Explicit Robust Schemes for Implementation of a Class of Principal Value-Based Constitutive Models: Symbolic and Numeric Implementation

S.M. Arnold
Lewis Research Center
Cleveland, Ohio

and

A.F. Saleeb, H.Q. Tan, and Y. Zhang
University of Akron
Akron, Ohio

May 1993
Explicit Robust Schemes for Implementation of a Class of Principal Value-Based Constitutive Models: Symbolic and Numeric Implementation

S. M. Arnold
National Aeronautics and Space Administration
Lewis Research Center
Cleveland, Ohio 44135

A. F. Saleeb, H. Q. Tan, and Y. Zang
University of Akron
Akron, Ohio 44325

Abstract
The issue of developing effective and robust schemes to implement a class of the Ogden-type hyperelastic constitutive models is addressed. To this end, special purpose functions (running under MACSYMA) are developed for the symbolic derivation, evaluation, and automatic FORTRAN code generation of explicit expressions for the corresponding stress function and material tangent stiffness tensors. These explicit forms are valid over the entire deformation range, since the singularities resulting from repeated principal-stretch values have been theoretically removed. The required computational algorithms are outlined, and the resulting FORTRAN computer code is presented.

1 Introduction
To a great extent, constitutive models of the so-called generalized Rivlin-Mooney type [1,2] (i.e., with the stored strain energy density written as a polynomial function in terms of the deformation invariants) have dominated the phenomenological theory of isotropic hyperelasticity [1-6]. Such models dominate the related computational literature on finite-strain elasticity [7-9] as well. Recently
though, alternative representations in terms of the principal stretches have become increasingly popular in nonlinear finite element analyses [6,8,10]. However, from the viewpoint of numerical implementation, the use of these models presents a number of unique and difficult problems, which do not arise in classical representations using the strain invariants. The main difficulty is that (in addition to being reasonably complicated functions of the strain components) taken separately, the main constituents of the deformation tensor (i.e., principal values and associated eigenvectors) are, in general, not uniquely defined and continuously differentiable functions. A careful consideration is thus called for in implementing constitutive models formulated in terms of these principal-strain measures; this was the main problem addressed by Saleeb and Arnold [11]. They bypassed the difficulty entirely by resorting to explicit derivations of appropriate forms of the material tangent-stiffness matrices, which are valid for the entire deformation range. The explicit expressions they developed [11] were for two specific forms of the Ogden-type, strain-energy functions, which actually encompass many of the popular representations currently in use for rubber materials. Results were obtained by simply applying a systematic limiting procedure for one type of tensor-valued function and its spectral representation.

Symbolic computation specializes in exact computation with numbers, formulas, vectors, matrices, equations and the like. Numerical computation, on the other hand, uses floating-point numbers to compute approximate solutions to problems of practical interest. The two approaches are complementary and, when combined into an integrated form, can be very powerful in engineering applications. In particular, application of symbolic manipulation can provide significant incentive for the development of new constitutive theories and their applications, for example, finite element. Recently, a problem-oriented, self-contained, symbolic expert system, named SDICE (see [12-13]), was developed; it is capable of efficiently deriving, in analytical form, potential based constitutive models whose representations are in terms of the classical invariant formulation [14-15]. In addition, the FORTRAN code associated with the resulting analytical expressions can be automatically generated.

The objective of the present paper is to discuss three special purpose functions (SDIFF, SDIFFEV, and TEMPLATE) running under DOE MACSYMA [16]. These three functions have been developed to allow the derivation and automatic FORTRAN code generation of alternative potential based constitutive models composed of principal values and their associated eigenvectors, as discussed in reference 11. All three functions are written at the MACSYMA command level. In the future, these functions will be integrated into the collection of special purpose functions known as SDICE. This paper begins by reviewing highlights of the previous theoretical development and discussing the associated computer algorithm for the derivation of the explicit expressions for the second Piola Kirchhoff stress tensor $S_{ij}$ and the material moduli tensor $D_{ijkl}$. The paper concludes with the evaluation of a separable strain energy function, similar to that discussed in reference 11, and its associated FORTRAN
source code generation.

2 Background

The theoretical development of singularity-free representations for principal value-based constitutive models has been discussed at length in reference 11. For brevity, we will confine our discussion, for illustrative purposes, to hyperelastic isotropic materials whose strain energy function $W$ can be taken to have the following separable functional dependence:

$$W = W(\lambda_i) = \sum_{n=1}^{p} a_n (\lambda_1^{\alpha} + \lambda_2^{\alpha} + \lambda_3^{\alpha})$$

where $\lambda_i$ represents the principal values of the right Cauchy-Green deformation tensor $C_{ij}$. Denoting $n_i$ ($i = 1, 2, 3$) to be the associated eigenvectors of $C_{ij}$, we can define

$$C_{ij} = \sum_{l=1}^{3} \lambda(i)N_{ij}^{(l)}$$

where $N_{ij}^{(l)}$ is defined as

$$N_{ij}^{(l)} = n_i^l n_j^l$$

and is often referred to as the (orthogonal) eigenprojection operator related to the associated eigenvectors of $C_{ij}$.

Equation (2) is valid for the case when all three eigenvalues, $\lambda_i$, are distinct. However, for the case when two eigenvalues are the same (i.e., double coalescence $\lambda_1 \neq \lambda_2 = \lambda_3 = \lambda$) we have

$$C_{ij} = (\lambda_1 - \lambda)N_{ij}^{(1)} + \lambda \delta_{ij}$$

And for the case of triple coalescence ($\lambda_1 = \lambda_2 = \lambda_3 = \lambda$), we have

$$C_{ij} = \lambda \sum_{l=1}^{3} N_{ij}^{(l)} = \lambda \delta_{ij}$$

Similarly, by manipulating equations (2) and (4), we can obtain explicit expressions for $N_{ij}^{(r)}$ in terms of $C_{ij}$:

$$N_{ij}^{(r)} = \frac{1}{(\lambda_r - \lambda_s)(\lambda_s - \lambda_t)} [(C_{ij} - \lambda_s \delta_{ij})(C_{ij} - \lambda_t \delta_{ij})]$$

and
In the preceding equations, the $r$, $s$, and $t$ are any cyclic permutation of $(1, 2, \text{or } 3)$. These definitions, equations (2) through (7), will prove very useful in obtaining the pertinent singularity-free directional derivatives of both the strain-energy potential function $W$ and the stress function $S_{ij} = S_{ij}(C_{ij})$.

The explicit singularity-free expressions for the second Piola Kirchhoff stress tensor $S_{ij}(C_{ij})$ are defined as

$$S_{ij} = 2 \frac{\partial W}{\partial C_{ij}} = S_{ij}(C_{ij})$$

and those for the material moduli tensor $D_{ijkl}(C_{ij})$ are obtained by applying the directional derivative formula to $S_{ij}$, that is

$$D_{ijkl} = 2 \frac{\partial S_{ij}}{\partial C_{kl}} = 4 \frac{\partial^2 W}{\partial C_{ij} \partial C_{kl}} = D_{ijkl}(C_{ij})$$

As a result, the explicit expressions of the functional dependence of tensors $S_{ij}$ and $D_{ijkl}$ on $C_{ij}$ can be obtained directly for the following three cases: 1) all three eigenvalues are distinct; 2) a single singularity ($\lambda_1 \neq \lambda_2 = \lambda_3 = \lambda$, i.e., double coalescence) is present; or 3) a double singularity ($\lambda_1 \neq \lambda_2 = \lambda_3 = \lambda$, i.e., triple coalescence) is present.

### 3 Computer Algorithm

The objective of the present study was to construct three special purpose functions (SDIFF, SDIFFEV, and TEMPLATE) written at the MACSYMA command level that can, respectively,

1. Derive explicit expressions for the stress tensor $S_{ij}$ (eqs. (8)) and material tensor $D_{ijkl}$ (eqs. (9)) given three, one, or no distinct eigenvalues

2. Evaluate symbolically the expressions generated by SDIFF for a given strain-energy function $W$

3. Evaluate the expressions generated by SDIFF and use the built-in MACSYMA function gentran to automatically generate the associated FORTRAN code needed to evaluate the expressions numerically for a given function, $W$.

These special purpose functions contain a list of built-in MACSYMA instructions (factor, expand, ev, ratsubst, diff, limit, and for-loops, to name a few) arranged in a specific algorithmic order. Each function, then, can be thought of as a macro command.
3.1 SDIFF (case)

Issuing the command SDIFF invokes the following algorithm (consisting of 15 steps) for automatic derivation of $S_{ij}$ and $D_{ij}$ in. In this context, case $\equiv 1$ indicates that all three eigenvalues are distinct; case $\equiv 2$ indicates that only one is distinct; and case $\equiv 3$, that none are distinct.

To obtain $S_{ij}$,

1. Differentiate $W$ with respect to $C_{ij}$ (see eq. (8))

$$S_{ij} = \sum_{i=1}^{3} 2 \frac{\partial W}{\partial \lambda_{(1)}} \frac{\partial \lambda_{(1)}}{\partial C_{ij}}$$

2. Apply the special directional derivative rules obtained from equation (2), that is,

$$N_{ij}^{(1)} = \frac{\partial \lambda_{(1)}}{\partial C_{ij}}$$

whose value is given in equation (6).

3. Obtain typical scalar derivatives by using the built-in diff command:

$$s(\lambda_{(1)}) = 2 \frac{\partial W}{\partial \lambda_{(1)}}$$

4. Multiply the results $s(\lambda_{(1)})$ and $N_{ij}^{(1)}$, then sum and factor out coefficients of like terms (i.e., $C_{ik}C_{kj}$, $C_{ij}$, and $\delta_{ij}$), thereby obtaining the functional dependence of $S_{ij}$ on $C_{ij}$. In the case of three distinct eigenvalues,

$$S_{ij} = aC_{ik}C_{kj} + bC_{ij} + c\delta_{ij}$$

where $\delta_{ij}$ is the second order identity tensor and

$$a = -m[s(\lambda_1)(\lambda_2 - \lambda_3) + s(\lambda_2)(\lambda_3 - \lambda_1) + s(\lambda_3)(\lambda_1 - \lambda_2)]$$

$$b = m[s(\lambda_1)(\lambda_3^2 - \lambda_2^2) + s(\lambda_2)(\lambda_1^2 - \lambda_3^2) + s(\lambda_3)(\lambda_1^2 - \lambda_2^2)]$$

$$c = -m[s(\lambda_1)\lambda_2\lambda_3(\lambda_2 - \lambda_3) + s(\lambda_2)\lambda_3\lambda_1(\lambda_3 - \lambda_1) + s(\lambda_3)\lambda_1\lambda_2(\lambda_1 - \lambda_2)]$$

and

$$m = \frac{1}{(\lambda_1 - \lambda_2)(\lambda_2 - \lambda_3)(\lambda_3 - \lambda_1)}$$
To obtain \( D_{ijkl} \):

(5) Differentiate \( S_{ij} \) with respect to \( C_{kl} \) (see eq. (9)):

\[
D_{ijkl} = 2a[\frac{1}{2}(\delta_{ik} \delta_{ml} + \delta_{il} \delta_{mk})C_{mj} + \frac{1}{2}C_{im}(\delta_{jk} \delta_{ml} + \delta_{jl} \delta_{mk})]
\]

\[+ \sum_{r=1}^{3} \frac{\partial a}{\partial \lambda_r} \frac{\partial \lambda_r}{\partial C_{kl}} C_{im} C_{mj} + b[\frac{1}{2}(\delta_{ik} \delta_{jl} + \delta_{il} \delta_{jk})]
\]

\[+ \sum_{r=1}^{3} \frac{\partial b}{\partial \lambda_r} \frac{\partial \lambda_r}{\partial C_{kl}} C_{ij} + \sum_{r=1}^{3} \frac{\partial c}{\partial \lambda_r} \frac{\partial \lambda_r}{\partial C_{kl}} \delta_{ij} \]  \hspace{1cm} (18)

(6) Apply the special directional derivative rule

\[
N_{ij}^{(r)} = \frac{\partial \lambda_r}{\partial C_{ij}}
\]  \hspace{1cm} (19)

(7) Obtain the nine scalar derivatives,

\[
\frac{\partial a}{\partial \lambda_r}, \frac{\partial b}{\partial \lambda_r}, \frac{\partial c}{\partial \lambda_r}
\]  \hspace{1cm} (20)

of equations (14) to (16) for \( r = 1, 2, \) and 3.

(8) Substitute the preceding expressions and group-like terms, thus giving

\[
D_{ijkl} = 2a_1 C_{ik}^2 C_{ij}^2 + 2a_2 (C_{ik} C_{ij}^2 + C_{ij} C_{ik}^2) + 2a_3 (\delta_{ik} C_{ij}^2 + C_{ij}^2 \delta_{ij})
\]

\[+ 2a_4 (C_{ik} C_{ij}) + 2a_5 (\delta_{ik} C_{ij} + C_{ik} \delta_{ij} + \delta_{ik} C_{ij} + C_{ij} \delta_{ik})
\]

\[+ 2a_6 (\delta_{ik} \delta_{jl} + \delta_{ik} \delta_{jl}) \]  \hspace{1cm} (21)

(9) For comparison of equation (21) to the forms described in reference 11, section 4, we make use of the symmetry properties of \( C_{ij} \) and \( \delta_{ij} \), and define two second order symmetric tensors, \( P \) and \( Q \),

\[
P_{ijkl}(G, H) = G_{ik} H_{jl} + G_{il} H_{jk}
\]  \hspace{1cm} (22)

\[
Q_{ijkl}(G, H) = G_{ik} H_{jl} + G_{ij} H_{jk} + G_{ji} H_{ik} + G_{jk} H_{il}
\]  \hspace{1cm} (23)

such that upon substitution we obtain

\[
D_{ijkl} = a_1 P(C_{ik}^2, C_{ij}^2) + a_2 [P(C_{ik}^2, C_{ij}) + P(C_{ij}^2, C_{ik}^2)]
\]

\[+ a_3 [Q(C_{ik}^2 \delta_{ij}) + P(\delta_{ik}, C_{ij}^2)] + a_4 P(C_{ik}, C_{ij})
\]
\[ a_5 [Q(C_{kl}, \delta_{ij}) + Q(\delta_{kl}, C_{ij})] + 2a_6 I_{ijkl} \] (24)

where \( a_1, a_2, \ldots a_6 \) are as defined in reference 11 and the preceding equation (eq. (24)) is directly comparable to equations 4.6a in reference 11. Note that

\[ I_{ijkl} = \frac{1}{2} [\delta_{ik} \delta_{jl} + \delta_{il} \delta_{jk}] \] (25)

\[ C_{ij}^2 = C_{im} C_{mj} \] (26)

in the foregoing expressions.

Next, given the case of nondistinct eigenvalues, for example, case II when \( \lambda_1 \neq \lambda_2 = \lambda_3 = \lambda \), or case III when \( \lambda_1 = \lambda_2 = \lambda_3 \), we must

(10) Remove the singularity (case II) or singularities (case III) by defining an appropriate "path" for taking the limit of \( a, b, c \) and \( C_{ik} C_{kj} \) in equations (13); that is,

- For case II
  \[ \lambda_1, \lambda_2 = \lambda + \Delta, \lambda_3 = \lambda - \Delta \]
- For case III
  \[ \lambda_1 = \lambda, \lambda_2 = \lambda + \Delta, \lambda_3 = \lambda - \Delta \]

(11) Substitute the preceding eigenvalues into the expressions for \( a, b, c \) in equations (13), and take the limit of the numerator and denominator of \( a, b, \) and \( c \) as \( \Delta \to 0 \).

(12) If both limits are zero, apply l'Hospital's rule recursively to the now equivalent one dimensional problem. For example, given case II, we obtain

\[ \lim_{\Delta \to 0} a(\Delta) = \frac{1}{(\lambda_1 - \lambda)^2} [s(\lambda) - s(\lambda) - (\lambda_1 - \lambda)s'(\lambda)] \] (27)

\[ \lim_{\Delta \to 0} b(\Delta) = \frac{1}{(\lambda_1 - \lambda)^2} [-2\lambda[s(\lambda) - s(\lambda)] + (\lambda_1^2 - \lambda^2)s'(\lambda)] \] (28)

\[ \lim_{\Delta \to 0} c(\Delta) = \frac{1}{(\lambda_1 - \lambda)^2} [\lambda^2 s(\lambda) + \lambda_1 (\lambda_1 - 2\lambda) s(\lambda) - \lambda_1 \lambda (\lambda_1 - \lambda) s'(\lambda)] \] (29)

where

\[ s'(\lambda_r) = \frac{\partial s(\lambda_r)}{\partial \lambda_r} = \frac{2\partial^2 W}{\partial \lambda_r \partial \lambda_r} \] (30)
(13) Simplify $C_{ik}C_{kj}$ by using the definition of $C_{ij}$ and $N_{ij}$, that is,

$$C_{ij}^2 = \lambda_1 N_{ij}^{(1)} + \lambda_2 N_{ij}^{(2)} + \lambda_3 N_{ij}^{(3)}$$

In addition, by using $\delta_{ij} = N_{ij}^{(1)} + N_{ij}^{(2)} + N_{ij}^{(3)}$ and equation (4), for case II we obtain

$$C_{ik}C_{kj} = \frac{1}{(\lambda_1 - \lambda)}[(\lambda_1^2 - \lambda^2)C_{ij} + (\lambda^2\lambda_1 - \lambda\lambda_1^2)\delta_{ij}]$$

and with equation (5) for case III we have

$$C_{ik}C_{kj} = \lambda \delta_{ij}.$$  

(14) Substitute the limiting values of $a, b, c$ and $C_{ik}C_{kj}$ into equations (13) and group like terms to obtain the modified stress function, $S_{ij}$, and the $\bar{a}$ and $\bar{b}$ values for case II:

$$S_{ij} = \bar{a}C_{ij} + \bar{b}\delta_{ij}$$

where

$$\bar{a} = \frac{s(\lambda_1) - s(\lambda)}{\lambda_1 - \lambda}$$

and

$$\bar{b} = -\frac{[s(\lambda_1)\lambda - s(\lambda)\lambda_1]}{\lambda_1 - \lambda}$$

For case III,

$$S_{ij} = s(\lambda)\delta_{ij}$$

(15) Repeat steps 5 through 10, but now use the appropriate modified stress function. For case II, this results in,

$$D_{ijkl} = 2(\frac{\partial \bar{a}}{\partial \lambda_1}N_{ij}^{(1)} + \frac{\partial \bar{a}}{\partial \lambda}N_{ij}^{(2)})C_{kl} + 2\bar{a}\delta_{ijkl} + 2\left(\frac{\partial \bar{a}}{\partial \lambda_1}N_{ij}^{(1)} + \frac{\partial \bar{a}}{\partial \lambda}N_{ij}^{(2)}\right)\delta_{kl}$$

And for case III,

$$D_{ijkl} = 2\frac{\partial \bar{a}}{\partial \lambda_p} \frac{\partial \lambda_p}{\partial C_{kl}} \delta_{ij}$$

where the special derivative rule of equation (7) is now used.
The value in automating the foregoing procedure is evident: not only does this special purpose function relieve the user of the tedious manual derivation process but it also ensures analytical accuracy. This was illustrated prior to the publication of reference 11 in that a number of errors in the hand derivation were detected, verified and corrected. Furthermore, as will be discussed in a sequel paper [17], this automated derivation procedure facilitated the generalization of the preceding expressions to the general nonseparable case, which to the author's knowledge, has eluded researchers to date. Also, it should be apparent that this derivation process needs to be executed only once. However, with each new definition of \( W \) evaluation of \( s(\lambda(\gamma)) \) and \( s'(\lambda(\gamma)) \) is required in order to specialize the needed coefficients; for example, \( a, b \) and \( c \), and \( a_1, a_2, \ldots a_6 \). As a consequence, this motivated the development of SDIFFEV, as described in the next section.

3.2 SDIFFEV(case, W)

The function SDIFFEV symbolically evaluates the explicit expressions for the stress function \( S_{ij} \) and material moduli tensor \( D_{ijkl} \), which were generated by SDIFF and stored in a LISP [18] level disk file. Only the coefficients of these expressions need be changed when a different strain-energy function is specified. The evaluation algorithm is illustrated here in pseudo code:

\[
\text{SDIFFEV(case, W)}
\]
\[
\text{IF} \quad (\text{diff}(W, \lambda_1, \lambda_2), \text{diff}(W, \lambda_2, \lambda_3), \text{diff}(W, \lambda_3, \lambda_1)) = 0 \quad \text{THEN}
\]
\[
\text{Display message: W is separable.}
\]
\[
\text{SEP} = 1
\]
\[
\text{ELSE Display message: W is non separable. SEP = 2}
\]
\[
\text{ENDIF}
\]
\[
\text{IF case = 1 THEN .}
\]
\[
\text{Call Subroutine A}
\]
\[
\text{ELSE IF case = 2 THEN}
\]
\[
\quad \text{Call Subroutine B}
\]
\[
\text{ELSE IF case = 3 THEN}
\]
\[
\quad \text{Call Subroutine C}
\]
\[
\text{END IF}
\]
\[
\text{End}
\]

\[
\text{Subroutine A}
\]
\[
\text{IF SEP = 2 THEN}
\]
\[
\text{Do loop \( i = 1, 6 \)}
\]
\[
\text{\( a[i] = e[a[i]] \) (\( e[a[i]] \) are the coefficients of tensor D stored on the disk file produced by SDIFF(1))}
\]
\[
\text{End loop}
\]
\[
\text{ELSE IF SEP = 1 THEN}
\]
s[2,1] = s[3,1] = s[3,2] = 0
ENDIF
Do loop i = 1, 6
  a[i] = ev(ea[i])
End loop
Do loop i = 1, 3
  s[i] = 2*diff(W, A, 1)
  s[i,i] = 2*diff(W, A, 2)
IF SEP = 2 THEN
  Do loop j = 1, 3
    s[i,j] = diff(W, A, 2)
  End loop
ENDIF
End loop
Call OPTION
Return

Subroutine B
W = ev(W, λ_3 = λ_2)
IF SEP = 2 THEN
  Do loop i = 1, 3
    b[i] = eb[i] (eb[i] are the coefficients of tensor D stored on the disk file produced
    by SDIFF (2))
  End loop
ELSE IF SEP = 1 THEN
  s[2,1] = 0
  Do loop i = 1, 6
    b[i] = ev(eb[i])
  End loop
  Do loop i = 1, 2
    s[i] = 2*diff(W, A, 1)
    s[i,i] = 2*diff(W, A, 2)
  IF SEP = 2 THEN
    Do loop j = 1, 2
      s[i,j] = diff(W, A, 2)
    End loop
  ENDIF
  End loop
Call OPTION
Return
Subroutine C
\[ W = \text{ev}(W, \lambda_3 = \lambda_2 = \lambda_1) \]
\[ s[i] = 2 * \text{diff}(W, \lambda_i, 1) \]
\[ s[i, j] = 2 * \text{diff}(W, \lambda_i, 2) \]
Call OPTION
Return

Subroutine OPTION

Display the formulae S[i,j] and D[i,j,k,l]. Then, ask if user wants to see the symbolic form for the given function W, the intermediate step evaluations, and the derivatives of W.
READ(type y, or n to the question)
DISPLAY the options user may choose
Return

3.3 TEMPLATE

The function TEMPLATE is similar to the function SDIFFEV in that both will evaluate the explicit expressions obtained from SDIFF. As a result neither can be employed unless preceded by an invocation of SDIFF. TEMPLATE, however, will automatically generate the associated FORTRAN source code needed to evaluate the expressions numerically for a given potential function W. Code generation is accomplished by utilizing the built-in MACSYMA function gentran, and a number of template files. The template files can be thought of as a framework for the generation of four basic FORTRAN subroutines (i.e., the main driving routine COMPSD and the three subroutines – one each for case I, case II, and case III) and numerous functions. Appendix A contains the template file for the main driving routine COMPSD. This subroutine is constructed for easy implementation into a finite element code; the input requirements are the strain tensor \( C_m \) (denoted as cmu) and its associated eigenvalues (i.e., \( \lambda_1, \lambda_2, \lambda_3 \) denoted by \( g_{11}, g_{12}, \) and \( g_{13} \) respectively), and the outputs are the stress tensor \( S_n \) (denoted as s), and the material moduli tensor \( D_{nm} \) (denoted as d). Here, \( n \) and \( m \) run from 1 to 6. The only automated code generation required is that for the subroutines COMPSD1, COMPSD2, and COMPSD3. These codes are generated by issuing the command \(<\text{gentranin}>\). The subroutines COMPSD1, COMPSD2, and COMPSD3 are associated with case I (\( \lambda_1 \neq \lambda_2 \neq \lambda_3 \)), case II (\( \lambda_1 = \lambda_2 = \lambda_3 \)), and case III (\( \lambda = \lambda_1 = \lambda_2 = \lambda_3 \)), described in Section 2.0. The template files corresponding to these three cases are shown, respectively in appendixes B, C, and D. Note that in these routines, most of the FORTRAN code is automatically generated, since it pertains to the definition of coefficients \( a, b, c; a_1, a_2, \ldots, a_6 \), and the first \( (s_1, s_2, s_3, \text{see eq. (12)}) \) and second scalar \( (s_{11}, s_{22}, s_{33}, \text{see eq. (30)}) \) derivatives of the strain energy function W. The
gentran commands are enclosed by double inequality signs, that is, $\ll \gg$. Finally, all functions that are associated with a given case have been included in the corresponding appendix.

4 Example

As an example, consider the case in which the strain energy function $W$ of equation (1) consists of only two terms; that is,

$$W = x_1(g_1 l_1^2 + g_2 l_2^2 + g_3 l_3^2) + x_2(g_1 l_1^2 + g_2 l_2^2 + g_3 l_3^2)$$

where $x_1$, $x_2$, $y_1$, and $y_2$ are material coefficients and $g_1 = \lambda_1$, $g_2 = \lambda_2$, and $g_3 = \lambda_3$. After defining $W$, we can symbolically obtain the analytical expressions for $S_{ij}$ and $D_{ijkl}$ (given the case of three distinct eigenvalues) by merely issuing the command

```macsyma
sdiffev(1, W);
```

at the MACSYMA command level. Case II or III can just as easily be obtained by substituting a 2 or 3 in place of the 1 in this command. The resulting output is shown in appendix E where the expressions for the coefficients $a, b, c$ and $a_1, a_2, \ldots, a_6$ could be further simplified and manipulated, if desired, by using other MACSYMA built-in functions. Typically, however, the analyst will ultimately desire a FORTRAN code for the resulting expressions in order to solve a given structural problem using the foregoing constitutive model. This code, described in the previous section, can easily be obtained by issuing the command

```macsyma
template();
```

at the MACSYMA command level. The generated FORTRAN code will then be stored in a file named temp.f. The automatically generated FORTRAN code for the above example is shown in appendix F.

5 Summary of Results

Taken separately, the main constituents of the deformation tensor (i.e., principal values and associated eigenvectors) are, in general, not uniquely defined and continuously differentiable functions. Careful consideration is thus called for in implementing constitutive models formulated in terms of these principal-strain measures. This difficulty can be entirely bypassed by resorting to explicit symbolic derivations of appropriate forms of the material tangent-stiffness matrices, which are valid over the entire deformation range. Furthermore, to enhance effective utilization and implementation of the present results, automatic FORTRAN generation of these explicit expressions has been pursued and
achieved. As a result, three special purpose functions (SDIFF, SDIFFEV and TEMPLATE), running under MACSYMA, have been developed and verified.

References


APPENDIX A: Template File Associated With COMPSD
The Main Driver Routine

c ****************************************************
c This is the template subroutine to calculate
c tensor S and D. inputs are eigenvalues g11,g12,g13,
c and cmu(6). cmu is assumed to be engineering strain(e),
c e.g. the Cauchy-green deformation tensor cm(3,3) is related
to cmu(6) in the following fashion:
c cm(1,1)=cmu(1), cm(2,2) = cmu(2), cm(3,3) =cmu(3),
c cmu(4)=2*cm(1,2), cm(5)=2*cm(2,3),cmu(6)=2*cm(1,3).
c The outputs are the second order tensor S(6)
c and forth order tensor D(6,6) are related in the
c following way:
S=D*C
S(1,1) = S(1)
c S(2,2) = S(2)
c S(3,3) = S(3)
c S(1,2) = S(4)
c S(2,3) = S(5)
c S(3,1) = S(6)
c C(1,1) = C(1)
c C(2,2) = C(2)
c C(3,3) = C(3)
c C(1,2) = C(4)
c C(2,3) = C(5)
c C(3,1) = C(6)
c
subroutine compsd(g11,g12,g13,cmu,s,d)
real*8 g11,g12,g13,ts(3,3),td(3,3,3,3)
real*8 delt(3,3),delt4(3,3,3,3),s(6),d(6,6)
real*8 cmu(6),cm(3,3)
c
converts cmu(6) to matrix cm(3,3) in a way that
cm(1,2)=cm(2,1)=cmu(4),cm(2,3)=cm(3,2)=cmu(5),
cm(1,3)=cm(3,1)=cmu(6).
do 5 i=1,3
   do 5 j=1,3
      if (i.eq.j) then
         iq=i
         cm(i,j)=cmu(iq)
      else if (i.ne.j) then
         if((i+j).eq.3) iq=4
         if((i+j).eq.4) iq=6
         if((i+j).eq.5) iq=5
         cm(i,j)=cmu(iq)/2
      end if
      continue
   5 continue

  c
  c    Initiates the second identity tensor delt(3,3) which
  c    is a 2X2 identity matrix.
  c
  do 6 i=1,3
     delt(i,i)=1.0
  6 continue
  c
  c    Computes the forth order identity tensor delt4(3,3,3,3)
  c    by definition.
  c
  do 7 i=1,3
     do 7 j=i,3
        delt4(i,j,i,j)=delt(i,i)*delt(j,j)+delt(i,j)*delt(j,i)
        delt4(i,j,j,i)=delt4(i,j,i,j)
  7 continue

  c********************************************************************
  c For different eigenvalues gl1,gl2,gl3 the computation
  c is different. case1 is gl1#gl2#gl3 call subroutine comsd1.
  c case2 is gl3=gl2#gl1 or gl1=gl3#gl2 or gl1=gl2#gl3 then
  c call subroutine comsd2. case3 is gl1=gl2=gl3 call subroutine
  c comsd3.
  c*********************************************************************

 if ((gl1.ne.gl2).and.(gl2.ne.gl3).and.(gl1.ne.gl3)) then
    call comsd1(gl1,gl2,gl3,delt,delt4,cm,ts,td)
else if((gl2.eq.gl3).and.(gl1.ne.gl3)) then
call compsd2(gl1, gl2, delt, delt4, cm, ts, td)
else if((gl1.eq.gl2).and.(gl3.ne.gl2)) then
  gl1=gl3
  call compsd2(gl1, gl2, delt, delt4, cm, ts, td)
else if((gl1.eq.gl3).and.(gl2.ne.gl3)) then
  gl1=gl2
  gl2=gl3
  call compsd2(gl1, gl2, delt, delt4, cm, ts, td)
else
  call compsd3(gl1, delt, delt4, ts, td)
end if

c
Rewrite the tensor ts(i,j) td(i,j,k,l) to S(i) and D(i,j)
respectively by using the symmetric property.
converts ts(3,3) s(6) and td(3,3,3,3) to D(6,6)
c
do 8 i=1,3
do 8 j=i,3
  if (i.eq.j) iq=i
  if (i.eq.1.and.j.eq.2) iq=4
  if (i.eq.2.and.j.eq.3) iq=5
  if (i.eq.1.and.j.eq.3) iq=6
    s(iq)=ts(i,j)
  continue
continue
do 9 i=1,3
do 9 j=i,3
d(i,j)=td(i,i,j,j)
continue
continue
do 10 i=1,3
d(i,4)=td(i,i,1,2)+td(i,i,2,1)
d(i,5)=td(i,i,2,3)+td(i,i,1,3)
d(i,6)=td(i,i,3,1)+td(i,i,1,3)
continue
d(4,4)=(td(1,2,1,2)+td(1,2,2,1)+td(2,1,1,2)+td(2,1,2,1))/2.
d(4,5)=(td(1,2,2,3)+td(1,2,3,2)+td(2,1,2,3)+td(2,1,3,2))/2.
d(4,6)=(td(1,2,1,3)+td(1,2,3,1)+td(2,1,1,3)+td(2,1,3,1))/2.
d(5,5)=(td(2,3,2,3)+td(2,3,3,2)+td(3,2,2,3)+td(3,2,3,2))/2.
d(5,6)=(td(2,3,1,3)+td(2,3,3,1)+td(3,2,1,3)+td(3,2,3,1))/2.
\[ d(6,6) = (td(3,1,1,3)+td(3,1,3,1)+td(1,3,1,3)+td(1,3,3,1))/2. \]

do 11 i = 1,6
    do 11 j = 1,6
        d(i,j) = d(j,i)
continue

prints out the inputs \( g_{l1}, g_{l2}, g_{l3}, c_{mu}(6) \) and outputs \( S \) and \( D \)

print*, 'g_{l1}=', gl1
print*, 'g_{l2}=', gl2
print*, 'g_{l3}=', gl3
print*, 'Input tensor C(6):'
print*, (cmu(i), i = 1,6)
print*, "second order tensor S(6):
print*, (s(i), i=1,6)
print*, "The forth order tensor D(6,6):"
   do 101 i=1,6
        print*,(d(i,j),j=1,6)
continue
return
end

subroutine compsdl(gll,gl2,gl3,delt,delt4,cmu,ts,td)

<<
genranin("case11.tem")$
>>
subroutine compsdl2(gll,gl2,delt,delt4,cmu,ts,td)

<<
genranin("case22.tem")$
>>
subroutine compsdl3(gll,delt,delt4,ts,td)

<<
genranin("case3.tem")$
>>
This subroutine computes P and Q forth order tensors which we define in tensor D.

subroutine pqcom(cm1,cm2,p,q)
real*8 cm1(3,3),cm2(3,3), p(3,3,3,3),q(3,3,3,3)
do 100 i=1,3
do 100 j=1,3
do 100 k=1,3
   do 100 l=1,3
      p(i,j,k,l)=cm1(i,k)*cm2(j,l)+cm1(i,l)*cm2(j,k)
      q(i,j,k,l)=p(i,j,k,l)+cm1(j,l)*cm2(i,k)+cm1(j,k)*cm2(i,l)
100 continue
return
end

This subroutine computes matrix product cmXcm.

subroutine product(mat1,cmm)
real*8 mat1(3,3),cmm(3,3),sum
do 25 i=1,3
do 25 j=1,3
   sum=0.0
   do 26 k=1,3
      sum=sum+mat1(i,k)*mat1(k,j)
26 continue
   cmm(i,j)=sum
25 continue
return
end
APPENDIX B: Template File Associated With COMPSD1
Valid For Three Distinct Eigenvalues

real*8 g11,g12,g13,ts(3,3),td(3,3,3,3)
real*8 cm(3,3),delt(3,3),delt4(3,3,3,3),p(3,3,3,3)
real*8 q(3,3,3,3),cmm(3,3),p1(3,3,3,3),p21(3,3,3,3)
real*8 p31(3,3,3,3),q11(3,3,3,3),q12(3,3,3,3),p22(3,3,3,3)
real*8 q21(3,3,3,3),q22(3,3,3,3),a,b,c,a1,a2,a3,a4,a5,a6

Obtains cmm(3,3)=cm(3,3)*cm(3,3) from subroutine product
call product(cm,cmm)
Uses the formula we derived in code to compute second order
tensor ts(3,3).
<<
gentran(for i:1 thru 3 do
(for j:1 thru 3 do
(ts[i,j]:a(g11,g12,g13)*cmm[i,j]+b(g11,g12,g13)
 *cm[i,j]+c(g11,g12,g13)*delt[i,j]))$ 
>>
Call subroutine to compute all the functions we defined
when we derived forth order tensor td, namely P(i,j,k,1)
and Q(i,j,k,1) which are the functions of cm(3,3) and
the matrix product cmm(3,3).
call pqcom(cmm,cmm,p1,q)
call pqcom(cmm,cm,p21,q)
call pqcom(cm,cmm,p22,q)
call pqcom(cm,cm,p31,q)
call pqcom(cmm,delt,p,q11)
call pqcom(delt,cmm,p,q12)
call pqcom(cm,delt,p,q21)
call pqcom(delt,cm,p,q22)
Computes forth order tensor $td(i,j,k,l)$

```

gentran(for i:1 thru 3 do
  (for j:1 thru 3 do
    (for k:1 thru 3 do
      (for l:1 thru 3 do
        $td[i,j,k,l] := (a1(g11,g12,g13)*p1[i,j,k,l] + a2(g11,g12,g13)*p21[i,j,k,l] + a3(g11,g12,g13)*q11[i,j,k,l] + a5(g11,g12,g13)*q21[i,j,k,l])$
      ))
    ))
  ))

return
end
```

$a,b,c,a1-a6$ are the coefficients we derived in code.

```

gentran(a(g11,g12,g13) := block(type(function,a),
  type("real*8",g11,g12,g13),
  type("real*8",a,s1,s2,s3),
  a:eval(ta))$  

gentran(b(g11,g12,g13) := block(type(function,b),
  type("real*8",b,g11,g12,g13),
  type("real*8",s1,s2,s3),
  b:eval(tb))$  

gentran(c(g11,g12,g13) := block(type(function,c),
  type("real*8",c,g11,g12,g13),
  type("real*8",s1,s2,s3),
  c:eval(tc))$
```

21
\begin{verbatim}
<<
gentran(a1(g11,g12,g13) := block(type(function,a1),
   type("real*8",a1,g11,g12,g13),
   type("real*8",s1,s2,s3,s11,s22,s33),
   a1:eval(ta1))$
>>
<<
gentran(a2(g11,g12,g13) := block(type(function,a2),
   type("real*8",a2,g11,g12,g13),
   type("real*8",s1,s2,s3,s11,s22,s33),
   a2:eval(ta2))$
>>
<<
gentran(a3(g11,g12,g13) := block(type(function,a3),
   type("real*8",a3,g11,g12,g13),
   type("real*8",s1,s2,s3,s11,s22,s33),
   a3:eval(ta3))$
>>
<<
gentran(a4(g11,g12,g13) := block(type(function,a4),
   type("real*8",a4,g11,g12,g13),
   type("real*8",s1,s2,s3,s11,s22,s33),
   a4:eval(ta4))$
>>
<<
gentran(a5(g11,g12,g13) := block(type(function,a5),
   type("real*8",a5,g11,g12,g13),
   type("real*8",s1,s2,s3,s11,s22,s33),
   a5:eval(ta5))$
>>
<<
gentran(a6(g11,g12,g13) := block(type(function,a6),
   type("real*8",a6,g11,g12,g13),
   type("real*8",s1,s2,s3,s11,s22,s33),
   a6:eval(ta6))$
>>
\end{verbatim}
The $s_1, s_2, s_3, s_{11}, s_{22}, s_{33}$ are derivatives of $W$

```c
function s1(g11,g12,g13)
<<cut(var);>>
<<
gentran(type( "real*8",s1,g11,g12,g13),
   s1:2*eval(diff(w,'g11,1)))$
>>
return
end

c
function s2(g11,g12,g13)
<<cut(var);>>
<<
gentran(type( "real*8",s2,g11,g12,g13),
   s2:2*eval(diff(w,'g12,1)))$
>>
return
end

c
function s3(g11,g12,g13)
<<cut(var);>>
<<
gentran(type( "real*8",s3,g11,g12,g13),
   s3:2*eval(diff(w,'g13,1)))$
>>
return
end

c
function s_{11}(g11,g12,g13)
<<cut(var);>>
<<
gentran(type( "real*8",s_{11},g11,g12,g13),
   s_{11}:2*eval(diff(w,'g11,2)))$
>>
return
end
```
function s22(gl1,gl2,gl3)
  <<cut(var);>>
  <<
    gentran(type("real*8",s22,gl1,gl2,gl3),
    s22:2*eval(diff(w,'gl2,2)))$
  >>
  return
end

function s33(gl1,gl2,gl3)
  <<cut(var);>>
  <<
    gentran(type( "real*8",s33,gl1,gl2,gl3),
    s33:2*eval(diff(w,'gl3,2)))$
  >>
  return
end
APPENDIX C: Template File Associated With COMPSD2
Valid For Double Coalescence Case

real*8 gll,gl2,ts(3,3),td(3,3,3,3)
real*8 cm(3,3),delt(3,3),delt4(3,3,3,3),p1(3,3,3,3)
real*8 q2(3,3,3,3),qi(3,3,3,3),p(3,3,3,3),q(3,3,3,3)
real*8 b1,b2,b3, abar,bbar

Computes second order tensor ts(i,j) based on the formula derived in code.

<<
gentrans(i:1 thru 3 do
 (j:1 thru 3 do
 (ts[i,j]:abar(gll,gl2)*cm[i,j]+bbar(gll,gl2)*delt[i,j])))
>>

Call subroutine to get P, Q which are defined in code.

call pqcom(cm,cm,p1,q)
call pqcom(cm,delt,p,q1)
call pqcom(delt,cm,p,q2)

Computes tensor td(i,j,k,l).

<<
gentrans(i:1 thru 3 do
 (j:1 thru 3 do
 (k:1 thru 3 do
 (l:1 thru 3 do
 (td[i,j,k,l]:b1(gll,gl2)*p1[i,j,k,l]+b2(gll,gl2)*q1[i,j,k,l]+q2[i,j,k,l]+b3(gll,gl2)*delt4[i,j,k,l])))
>>
return
end
abar, bbar are b1, b2, b3 functions derived in code.

```
gentran(abar(gl1, gl2):=block(type(function, abar),
    type("real*8", abar, gl1, gl2),
    type("real*8", ss1, ss2),
    abar: eval(abar)) )$

>>

gentran(bbar(gl1, gl2):=block(type(function, bbar),
    type("real*8", bbar, gl1, gl2),
    type("real*8", ss1, ss2),
    bbar: eval(bbar)) )$

>>

gentran(b1(gl1, gl2):=block(type(function, b1),
    type("real*8", b1, gl1, gl2),
    type("real*8", ss1, ss2, ss11, ss22),
    b1: eval(tb1)) )$

>>

gentran(b2(gl1, gl2):=block(type(function, b2),
    type("real*8", b2, gl1, gl2),
    type("real*8", ss1, ss2, ss11, ss22),
    b2: eval(tb2)) )$

>>

gentran(b3(gl1, gl2):=block(type(function, b3),
    type("real*8", b3, gl1, gl2),
    type("real*8", ss1, ss2, ss11, ss22),
    b3: eval(tb3)) )$

>>

neww: subst([gl3=gl2], w)$
```
ss1, ss2, ss11, ss22 are derivatives of W.

function ss1(gll, gl2)
<<cut(var);>>
<<
gentran(type("real*8", ss1, gll, gl2),
    ss1: 2*eval(diff(neww, 'gll, 1)))$
>>
return
end

c
function ss2(gll, gl2)
<<cut(var);>>
<<
gentran(type("real*8", ss2, gll, gl2),
    ss2: 2*eval(diff(neww, 'gl2, 1)))$
>>
return
end

c
function ss11(gll, gl2)
<<cut(var);>>
<<
gentran(type("real*8", ss11, gll, gl2),
    ss11: 2*eval(diff(neww, 'gl1, 2)))$
>>
return
end

c
function ss22(gll, gl2)
<<cut(var);>>
<<
gentran(type("real*8", ss22, gll, gl2),
    ss22: 2*eval(diff(neww, 'gl2, 2)))$
>>
return
end
APPENDIX D: Template File Associated With COMPSD3
Valid For The Triple Coalesence Case

c
real*8 gl1,ts(3,3),td(3,3,3,3),delt(3,3),delt4(3,3,3,3)
real*8 cc1,abbar
c
<<
gentran(for i:1 thru 3 do
 (for j:1 thru 3 do
  (ts[i,j]:abbar(gl1)*delt[i,j])))$
>>
<<
gentran(for i:1 thru 3 do
 (for j:1 thru 3 do
  (for k:1 thru 3 do
   (for l:1 thru 3 do
    (td[i,j,k,l]:cc1(gl1)*delt4[i,j,k,l]))))))$
return
end
<<
gentran(abbar(gl1):=block(type(function,abbar),
 type("real*8", abbar,gl1),
 abbar:eval(abbar)))$
>>
<<
gentran(cc1(gl1):=block(type(function,cc1),
 type("real*8", cc1,gl1),
 cc1:eval(cc1)))$
>>
<<
www:subst(['gl3='gl1, 'gl2='gl1],w)$
c

function sss1(gll)

<<cut(var);>>

<<<
gentran(type("real*8", sss1,gll),
        sss1:2*eval(diff(www,'gll,1)))$

>>>
return
dern

c

function sss11(gll)

<<cut(var);>>

<<<
gentran(type("real*8", sss11,gll),
        sss11:2*eval(diff(www,'gll,2)))$

>>>
return
dern
c6) sdiffev(l,w);

w is a separable function.

\[ w = (g_{13} + g_{12} + g_{11}) x_2 + (g_{13} + g_{12} + g_{11}) x_1 \]

This is case 1 with distinct eigenvalues \( g_{11} \neq g_{12} \neq g_{13} \).

Please type \( y \) if your answer is yes, otherwise type \( n \) to skip it.

Do you want to display the second order tensor \( s[i,j] \)? \( y; \)

\[ s = a c_{i,j} + c_{i,k} \delta_{k,j} + b c_{i,j} \]

Do you want to display \( c[i,j] \) and \( \delta[i,j] \)? \( y; \)

\[ \delta_{i,j} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \]

\[ c_{i,j} = g_{13} n_3 \ n_3 + g_{12} n_2 \ n_2 + g_{11} n_1 \ n_1 \]

\( n_1, n_2, n_3 \) are eigenvectors associated with eigenvalues \( g_{11}, g_{12}, g_{13} \).

If \( c[i,j] \) is given then the eigenvectors can be computed
Do you want to display a, b, c in s[i,j] form?

\[
\begin{align*}
a &= - \frac{s_3}{t_31 \left( t_31 - t_21 \right)} + \frac{s_2}{t_21 \left( t_31 - t_21 \right)} - \frac{s_1}{t_21 \ t_31} \\

b &= - \frac{s_3 \left( 2 \ t_31 - t_21 - 2 \ g_{13} \right)}{t_31 \left( t_31 - t_21 \right)} + \frac{s_1 \left( t_31 + t_21 + 2 \ g_{11} \right)}{t_21 \ t_31} \\
c &= - \frac{s_3 \left( t_31 - g_{13} \right) \left( t_31 - t_21 - g_{13} \right)}{t_31 \left( t_31 - t_21 \right)} - \frac{s_2 \left( t_21 - g_{12} \right) \left( t_31 + g_{11} \right)}{t_21 \left( t_31 - t_21 \right)} - \frac{s_1 \left( t_21 + g_{11} \right) \left( t_31 + g_{11} \right)}{t_21 \ t_31}
\end{align*}
\]

Do you want to display t21, t31, s1, s2, s3?

\[
\begin{align*}
t_21 &= g_{12} - g_{11} \\
t_31 &= g_{13} - g_{11}
\end{align*}
\]

s1, s2, s3 are the first derivatives of W with respect to g_{11}, g_{12}, g_{13}.

\[
\begin{align*}
sl &= g_{11} \\
x_2 &= y_2 + g_{11} \\
x_1 &= y_1
\end{align*}
\]
$$y^2 - 1 \quad y^1 - 1$$
$$s_2 = g_{12} \quad x_2 \ y^2 + g_{12} \quad x_1 \ y^1$$
$$y^2 - 1 \quad y^1 - 1$$
$$s_3 = g_{13} \quad x_2 \ y^2 + g_{13} \quad x_1 \ y^1$$

Do you want to display the forth order tensor $d[i,j,k,l]$?

$$d_{i,j,k,l} = [a \ \delta^4 + a (q(\delta, c) + q(c, \delta))$$
$$\quad + a (q(\delta, c) + q(c, \delta)) + a (p(c, c) + p(c, c))$$

Do you want to display the functions $p, q$ and $\delta^4$?

$$p(g, h) = g \quad h + g \quad h$$
$$i, k \quad j, l \quad i, l \quad j, k$$

$$q(g, h) = g \quad h + h \quad g + 2 \quad g \quad h$$
$$i, k \quad j, l \quad i, k \quad j, l \quad i, l \quad j, k$$

$$\delta^4 = \delta^4 \quad \delta^4 + \delta^4 \quad \delta^4$$
$$i, k \quad j, l \quad i, l \quad j, k$$

Do you want to continue displaying a

$$1$$

y;
\[
a = \frac{2s_3(2t_{31} - t_{21})}{t_{31} (t_{31} - t_{21})} - \frac{2s_1(t_{31} + t_{21})}{t_{21} t_{31}} - \frac{s_{33}}{t_{31} (t_{31} - t_{21})}.
\]

Do you want to continue displaying a ?

\[
a = \frac{2}{y;}
\]

\[
s_1(2t_{31} + 3t_{21} t_{31} + 4g_{11} t_{31} + 2t_{21} + 4g_{11} t_{21})
\]

\[
a = \frac{3}{t_{21} t_{31}}.
\]

\[
s_2(2t_{31} - 3t_{21} t_{31} + 4g_{11} t_{31} - t_{21} - 8g_{11} t_{21})
\]

\[
s_3(t_{31} + 3t_{21} t_{31} + 8g_{11} t_{31} - 2t_{21} - 4g_{11} t_{21})
\]

\[
s_{33}(2t_{31} - t_{21} - 2g_{13}) + s_{11}(t_{31} + t_{21} + 2g_{11})
\]

\[
a = \frac{2}{t_{31} (t_{31} - t_{21})} - \frac{2}{t_{21} t_{31}}.
\]

33
\[
\begin{align*}
\text{s22} \left( t_{31} - t_{21} + g_{12} + g_{11} \right) \\
+ \frac{2}{t_{21}} \left( t_{31} - t_{21} \right)
\end{align*}
\]

Do you want to continue displaying a? 

\[
y; \quad 3
\]

\[
a = - s_3 \left( t_{31} - 2 t_{21} t_{31} - g_{11} t_{31} + t_{21} t_{31} - 3 g_{11} t_{21} t_{31} - 4 g_{11} t_{31} \right) \\
+ 2 g_{11} t_{21} + 2 g_{11} t_{21})/(t_{31} (t_{31} - t_{21} ))
\]

\[
- s_1 \left( t_{21} t_{31} + 2 g_{11} t_{31} + t_{21} t_{31} + 3 g_{11} t_{21} t_{31} + 2 g_{11} t_{31} \\
+ 2 g_{11} t_{21} + 2 g_{11} t_{21})/(t_{21} t_{31} )
\]

\[
+ s_2 \left( t_{21} t_{31} + 2 g_{11} t_{31} - 2 t_{21} t_{31} - 3 g_{11} t_{21} t_{31} + 2 g_{11} t_{31} + t_{21} \\
- g_{11} t_{21} - 4 g_{11} t_{21})/(t_{21} (t_{31} - t_{21} ))
\]

\[
\text{s33} \left( t_{31} - g_{13} \right) \left( t_{31} - t_{21} - g_{13} \right) \\
- \frac{2}{t_{31}} \left( t_{31} - t_{21} \right) + \frac{2}{t_{21}} \left( t_{31} - t_{21} \right)
\]

\[
\text{s11} \left( t_{21} + g_{11} \right) \left( t_{31} + g_{11} \right) \\
- \frac{2}{t_{21}} \frac{2}{t_{31}}
\]

34
Do you want to continue displaying a ?

\[
y;  
\]

\[
a = \frac{2 \cdot s_3 \left(t_{21} + g_{11}\right) \left(t_{31} + 4 \cdot g_{11} \cdot t_{31} + t_{21} - 2 \cdot g_{11} \cdot t_{21}\right)}{4} 
\]

\[
+ \frac{2 \cdot s_1 \left(t_{31} + t_{21} + 2 \cdot g_{11}\right) \left(t_{31} + 2 \cdot g_{11} \cdot t_{31} + t_{21} + 2 \cdot g_{11} \cdot t_{21}\right)}{3 \cdot 3} 
\]

\[
- \frac{2 \cdot s_2 \left(t_{31} + 2 \cdot g_{11}\right) \left(t_{31} + t_{21} + 2 \cdot g_{11} \cdot t_{21} - 2 \cdot g_{11} \cdot t_{21}\right)}{t_{21} \cdot \left(t_{31} - t_{21}\right)} 
\]

\[
- \frac{s_{33} \left(2 \cdot t_{31} - t_{21} - 2 \cdot g_{13}\right)}{2} 
\]

\[
+ \frac{s_{11} \left(t_{31} + t_{21} + 2 \cdot g_{11}\right)}{t_{21} \cdot \left(t_{31} - t_{21}\right)} 
\]

\[
+ \frac{s_{22} \left(t_{31} - t_{21} + g_{12} + g_{11}\right)}{2 \cdot \left(t_{31} - t_{21}\right)} 
\]

Do you want to continue displaying a ?

\[
y; 
\]

35
\[ a = s_1 \left( t_{21} t_{31} + 2 g_{11} t_{31} + t_{21} t_{31} + 6 g_{11} t_{21} t_{31} + 6 g_{11} t_{31} \right) + 5 \]

\[ + t_{21} t_{31} + 6 g_{11} t_{21} t_{31} + 9 g_{11} t_{21} t_{31} + 4 g_{11} t_{31} + 2 g_{11} t_{21} + 6 g_{11} t_{21} \]

\[ + 6 g_{11} t_{21} + 4 g_{11} t_{21} \right) / \left( t_{21} t_{31} \right) \]

\[ + s_3 \left( t_{21} t_{31} + 2 g_{11} t_{31} - 2 t_{21} t_{31} - 6 g_{11} t_{21} t_{31} - 3 g_{11} t_{31} \right) + 2 g_{11} t_{21} + 6 g_{11} t_{21} \]

\[ + 4 g_{11} t_{21} \right) / \left( t_{21} \left( t_{31} - t_{21} \right) \right) - s_2 \]

\[ s_3 \left( t_{31} - g_{13} \right) \left( t_{31} - t_{21} - g_{13} \right) \left( 2 t_{31} - t_{21} - 2 g_{13} \right) \]

\[ s_{33} \left( t_{31} - g_{13} \right) \left( t_{31} - t_{21} - g_{13} \right) \left( 2 t_{31} - t_{21} - 2 g_{13} \right) \]

\[ - \frac{2}{2} \frac{t_{31}}{t_{31} - t_{21}} \]

\[ s_{11} \left( t_{21} + g_{11} \right) \left( t_{31} + g_{11} \right) \left( t_{31} + t_{21} + 2 g_{11} \right) \]

\[ + \frac{2}{2} \frac{t_{21}}{t_{21} t_{31}} \]

36
\[
\begin{align*}
\text{s22} & \ (t21 - gl2) \ (t31 + gl1) \ (t31 - t21 + gl2 + gl1) \\
& - \ \frac{\text{s22} \ (t21 - gl2) \ (t31 + gl1) \ (t31 - t21 + gl2 + gl1)}{t21 \ (t31 - t21)} \\

\text{Do you want to continue displaying a } \? & \ \\
6 & \\

y; & \\

a & = \\
6 & \\

\begin{align*}
2 \ & gl1 \ s1 \ (t21 + gl1) \ (t31 + gl1) \ (t31 + t21 + t31 + t31 + t21 + gl1) \\
& - \ \frac{2 \ gl1 \ s1 \ (t21 + gl1) \ (t31 + gl1) \ (t31 + t21 + t31 + t31 + t21 + gl1)}{t21 \ (t31 - t21)} \\
& + \ \frac{2 \ gl1 \ s2 \ (t21 + gl1) \ (t31 + gl1) \ (t31 - 2 \ t21 + t31 + gl1 \ t31 + t21)}{t21 \ (t31 - t21)} \\
& - \ \frac{2 \ gl1 \ t21)}{(t21 \ (t31 - t21))} - 2 \ gl1 \ s3 \ (t21 + gl1) \ (t31 + gl1) \\
& \ (t31 - 2 \ t21 \ t31 - 2 \ gl1 \ t31 + t21 + gl1 \ t21)}{(t31 \ (t31 - t21))} \\

2 \ & s33 \ (t31 - gl3) \ (t31 - t21 - gl3) \ s22 \ (t21 - gl2) \ (t31 + gl1) \\
& - \ \frac{2 \ s33 \ (t31 - gl3) \ (t31 - t21 - gl3) \ s22 \ (t21 - gl2) \ (t31 + gl1)}{t21 \ (t31 - t21)} \\

2 \ & s11 \ (t21 + gl1) \ (t31 + gl1) \\
& - \ \frac{2 \ s11 \ (t21 + gl1) \ (t31 + gl1)}{t21 \ (t31 - t21)} \\
\end{align*}
\]
Do you want to display $s_{11}$?
\[ y; \]
\[ s_{11} = g_{11} x_{2} (y_{2} - 1) y_{2} + g_{11} x_{1} (y_{1} - 1) y_{1} \]

Do you want to display $s_{22}$?
\[ y; \]
\[ s_{22} = g_{12} x_{2} (y_{2} - 1) y_{2} + g_{12} x_{1} (y_{1} - 1) y_{1} \]

Do you want to display $s_{33}$?
\[ y; \]
\[ s_{33} = g_{13} x_{2} (y_{2} - 1) y_{2} + g_{13} x_{1} (y_{1} - 1) y_{1} \]

(d6) done
APPENDIX F: Listing of Automatically Generated FORTRAN Code

This is the template subroutine to calculate tensor S and D. inputs are eigenvalues gll, gl2, gl3, and cmu(6). cmu is assumed to be engineering strain(e), e.g. the Cauchy-green deformation tensor cm(3,3) is related to cmu(6) in the following fashion:

\[
\begin{align*}
\text{cm}(1,1) &= \text{cmu}(1), \\
\text{cm}(2,2) &= \text{cmu}(2), \\
\text{cm}(3,3) &= \text{cmu}(3), \\
\text{cmu}(4) &= 2 \times \text{cm}(1,2), \\
\text{cm}(5) &= 2 \times \text{cm}(2,3), \\
\text{cmu}(6) &= 2 \times \text{cm}(1,3).
\end{align*}
\]

The outputs are the second order tensor S(6) and forth order tensor D(6,6) are related in the following way:

\[
\begin{align*}
S &= D \times C \\
S(1,1) &= S(1), \\
S(2,2) &= S(2), \\
S(3,3) &= S(3), \\
S(1,2) &= S(4), \\
S(2,3) &= S(5), \\
S(3,1) &= S(6)
\end{align*}
\]

\[
\begin{align*}
C(1,1) &= C(1), \\
C(2,2) &= C(2), \\
C(3,3) &= C(3), \\
C(1,2) &= C(4), \\
C(2,3) &= C(5), \\
C(3,1) &= C(6)
\end{align*}
\]

subroutine compsd(gll, gl2, gl3, cmu, s, d)
real*8 g11, gl2, gl3, ts(3,3), td(3,3,3,3)
real*8 delt(3,3), delt4(3,3,3,3), s(6), d(6,6)
real*8 cmu(6), cm(3,3)
converts cmu(6) to matrix cm(3,3) in a way that
\[
\begin{align*}
\text{cm}(1,2) &= \text{cm}(2,1) = \text{cmu}(4), \\
\text{cm}(2,3) &= \text{cm}(3,2) = \text{cmu}(5), \\
\text{cm}(1,3) &= \text{cm}(3,1) = \text{cmu}(6)
\end{align*}
\]
do 5 i=1,3
    do 5 j=1,3
        if (i.eq.j) then
            iq=i
            cm(i,j)=cmu(iq)
        else if (i.ne.j) then
            if((i+j).eq.3) iq=4
            if((i+j).eq.4) iq=6
            if((i+j).eq.5) iq=5
            cm(i,j)=cmu(iq)/2
        end if
    continue
5

Initiates the second identity tensor delt(3,3) which is a 2X2 identity matrix

do 6 i=1,3
    delt(i,i)=1.0
6
continue

Computes the forth order identity tensor delt4(3,3,3,3) by definition.

do 7 i=1,3
    do 7 j=1,3
        delt4(i,j,i,j)=delt(i,i)*delt(j,j)+delt(i,j)*delt(j,i)
        delt4(i,j,j,i)=delt4(i,j,i,j)
    continue
7

For different eigenvalues gl1,gl2,gl3 the computation is different.
case1 is gl1#gl2#gl3 call subroutine comsd1.
case2 is gl3=gl2#gl1 or gl1=gl3#gl2 or gl1=gl2#gl3 then
call subroutine compsd2.
case3 is gl1=gl2=gl3 call subroutine cccpsd3.

******************************************************************
if ((g11.ne.g12).and.(g12.ne.g13).and.(g11.ne.g13)) then  
call compsdl(g11,g12,g13,ae_t,delt4,cm,ts,td)  
else if((g12.eq.g13).and.(g11.ne.g13)) then  
call compsd2(g11,g12,delt,delt4,cm,ts,td)  
else if((g11.eq.g12).and.(g13.ne.g12)) then  
g11=g13  
call compsd2(g11,g12,delt,delt4,cm,ts,td)  
else if((g11.eq.g13).and.(g12.ne.g13)) then  
g11=g12  
g12=g13  
call compsd2(g11,g12,delt,delt4,cm,ts,td)  
else  
call compsd3(g11,delt,delt4,ts,td)  
end if  

Rewrite the tensor ts(l,j) td(i,j,k,l) to S(i) and D(i,j) respectively by using the symmetric property.

converting ts(3,3) s(6) and td(3,3,3,3) to D(6,6)

do 8 i=1,3  
do 8 j=i,3  
    if (i.eq.j) iq=i  
    if (i.eq.1.and.j.eq.2) iq=4  
    if (i.eq.2.and.j.eq.3) iq=5  
    if (i.eq.1.and.j.eq.3) iq=6  
    s(iq)=ts(i,j)  
8 continue  

do 9 i=1,3  
do 9 j=i,3  
    d(i,j)=td(i,i,j,j)  
9 continue  

do 10 i=1,3  
    d(i,4)=td(i,i,1,2)+td(i,i,2,1)  
    d(i,5)=td(i,i,2,3)+td(i,i,3,2)  
    d(i,6)=td(i,i,3,1)+td(i,i,1,3)  
10 continue
d(4,4) = (td(1,2,1,2)+td(1,2,2,1)+td(2,1,1,2)+td(2,1,2,1))/2.
d(4,5) = (td(1,2,2,3)+td(1,2,3,2)+td(2,1,2,3)+td(2,1,3,2))/2.
d(4,6) = (td(1,2,1,3)+td(1,2,3,1)+td(2,1,1,3)+td(2,1,3,1))/2.
d(5,5) = (td(2,3,2,3)+td(2,3,3,2)+td(3,2,2,3)+td(3,2,3,2))/2.
d(5,6) = (td(2,3,1,3)+td(2,3,3,1)+td(3,2,1,3)+td(3,2,3,1))/2.
d(6,6) = (td(3,1,1,3)+td(3,1,3,1)+td(1,3,1,3)+td(1,3,3,1))/2.

do 11 i=1,6
  do 11 j=1,6
    d(i,j) = d(j,i)
 continue

prints out the inputs gll, c12, g13, cmu(6)
and outputs S and D

print*, ' gll=', gll
print*, ' g12=', g12
print*, ' g13=', g13
print*, 'Input tensor C(6):'
print*, (cmu(i), i=1,6)
print*, "second order tensor S(6):"
print*, (s(i), i=1,6)
print*, "The forth order tensor D(6,6):"
do 101 i=1,6
print*, (d(i,j), j=1,6)
101 continue
return
end

subroutine comsdl1(gll, c12, g13, delt, delt4, cm, ts, td)

real*8 gll, g12, g13, ts(3,3), td(3,3,3,3)
real*8 cm(3,3), delt(3,3), delt4(3,3,3,3), p(3,3,3,3)
real*8 q(3,3,3,3), cm(3,3), p1(3,3,3,3), p21(3,3,3,3)
real*8 p31(3,3,3,3), q11(3,3,3,3), q12(3,3,3,3), p22(3,3,3,3)
real*8 q21(3,3,3,3), q22(3,3,3,3), a, b, c, a1, a2, a3, a4, a5, a6
Obtains $\mathbf{cmm}(3,3) = \mathbf{cm}(3,3) \times \mathbf{cm}(3,3)$ from subroutine product.

```
call product(cm, cm)
c```

Uses the formula we derived in code to compute second order tensor $\mathbf{ts}(3,3)$.

```
do 25037 i=1,3
   do 25038 j=1,3
      $\mathbf{ts}(i,j) = a(g_{ll},g_{l2},g_{l3}) \cdot \mathbf{cmm}(i,j) + b(g_{ll},g_{l2},g_{l3}) \cdot \mathbf{cm}(i,j)
                           c(g_{ll},g_{l2},g_{l3}) \cdot \delta(i,j)$
   continue
25038 continue
25037 continue
```

call subroutine to compute the functions we defined when we derived forth order tensor $\mathbf{td}$, namely $P(i,j,k,l)$ and $Q(l,j,k,l)$ which are the functions of $\mathbf{cm}(3,3)$ and the matrix product $\mathbf{cmm}(3,3)$.

```
call pqcom(cm, cm, pl, q)
call pqcom(cm, cm, p21, q)
call pqcom(cm, cm, p22, q)
call pqcom(cm, cm, p31, q)
call pqcom(cm, delt, p, q11)
call pqcom(delt, cm, p, q12)
call pqcom(cm, delt, p, q21)
call pqcom(delt, cm, p, q22)
c```

computes forth order tensor $\mathbf{td}(i,j,k,l)$

```
do 25039 i=1,3
   do 25040 j=1,3
      do 25041 k=1,3
         do 25042 l=1,3
            $\mathbf{td}(i,j,k,l) = a_1(g_{ll},g_{l2},g_{l3}) \cdot p_1(i,j,k,l) + a_2(g_{ll},g_{l2},g_{l3}) \cdot (p_21(i,j,k,l)+p_22(i,j,k,l)) + a_4(g_{ll},g_{l2},g_{l3}) \cdot p_31(i,j,k,l)+a_3(g_{ll},g_{l2},g_{l3}) \cdot (q_{11}(i,j,k,l)+q_{12}(i,j,k,l))+a_5(g_{ll},g_{l2},g_{l3}) \cdot (q_{21}(i,j,k,l)+q_{22}(i,j,k,l))+a_6(g_{ll},g_{l2},g_{l3})$
         continue
25041 continue
25040 continue
25039 continue
```

43
*delt4(i,j,k,l)
continue
continue
continue
continue
return
dend

a,b,c,a1-a6 are the coefficients we derived in code.

real*8 function a(gl1,gl2,gl3)
real*8 gl1,gl2,gl3,s1,s2,s3
a=-s1(gl1,gl2,gl3)/(-gl1+gl2)/(+gl1+gl3)+s2(gl1,gl2,gl3)/
. (-gl1+gl2)/(-gl2+gl3)s3(gl1,gl2,gl3)/(-gl1+gl3)/(-gl2+gl3)
return
dend

real*8 function b(gl1,gl2,gl3)
real*8 gl1,gl2,gl3,s1,s2,s3
b=s3(gl1,gl2,gl3)*(gl1+gl2)/(-gl1+gl3)-s2(gl1,gl2,gl3)*
. (+gl1+gl2)/(-gl2+gl3)+s1(gl1,gl2,gl3)*(gl1+gl2)/
. (-gl1+gl2)/(-gl1+gl3)
return
dend

real*8 function c(gl1,gl2,gl3)
real*8 gl1,gl2,gl3,s1,s2,s3
c=-s1(gl1,gl2,gl3)*gl2*gl3/(-gl1+gl2)/(-gl1+gl3)+s2(gl1,
. gl2,gl3)*gl3/(-gl1+gl2)/(-gl2+gl3)-s1*3(gl1,gl2,gl3)*gl2/
. (-gl1+gl3)/(-gl2+gl3)
return
dend
real*8 function a1(gl1, gl2, gl3)
real*8 gl1, gl2, gl3, s1, s2, s3, s11, s22, s33

a1 = -s11(gl1, gl2, gl3)/(-gl1 + gl2)**2/(-gl1 + gl3)**2 + 2.0*
s2(gl1, gl2, gl3)*(-gl1 - 2.0*gl2 + gl3)/(-gl1 + gl2)**3/
(-gl1 + gl3)**3 - s22(gl1, gl2, gl3)/(-gl1 + gl2)**3/(-gl1 + gl3)**2*
-s33(gl1, gl2, gl3)/(-gl1 + gl2)**2/(-gl1 + gl3)**2 - 2.0*
s1(gl1, gl2, gl3)*(-g11 + gl2 + gl3)/(-gl1 + gl2)**3/
(-gl1 + gl3)**3 + 2.0*s3(gl1, gl2, gl3)*(-g11 - gl2 + 2.0*gl3)/
(-gl1 + gl3)**3
return
d end

real*8 function a2(g11, g12, g13)
real*8 g11, g12, g13, s1, s2, s3, s11, s22, s33
a2 = s33(g11, g12, g13)*g11 + g12)/(-gl1 + gl3)**2/(-gl1 + gl2)**2 +
s22(g11, g12, g13)*g12 + g13)/(-gl1 + gl2)**2/(-g12 + gl3)**2 +
s11(g11, g12, g13)*(g12 + g13)*(-gl1 + gl2)**2/(-g11 + gl3)**2*
-s3(g11, g12, g13)*(-g11 + g12)**3/(-2.0*g11**2 - 3.0*g11* gl1 + 2.0*g12**2 +
3.0*g11 + g13 + 3.0*g12 + g13 + g13**2)/(-g11 + g13)**3/(-gl1 + gl3)**3 - s2(g11, g12, g13)*(-2.0*gl1**2 - 3.0*gl1* gl12 + g12 - 2.0*gl1 + g12)**2 +
g13 - 3.0*g12 + g13 + 2.0*g13**2)/(-gl1 + g12)**3/(-g11 + g13)**3 +
s1(g11, g12, g13)*(-g11 + 2.0*g11 + g12 + 2.0*g12 + g12**2 - 3.0*gl1*
g13 + 3.0*g12 + g13 + 2.0*g13**2)/(-gl1 + g12)**3/(-g11 + g13)**3
return
d end

real*8 function a3(g11, g12, g13)
real*8 g11, g12, g13, s1, s2, s3, s11, s22, s33
a3 = -s11(g11, g12, g13)*g12 + g13)/(-gl1 + gl2)**2/(-gl1 + gl3)**2 - g11*
s22(g11, g12, g13)*g13)/(-gl1 + gl2)**2/(-g12 + gl3)**2 - g11*
s33(g11, g12, g13)*g12)/(-gl1 + gl2)**2/(-g12 + gl3)**2 +
s2(g11, g12, g13)*g12)/(-g11 + g13)**2 + s2(g11, g12, g13)*g12)/
g13 - g11 * g12 * g13 - 2.0* g12**2 * g13 + g11 * g13**2 + g12 * g13**2)/
(-g11 + gl1g13)**3/(-g12 + gl1g13)**3 - s1(g11, g12, g13)*g11**3 - 2.0*
g11**2* g12 + 1.0* g11 + g12**2 - 2.0* g11**2* g13 - g11 * g12 + g13**2*
g11**2* g13 + g12**2* g13 + g11**2* g13 + g12**2* g13 + g12**2* g13 +
g12**2* g13 + g11**2* g13 + g12**2* g13 + g12**2* g13 + g12**2* g13 +
g12**2* g13 + g13**3)/(-g11 + gl1g13)**3
return
d end
real*8 function a4(gll,gl2,gl3)
real*8 gll,gl2,gl3,s1,s2,s3,s11,s22,s33
a4=-s33(gll,gl2,gl3)*(g11+gl2)**2/(-gll+gl3)**2/(-gl2+gl3)**2-
   s22(gll,gl2,gl3)*(gll+gl3)**2/(-gll+gl1)**2/(-gl2+gl3)**2-
   -s11(gll,gl2,gl3)*(gl2+gl3)**2/(-gll+gl2)**2/(-gl2+gl3)**2-
   (-gll+gl2)**2/(-gll+gl3)**2+2.0*s2(gll,gl2,gl3)*(gll+gl3)*
   (g1**2-g11*gl2+g12**2+g11*gl3-g12*gl3+gl3**2)/(-gl1+g12)**3/(-gl2+gl3)**3-
   g12**3/(-gl2+gl3)**3-2.0*s1(gll,gl2,gl3)*(gll+gl3)**2/(-gll+g12)**3/(-gl2+gl3)**3-
   *g11*gl2+gl2**2-g11*gl3+g12+gl3**2)/(-gll+g12)**3/(-gl2+gl3)**3-
   )**3+2.0*s3(gll,gl2,gl3)*(gl1+gl2)*(-gll**2-gll*gl1*gl2+gll**2+gll**3)/(-gll+gl2)**3/(-gl2+gl3)**3-
   g13+g12+g13**3)/(-gll+gl2)**3/(-gll+gl3)**3
return
end

real*8 function a5(gll,gl1,gl2,gl3)
real*8 gll,gl1,gl2,gl3,s1,s2,s3,s11,s22,s33
a5=gll*s33(gll,gl1,gl2,gl3)*gll*(g11+gl2)/(-gll+gl3)**2/(-gl2+gl3)
   +2.0*s2(gll,gl1,gl2,gl3)*gll*(gll+g12)/(-gll+g13)**2/(-g12+g13)**2-
   +s11(gll,gl1,gl2,gl3)*(gll+g12)*(-gll**2-gll*gl2+gll**2+gll**3)/(-gll+g12)**3/(-gll+g13)**3-
   +g13+g12+g12**2+g13**2+g12*gl1**2+g12**2+g12**3+gll**2+
   3*gl1**2+g12**2+g13**2+g12**2+g13**2+gll**3+gll**2+g13**2-
   2.0*gll**2+g12**2+gll**3+gll**2+g12**3+gll**3)/(-gll+g13)**3/(-gl1+g13)**3-
   +g12**2+g11*gl2**3+g11**2+g12**2+gll**2+g12**2+g12**2+g12**3+
   gll**2+g12**3/g1**3/(-gll+g13)**3/(-gll+g13)**3-s2(gll,gl1,gl2,gl3)*(g1**3-g12-2.0*g11
   +**2+g12**2+g11*gl2**3+g11**2+g12**2+g12**2+g12**3+gll**3-g12**2+g12**3+
   g13+g12**3+g13**2+g12**2+gll*gl1**2+g12**2+g12**2+g12**3+
   g11*gl3**2+g12**3/g1**3/(-gll+g12)**3/(-gll+g12)**3-s1(gll,gl1,gl2,gl3)+
   g13*+g11**2+g12**2+g11**2+g12**2+g12**2+g12**2+g12**3+/(-gll+g12)**3/(-gll+g12)**3+s1(gll,gl1,gl2,gl3)+
   +g13**2+g12**2+g13**2+gll**3+g12**3+gll**3/(-gll+g12)**3/(-gll+g12)**3
return
end
real*8 function a6(gll,gl2,gl3)
real*8 gl1,gl2,gl3,s1,s2,s3,sll,s22,s33
a6=-s11(gl1,gl2,gl3)*gl2**2*gl3**2/(-gl1+gl2)**2/(-gl1+gl3)**2
-g11**2*s22(gl1,gl2,gl3)*gl3**2/(-gl1+gl2)**2/(-gl2+gl3)**2-gll
**2*s33(gl1,gl2,gl3)*gl2**2/(-gl1+gl3)**2/(-gl2+gl3)**2-2.0*g11
*s3(gl1,gl2,gl3)*gl2**2*gl3**2/(-gl1+gl2)**2/(-gl2+gl3)**2-2.0*gll
**2/(-gl1+gl3)**2/(-gl2+gl3)**2-2.0*gl1*gl3-2.0*
gl2*gl3+gl3**2)/(-gl1+gl3)**3/(-gl2+gl3)**3+2.0*gl1*s2(gl1,gl2,
gl3)*gl2*gl3*(gl1**2-2.0*gl1*gl2+gl2**2+gl1*gl3-2.0*gl2*gl3+
gl3**2)/(-gl1+gl2)**3/(-gl2+gl3)**3-2.0*gl1*s1(gl1,gl2,gl3)*s2
(s1+2.0*gl1*gl2+gl2**2-2.0*gl1*gl3+gl2*gl3+gl3**2)/
(-gl1+gl2)**3/(-gl1+gl3)**3
return
end

The s1,s2,s3,s11,s22,s33 are derivatives of W

function s1(gl1,gl2,gl3)
common /param/ x1,x2,y1,y2
real*8 x1,x2,y1,y2
real*8 s1,gl1,gl2,gl3
s1=2.0*(gl1**(-1+y1)*x1*y1+gl1**(-1+y2)*x2*y2)
return
end

function s2(gl1,gl2,gl3)
common /param/ x1,x2,y1,y2
real*8 x1,x2,y1,y2
real*8 s2,gl1,gl2,gl3
s2=2.0*(gl2**(-1+y1)*x1*y1+gl2**(-1+y2)*x2*y2)
return
end

function s3(gl1,gl2,gl3)
common /param/ x1,x2,y1,y2
real*8 x1,x2,y1,y2
real*8 s3,gl1,gl2,gl3
s3=2.0*(gl3**(-1+y1)*x1*y1+gl3**(-1+y2)*x2*y2)
return
end
function s11(g11,g12,g13)
common /param/ x1,x2,y1,y2
real*8 x1,x2,y1,y2
real*8 s11,g11,g12,g13
s11=2.0*(g11**(-2+y1)*x1*(-1.0+y1)*y1+g11**(-2+y2)*x2* (-1.0+y2)*y2)
return
end

function s22(g11,g12,g13)
common /param/ x1,x2,y1,y2
real*8 x1,x2,y1,y2
real*8 s22,g11,g12,g13
s22=2.0*(g12**(-2+y1)*x1*(-1.0+y1)*y1+g12**(-2+y2)*x2* (-1.0+y2)*y2)
return
end

function s33(g11,g12,g13)
common /param/ x1,x2,y1,y2
real*8 x1,x2,y1,y2
real*8 s33,g11,g12,g13
s33=2.0*(g13**(-2+y1)*x1*(-1.0+y1)*y1+g13**(-2+y2)*x2* (-1.0+y2)*y2)
return
end

subroutine compsd2(g11,g12,delt,delt4,cm,ts,td)
real*8 g11,g12,ts(3,3),td(3,3,3,3)
real*8 cm(3,3),delt(3,3),delt4(3,3,3,3),pl(3,3,3,3)
real*8 q2(3,3,3,3),qi(3,3,3,3),p(3,3,3,3),q(3,3,3,3)
real*8 b1,b2,b3,abar,bbar
Computes second order tensor $ts(i,j)$ based on the formula derived in code.

\[ \text{do } 25043 \ i=1,3 \]
\[ \text{do } 25044 \ j=1,3 \]
\[ ts(i,j)=abar(gll,gl2)*cm(i,j)+bbar(gll,gl2)*delt(i,j) \]
\[ \text{continue} \]
\[ \text{continue} \]

Call subroutine to get $P, Q$ which are defined in code.

\[ \text{call pqcom(cm,cm,p1,q)} \]
\[ \text{call pqcom(cm,delt,p,q1)} \]
\[ \text{call pqcom(delt,cm,p,q2)} \]

Computes tensor $td(i,j,k,l)$.

\[ \text{do } 25045 \ l=1,3 \]
\[ \text{do } 25046 \ j=1,3 \]
\[ \text{do } 25047 \ k=1,3 \]
\[ \text{do } 25048 \ i=1,3 \]
\[ td(i,j,k,l)=b1(gll,gl2)*p1(i,j,k,l)+b2(gll,gl2)*(q1(i,j,k,l)+q2(i,j,k,l))+b3(gll,gl2)*delt4(i,j,k,l) \]
\[ \text{continue} \]
\[ \text{continue} \]
\[ \text{continue} \]
\[ \text{continue} \]
\[ \text{return} \]
\[ \text{end} \]

$abar, bbar$ are $b1, b2, b3$ functions derived in code.

\[ \text{real*8 function abar(gll,gl2)} \]
\[ \text{real8 gll,gl2,ss1,ss2} \]
\[ abar=(-ss1(gll,gl2)+ss2(gll,gl2))/(-gll+gl2) \]
\[ \text{return} \]
\[ \text{end} \]
real*8 function bbar(gl1,gl2)
real*8 gl1,gl2,ss1,ss2
bbar=(-gl1*ss2(gl1,gl2)+ss1(gl1,gl2)*gl2)/(-gl1+gl2)
return
end

real*8 function b1(gl1,gl2)
real*8 gl1,gl2,ss1,ss2,ssl,ss2
b1=2.0*sl(gl1,gl2)/(-gl1+gl2)**3-2.0*ss2(gl1,gl2)/(-gl1+gl2)**3+ss11(gl1,gl2)/(-gl1+gl2)**2+ss22(gl1,gl2)/(-gl1+gl2)**2 return
end

real*8 function b2(gl1,gl2)
real*8 gl1,gl2,ss1,ss2,ssl,ss2
b2=-gl1*ss22(gl1,gl2)/(-gl1+gl2)**2-ss11(gl1,gl2)*gl2/(-gl1+gl2)**2+ssl(gl1,gl2)*(gl1+gl2)/(-gl1+gl2)**3+ss22(gl1,gl2)/(-gl1+gl2)**2 return
end

real*8 function b3(gl1,gl2)
real*8 gl1,gl2,ss1,ss2,ssl,ss2
b3=2.0*gl1*ss1(gl1,gl2)*gl2/(-gl1+gl2)**3-2.0*gl1*ss2(gl1,gl2)**3+ss11(gl1,gl2)*gl1**2/(-gl1+gl2)**2+ss22(gl1,gl2)/(-gl1+gl2)**2 return
end

ss1,ss2,ssl,ss2 are derivatives of W.

function ss1(gl1,gl2)
common /param/ x1,x2,y1,y2
real*8 x1,x2,y1,y2
real*8 ss1,gl1,gl2
ss1=2.0*(gl1**(-1+y1)*x1*y1+gl1**(-1+y2)*x2*y2)
return
end

50
function ss2(g11,g12)
common /param/ x1,x2,y1,y2
real*8 x1,x2,y1,y2
real ss2,g11,g12
ss2=2.0*(2.0*g12**(-1+y1)*x1*y1+2.0*g12**(-1+y2)*x2*y2)
return
end

function ss11(g11,g12)
common /param/ x1,x2,y1,y2
real*8 x1,x2,y1,y2
real*8 ss11,g11,g12
ss11=2.0*(g11**(-2+y1)*x1*(-1.0+y1)*y1+g11**(-2+y2)*x2*(-1.0+y2)*y2)
return
end

function ss22(g11,g12)
common /param/ x1,x2,y1,y2
real*8 x1,x2,y1,y2
real*8 ss22,g11,g12
ss22=2.0*(2.0*g12**(-2+y1)*x1*(-1.0+y1)*y1+2.0*g12**(-2+y2)*x2*(-1.0+y2)*y2)
return
end

subroutine compsd3(g11,delt,delt4,ts,td)
real*8 g11,ts(3,3),td(3,3,3,3),delt(3,3),delt4(3,3,3,3)
real*8 cc1,abbar

do 25049 i=1,3
   do 25050 j=1,3
      ts(i,j)=abbar (g11)*delt(i,j)
   25050 continue
25049 continue
do 25051 i=1,3
   do 25052 j=1,3
      do 25053 k=1,3
         do 25054 l=1,3
            td(i,j,k,l)=cc1(g11)*delt4(i,j,k,l)
         continue
      continue
   continue
25051 continue
25054 return
25053 continue
25052 continue
25051 continue

real*8 function abbar(g11)
real*8 g11
abbar=sssl(g11)
return
end

real*8 function cc1(g11)
real*8 g11
ccl=sssl1(g11)
return
end

function sssl1(g11)
common /param/ x1,x2,y1,y2
real*8 x1,x2,y1,y2
real*8 sssl1,g11
sssl1=2.0*(3.0*g11**(-1+y1)*x1*y1+3.0*g11**(-1+y2)*x2*y2)
return
end
function sss11(g11)
common /param/ x1,x2,y1,y2
real*8 x1,x2,y1,y2
real*8 sss11,g11

sss11=2.0*(3.0*g11**(-2+y1)*x1*(-1.0+y1)*y1+3.0*g11**(-2+y2)*x2*(-1.0+y2)*y2)
return
end

This subroutine computes P and Q forth order tensors which we define in tensor D

subroutine pqcom(cml,cm2,p,q)
real*8 cml(3,3),cm2(3,3),p(3,3,3,3),q(3,3,3,3)
do 100 i=1,3
do 100 j=1,3
do 100 k=1,3
do 100 l=1,3
   p(i,j,k,l)=cm1(i,k)*cm2(j,l)+cm1(i,l)*cm2(j,k)
   q(i,j,k,l)=p(i,j,k,l)+cm1(j,l)*cm2(i,k)+cm1(j,k)*cm2(i,l)
100 continue
return
end

This subroutine computes matrix product cmXcm.

subroutine product(mat1,cmm)
real*8 mat1(3,3),cmm(3,3),sum
do 25 i=1,3
do 25 j=1,3
   sum=0.0
do 26 k=1,3
      sum=sum+mat1(i,k)*mat1(k,j)
   continue
   cmm(i,j)=sum
25 continue
return
end
## Explicit Robust Schemes for Implementation of a Class of Principal Value-Based Constitutive Models: Symbolic and Numeric Implementation

### Authors
S.M. Arnold, A.F. Saleeb, H.Q. Tan, and Y. Zhang

### National Aeronautics and Space Administration
Lewis Research Center
Cleveland, Ohio 44135–3191

### Performing Organization Name(s) and Address(es)
National Aeronautics and Space Administration
Washington, D.C. 20546–0001

### Performing Organization Report Number
E-7788

### Sponsoring/Monitoring Agency Name(s) and Address(es)
National Aeronautics and Space Administration
Washington, D.C. 20546–0001

### Sponsoring/Monitoring Agency Report Number
NASA TM–106124

### Distribution/Availability Statement
Unclassified - Unlimited
Subject Category 49 59

### Abstract
The issue of developing effective and robust schemes to implement a class of the Ogden-type hyperelastic constitutive models is addressed. To this end, special purpose functions (running under MACSYMA) are developed for the symbolic derivation, evaluation, and automatic FORTRAN code generation of explicit expressions for the corresponding stress function and material tangent stiffness tensors. These explicit forms are valid over the entire deformation range, since the singularities resulting from repeated principal-stretch values have been theoretically removed. The required computational algorithms are outlined, and the resulting FORTRAN computer code is presented.