NASA Technical Memorandum 106123

Explicit Robust Schemes for Implementation of General Principal Value-Based Constitutive Models

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

NASA

Explicit Robust Schemes for Implementation of General Principal Value-Based Constitutive Models

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

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

Abstract

The issue of developing effective and robust schemes to implement general hyperelastic constitutive models is addressed. To this end, special purpose functions are used to symbolically derive, evaluate, and automatically generate the associated FORTRAN code for the explicit forms of the corresponding stress function and material tangent stiffness tensors. These explicit forms are valid for the entire deformation range (i.e.,with both distinct and repeated principal-stretch values). The analytical form of these explicit expressions is given here for the case in which the strain-energy potential W is taken as a *nonseparable* polynomial function of the principle stretches.

1 Introduction

Recently, constitutive models of rubber hyperelasticity, using alternative representations in terms of the principal stretches as opposed to deformation invariants, have become increasingly popular in nonlinear finite element analyses [1-3]. Two of our recent publications have discussed in detail the theoretical development [4] and symbolic and numeric implementation [5] of explicit forms for the second Piola Kirchhoff stress tensor and the material tangent stiffness tensor. These forms correspond to *a class of Ogden type* hyperelastic constitutive models, based on principal-stretch values. These explicit expressions are

significant since they are valid for the entire deformation range, even though the main constituents of the deformation tensor (i.e., principle values and associated eigenvectors) are, in general, neither uniquely defined nor continuously differentiable over the entire range. The two specific forms of the Ogden-type strain energy functions addressed in reference 4 encompass many of the popular representations currently in use for rubber materials. However, those functions were restricted to special forms of nonseparable representations of the strain energy density functions, with the restricted nonseparable form given in reference 4, section 5, dealing with the important and practical case of incompressible and slightly compressible solids. To date, comparable treatments for the *general nonseparable* forms of the models are not available in the literature. Indeed, it is the extension of our earlier results [4] and recent developments [5] to deal with this latter case that constitutes our main objective in the present paper.

By cleverly applying symbolic manipulation packages so as to control expression growth new constitutive theories can be developed and applied (e.g., finite element; see, [6] and [7]). Symbolic computation uses numbers, formulas, vectors, matrices, equations and the like to compute exact solutions; whereas numerical computation uses floating-point numbers to compute approximate solutions to problems of practical interest. Here, we will utilize three recently developed [5] special purpose symbolic functions (SDIFF, SDIFFEV, and TEMPLATE) running under DOE MACSYMA [8]. These special purpose functions allow the derivation and automatic FORTRAN code generation of alternative *generalized* potential based constitutive models composed of principal values and their associated eigenvectors.

This paper begins by reviewing highlights of our previous work in developing the theory of explicit forms [4] and implementing them symbolically and numerically [5]. Following this review, the results of the derivation of the generalized expressions for the second Piola Kirchhoff stress tensor S_{ij} and the material moduli tensor D_{ijkl} are given. The paper then concludes with a discussion of the template files required to automatically generate the associated FORTRAN source code.

2 Background

The theoretical development of a *singularity-free* representation of principal value-based constitutive models has been discussed at length in reference [4]. Here, we will confine our discussion to hyperelastic isotropic materials whose strain energy function W is taken to be a general function of the principal stretches, that is,

$$W = W(\lambda_1, \lambda_2, \lambda_3) \tag{1}$$

where $\lambda_1, \lambda_2, \lambda_3$ are 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,

$$C_{ij} = \sum_{l=1}^{3} \lambda_{(l)} N_{ij}^{(l)} \tag{2}$$

where $N_{ij}^{(l)}$, which is often referred to as the (orthogonal) *eigenprojection* operator related to the associated eigenvectors of C_{ij} is defined as

$$N_{ij}^{(l)} = n_i^l n_j^l \tag{3}$$

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

$$C_{ij} = (\lambda_1 - \lambda) N_{ij}^{(1)} + \lambda \delta_{ij} \tag{4}$$

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

$$C_{ij} = \lambda \sum_{l=1}^{3} N_{ij}^{(l)} = \lambda \delta_{ij}. \tag{5}$$

Similarly, through suitable manipulation of equations (2) and (4), explicit expressions for $N_{ij}^{(r)}$ in terms of C_{ij} can be obtained for the case of three distinct eigenvalues,

$$N_{ij}^{(r)} = \frac{1}{(\lambda_r - \lambda_s)(\lambda_s - \lambda_t)} [(C_{ij} - \lambda_s \delta_{ij})(C_{ij} - \lambda_t \delta_{ij})] \tag{6}$$

and for the case of double coalescence,

$$N_{ij}^{(r)} = \frac{1}{(\lambda_r - \lambda)} (C_{ij} - \lambda \delta_{ij}). \tag{7}$$

In the preceding equations r, s, and t represent any cyclic permutation of (1, 2, or 3). These definitions will prove very useful in obtaining the pertinent singularity-free directional derivatives of both the potential 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

$$S_{ij} = 2 \frac{\partial W}{\partial C_{ij}} \equiv S_{ij}(C_{ij}) \tag{8}$$

Those for the material moduli tensor $D_{ijkl}(C_{ij})$ can then be obtained by applying the directional derivative formula to S_{ij}, that is,

$$D_{ijkl} = 2 \frac{\partial S_{ij}}{\partial C_{kl}} = 4 \frac{\partial^2 W}{\partial C_{ij} \partial C_{kl}} \equiv D_{ijkl}(C_{ij}) \tag{9}$$

As a result, the explicit expressions for the tensors $S_{ij}(C_{ij})$ and $D_{ijkl}(C_{ij})$ can be obtained directly for the following three cases: case I - when all three eigenvalues are distinct; case II - when a single singularity is present ($\lambda_1 \neq \lambda_2 = \lambda_3 = \lambda$, i.e., double coalescence); or case III - when a double singularity is present ($\lambda_1 \neq \lambda_2 = \lambda_3 = \lambda$, i.e., triple coalescence).

The derivation and implementation process for the formulations described was recently automated [5] by constructing three special purpose functions (SDIFF, SDIFFEV, and TEMPLATE), written at the MACSYMA command level, that can respectively,

(1) Derive the 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 automatically generate (using the built-in MACSYMA function **gentran**) the associated FORTRAN code needed to evaluate the expressions numerically for a given potential function, W

These three 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. Thus each special purpose function can be thought of as a macro command.

3 Symbolic Derivation

Let us begin by assuming that W is a nonseparable function of λ_1, λ_2, and λ_3. For example,

$$W = \sum_{n=1}^{p} [x_n(\lambda_1 + \lambda_2 + \lambda_3)^{\alpha_n} + y_n(\lambda_1\lambda_2 + \lambda_2\lambda_3 + \lambda_3\lambda_1)^{\beta_n} + z_n(\lambda_1\lambda_2\lambda_3)^{\gamma_n}]$$

As a result, when the special purpose function SDIFF is invoked, the scalar derivative of W with respect to each eigenvalue will no longer be a function of that eigenvalue only, as discussed in reference 5, but will instead be a function of all three eigenvalues, that is,

$$s_{(l)}(\lambda_1, \lambda_2, \lambda_3) = 2\frac{\partial W}{\partial \lambda_{(l)}}.$$

Furthermore, in deriving D_{ijkl}, the mixed second derivatives ($\frac{\partial^2 W}{\partial \lambda_i \lambda_j}$) must also be taken into account in the procedure. To derive the generalized explicit expression for the three cases, one need only issue the command SDIFF upon invoking MACSYMA, as shown here:

- Case I -three distinct eigenvalues ($\lambda_1 \neq \lambda_2 \neq \lambda_3$)

$$SDIFF(1)$$

- Case II - double coalescence ($\lambda_1 \neq \lambda_2 = \lambda_3 = \lambda$)

$$SDIFF(2)$$

- Case III - triple coalescence ($\lambda_1 = \lambda_2 = \lambda_3 = \lambda$)

$$SDIFF(3)$$

Note that the resulting derived expressions have been manipulated and condensed so as to streamline their reporting and to facilitate their comparison with previous work [4].

3.1 Results for Case I

The explicit expression for the second Piola Kirchhoff stress tensor is

$$S_{ij} = aC_{ik}C_{kj} + bC_{ij} + c\delta_{ij} \tag{10}$$

where δ_{ij} is the second order identity tensor and a,b, and c are defined as

$$a = -m[s_1(\lambda_2 - \lambda_3) + s_2(\lambda_3 - \lambda_1) + s_3(\lambda_1 - \lambda_2)] \tag{11}$$

$$b = m[s_1(\lambda_2^2 - \lambda_3^2) + s_2(\lambda_3^2 - \lambda_1^2) + s_3(\lambda_1^2 - \lambda_2^2)] \tag{12}$$

$$c = -m[s_1\lambda_2\lambda_3(\lambda_2 - \lambda_3) + s_2\lambda_3\lambda_1(\lambda_3 - \lambda_1) + s_3\lambda_1\lambda_2(\lambda_1 - \lambda_2)] \tag{13}$$

and where

$$m = \frac{1}{(\lambda_1 - \lambda_2)(\lambda_2 - \lambda_3)(\lambda_3 - \lambda_1)} \tag{14}$$

The explicit expression for the material moduli tensor $D_{ijkl}(C_{ij})$ is

$$\begin{aligned} D_{ijkl} = {} & a_1 P(C_{kl}^2, C_{ij}^2) + a_2[P(C_{kl}^2, C_{ij}) + P(C_{kl}, C_{ij}^2)] \\ & + a_3[Q(C_{kl}^2\delta_{ij}) + P(\delta_{kl}, C_{ij}^2)] + a_4 P(C_{kl}, C_{ij}) \\ & + a_5[Q(C_{kl}, \delta_{ij}) + Q(\delta_{kl}, C_{ij})] + 2a_6 I_{ijkl} \end{aligned} \tag{15}$$

where two second order symmetric tensors P and Q have been introduced and are defined as

$$P_{ijkl}(G, H) = G_{ik}H_{jl} + G_{il}H_{jk} \tag{16}$$

$$Q_{ijkl}(G,H) = G_{ik}H_{jl} + G_{ij}H_{jk} + G_{jl}H_{ik} + G_{jk}H_{il} \tag{17}$$

and the notation

$$I_{ijkl} = \frac{1}{2}[\delta_{ik}\delta_{jl} + \delta_{il}\delta_{jk}] \tag{18}$$

$$C_{ij}^2 = C_{im}C_{mj} \tag{19}$$

has been used in equation (15). Here the coefficients $a_1, a_2, ..., a_6$ are defined as

$$a_1 = \sum_{r=1}^{3}\eta_r + \sum_{r=1}^{3}\sum_{s=1,r\neq s}^{3}\xi_{rs} \tag{20}$$

$$a_2 = \sum_{r=1}^{3}(\lambda_r - I_1)\eta_r - \frac{1}{2}\sum_{r=1}^{3}\sum_{s=1,r\neq s}^{3}(2I_1 - \lambda_r - \lambda_s)\xi_{rs} \tag{21}$$

$$a_3 = \sum_{r=1}^{3}\frac{I_3\eta_r}{\lambda_r} + \frac{1}{2}\sum_{r=1}^{3}\sum_{s=1,r\neq s}^{3}(\lambda_r + \lambda_s)(I_1 - \lambda_r - \lambda_s)\xi_{rs} \tag{22}$$

$$a_4 = \sum_{r=1}^{3}(I_1 - \lambda_r)^2\eta_r + \frac{1}{2}\sum_{r=1}^{3}\sum_{s=1,r\neq s}^{3}(I_1 - \lambda_r)(I_1 - \lambda_s)\xi_{rs} \tag{23}$$

$$a_5 = \sum_{r=1}^{3}(\mu_r + \frac{I_3\eta_r(\lambda_r - I_1)}{\lambda_r}) - \frac{1}{2}\sum_{r=1}^{3}\sum_{s=1,r\neq s}^{3}(I_2 + \lambda_r\lambda_s)(I_1 - \lambda_r - \lambda_s)\xi_{rs} \tag{24}$$

$$a_6 = \sum_{r=1}^{3}(\frac{I_3}{\lambda_r})^2\eta_r + (\lambda_r - I_1)\mu_r + \sum_{r=1}^{3}\sum_{s=1,r\neq s}^{3}I_3(I_1 - \lambda_r - \lambda_s)\xi_{rs} \tag{25}$$

where

$$\mu_r = \frac{s_r}{(\lambda_r - \lambda_s)(\lambda_r - \lambda_t)}$$

$$\eta_r = \frac{[s_{rr} + (\lambda_s - \lambda_r)(\mu_r + \mu_t) + (\lambda_t - \lambda_r)(\mu_r + \mu_s)]}{(\lambda_r - \lambda_s)^2(\lambda_r - \lambda_t)^2}$$

$$\xi_{rs} = \frac{s_{rs}}{(\lambda_r - \lambda_s)^2(\lambda_r - \lambda_t)(\lambda_s - \lambda_t)}$$

and

$$I_1 = \lambda_1 + \lambda_2 + \lambda_3$$

$$I_2 = \lambda_1\lambda_2 + \lambda_2\lambda_3 + \lambda_1\lambda_3$$

$$I_3 = \lambda_1\lambda_2\lambda_3$$

Note that in the preceding expression the following differentiation notation has been introduced:

$$s_l = s_l(\lambda_1, \lambda_2, \lambda_3) = 2\frac{\partial W}{\partial \lambda_{(l)}} \tag{26}$$

and

$$s_{ij} = s_{ij}(\lambda_1, \lambda_2, \lambda_3) = \frac{\partial s_i(\lambda_1, \lambda_2, \lambda_3)}{\partial \lambda_j} = \frac{\partial^2 W}{\partial \lambda_i \lambda_j} \tag{27}$$

For example, $s_{11} = 2\frac{\partial^2 W}{\partial \lambda_1^2}; s_{22} = 2\frac{\partial^2 W}{\partial \lambda_2^2}; s_{33} = 2\frac{\partial^2 W}{\partial \lambda_3^2}; s_{12} = s_{21} = \frac{\partial^2 W}{\partial \lambda_1 \lambda_2}; s_{13} = s_{31} = \frac{\partial^2 W}{\partial \lambda_1 \lambda_3}; s_{23} = s_{32} = \frac{\partial^2 W}{\partial \lambda_2 \lambda_3}$. A comparison of the preceding expressions and those obtained earlier for the two special Ogden-type strain energy forms [4], shows that the expressions are identical except for the additional double-summation terms (containing the cross derivative terms) in the coefficients $a_1, a_2, ...a_6$ (see equations (20)-(25)) comprising the material moduli tensor D_{ijkl}. Thus the previous work is now merely a special case of the present generalized expressions.

3.2 Results for Case II

In this case, a single singularity ($\lambda_1 \neq \lambda_2 = \lambda_3 = \lambda$) is analytically removed, thereby yielding

$$S_{ij} = \overline{a} C_{ij} + \overline{b} \delta_{ij} \tag{28}$$

with

$$\overline{a} = \frac{s_1 - s_2}{(\lambda_1 - \lambda)} \tag{29}$$

$$\overline{b} = -\frac{[s_1 \lambda - s_2 \lambda_1]}{(\lambda_1 - \lambda)} \tag{30}$$

and a reduced material moduli tensor

$$D_{ijkl} = b_1 P(C_{kl}, C_{ij}) + b_2[Q(C_{kl}, \delta_{ij}) + Q(\delta_{kl}, C_{ij})] + b_3 I_{ijkl} \tag{31}$$

where

$$b_1 = \frac{1}{(\lambda_1 - \lambda_2)^3}\{(\lambda_1 - \lambda_2)[s_{11} + s_{22} - 2s_{12}] - 2[s_1 - s_2]\}$$

$$b_2 = \frac{1}{(\lambda_1 - \lambda_2)^3}\{(\lambda_2 - \lambda_1)[\lambda_2 s_{11} + \lambda_1 s_{22} - 2(\lambda_1 + \lambda_2)s_{12}] + (\lambda_1 + \lambda_2)[s_1 - s_2]\}$$

$$b_3 = \frac{1}{(\lambda_1 - \lambda_2)^3}\{(\lambda_1 - \lambda_2)[\lambda_1^2 s_{22} + \lambda_2^2 s_{11} - 2\lambda_1 \lambda_2 s_{12}] - 2\lambda_1 \lambda_2 [s_1 - s_2]\}$$

Again, in comparing the coefficients a and b, and, b_1, b_2, and b_3 to those obtained in previous work [4], the only difference seen is the appearance of the cross derivative term (s_{12}) in coefficients b_1, b_2, and b_3.

3.3 Results for Case III

Finally, in the case of a double singularity ($\lambda_1 = \lambda_2 = \lambda_3 = \lambda$), the explicit expression for the stress tensor becomes

$$S_{ij} = s_\lambda(\lambda)\delta_{ij} \tag{32}$$

whereas the material moduli tensor becomes

$$D_{ijkl} = 2s_{\lambda\lambda}(\lambda)\delta_{ijkl}. \tag{33}$$

These are identical to the previous results, as one would expect.

The value of automating the foregoing derivation procedure is apparent in that not only does this special purpose function SDIFF relieve the user of the tedious manual derivation process, but it also ensures analytical accuracy. This was illustrated prior to the publication of reference 4, in that a number of errors in the hand derivation were detected, verified, and corrected. Also, because the derivation process needs to be executed only once, except for the evaluation of the scalar derivatives in equations (26) and (27) for each new definition of W, a second special purpose function, SDIFFEV, as described in [5], was developed. This function is used to symbolically evaluate the foregoing expressions.

3.4 FORTRAN Code Generation

The function TEMPLATE is similar to the function SDIFFEV in that both functions will evaluate the explicit expressions obtained from SDIFF. TEMPLATE, however, will automatically generate the associated FORTRAN source code needed to numerically evaluate the expressions for a given potential function W. Code generation is accomplished by utilizing the MACSYMA built-in function **gentran** and a number of template files. The template files can be thought of as a framework for the FORTRAN generation of four subroutines (the main driving routine COMPSD and three subroutines, one each for case I , case II, and case III) and numerous functions. The template file for the main driving routine COMPSD is shown in appendix A. 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 λ_1, λ_2, and λ_3 (denoted by gl1, gl2, and gl3 respectively). 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. Clearly, the only code generation required is that of subroutines COMPSD1, COMPSD2, and COMPSD3. Code generation is initiated by issuing the command **gentranin,** preceded by and followed by less than and greater than symbols, respectively.

The subroutines COMPSD1, COMPSD2, and COMPSD3 are associated with case I ($\lambda_1 \neq \lambda_2 \neq \lambda_3$), case II ($\lambda_1, \lambda = \lambda_2 = \lambda_3$), 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 in appendices B,C and D, respectively. Note that in these routines, most of the FORTRAN code is automatically generated, since it pertains to the definition of coefficients a,b,c ; $a_1, a_2, ..., a_6$, and the first and second scalar derivatives of the strain energy function W, (i.e., $s_1, s_2, s_3, s_{11}, s_{22}$, and s_{33}). Also, the **gentran** commands are again preceded and followed by double inequality signs (that is, $\ll \gg$). All functions that are associated with a given case have been included in the corresponding appendix. As a result, with the appropriate template files, the FORTRAN source code associated with any general nonseparable or separable strain-energy function can easily be generated.

4 Summary of Results

Taken separately, the main constituents of the deformation tensor (i.e., the 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 was entirely bypassed by resorting to explicit symbolic derivations of appropriate forms of the material tangent-stiffness matrices which are valid for the entire deformation range. Furthermore, to enhance effective utilization and implementation of the present results, automatic FORTRAN code generation of the present *generalized explicit expressions* was pursued and achieved. As a result, nonseparable forms dealing with the important practical case of incompressible and slightly compressible solids can easily be generated. Finally, the generic analytical forms of these explicit expressions have been given for three cases: (1) distinct eigenvalues, (2) one distinct eigenvalue, and (3) no distinct eigenvalues.

In the future we will broaden our scope of application to include not only deformation constitutive models but also damage representations as well. An example that immediately comes to mind, where the above singularity-free representations will be important, is a maximum principle stress (or strain) damage formulation. Using this work as a building block, we can then envision moving to even more sophisticated damage formulations involving even higher tensorial representations.

References

[1] Finney, R.H.; and Kumar, A.: Development of Material Constants for Nonlinear Finite Element Analysis. J. Rubber Chem. Tech., Vol.. 61, pp. 879-891, 1988.

[2] Sussman, T.; and Bathe, K.J.: A Finite Element Formulation for Nonlinear Incompressible Elastic and Inelastic Analysis. Comp. Struct. , Vol. 26, pp. 357-409, 1987.

[3] Chang, T.Y.; Saleeb, A.F.; and Li, G.: Large Strain Analysis of Rubber-Like Materials by a Mixed Finite Element. Computational Mech., Vol. 8, No.4, pp. 221-233, 1991.

[4] Saleeb.A.F.; and Arnold, S.M.: Explicit Robust Schemes for Implementation of a Class Of Principle Value-Based Constitutive Models: Theoretical Development. NASA TM 105345, 1991.

[5] Arnold, S.M.; et al.: Explicit Robust Schemes for Implementation of a Class of Principal Value-Based Constitutive Models: Symbolic and Numeric Implementation. NASA TM 106124, 1993.

[6] Arnold, S.M.; and Tan, H.Q.: Symbolic Derivation of Potential Based Constitutive Equations. Computational Mech., Vol. 6, pp. 237-246, 1990.

[7] Arnold, S.M.; Tan, H.Q.; and Dong, X.: Application of Symbolic Computations to The Constitutive Modeling of Structural Materials. Symbolic Computations and Their Impact on Mechanics, Noor, A.K., Elishakoff, I. and Hulbert, G., eds., PVP-Vol. 205, ASME, pp.215-229.,1990.

[8] MATHLAB Group: MACSYMA Reference Manual. Version 10. Laboratory for Computer Science, Massachusetts Institute of Technology, Cambridge, MA, 1984.

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 gl1,gl2,gl3,
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
c       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:
c       S=D*C
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(gl1,gl2,gl3,cmu,s,d)
        real*8 gl1,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)
c
c       converts cmu(6) to matrix cm(3,3) in a way that
c       cm(1,2)=cm(2,1)=cmu(4),cm(2,3)=cm(3,2)=cmu(5),
c       cm(1,3)=cm(3,1)=cmu(6).
c
```

```
        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=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)
 7     continue
c
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 compsd2. case3 is gl1=gl2=gl3 call subroutine
c       compsd3.
c**********************************************************************
        if ((gl1.ne.gl2).and.(gl2.ne.gl3).and.(gl1.ne.gl3)) then
        call compsd1(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
c         Rewrite the tensor ts(i,j) td(i,j,k,l)to S(i) and D(i,j)
c         respectively by using the symetric property.
c         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
 8        continue
          do 9 i=1,3
            do 9 j=i,3
               d(i,j)=td(i,i,j,j)
            continue
 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)
 11   continue
c
c     prints out the inputs gl1,gl2,gl3,cmu(6) and outputs S and D
c
      print*, 'gl1=', gl1
      print*, 'gl2=', gl2
      print*, 'gl3=', 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)
 101  continue
      return
      end
c
      subroutine compsd1(gl1,gl2,gl3,delt,delt4,cm,ts,td)
<<
      gentranin("case1.tem")$
>>
      subroutine compsd2(gl1,gl2,delt,delt4,cm,ts,td)
<<
      gentranin("case2.tem")$
>>
      subroutine compsd3(gl1,delt,delt4,ts,td)
<<
      gentranin("case3.tem")$

>>
```

```
c
c       This subroutine computes P and Q forth order tensors
c       which we define in tensor D.
c
        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
c
c       This subroutine computes matrix product cmXcm.
c
        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 gl1,gl2,gl3,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
c
c     Obtains cmm(3,3)=cm(3,3)*cm(3,3) from subroutine product
c
      call product(cm,cmm)
c     Uses the formula we derived in code to compute second order
c     tensor ts(3,3).
<<
      gentran(for i:1 thru 3 do
       (for j:1 thru 3 do
         (ts[i,j]:a(gl1,gl2,gl3)*cmm[i,j]+b(gl1,gl2,gl3)
           *cm[i,j]+c(gl1,gl2,gl3)*delt[i,j])))$
>>
c     Call subroutine to compute all the functions we defined
c     when we derived forth order tenosor td, namely P(i,j,k,l)
c     and Q(i,j,k,l) which are the functions of cm(3,3) and
c     the matrix product cmm(3,3).
c
      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)
```

```
c
c       Computes forth order tensor td(i,j,k,l)
c
<<
        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(gl1,gl2,gl3)*p1[i,j,k,l]+a2(gl1,gl2,gl3)
            *(p21[i,j,k,l]+p22[i,j,k,l])+a4(gl1,gl2,gl3)*p31[i,j,k,l]
            +a3(gl1,gl2,gl3)*(q11[i,j,k,l]+q12[i,j,k,l])
            +a5(gl1,gl2,gl3)*(q21[i,j,k,l]+q22[i,j,k,l])
            +a6(gl1,gl2,gl3)*delt4[i,j,k,l])))))$
>>
        return
        end
c
c       a,b,c,a1-a6 are the coefficients we derived in code.
c
<<
        gentran(a(gl1,gl2,gl3):=block(type(function,a),
                                      type("real*8",gl1,gl2,gl3),
                                      type("real*8",a,s1,s2,s3),
                                      a:eval(ta)))$
>>
<<
        gentran(b(gl1,gl2,gl3):=block(type(function,b),
                                      type("real*8",b,gl1,gl2,gl3),
                                      type("real*8",s1,s2,s3),
                                      b:eval(tb)))$
>>
<<

        gentran(c(gl1,gl2,gl3):=block(type(function,c),
                                      type("real*8",c,gl1,gl2,gl3),
                                      type("real*8",s1,s2,s3),
                                      c:eval(tc)))$
>>
```

```
<<
        gentran(a1(gl1,gl2,gl3):=block(type(function,a1),
                                       type("real*8",a1,gl1,gl2,gl3),
                        type("real*8",s1,s2,s3,s11,s22,s33,s21,s32,s31),
                         a1:eval(a1)))$
>>
<<
        gentran(a2(gl1,gl2,gl3):=block(type(function,a2),
                        type("real*8",a2,gl1,gl2,gl3),
                        type("real*8",s1,s2,s3,s11,s22,s33,s21,s32,s31),
                        a2:eval(a2)))$
>>
<<
        gentran(a3(gl1,gl2,gl3):=block(type(function,a3),
                        type("real*8",a3,gl1,gl2,gl3),
                        type("real*8",s1,s2,s3,s11,s22,s33,s21,s32,s31),
                        a3:eval(a3)))$
>>
<<

        gentran(a4(gl1,gl2,gl3):=block(type(function,a4),
                        type("real*8",a4,gl1,gl2,gl3),
                        type("real*8",s1,s2,s3,s11,s22,s33,s21,s32,s31),
                        a4:eval(a4)))$
>>
<<
        gentran(a5(gl1,gl2,gl3):=block(type(function,a5),
                        type("real*8",a5,gl1,gl2,gl3),
                        type("real*8",s1,s2,s3,s11,s22,s33,s21,s32,s31),
                        a5:eval(a5)))$
>>
<<
        gentran(a6(gl1,gl2,gl3):=block(type(function,a6),
                        type("real*8",a6,gl1,gl2,gl3),
                        type("real*8",s1,s2,s3,s11,s22,s33,s21,s32,s31),
                        a6:eval(a6)))$
>>
```

```
c
c       The s1,s2,s3,s11,s22,s33,s21,s32,s31 are derivatives of W
c
        function s1(gl1,gl2,gl3)
        <<cut(var);>>
<<
        gentran(type(  "real*8",s1,gl1,gl2,gl3),
                       s1:2*eval(diff(w,'gl1,1)))$
>>
        return
        end
c
        function s2(gl1,gl2,gl3)
        <<cut(var);>>
<<
        gentran(type(   "real*8",s2,gl1,gl2,gl3),
                        s2:2*eval(diff(w,'gl2,1)))$
>>
        return
        end
c
        function s3(gl1,gl2,gl3)
        <<cut(var);>>
<<
        gentran(type(  "real*8",s3,gl1,gl2,gl3),
                       s3:2*eval(diff(w,'gl3,1)))$
>>
        return
        end
c
        function s11(gl1,gl2,gl3)
        <<cut(var);>>
<<
        gentran(type(  "real*8",s11,gl1,gl2,gl3),
                       s11:2*eval(diff(w,'gl1,2)))$
>>
        return
        end
```

```
c
      function s22(gl1,gl2,gl3)
      <<cut(var);>>
<<
      gentran(type("real*8",s22,gl1,gl2,gl3),
                    s22:2*eval(diff(w,'gl2,2)))$
>>
      return
      end
c
      function s33(gl1,gl2,gl3)
      <<cut(var);>>
<<
      gentran(type(  "real*8",s33,gl1,gl2,gl3),
                     s33:2*eval(diff(w,'gl3,2)))$

>>
      return
      end
c
      function s21(gl1,gl2,gl3)
      <<cut(var);>>
<<
      gentran(type(  "real*8",s21,gl1,gl2,gl3),
                     s21:2*eval(diff(w,'gl2,1,'gl1,1)))$
>>
      return
      end
c
      function s31(gl1,gl2,gl3)
      <<cut(var);>>
<<
      gentran(type(  "real*8",s31,gl1,gl2,gl3),
                     s31:2*eval(diff(w,'gl3,1,'gl1,1)))$
>>
      return
      end
```

```
c
      function s32(gl1,gl2,gl3)
      <<cut(var);>>
<<
      gentran(type( "real*8",s32,gl1,gl2,gl3),
                    s32:2*eval(diff(w,'gl3,1,'gl2,1)))$
>>
      return
      end
```

APPENDIX C: Template File Associated With COMPSD2
Valid For Double Coalesence Case

```
      real*8 gl1,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),q1(3,3,3,3),p(3,3,3,3),q(3,3,3,3)
      real*8 b1,b2,b3, abar,bbar
c
c     Computes second order tensor ts(i,j) based on the formula
c     derived in code.
c
<<
      gentran(for i:1 thru 3 do
       (for j:1 thru 3 do
        (ts[i,j]:abar(gl1,gl2)*cm[i,j]+bbar(gl1,gl2)*delt[i,j])))$
>>
c
c     Call subroutine to get P, Q which are defined in code.
c
      call pqcom(cm,cm,p1,q)
      call pqcom(cm,delt,p,q1)
      call pqcom(delt,cm,p,q2)
c
c     Computes tensor td(i,j,k,l).
c
<<
      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]:b1(gl1,gl2)*p1[i,j,k,l]+b2(gl1,gl2)*
          (q1[i,j,k,l]+q2[i,j,k,l])+b3(gl1,gl2)*delt4[i,j,k,l])))))$
>>
      return
      end
```

```
c
c       abar,bbar are b1, b2, b3 functions derived in code.
c
<<
        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,ss21),
                                   b1:eval(b1)))$
>>
<<
        gentran(b2(gl1,gl2):=block(type(function,b2),
                                   type("real*8",b2,gl1,gl2),
                                   type("real*8", ss1,ss2,ss11,ss22,ss21),
                                   b2:eval(b2)))$
>>
<<
        gentran(b3(gl1,gl2):=block(type(function,b3),
                                   type("real*8",b3,gl1,gl2),
                                   type("real*8", ss1,ss2,ss11,ss22,ss21),
                                   b3:eval(b3)))$
>>
<<
        neww:subst(['gl3='gl2],w)$
>>
```

```
c
c       ss1,ss2,ss11,ss22,ss21 are derivatives of W.
c
        function ss1(gl1,gl2)
        <<cut(var);>>
<<
        gentran(type("real*8",ss1,gl1,gl2),
                    ss1:2*eval(diff(neww,'gl1,1)))$
>>
        return
        end
c
        function ss2(gl1,gl2)
        <<cut(var);>>
<<
        gentran(type("real*8",ss2,gl1,gl2),
                    ss2:2*eval(diff(neww,'gl2,1)))$
>>
        return
        end
c
        function ss11(gl1,gl2)
        <<cut(var);>>
<<
        gentran(type("real*8",ss11,gl1,gl2),
                    ss11:2*eval(diff(neww,'gl1,2)))$
>>
        return
        end
c
        function ss21(gl1,gl2)
        <<cut(var);>>
<<
        gentran(type("real*8",ss21,gl1,gl2),
                ss21:2*eval(diff(neww,'gl2,1,'gl1,1)))$
>>
        return
        end
```

```
c
      function ss22(gl1,gl2)
      <<cut(var);>>
<<
      gentran(type("real*8",ss22,gl1,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(gl1)
      <<cut(var);>>
<<
      gentran(type("real*8",sss1,gl1),
        sss1:2*eval(diff(www,'gl1,1)))$
>>
      return
      end
c
      function sss11(gl1)
      <<cut(var);>>
<<
      gentran(type("real*8",sss11,gl1),
              sss11:2*eval(diff(www,'gl1,2)))$
>>
      return
      end
```

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (*Leave blank*)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	April 1993	Technical Memorandum

4. TITLE AND SUBTITLE

Explicit Robust Schemes for Implementation of General Principal Value-Based Constitutive Models

5. FUNDING NUMBERS

WU–510–01–50

6. AUTHOR(S)

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

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

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

8. PERFORMING ORGANIZATION REPORT NUMBER

E–7787

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

National Aeronautics and Space Administration
Washington, D.C. 20546–0001

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

NASA TM–106123

11. SUPPLEMENTARY NOTES

S.M. Arnold, NASA Lewis Research Center, Cleveland, Ohio; A.F. Saleeb, University of Akron, Department of Civil Engineering, Akron, Ohio 44325; and H.Q. Tan and Y. Zhang, University of Akron, Department of Mathematical Sciences, Akron, Ohio, 43325. Responsible person, S.M. Arnold, (216) 433–3334.

12a. DISTRIBUTION/AVAILABILITY STATEMENT

Unclassified - Unlimited
Subject Category ~~49~~ 59

12b. DISTRIBUTION CODE

13. ABSTRACT (*Maximum 200 words*)

The issue of developing effective and robust schemes to implement general hyperelastic constitutive models is addressed. To this end, special purpose functions are used to symbolically derive, evaluate, and automatically generate the associated FORTRAN code for the explicit forms of the corresponding stress function and material tangent stiffness tensors. These explicit forms are valid for the entire deformation range (i.e., with both distinct and repeated principal-stretch values). The analytical form of these explicit expressions is given here for the case in which the strain-energy potential W is taken as a *nonseparable* polynomial function of the principle stretches.

14. SUBJECT TERMS

Hyperelastic; Constitutive models; Symbolic computation; Principal value

15. NUMBER OF PAGES

29

16. PRICE CODE

A03

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	