a8)') (label0(i),i=1,7)
2499      write('pixout','(" $style font=",a8," ffont=",f4.2,
2500      1 " ", btitle=.t. $"')' '6h'//font,ffont
2501      end if
2502

```

```

2503         return
2504
2505 c-----
2506
2507         entry pixnos
2508
2509 c         Number pix frame in lower left corner
2510
2511         nframe = nframe + 1
2512         call gtstrm
2513         call lincnt( 61 )
2514         write(ioplot,'(1x,i3)') nframe
2515         return
2516
2517 c-----
2518
2519         end

```

```

2520      subroutine pixver( line0,nz0,z0)
2521
2522      character line0*8, line*8, mark*8
2523
2524      dimension z0(nz0), xx(2), yy(2)
2525
2526      common / pixcom1 / line(6), mark(6)
2527      common / pixcom2 / jline,jmark,axmin,axmax,aymin,aymax,nframe
2528
2529 c-----
2530
2531 c      draw vertical lines across entire graph at x = (z0(i),i=1,nz0)
2532
2533      call cmatch( line0,6,line,jline1 )
2534      dyy = 0.0005*(aymax - aymin)
2535      yy(1) = aymin + dyy
2536      yy(2) = aymax - dyy
2537
2538      do 100 n=1,nz0
2539 c      reset jline; note that pixadd advances jline
2540      jline = jline1
2541      xx(1) = z0(n)
2542      xx(2) = z0(n)
2543      call pixadd( 2,xx,yy,0,0,0 )
2544      100 continue
2545
2546      return
2547
2548 c-----
2549
2550      entry pixhor( line0,nz0,z0 )
2551
2552 c      draw horizontal lines across entire graph at y = (z0(i),i=1,nz0)
2553
2554      call cmatch( line0,6,line,jline1 )
2555      dxx = 0.0005*(axmax - axmin)
2556      xx(1) = axmin + dxx
2557      xx(2) = axmax - dxx
2558
2559      do 200 n=1,nz0
2560 c      reset jline; note that pixadd advances jline
2561      jline = jline1
2562      yy(1) = z0(n)
2563      yy(2) = z0(n)
2564      call pixadd( 2,xx,yy,0,0,0 )
2565      200 continue
2566
2567      return
2568
2569 c-----
2570
2571      end

```

```

2572      subroutine minmax( nx,x,xmm )
2573      dimension x(nx), y(nx), xmm(2), ymm(2), jmm(2)
2574
2575      c      find minimum (xmm(1)) and maximum (xmm(2)) of array x
2576      c      the array x may not be monotone
2577      c      if(nx .gt. 0), no initial values for xmm
2578      c      if(nx .lt. 0), continue comparison to input xmm.
2579      c      if(nx .eq. 0), no action, return
2580
2581      if(nx .eq. 0) return
2582      if(nx .gt. 0) then
2583          xmm(1) = x(1)
2584          xmm(2) = x(1)
2585      end if
2586      inx = iabs(nx)
2587      do 100 i=1,inx
2588          xmm(1) = amin1(xmm(1),x(i))
2589          xmm(2) = amax1(xmm(2),x(i))
2590      100 continue
2591      return
2592
2593      entry mesh( nx,x,xmm )
2594      c      form equally spaced array x for the interval (xmm(1),xmm(2))
2595      nx1 = nx - 1
2596      dx = (xmm(2)-xmm(1))/float(nx1)
2597      x(1) = xmm(1)
2598      do 200 i=2,nx1
2599          x(i) = float(i-1)*dx + xmm(1)
2600      200 continue
2601      x(nx) = xmm(2)
2602      return
2603
2604      entry minmax2( nx,x,y,xmm,ymm,jmm )
2605      c      find maximum and minimum of array (y), return values in (ymm)
2606      c      return also corresponding (x) and index values in (xmm) and (jmm)
2607
2608      xmm(1) = x(1)
2609      xmm(2) = x(1)
2610      ymm(1) = y(1)
2611      ymm(2) = y(1)
2612      jmm(1) = 1
2613      jmm(2) = 1
2614      do 300 i=2,nx
2615          if(y(i) .lt. ymm(1)) then
2616              jmm(1) = i
2617              xmm(1) = x(i)
2618              ymm(1) = y(i)
2619          end if
2620          if(y(i) .gt. ymm(2)) then
2621              jmm(2) = i
2622              xmm(2) = x(i)
2623              ymm(2) = y(i)
2624          end if
2625      300 continue
2626      return
2627      end

```

```

2628      subroutine lbound( xmin,ymin,nx,x,y,nx1,x1,y1 )
2629      dimension x(nx),y(nx),x1(nx),y1(nx)
2630
2631      c      (l)ower (bound)
2632      c      define arrays (x1,y1) from arrays (x,y)
2633      c      so that for each (x1,y1), (x1 > xmin) and (y1 > ymin)
2634
2635      nxa = 0
2636      do 100 i=1,nx
2637      if((x(i) .lt. xmin) .or. (y(i) .lt. ymin)) go to 100
2638      nxa = nxa + 1
2639      x1(nxa) = x(i)
2640      y1(nxa) = y(i)
2641 100 continue
2642      nx1 = nxa
2643      return
2644
2645      entry ubound( xmax,ymax,nx,x,y,nx1,x1,y1 )
2646      c      (u)pper (bound)
2647      c      define arrays (x1,y1) from arrays (x,y)
2648      c      so that for each (x1,y1), (x1 < xmax) and (y1 < ymax)
2649
2650      nxa = 0
2651      do 200 i=1,nx
2652      if((x(i) .gt. xmax) .or. (y(i) .gt. ymax)) go to 200
2653      nxa = nxa + 1
2654      x1(nxa) = x(i)
2655      y1(nxa) = y(i)
2656 200 continue
2657      nx1 = nxa
2658      return
2659      end

```

```

2660      subroutine cmatch( item,nlist,list,index )
2661 c      find match for character string (item) in character array (list)
2662
2663      character item*(*),list*(*)
2664      dimension list(nlist), x(nx), ix(nix)
2665
2666      index = 0
2667      do 100 n=1,nlist
2668      if(item .ne. list(n)) go to 100
2669      index = n
2670      go to 102
2671 100 continue
2672 102 return
2673 c-----
2674      entry rmatch( x0,nx,x,index )
2675 c      find match for real variable (x0) in real array (x)
2676
2677      index = 0
2678      ax0 = abs(x0)
2679      do 200 n=1,nx
2680      if(abs(x0-x(n)) .gt. 1.0e-10*amax1(ax0,abs(x(n)))) go to 200
2681      index = n
2682      go to 202
2683 200 continue
2684 202 return
2685 c-----
2686      entry imatch( ix0,nix,ix,index )
2687 c      find match for integer variable (ix0) in integer array (ix)
2688
2689      index = 0
2690      do 300 n=1,nix
2691      if(ix0 .ne. ix(n)) go to 300
2692      index = n
2693      go to 302
2694 300 continue
2695 302 return
2696      end

```



```

2697      function lastch( ch )
2698
2699 c      determine the position of the last nonblank character
2700
2701      character ch*(*)
2702
2703      nch = len(ch)
2704      lastch = 0
2705      do 100 i=nch,1,-1
2706      if(ch(i:i) .ne. " ") then
2707      lastch = i
2708      go to 110
2709      end if
2710      100 continue
2711      110 return
2712
2713      end

```

```
2714 */
2715 * rcft i=cha lib=(cgsh,gh,cgscft,cftmath)
2716 *destroy cha bcha mcha
```

REFERENCES

1. J. Von Neumann and R. D. Richtmyer, "A Method for the Numerical Calculation of Hydrodynamic Shocks," J. Appl. Phys. 21, pp. 232-237 (1950).
2. D. C. Wallace, "Thermoelastic-Plastic Flow in Solids," Los Alamos National Laboratory Report LA-10119, June (1985).
3. D. C. Wallace, "Irreversible Thermodynamics of Plastic Flow in Solids," Phys. Rev. B, 22, pp. 1477-1486 (1980).
4. P. S. Follansbee and U. F. Kocks, "A Constitutive Description of the Deformation of Copper Based on the Use of the Mechanical Threshold Stress as an Internal State Variable," Acta Metall. 36, pp. 81-93 (1988).
5. P. S. Follansbee, "The Rate-Dependence of Structure Evolution in Copper and Its Influence on the Stress-Strain Behavior at Very High Rates," IMPACT '87 International Conference on Impact Loading and Dynamic Behavior of Materials, Bremen, FDR, 18-21 May (1987).
6. R. N. Thurston, "Wave Propagation in Fluids and Normal Solids," in Physical Acoustics, Vol. I-Part A, Warren P. Mason, Editor, Academic Press, New York, pp. 1-110 (1964).
7. D. C. Wallace, "Thermoelastic Theory of Stressed Crystals and Higher-Order Elastic Constants," Solid State Phys. (Academic Press, Inc., New York) 25, pp. 301-404 (1970).
8. K. Brugger, "Pure Modes for Elastic Waves in Crystals," J. Appl. Phys. 36, pp. 759-768 (1965).
9. M. W. Guinan and D. J. Steinberg, "Pressure and Temperature Derivatives of the Isotropic Polycrystalline Shear Modulus for 65 Elements," J. Physics and Chem. of Solids 35, pp. 1501-1512 (1974).
10. Single Crystal Elastic Constants and Calculated Aggregate Properties: A Handbook, Edited by G. Simmons and H. Wang, M.I.T. Press, Cambridge, MA (1971).
11. Thermophysical Properties of Matter, Vol. 4, Edited by Y. S. Touloukian and E. H. Buyco, Plenum Press, New York, p. 57 (1970).
12. Thermophysical Properties of Matter, Vol. 12, Edited by Y. S. Touloukian, R. K. Kirby, R. E. Taylor, and P. D. Desai, Plenum Press, New York, p. 77 (1975).
13. J. W. Swegle and D. E. Grady, "Shock Viscosity and the Prediction of Shock-Wave Risetimes," J. Appl. Phys. 58, pp. 692-701 (1985).

14. P. van't Klooster, N. J. Trappeniers, and S. N. Biswas, "Effect of Pressure on the Elastic Constants of Noble Metals from - 196 to + 25°C and up to 2500 bars," *Physica* 97B, pp. 65-75 (1979).
15. N. J. Trappeniers, S. N. Biswas, and C. A. Ten Seldom, "Effect of Pressure on the Shear Modulus of Polycrystalline Aluminum, Copper, and Silver," *Physica* 85B, pp. 20-32 (1977).
16. D. L. Tonks, Los Alamos National Laboratory, unpublished (1990).
17. T. J. Ahrens and G. E. Duvall, "Stress Relaxation Behind Elastic Shock Waves in Rocks," *J. Geophys. Res.* 71, pp. 4349-4360 (1966).
18. R. J. Clifton, "Plastic Waves: Theory & Experiment," in Mechanics Today, Edited by S. Nemat-Nasser, Vol. 1, Pergamon Press, New York, pp. 102-167 (1972).
19. Y. Partom, "Elastic Precursor Decay Calculation," *J. Appl. Phys.* 59, pp. 2716-2727 (1986).
20. R. H. Warnes and D. L. Tonks, "Measurement and Analysis of 3-GPa Shock-Wave Profiles in Annealed OFE Copper," Shock Compression of Condensed Matter - 1989. Edited by S. C. Schmidt, J. N. Johnson, and L. W. Davison, Elsevier Science Publishers B. V. Amsterdam, pp. 329-332 (1990).
21. D. L. Tonks and J. N. Johnson, "Shock-Wave Evolution of the Mechanical Threshold Stress in Copper," Shock Compression of Condensed Matter - 1989. Edited by S. C. Schmidt, J. N. Johnson, and L. W. Davison, Elsevier Science Publishers B. V. Amsterdam, pp. 333-336 (1990).
22. Y. Partom, "Understanding the Swegle-Grady $\dot{\epsilon} \sim \sigma^4$ Relation for Steady Plastic Waves," Shock Compression of Condensed Matter - 1989. Edited by S. C. Schmidt, J. N. Johnson, and L. W. Davison, Elsevier Science Publishers B. V. Amsterdam, pp. 317-319 (1990).
23. J. N. Johnson and L. M. Barker, "Dislocation Dynamics and Steady Plastic Wave Profiles in 6061-T6 Aluminum," *J. Appl. Phys.* 40, pp. 4321-4334 (1969).
24. C. A. Forest, "The Impulse Time-Integral Function and Lagrangian Analysis," *Bull. Am. Phys. Soc.* 34, p. 1736 (1989).
25. G. T. Gray III and P. S. Follansbee, "Influence of Peak Pressure and Pulse Duration on Substructure Development and Threshold Stress Measurements in Shock-Loaded Copper," in Impact Loading and Dynamic Behavior of Materials, edited by C. Y. Chiem, H.-D. Kunze, and L. W. Meyer, Deutsche Gesellschaft fur Metalkunde, Vol. 2, pp. 541-548, Springer-Verlag (1988).
26. R. Courant and K. O. Friedrichs, Supersonic Flow and Shock Waves, Interscience Publishers, Inc., New York, pp. 121-124 (1948).

This report has been reproduced directly from
the best available copy.

Available to DOE and DOE contractors from
the Office of Scientific and Technical Information
P.O. Box 62
Oak Ridge, TN 37831
prices available from
(615) 576-8401, FTS 626-8401

Available to the public from
the National Technical Information Service
U.S. Department of Commerce
5285 Port Royal Rd.
Springfield, VA 22161

Microfiche A01

NTIS		NTIS		NTIS		NTIS	
Page Range	Price Code	Page Range	Price Code	Page Range	Price Code	Page Range	Price Code
001-025	A02	151-175	A08	301-325	A14	451-475	A20
026-050	A03	176-200	A09	326-350	A15	476-500	A21
051-075	A04	201-225	A10	351-375	A16	501-525	A22
076-100	A05	226-250	A11	376-400	A17	526-550	A23
101-125	A06	251-275	A12	401-425	A18	551-575	A24
126-150	A07	276-300	A13	426-450	A19	576-600	A25
						601-up*	A99

*Contact NTIS for a price quote.

END

DATE FILMED

02 / 07 / 91