Improvements on the 3-D RIPLE program: the implementation of friction and power in a forming simulation program

Citation for published version (APA):
Improvements on the 3-D RIPLE program

The implementation of friction and power in a forming simulation program

A.P. Vloemans
Research report

July 1993
WPA 1560

Guestresearch at the Mechanical Engineering Laboratory
Agency of Industrial Science and Technology
Ministry of International Trade and Industry
Tsukuba, Japan
Summary

RIPLE is a final element method simulation program using the matrix method. RIPLE is an abbreviation of RIgid PLastic deformation analysis code and it’s especially designed for large plastic deformation processes such as massive and thick plate forming.

In the 3-D RIPLE program the computation of friction and deformation power was not build in yet. So friction had to be implemented in two ways: According to Coulomb and according to Von Mises.

To build in friction (both Coulomb and Von Mises) a description for the relative average velocity of the element surface(s) dealing with friction was found and used in the formula describing the friction power. From this formula the first and second derivatives were taken and implemented in respectively the element load vector and the element stiffness matrix in the program.

Also other descriptions for the average velocity, the element load vector and the element stiffness matrix were implemented and testruns were done to compute the total deformation power. So also the computation of the power had to be build in. The values for the power were compared to the results of the 2-D RIPLE program simulating the same process (compressing a cylinder) and the analysis. This 2-D program is considered to be very reliable because it’s results are almost exactly the same as the analysis. After comparing the version with the best agreement was chosen.

In general the program has to deal with two kinds of forming processes: with constant and with changing boundary conditions. For processes with constant boundaries, it’s easier to implement friction in the program and testruns go faster. That’s why friction was first build in in the program for this kind of processes.

After this the implementation of friction was adapted to make it suitable for processes with changing boundary conditions in time. The result is a FEM program simulating all kinds of metal forming processes including friction and giving reliable outputs for the required power.
# Symbols

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Definition</th>
<th>Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>$Y_0$</td>
<td>initial flowstress</td>
<td>[N/mm²]</td>
</tr>
<tr>
<td>$K$</td>
<td>characteristic deformation resistance</td>
<td>[N/mm²]</td>
</tr>
<tr>
<td>$m$</td>
<td>strainrate hardening</td>
<td>[-]</td>
</tr>
<tr>
<td>$n$</td>
<td>strain hardening</td>
<td>[-]</td>
</tr>
<tr>
<td>$\Phi$</td>
<td>deformation power</td>
<td>[W]</td>
</tr>
<tr>
<td>$\sigma'$</td>
<td>deviatorial stress vector</td>
<td>[N/mm²]</td>
</tr>
<tr>
<td>$\sigma$</td>
<td>stress</td>
<td>[N/mm²]</td>
</tr>
<tr>
<td>$\varepsilon$</td>
<td>strain</td>
<td>[-]</td>
</tr>
<tr>
<td>$\dot{\varepsilon}$</td>
<td>strainrate</td>
<td>[1/s]</td>
</tr>
<tr>
<td>$\ddot{\varepsilon}$</td>
<td>strainratevector</td>
<td>[1/s]</td>
</tr>
<tr>
<td>$V$</td>
<td>volume</td>
<td>[mm³]</td>
</tr>
<tr>
<td>$\lambda$</td>
<td>mean stress</td>
<td>[N/mm²]</td>
</tr>
<tr>
<td>$C$</td>
<td>matrix notation of the Kronecker Delta</td>
<td>[-]</td>
</tr>
<tr>
<td>$S_T$</td>
<td>traction surface</td>
<td>[mm²]</td>
</tr>
<tr>
<td>$T$</td>
<td>traction vector</td>
<td>[N/mm²]</td>
</tr>
<tr>
<td>$U$</td>
<td>velocity vector field</td>
<td>[mm/s]</td>
</tr>
<tr>
<td>$M$</td>
<td>number of elements</td>
<td>[-]</td>
</tr>
<tr>
<td>$N$</td>
<td>number of nodepoints</td>
<td>[-]</td>
</tr>
<tr>
<td>$u$</td>
<td>velocity vector</td>
<td>[mm/s]</td>
</tr>
<tr>
<td>$Y$</td>
<td>yield stress</td>
<td>[N/mm²]</td>
</tr>
<tr>
<td>$P$</td>
<td>P-matrix</td>
<td>[mm·s]</td>
</tr>
<tr>
<td>$Q$</td>
<td>Q-matrix</td>
<td>[N·s]</td>
</tr>
<tr>
<td>$F$</td>
<td>F-matrix</td>
<td>[N]</td>
</tr>
<tr>
<td>$H$</td>
<td>H-matrix</td>
<td>[mm²]</td>
</tr>
<tr>
<td>$S$</td>
<td>stiffness matrix</td>
<td>[N·s/mm]</td>
</tr>
<tr>
<td>$R$</td>
<td>load vector</td>
<td>[N]</td>
</tr>
<tr>
<td>$\lambda_{(n)}$</td>
<td>mean stress vector</td>
<td>[N/mm²]</td>
</tr>
<tr>
<td>$\tau_{fr}$</td>
<td>friction stress</td>
<td>[N/mm²]</td>
</tr>
<tr>
<td>$m$</td>
<td>friction coefficient</td>
<td>[-]</td>
</tr>
<tr>
<td>$\mu$</td>
<td>friction coefficient</td>
<td>[-]</td>
</tr>
<tr>
<td>$\bar{\sigma}$</td>
<td>effective stress</td>
<td>[N/mm²]</td>
</tr>
</tbody>
</table>
\( \sigma_n \) : normal stress \([\text{N/mm}^2]\)

\( A \) : area of the element face under friction \([\text{mm}^2]\)

\( V \) : averaged relative velocity of the friction surface \([\text{mm/s}]\)

\( VX \) : averaged relative surface velocity in x-direction \([\text{mm/s}]\)

\( VY \) : averaged relative surface velocity in y-direction \([\text{mm/s}]\)

\( VZ \) : averaged relative surface velocity in z-direction \([\text{mm/s}]\)

\( U_{xx_i} \) : velocity of node i in x-direction \([\text{mm/s}]\)

\( U_{yy_i} \) : velocity of node i in y-direction \([\text{mm/s}]\)

\( U_{zz_i} \) : velocity of node i in z-direction \([\text{mm/s}]\)

\( U_p \) : punch velocity vector \([\text{mm/s}]\)

\( U_pX \) : punch velocity in x-direction \([\text{mm/s}]\)

\( U_pY \) : punch velocity in y-direction \([\text{mm/s}]\)

\( U_pZ \) : punch velocity in z-direction \([\text{mm/s}]\)

\( P \) : total power \([\text{W}]\)

\( P_{fr} \) : friction power \([\text{W}]\)

\( P_{def} \) : deformation power \([\text{W}]\)

\( VK1,..,VK4 \) : velocities of the 4 nodepoints of a surface \([\text{mm/s}]\)

\( R(t) \) : B-spline curve \([-]\)

\( N_{ip}(t) \) : B-spline basis functions \([-]\)

\( P_i \) : control points \([-]\)

\( T \) : knot vector \([-]\)

\( t_i \) : knots \([-]\)

\( S(u,w) \) : tensor produkt B-spline surface \([-]\)

\( M_{ip}(u) \) : B-spline basis functions \([-]\)

\( N_{j,q}(w) \) : B-spline basis functions \([-]\)

\( P_{ij} \) : control points \([-]\)

\( T_m,T_N \) : knot vectors \([-]\)

\( X \) : nodepoint coordinate vector \([\text{mm}]\)
<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>$S$</td>
<td>punch surface coordinate vector</td>
<td>[mm]</td>
</tr>
<tr>
<td>$D$</td>
<td>distance between $X_0$ and $S_0$</td>
<td>[mm]</td>
</tr>
<tr>
<td>$V_N$</td>
<td>nodepoint velocity vector</td>
<td>[mm/s]</td>
</tr>
<tr>
<td>$\Delta t$</td>
<td>timestep</td>
<td>[s]</td>
</tr>
<tr>
<td>$L$</td>
<td>transformation matrix</td>
<td>[-]</td>
</tr>
</tbody>
</table>
Preface

After some years of studying at the Eindhoven University of Technology I became more and more interested in doing a period of study in a foreign country. Professor Kals, my graduate professor and always supporting his students to spend some time of their study abroad, gave me the opportunity to do a traineeship in Japan at the Mechanical Engineering Laboratory in Tsukuba. The Plasticity and Forming Division in this laboratory handles in the same field as my graduate section at the TUE.
At that time some students of my section already proceeded me at the MEL and their enthusiasm made me very curious. Especially Japan always seemed very interesting to me because of its economical and technological development. But it also seemed always rather inaccessible, so when I got this chance to go there a decision was very easy to make.
The mechanical Engineering Laboratory is a governmental laboratory situated in Tsukuba "Science City", which is an amazing place because of the presence of many laboratories, doing research in all kinds of disciplines. Around 20.000 researchers work here, some 3.000 of them are from abroad.
So at the MEL I worked at the Plasticity and Forming Division for three months on an FEM program, called RIPLE. At the end I also did some experiments about square deepdrawing using sidetools. All this time I worked together with Mr. E. Sato, a PHD student working at the MEL.
I can say our co-operation was very useful, very pleasant and very instructive for me. I should like to thank him for everything he did for me during my stay in Japan.

On the other hand this period taught me a lot about Japan, the Japanese and the way they work and live. After all, these three months were very educative for me and I want to thank the director of the division, Dr. T. Sano, for his hospitality and giving me this opportunity to visit the MEL and I want to thank all the others who made my stay very pleasant.

Rob Vloemans
Contents

Summary 2

Symbols 3

Preface 6

1. Introduction 9
   1.1 The computer system at the MEL 9
   1.2 The principal of the RIPLE program 10
      1.2.1 The structure of RIPLE 10
      1.2.2 The theory 11
   2. Friction in RIPLE 13
      2.1 Introduction 13
      2.2 Von Mises and Coulomb 14
      2.3 Implementation of friction in RIPLE under constant boundary conditions 15
         2.3.1 The theory 15
         2.3.2 The average relative velocity 15
         2.3.3 The load vector 16
         2.3.4 The element stiffness matrix 17
   3. Power in RIPLE 19
      3.1 Introduction 19
      3.2 The volume of an element 20
   4. Choosing the best implementation of friction 22
      4.1 Results 22
      4.2 Conclusions 24
      4.3 Stress and strain distributions 26
   5. Friction under changing boundary conditions 32
      5.1 Introduction 32
      5.2 The description of the punch surface using B-splines 32
         5.2.1 Description of a curve 32
         5.2.2 Description of a surface 33
      5.3 Node touching and node separating from the tool 34
         5.3.1 The touching of the tool 34
5.3.2 The Newton Raphson iteration method in case of the die 35
5.3.3 The seperation from the tool 37

5.4 **Stress and strain distributions (without friction)** 39

5.5 **Implementation of friction under changing boundary conditions** 42
  5.5.1 How to compute the friction power 42
  5.5.2 Derivation of the normal stress 42
  5.5.3 Derivation of the transfermatrix 43

5.6 **Stress and strain distributions (with friction)** 44

References 49

Appendices A till D 50
1. Introduction

1.1. The computer system at the MEL

At the Agency of Industrial Science and Technology (AIST) in Tsukuba there is a computer centre called RIPS. RIPS stands for Research Information Processing System and to support research activities, the AIST research institutes in Tsukuba can use the facilities of this centre freely.

In the RIPS centre different (super)computersystems are available of which we mainly used the CRAY X-MP/216 supercomputer. From the MEL the SUN spark worksta-tion and several p.c. were connected with CRAY by means of a network called Ethernet. The operating system being applied is UNIX V. The relations within this network are showed in figure 1.

Figure 1: Relationships between the computersystems.
1.2 The principal of the RIPLE program

1.2.1 The structure of RIPLE

To make clear the structure of the RIPLE program, figure 2 gives the flowchart.

The information in the first four blocks are written down in the inputfile. The parameter card gives the kind of simulation, the number of stages, information about the convergence rate, whether friction is being used or not (if so, which kind of friction) and the definition of the initial velocity field.

In the model definition card the mesh of the workpiece is defined, the boundary conditions and the distribution of the load on the workpiece or the tool velocity.
are given. This card also contains the material properties and the friction factor if friction is used.
The other blocks are the steps which the program successively follows when it’s running. More about this will become clear in the next paragraph.

1.2.2. The theory

RIPLE is an abbreviation of RIigid PLastic analysis code. It’s developed at the MEL and it’s written in Fortran.77. It can be used to simulate all kinds of forming processes, like forging, compression, deepdrawing, bulging, etc. It uses the formula for rigid plastic deformation:

\[ \sigma = Y_0 + K \cdot \varepsilon^m \cdot \varepsilon^n \]  \hspace{1cm} (1)

So RIPLE is especially useful for simulations of processes with large plastic deformation.

Under the condition that the entire rigid-plastic body is deformed plastically, the next energy-equation is used in the RIPLE program, which is an FEM program using the general matrix-method, developed by Lee and Kobayashi [1], see also [2]. This method is based on one of the variational principles for rigid plastic deformation, and is particular suited for problems involving large plastic deformation.

\[ \Phi = \int_{V} \sigma \cdot \dot{\varepsilon} dV + \int_{V} \lambda C \cdot \dot{\varepsilon} dV - \int_{S_T} T^T U dS \]  \hspace{1cm} (2)

The first integral describes the energy dissipated in the body. \( \sigma \) is the deviatorial stress vector. The strainrate vector is derived from the velocity vector field \( U \).

In the second integral, \( \lambda \) is the Lagrange multiplier which is identified as the mean stress. \( C \) is the proper matrix vector notation of the Kronecker delta. This matrix multiplied with the strainrate implies the incompressability condition.

The third integral describes the attraction between the surfaces of the workpiece and the tool(s). \( T \) is the traction vector, specified on the boundary \( S_T \).

The body, \( V \), is devided into \( M \) elements and these elements are connected together with \( N \) nodepoints.

When the proper equations for the stress and the strainrate are substituted into equation (1), the requirement of this equation to be stationary (\( \delta \Phi = 0 \)) leads to a nonlinear stiffness relation for each of the \( M \) elements.

In this case, the three dimensional version of the RIPLE-program is of interest. The program is designed for using 8-node elements. So the nonlinear relationship for every
element represents a system of 24 equations with 25 unknowns: u and λ, where u is the vector containing the velocities in x, y and z direction of each nodal point associated with the element.

To make the system solvable, a 25th equation must be added. This is the incompressibility condition.

To linearize and solve the nonlinear stiffness equation for each element, the Newton-Raphson iteration method can be used: consider a small perturbation $Δu(n)$ of the velocity vector $u(n)$ for the nth iteration step, such that $u(n)=u(n-1)+Δu(n)$. Now, the iteration is repeated until the velocity field becomes satisfactory. This means that $Δu/u$ is equal or smaller than a desired value. This value can be given in the input file of the program.

Now the element perturbation matrix equation has become as follow:

$$\begin{bmatrix} YP_{(n-1)} & Q \\ Q & 0 \end{bmatrix} \begin{bmatrix} Y(n-1) \\ λ_{(n)} \end{bmatrix} = \begin{bmatrix} F \\ Q^T u_{(n-1)} \end{bmatrix}$$

In this equation, the first matrix left of the equation mark is the element stiffness matrix and both matrices on the right side make the load vector. Each iteration these matrices are updated according to the new velocity field.

Equation (3) can be written like:

$$S_{(n-1)}[\frac{Δu}{λ}]_{(n)} = R_{(n-1)}$$

where m means "element number".

All computed element stiffness matrices and load vectors for each element are assembled after each iteration to make respectively the total stiffness matrix and the total load vector:

$$S_{(n-1)}[\frac{Δu}{λ}]_{(n)} = R_{(n-1)}$$

where:

$$S_{(n)} = \sum_{m=1}^{M} S^{(m)}_{(n)}$$

$$R_{(n)} = \sum_{m=1}^{M} R^{(m)}_{(n)}$$

$$Δu_{(n)} = \sum_{m=1}^{M} Δu^{(m)}_{(n)}$$

$$λ_{(n)} = \sum_{m=1}^{M} \frac{\lambda^{(m)}_{(n)}}{Y^{(m)}_{(n)}}$$
2. Friction in RIPLE

2.1. Introduction

There is a two-dimensional and a three-dimensional version of the RIPLE program. Both are derived from the principle mentioned in §1.2.2. In the two-dimensional program friction is already build in and works satisfactory. In the three-dimensional version only a beginning is made for the subroutine about friction, but it is not operational. So friction has to be build in in a reliable way. An easy way to see if there is any agreement with the two-dimensional program is doing a simulation of cylinder compression with a flat punch, see figure 3. Here also the power is computed, see chapter 4. First without and later, after friction was build in, with friction. This cylinder has the same size and the same material properties as the one being used in the simulation with the two dimensional version of RIPLE, see [3]. Here, the simulation was verified analytically and there was a good agreement. For the mesh, see figure 4. Because of symmetry, only the deformation of 1/8 of the cylinder was calculated.

![Figure 3: The compression of a cylinder with a flat punch.](image1)

![Figure 4: The mesh of 1/8 of the cylinder.](image2)

The radius is 100 mm and the height is 200 mm. The punch velocity is -4 mm/s. The mesh was designed in such a way that not too much node points and elements were defined, so that the computation time would be restricted. This is because of the foreseeing that many test runs would be done. Anyway we can say that the mesh is fine enough to get reliable results. In appendix A the input file of the simulation is given.
A simulation of cylinder compressing with a flat punch is easy, because the boundary conditions of the node points touching the punch don't change during the process. If they change, like in case of deep drawing or cylinder compressing a with a rounded punch, implementation of friction, especially of Coulomb friction, is more complicated, see chapter 5.

2.2 Von Mises and Coulomb

In the three-dimensional program, two kinds of friction had to be build in:

Von Mises: \[ \tau_{fr} = \frac{m \cdot \sigma_f}{\sqrt{3}} \]  \( m \cdot \sigma_f = \frac{m \cdot \sigma}{\sqrt{3}} \)  \( \sqrt{3} \)  \( \sqrt{3} \)

Coulomb: \[ \tau_{fr} = \mu \cdot \sigma_n \]

At the Von Mises method, the flow stress \( \sigma_f \) is being used. This is equal to the effective stress, which is computed by RIPLE in subroutine ESCALL for each element (see the listing of the program, WPA report nr.1561).

It is more complicated to implement Coulomb friction, when boundary conditions change during the process. For each element it is easy to compute the stresses in X-, Y- and Z-direction. When the normal vector of each surface which has to deal with friction is parallel to one of the axis of the coordinate system - and keeps during deformation - Coulomb friction is easy to implement into the program. Otherwise, first the normal stresses at the node points of the friction surface has to be defined. This can be done after transferring the X-, Y- and Z-stresses from the global (X,Y,Z) coordinate system into the local coordinate system. The derivation of the transfer matrix is described in §5.5.3.

For cylinder compression with a flat punch, see figure 3, Coulomb friction can easily be implemented, because the orientation of the friction surfaces doesn't change. So the normal stress is equal to the element stress in Z-direction, any time during the process.

Now friction can be build in by supplement both the element stiffness matrix and the load vector, see §2.3.
2.3 Implementation of friction in RIPLE under constant boundary conditions

2.3.1. The theory

When the theory is correctly followed, friction has to be build in by adapting the load vector and the element stiffness matrix. For the load vector this must be done by taking the first derivative of the element friction power to the velocity in X, Y or Z-direction and subtract this to the components of the element load vector. The element stiffness matrix is adapted by adding the second derivatives of the element friction power to the velocities in X, Y and Z-direction to the involved components of the matrix. The formula for the frictionpower of one particular elementsurface is given as follow:

\[ P_{fr} = \tau_{fr} \cdot A \cdot V \]  

(12)

where:

- \( A \) = area of the element face under friction
- \( V \) = average relative* velocity of the element frictionsurface.

*The friction power is caused by the relative velocity between element surface and tool.

2.3.2. The average relative velocity

One of the questions that arise was how to define the averaged relative velocity (V). In the present program a suggestion was done, but that didn’t seem very realistic. So a new suggestion had to be done, see also figure 5:

![Figure 5: Velocities in X-, Y- and Z-direction of the nodepoints of a surface.](image-url)
In case of cylinder compression with a flat punch, the punch has no velocity in X- and Y-direction \((U_pX=U_pY=0)\), so we can say:

Average relative velocity in X-direction:

\[
V_X = \frac{1}{4} \sum_{i=1}^{4} (U_{x_i} - U_{pX}) = \frac{1}{4} \sum_{i=1}^{4} U_{x_i},
\]

(13)

Average relative velocity in Y-direction:

\[
V_Y = \frac{1}{4} \sum_{i=1}^{4} (U_{y_i} - U_{pY}) = \frac{1}{4} \sum_{i=1}^{4} U_{y_i},
\]

(14)

The velocities of punch and nodepoints in Z-direction are, of course, equal \((U_{xxi}=U_{pX})\), otherwise the contact wouldn't be guaranteed. So the average relative velocity in Z-direction is zero:

\[
V_Z = \frac{1}{4} \sum_{i=1}^{4} (U_{z_i} - U_{pZ}) = 0
\]

(15)

Average relative velocity of the friction surface:

\[
V = \sqrt{(V_X)^2 + (V_Y)^2}
\]

(16)

\(V_X\), \(V_Y\) and \(V_Z\) (=0) are the three orthogonal components of the vector, which forms the average velocity of the surface in X-, Y- and Z-direction.

So, \(V\) is the measure of this vector.

### 2.3.3 The load vector

Now, following the theory, the components of the element loadvector (24 components), dealing with friction, have to be decreased by the first derivatives of the surface friction power to the velocities in X-, Y- and Z-directions:

For the \(X_i\) components \((i=1,\ldots,4)\):

\[
\frac{\delta P_{fr}}{\delta U_{xx_i}} = \tau_{fr} A \frac{V_X}{4V}
\]

(17)

For the \(Y_i\) components \((i=1,\ldots,4)\):

\[
\frac{\delta P_{fr}}{\delta U_{yy_i}} = \tau_{fr} A \frac{V_Y}{4V}
\]

(18)
For the $Z_i$ components ($i=1,\ldots,4$):
\[
\frac{\delta P_{fr}}{\delta U_{zz_i}} = 0 \quad (19)
\]

### 2.3.4 The element stiffness matrix

The involved components of the element stiffness matrix (25*25 components) are increased by the second derivatives of the surface friction power to the velocities in $X$-, $Y$- and $Z$-directions:

For the $(X_i, X_i)$ and $(Y_i, Y_i)$ components ($i=1,\ldots,4$):
\[
\frac{\delta^2 P_{fr}}{\delta U^2_{xx_i}} = \frac{\delta^2 P_{fr}}{\delta U^2_{yy_i}} = \frac{\tau_{fr} \cdot A}{16V} \quad (20)
\]

For the $(Z_i, Z_i)$ components ($i=1,\ldots,4$):
\[
\frac{\delta^2 P_{fr}}{\delta U^2_{zz_i}} = 0 \quad (21)
\]

For the $(X_i, X_j)$ components ($i,j=1,\ldots,4$):
\[
\frac{\delta^2 P_{fr}}{\delta U^2_{xx_i}} = -\frac{\tau_{fr} \cdot A \cdot VX \cdot VX}{256V^3} \quad (22)
\]

For the $(Y_i, Y_j)$ components ($i,j=1,\ldots,4$):
\[
\frac{\delta^2 P_{fr}}{\delta U^2_{yy_i}} = -\frac{\tau_{fr} \cdot A \cdot VY \cdot VY}{256V^3} \quad (23)
\]

For the $(Z_i, Z_j)$ components ($i,j=1,\ldots,4$):
\[
\frac{\delta^2 P_{fr}}{\delta U^2_{zz_i}} = 0 \quad (24)
\]

For the $(X_i, Y_j)$ and $(Y_i, X_j)$ components ($i,j=1,\ldots,4$):
\[
\frac{\delta^2 P_{fr}}{\delta U_{xx_i}} \frac{\delta^2 P_{fr}}{\delta U_{yy_i}} = -\frac{\tau_{fr} \cdot A \cdot VX \cdot VY}{256V^3} \quad (25)
\]
For the \((y, z)\) and \((z, y)\) components \((i,j=1,\ldots,4)\):

\[
\frac{\delta^2 P_{fr}}{\delta U_{yy} \delta U_{zz}} = 0
\]  

(26)

For the \((z, x)\) and \((x, z)\) components \((i,j=1,\ldots,4)\):

\[
\frac{\delta^2 P_{fr}}{\delta U_{zz} \delta U_{xx}} = 0
\]  

(27)

In §4.1 the total power for the compression of a cylinder is computed by RIPLE using both ways of friction (Coulomb and Von Mises). The results are compared to the results of the 2-dimensional RIPLE program, which is considered to be reliable because of the agreement with the analysis.

In the program, we also tried the real average velocity of the element frictionsurface to compute the frictionpower. This is done because of the expected bad convergence when the relative average velocity (which is small) is used. This has of course consequences for the adaption of the element load vector and the element stiffness matrix (see appendix A). For the results, see chapter 4, where we also tried other averaged velocities and element load vectors and element stiffness matrices.
3. Power in RIPLE.

3.1 Introduction

The power always gives useful information about the proces concerned. Comparing the simulation power with the experiment power, it gives a good indication about the reliability of the program. If so, it gives useful information about the press being required for a certain proces.

So the subroutine POWER was implemented in the RIPLE program in such a way that after each stage the total power necessary to deform the body was computed and written on screen.

The power for one element can be devided in the element deformation power and, if appropriate, the element friction power:

\[ P = P_{\text{def}} + P_{\text{fr}} \tag{28} \]

where:

\[ P_{\text{def}} = \bar{\sigma} \cdot \bar{\varepsilon} \cdot \text{VOL} \tag{29} \]

where: VOL=volume of the element, see § 3.2.

and:

\[ P_{\text{fr}} = \sum_{m=1}^{M} \tau_{fr} \cdot A \cdot V \tag{30} \]

where: M=number of surfaces dealing with friction

Now, for each element the power is computed and after that, all the element powers are summized to get the total power necessary to deform the whole body.

The effective stress and strainrate in (29) are already calculated by RIPLE. The element volume was not calculated yet. So in §3.2 we tried to find a way to compute the element volume.
3.2 The volume of an element

To compute the volume of an eight node element, the element was split up into five tetrahedrons, see figure 6.

For a tetrahedron, according to [4], the volume is defined by taking the absolute value of the matrix (31), containing the X-, Y- and Z-distances between the node-points on the basis (L,M,N) and the nodepoint on the top (K), see figure 7.
The absolute value of the matrix is taken as follows:

\[
\begin{bmatrix}
XLK & XMK & XNK \\
\frac{1}{6} & YLK & YMK & YNK \\
& ZLK & ZMK & ZNK
\end{bmatrix}
\]

Here:

\[XLK = X_L - X_K\] (X-distance between L and K)
\[XMK = X_M - X_K\] (X-distance between M and K)
Etc.

\[DMN = YMK*ZNK - ZMK*YNK\]
\[DLN = YLK*ZNK - ZLK*YNK\]
\[DLM = YLK*ZMK - ZLK*YMK\]

\[VOLUME = (ABS(0.166667*(XLK*DMN-XMK*DLN+XNK*DLM)))\] (32)

The volume of the element is taken by summizing the volumes of the five tetrahedrons. Now, the element deformation power can be calculated.

For the implementation in the program, see subroutine POWER, appendix C.
4. Choosing the best implementation of friction

4.1 Results

To find the best implementation of friction, not only the theory is followed, as described in §2.3, but also other ways were tested. As said, testing is done by simulating cylinder compression with a flat punch. In the first place, the accuracy is of interest. This is defined by comparing the power output of the particular 3D RIPLE-version with the 2D RIPLE results and the analysis.

On the second place the number of iterations for each stage are of interest, because this defines the computing time.

To find out what is the best implementation, four ways of friction implementation were investigated.

For each version the total power (deformation- and friction power) was computed after 6 stages (=3 sec.), both using Coulomb and Von Mises friction, see figure 8.

As you can see, also the deformation power for cylinder compressing without friction is computed. This is done to check the accuracy of the 3D RIPLE program without using friction. Also the number of iterations of each stage for each version was investigated. This was
done for both Coulomb and Von Mises friction, see figure 10 and 11.

For the details about the four different ways of implementation, see appendix B1, B2, B3 and B4 which give each the subroutines ELSETF (computing the average velocity) and FRIC (adapting the element stiffness matrix and the load vector). Here, some comment is given:

*fem3a.f: The theory mentioned in §2.3 is followed exactly, but the average velocity is changed into the real average velocity (including the velocity in Z-direction) of the surface, not the relative average velocity between tool and workpiece. This was done because of better convergence when the average velocity has a big value. See appendix B1.

*fem3b.f: Here, the average velocity (not the relative) was used, which was proposed in the present program:

\[ V = \sqrt{(VK1)^2 + (VK2)^2 + (VK3)^2 + (VK4)^2} \]  

where:

\[ VK1 = \sqrt{U_{xx1}^2 + U_{yy1}^2 + U_{zz1}^2} \]  
\[ VK2 = \sqrt{U_{xx2}^2 + U_{yy2}^2 + U_{zz2}^2} \]  
\[ VK3 = \sqrt{U_{xx3}^2 + U_{yy3}^2 + U_{zz3}^2} \]  
\[ VK4 = \sqrt{U_{xx4}^2 + U_{yy4}^2 + U_{zz4}^2} \]

See figure 9.

The element load vector and the element stiffness matrix were adapted according to this average velocity and its consequences for the friction power. See appendix B2.

Figure 9: Velocities in X-, Y- and Z-direction.
*fem3c.f:* Here, the present element load vector and stiffness matrix were used (so the same as in fem3b.f), but the average velocity was changed into the one used in fem3a.f. See appendix B3.

*fem3d.f:* The theory of §2.3 is followed and the average relative velocity between the element surface under friction and the punch is used. Now we have the proper value of the velocity which cause the friction power. A slow convergence, but a very reliable output for the power is expected. See appendix B4.

### 4.2 Conclusions

* As expected, fem3d.f agrees most with fem2c.f and the analysis, because the theory is strictly followed (like the fem2c.f program and, of course, the analysis) and the proper average velocity is defined. The values are a little bit lower than the fem2c.f results and the analysis. This has to be explained by the rough mesh, so the volume is less than that of a perfect cylinder (like fem2c.f), and so is the surface of the up- and downside of the cylinder.

* Fem3a.f and fem3c.f give results very close to each other, despite of their different element load vector and stiffness matrix. But the averaged velocity is defined in the same way. The values of the results are higher than those of the fem3d.f program. Probably because of the higher value for the averaged velocity. So we can conclude that the average velocity has a big influence on the power results.

* Fem3b.f shows results which are definitely too high. This is caused by the wrong present average velocity, which is much too high.
Improvements on the 3-D RIPLE Program

number of iterations for "Coulomb" friction

![Graph showing iteration numbers for Coulomb friction](image1)

Figure 10: Iteration numbers for Coulomb friction.

number of iterations for "Von Mises" friction

![Graph showing iteration numbers for Von Mises friction](image2)

Figure 11: Iteration numbers for Von Mises friction.
4.3 Stress and strain distributions

Here some plots are given showing the output of the simulation of cylinder compression with a flat punch after 8 stages (=4 seconds). Here, fem3d.f is used. Both Coulomb and Von Mises frictions are used. The plots show the stress and the strain distributions. Also the nodepoints movements are shown.
The stress distribution of cylinder compression with a flat punch, using Coulomb friction after 8 stages (=4 seconds).
The strain distribution of cylinder compression with a flat punch, using Coulomb friction after 8 stages (=4 seconds).
The stress distribution of cylinder compression with a flat punch, using Von Mises friction after 8 stages (=4 seconds).
The strain distribution of cylinder compression with a flat punch, using Von Mises friction after 8 stages (=4 seconds).
The nodepoints movements of the FEM mesh of cylinder compression under Von Mises friction after 10 stages (=5 seconds).
5. Friction under changing boundary conditions

5.1 Introduction

So far, we only investigated a process at which all the boundaries for the velocities could be given before in the data file: cylinder compression with a flat punch. That means that all the node points involved follow these boundary conditions from the beginning of the process till the end, when the last stage is finished. So these node points are from the beginning in contact with the tool(s).

In reality this is not always the case. For example, at deep drawing all the boundaries of the node points cannot be given before in a data file, because for certain nodes the boundaries change during the process.

So the RIPLE program is designed in such a way that during the process boundary conditions are given to the node points as soon as they touch the surface of a tool (punch or die). To understand how this works it is necessary to know how toolsurfaces are described in RIPLE, see §5.2 and §5.3.

So for these kinds of processes the program is looking at the element node points instead of the element surfaces and that has consequences for the implementation of friction and power, see §5.5. A reliable way of implementation was investigated and checked by simulating the compression of a cylinder with a rounded punch. Here the boundaries change during the process, because from the beginning till the end, more and more node points of the uppersurface will touch the punch.

5.2 The description of the punch surface using B-splines

In RIPLE nonuniform rational B-splines (NURB) are used to describe the surfaces of the tools. For more details, see [5], [6], [7] and [8]. First the description of a curve is given.

5.2.1 Description of a curve

In general a curve can be described by using the B-spline method:

\[ R(t) = \sum_{i=0}^{n} N_{i,p}(t)P_i \]  

(38)

where \( t \) is a parameter and \( N_{i,p}(t) \) is a B-spline basis function. Let \( T = \{ t_0, \ldots, t_i, \ldots, t_m \} \) be a non-decreasing sequence of real numbers. \( T \) is called the knot vector and the
Improvements on the 3-D RIPLE Program

\( t_i \) values are the knots. \( P_i \) are the control points. The number of control points are related by the formula: \((p+1) + (n+1) = m+1\). The \( i^{th} \) normalized B-spline function of degree \( p \) (order \( p+1 \)) is defined as follows:

\[
N_{i,0}(t) = \begin{cases} 
1 & \text{if } t_i \leq t < t_{i+1} , \ t_i \leq t_{i+1} \\
0 & \text{otherwise} 
\end{cases}
\] (39)

\[
N_{i,p}(t) = \frac{t-t_i}{t_{i+p}-t_i} N_{i,p-1}(t) + \frac{t_{i+p+1}-t}{t_{i+p+1}-t_{i+1}} N_{i+1,p-1}(t) 
\] (40)

The \( N_{i,p}(t) \) functions are defined on the entire real line, but the focus is on the interval \( t \in [t_0,t_m] \). Note that \( N_{i,p}(t) \) is a \( p^{th} \) degree piecewise polynomial vector.

5.2.2 Description of a surface

In RIPLE the surface geometry of the tools is described by the tensor produkt B-spline surface.

Whereas a curve requires one parameter for its definition (\( t \)), a surface requires two: \( u \) and \( w \). A degree \( (p,q) \) tensor produkt B-spline surface has the form:

\[
S(u,w) = \sum_{i=0}^{m} \sum_{j=0}^{n} M_{i,p}(u) N_{j,q}(w) P_{ij} 
\] (41)

The control points, \( P_{ij} \), are arranged in a topological rectangular array called the control net. The \( M_{i,p}(u) \) and \( N_{j,q}(w) \) are the univariate B-spline basis functions defined by equations 39-40. Cubic B-spline surfaces are employed to describe the surfaces of the tools. Also open uniform knot vectors, \( T_M=T_N=\{0,0,0,0,1,2,3,4,4,4,4,4\} \), are used in both \( M_{i,p}(u) \) and \( N_{j,q}(w) \). Forty-nine sets control points, \( P_{ij} \) (\( i,j=1,7 \)), are used to describe the surfaces of the tools. For square deepdrawing, figure 12 gives the punch and die geometry expressed by tensor product B-spline surfaces.
5.3 Node touching and node separating from tool

5.3.1 The touching of the tool

Now the tool surface is modeled we can compute when and where a nodepoint will touch a tool.

Consider a nodepoint $X_0$ that is willing to touch a tool surface in one of the following stages. The distance $D_0$ between the intersection point on the corresponding surface $S_o(u,w)$ and $X_0$ is expressed as follows:

$$D_0 = |S_o(u,w)-(X_0+V_N\cdot\Delta t)|$$

(42)

$S_o(u,w)$ consists of an X-, Y- and Z-coordinate. These coordinates depend on the value for $u$ and $w$ (both unknown). $X_0$, also consisting of an X-, Y- and Z-coordinate, is the initial nodepoint position (known). $V_N$ (known) is the velocity of the nodepoint $X_0$ and $\Delta t$ is the timestep (unknown) necessary to reach the tool surface.

Now for each nodepoint which will touch the tool the timestep $\Delta t$ is calculated and the smallest will be compared with the incremental default timestep in RIPLE. If the smallest timestep is smaller than the default timestep, this smallest timestep will be used in the current step, otherwise the node will penetrate the tool. From this moment the node touches a tool it gets its boundary condition: it gets a tangential velocity.

For the die, $S_o(u,w)$, $X_0$ and $V_N$ in equation 42 can be written as follows:
Improvements on the 3-D RIPLE Program

\[ S_0(u,w) = \begin{bmatrix} x(u,w) \\ y(u,w) \\ z(u,w) \end{bmatrix} \quad X_0 = \begin{bmatrix} x_N \\ y_N \\ z_N \end{bmatrix} \quad V_N = \begin{bmatrix} v_x_N \\ v_y_N \\ v_z_N \end{bmatrix} \] (43) (44) (45)

For the punch, which is moving in Z-direction, \( S_0(u,w) \), \( X_0 \) and \( V_N \) have to be written as follows. See also figure 13.

\[ \begin{align*}
X_0 &= \begin{bmatrix} x_N \\ y_N \\ z_N \end{bmatrix} \\
S_0(u,w) &= \begin{bmatrix} x(u,w) \\ y(u,w) \\ z(u,w) + v_z \cdot \Delta t \end{bmatrix} \\
V_N &= \begin{bmatrix} v_x_N \\ v_y_N \\ v_z_N \end{bmatrix}
\] (46) (47) (48)

In the datafile the geometry of the tool(s) is (are) described. In appendix D the datafile is given for cylinder compression with a rounded punch.

To compute the proper values for \( u \), \( w \) and \( \Delta t \), the Newton Raphson iteration method is used, see §5.3.2.

5.3.2 The Newton Raphson iteration method in case of the die

Equation (42) can be solved by the Newton Raphson iteration method with an initial guess for the solution as \( \Delta t = 0 \), \( u = u^0 \) and \( w = w^0 \) when \( u^0 \) and \( w^0 \) is a point on the surface net.

In case of the die, the procedure of the Newton Raphson method is explained as follows: When (43), (44) and (45) are substituted in equation (42), we get:
Improvements on the 3-D RIPLE Program

\[ D_0 = \sqrt{(x(u,w) - x_N - vx_N \Delta t)^2 + (y(u,w) - y_N - vy_N \Delta t)^2 + (z(u,w) - z_N - vz_N \Delta t)^2} \]  

(49)

\[ \phi = D_0^2 = (x(u,w) - x_N - vx_N \Delta t)^2 + (y(u,w) - y_N - vy_N \Delta t)^2 + (z(u,w) - z_N - vz_N \Delta t)^2 \]  

(50)

For the Newton Raphson iteration method, the first and second derivatives of \( \phi \) have to be taken:

\[ \frac{\partial \phi}{\partial u} = 2(x(u,w) - x_N - vx_N \Delta t) \frac{\partial x(u,w)}{\partial u} + 2(y(u,w) - y_N - vy_N \Delta t) \frac{\partial y(u,w)}{\partial u} + 2(z(u,w) - z_N - vz_N \Delta t) \frac{\partial z(u,w)}{\partial u} \]  

(51)

\[ \frac{\partial \phi}{\partial w} = 2(x(u,w) - x_N - vx_N \Delta t) \frac{\partial x(u,w)}{\partial w} + 2(y(u,w) - y_N - vy_N \Delta t) \frac{\partial y(u,w)}{\partial w} + 2(z(u,w) - z_N - vz_N \Delta t) \frac{\partial z(u,w)}{\partial w} \]  

(52)

\[ \frac{\partial \phi}{\partial \Delta t} = -2vx_N(x(u,w) - x_N - vx_N \Delta t) - 2vy_N(y(u,w) - y_N - vy_N \Delta t) - 2vz_N(z(u,w) - z_N - vz_N \Delta t) \]  

(53)

\[ \frac{\partial^2 \phi}{\partial u^2} = 2\left( \frac{\partial x(u,w)}{\partial u} \right)^2 + 2(x(u,w) - x_N - vx_N \Delta t) \frac{\partial^2 x(u,w)}{\partial u^2} \]  

(54)

\[ \frac{\partial^2 \phi}{\partial w^2} = 2\left( \frac{\partial x(u,w)}{\partial w} \right)^2 + 2(y(u,w) - y_N - vy_N \Delta t) \frac{\partial^2 y(u,w)}{\partial w^2} \]  

(55)

\[ \frac{\partial^2 \phi}{\partial \Delta t^2} = 2(vx_N^2 + vy_N^2 + vz_N^2) \]  

(56)
For the k\textsuperscript{th} iteration step, $u_k$, $w_k$ and $\Delta t_k$ are computed as follows:
($\alpha$ is the Newton Raphson acceleration factor, to make a quicker convergence)

\begin{align}
    u_k &= u_{k-1} - \alpha \frac{\partial \phi}{\partial u} \\
    w_k &= w_{k-1} - \alpha \frac{\partial \phi}{\partial w} \\
    \Delta t_k &= \Delta t_{k-1} - \alpha \frac{\partial \Delta t}{\partial \Delta t} \frac{\partial \phi}{\partial \Delta t}
\end{align}

(57)\hspace{1cm}(58)\hspace{1cm}(59)

Now, for each iteration step (k) we can compute the value for $DNORM1$, see equation (54). The iterations of the Newton Raphson method will continue until $DNORM1$ becomes lower than 0.00001.

$$DNORM1 = \sqrt{\left( \frac{\partial \phi}{\partial u} \right)^2 + \left( \frac{\partial \phi}{\partial w} \right)^2 + \left( \frac{\partial \phi}{\partial \Delta t} \right)^2 \left( u^2 + w^2 + \Delta t^2 \right)}$$

(60)

For the punch, of course, the same procedure is followed, but here the tool velocity in Z-direction has to be taken into account. This will not be described here, because it’s almost similar as in case of the die.

5.3.3 The separation from the tool

A nodepoint $X_i$ on the surface is allowed to move only along tangential directions, and may be off the surface in the succeeding step. Therefore, only if there is a tensile contact normal force $X_i$ separate from the surface of the tool. Otherwise $X_i$ is assigned to the closest point $S_i(u, w)$ on the surface. The closest point $S_i(u, w)$ on the surface is found by minimizing the distance between $D_i$ between $X_i$ and $S_i(u, w)$:

$$D_i = |S_i(u, w) - X_i|$$

(61)

This minimization is done by the Newton Raphson iteration method, with as an initial guess for the solution $(u, w)$ the previous step’s $u'$ and $w'$.

$X_i$ and $S_i(u, w)$ are:
When (62) and (63) are substituted in (61), we get:

\[ D_1 = \sqrt{(x(u,w) - x_N)^2 + (y(u,w) - y_N)^2 + (z(u,w) - z_N)^2} \]  

(64)

\[ \phi = D_1^2 = (x(u,w) - x_N)^2 + (y(u,w) - y_N)^2 + (z(u,w) - z_N)^2 \]  

(65)

To employ the Newton Raphson iteration method, the first and second derivatives of \( \phi \) have to be taken:

\[
\frac{\partial \phi}{\partial u} = 2(x(u,w) - x_N) \frac{\partial x(u,w)}{\partial u} + 2(y(u,w) - y_N) \frac{\partial y(u,w)}{\partial u} + 2(z(u,w) - z_N) \frac{\partial z(u,w)}{\partial u} 
\]

(66)

\[
\frac{\partial \phi}{\partial w} = 2(x(u,w) - x_N) \frac{\partial x(u,w)}{\partial w} + 2(y(u,w) - y_N) \frac{\partial y(u,w)}{\partial w} + 2(z(u,w) - z_N) \frac{\partial z(u,w)}{\partial w} 
\]

(67)

\[
\frac{\partial^2 \phi}{\partial u^2} = 2 \left( \frac{\partial x(u,w)}{\partial u} \right)^2 + 2(x(u,w) - x_N) \frac{\partial^2 x(u,w)}{\partial u^2} + 2 \left( \frac{\partial y(u,w)}{\partial u} \right)^2 + 2(y(u,w) - y_N) \frac{\partial^2 y(u,w)}{\partial u^2} + 2 \left( \frac{\partial z(u,w)}{\partial u} \right)^2 + 2(z(u,w) - z_N) \frac{\partial^2 z(u,w)}{\partial u^2} 
\]

(68)

\[
\frac{\partial^2 \phi}{\partial w^2} = 2 \left( \frac{\partial x(u,w)}{\partial w} \right)^2 + 2(x(u,w) - x_N) \frac{\partial^2 x(u,w)}{\partial w^2} + 2 \left( \frac{\partial y(u,w)}{\partial w} \right)^2 + 2(y(u,w) - y_N) \frac{\partial^2 y(u,w)}{\partial w^2} + 2 \left( \frac{\partial z(u,w)}{\partial w} \right)^2 + 2(z(u,w) - z_N) \frac{\partial^2 z(u,w)}{\partial w^2} 
\]

(69)

Also here, for the \( k \)th iteration step \( u_k \) and \( w_k \) are computed as follows:

(\( \alpha \) is the Newton Raphson acceleration factor, to make a quicker convergence)
\[ u_k = u_{k-1} - \alpha \frac{\partial \phi}{\partial u} \]  

(70)

\[ w_k = w_{k-1} - \alpha \frac{\partial \phi}{\partial w} \]  

(71)

Now, for each iteration step (k) we can compute the value for DNORM2, see equation (66). The iterations of the Newton Raphson method will continue until DNORM2 becomes lower than 0.00001.

\[
DNORM2 = \sqrt{\left(\frac{\partial \phi}{\partial u}\right)^2 + \left(\frac{\partial \phi}{\partial w}\right)^2}
\]

(72)

### 5.4 Stress and strain distributions (without friction)

The following plots show the stress and strain distributions of the simulation of cylinder compression with a rounded punch after 6 stages (=3 seconds). These are all without friction.
The stress distribution of cylinder compression with a round punch without friction after 6 stages (=3 seconds).
The strain distribution of cylinder compression with a round punch without friction. After 6 stages (=3 seconds).
5.5 The implementation of friction under changing boundary conditions

To implement friction in this case at which boundary conditions are not constant in time, we look at the compression of a cylinder with a rounded punch. Here we know that at the start of the process only the nodepoint in the center of the upper surface will contact the punch, and during the process other nodepoints in the upper surface will come in contact.

As soon as contact occurs we have to take friction into account.

5.5.1 How to compute the friction power

The difficulty that arise here is the question: How to implement the friction power? In this case, for each element we check which nodepoints are in contact with the tool, and not, like in case of cylinder compression with a flat punch, which surface is under friction.

For computing the friction power the area and the average velocity of the surface to which the nodepoints in contact belong are computed in the same way as mentioned in §2.3.2. For the normal stress on the surface, in case of Coulomb friction, the average of the four normal stresses at the nodepoints is taken:

\[ \sigma_n = \frac{1}{4} (\sigma_n(i,1) + \sigma_n(i,2) + \sigma_n(i,3) + \sigma_n(i,4)) \]  

(73)

where:

- \( i \) = surface number
- \( 1, \ldots, 4 \) = nodepoint number

If a nodepoint doesn’t touch the punch surface, it’s normal stress is of course zero.

The derivation of the normal stresses at the nodepoints is described at §5.5.2.

When Von Mises friction is used, we don’t need the normal stress, because at the Von Mises method the flowstress \( (\sigma_f) \) is being used.

5.5.2 Derivation of the normal stress

When Coulomb friction (which is considered to be the best friction approximation in case of deep drawing) is desired to take into account, we have to know the normal stress at each nodepoint touching a tool surface.

At cylinder compression with a flat punch, the normal stress (in that case on the surface) was equal to the element stress in Z-direction.

But in case of a rounded punch, the direction of the normal vector of a nodepoint
touching the surface depends on the place of touching.

So a coördinate transformation matrix \((L)\) has to be derived, which changes the global \(X,Y,Z\) coördinate system into the local \(X',Y',Z'\) coördinate system at the nodepoint concerned. In this new coördinate system, the proper values for the normal stresses are computed.

5.5.3 Derivation of the transfermatrix

Equation (74) gives a general description how a coördinate system can be transferred to another:

\[
\begin{bmatrix}
X' \\
Y' \\
Z'
\end{bmatrix} = \begin{bmatrix}
L
\end{bmatrix} \begin{bmatrix}
X \\
Y \\
Z
\end{bmatrix}
\]  (74)

Given the definition of the B-spline surface, describing the punch or die geometry, we may derive the coördinate transformationmatrix \(L\). The basis vectors of the local curvilineair coördinate system consist of two unit vectors in parametrically tangential directions and one unit vector normal to the corresponding surface. These three vectors \((e'_1,e'_2,e'_3)\) form the transformation matrix \(L\) and are derived as follows:

\[
e'_1 = \frac{\partial S(u,w)}{\partial u}, \quad e'_2 = \frac{\partial S(u,w)}{\partial w}, \quad e'_3 = e'_1 \times e'_2
\]  (75)

\[
L = (e'_1,e'_2,e'_3)
\]

If \(S(u,w) = \begin{bmatrix} X(u,w) \\ Y(u,w) \\ Z(u,w) \end{bmatrix}\) is substituted in equation (75), then equation (75) can be rewritten as follows:

\[
e'_1 = \frac{\partial S(u,w)}{\partial u} = \begin{bmatrix} \frac{\partial x(u,w)}{\partial u} & \frac{\partial y(u,w)}{\partial u} & \frac{\partial z(u,w)}{\partial u} \\ \frac{\partial s(u,w)}{\partial u} & \frac{\partial y(u,w)}{\partial u} & \frac{\partial z(u,w)}{\partial u} \\ \frac{\partial s(u,w)}{\partial u} & \frac{\partial s(u,w)}{\partial u} & \frac{\partial s(u,w)}{\partial u} \end{bmatrix}
\]  (76)

\[
e'_2 = \frac{\partial S(u,w)}{\partial w} = \begin{bmatrix} \frac{\partial x(u,w)}{\partial w} & \frac{\partial y(u,w)}{\partial w} & \frac{\partial z(u,w)}{\partial w} \\ \frac{\partial s(u,w)}{\partial w} & \frac{\partial s(u,w)}{\partial w} & \frac{\partial s(u,w)}{\partial w} \\ \frac{\partial s(u,w)}{\partial w} & \frac{\partial s(u,w)}{\partial w} & \frac{\partial s(u,w)}{\partial w} \end{bmatrix}
\]  (77)
Improvements on the 3-D RIPLE Program

\[ e'_3 = e'_1 \times e'_2 = \left( \frac{\partial y(u,w)}{\partial u} \frac{\partial y(u,w)}{\partial w} \right) \left( \frac{\partial x(u,w)}{\partial u} \right) \left( \frac{\partial x(u,w)}{\partial w} \right) \]

\[ \left( \frac{\partial S(u,w)}{\partial u} \right) \left( \frac{\partial S(u,w)}{\partial w} \right) \left( \frac{\partial x(u,w)}{\partial u} \right) \left( \frac{\partial x(u,w)}{\partial w} \right) \]

\[ \left( \frac{\partial S(u,w)}{\partial u} \right) \left( \frac{\partial S(u,w)}{\partial w} \right) \left( \frac{\partial x(u,w)}{\partial u} \right) \left( \frac{\partial x(u,w)}{\partial w} \right) \]

\[ \frac{\partial S(u,w)}{\partial u} = \sqrt{\left( \frac{\partial x(u,w)}{\partial u} \right)^2 + \left( \frac{\partial y(u,w)}{\partial u} \right)^2 + \left( \frac{\partial z(u,w)}{\partial u} \right)^2} \]  \hspace{1cm} (78)

\[ \frac{\partial S(u,w)}{\partial w} = \sqrt{\left( \frac{\partial x(u,w)}{\partial w} \right)^2 + \left( \frac{\partial y(u,w)}{\partial w} \right)^2 + \left( \frac{\partial z(u,w)}{\partial w} \right)^2} \]  \hspace{1cm} (79)

5.6 Stress and stain distributions (with friction)

These plots show the stress and strain distributions in the simulation of cylinder compression with a rounded punch after 6 stages (=3 seconds). Here Coulomb and Von Mises friction is used.
The stress distribution at cylinder compression with a rounded punch using Coulomb friction ($\mu=0.1$), after 6 stages (=3 seconds).
The strain distribution at cylinder compression with a rounded punch using Coulomb friction ($\mu=0.1$), after 6 stages (=3 seconds).
The stress distribution at cylinder compression with a rounded punch using Von Mises friction (m=0.1), after 6 stages (=3 seconds).
The strain distribution at cylinder compression with a rounded punch using Von Mises friction \((m=0.1)\), after 6 stages (=3 seconds).
References


Appendices A till D
Appendix A
Inputfile for the programs fem3a.f till fem3d.f

```
TITLE
*** RIPLE CODE (CYLINDRICAL ELEMENT - 70) TEST RUN ***
MODEL
  1 126 70
STAGE
  10
CONVERGENCE
  1.0E-5
PRINT
  1
DEBUG
  0 0
ITERATION
  100
DISPLACEMENT
  2.00
FRICTION
  1
STIF
  1
CON1
  0.33300000 7 0.5000000
CON2
  100 0.01 0.00001
ENDPC
NODE
  1  0.00  0.00 100.00
  5 100.00  0.00 100.00 1
  6  0.00  0.00  80.00
 10 100.00  0.00  80.00 1
 11  0.00  0.00  60.00
 15 100.00  0.00  60.00 1
 16  0.00  0.00  40.00
 20 100.00  0.00  40.00 1
 21  0.00  0.00  20.00
 25 100.00  0.00  20.00 1
 26  0.00  0.00  0.00
 30 100.00  0.00  0.00 1
 31 23.10  9.57 100.00
 34 92.39 28.27 100.00 1
 35 23.10  9.57  80.00
 38 92.39 28.27  80.00 1
 39 23.10  9.57  60.00
 42 92.39 28.27  60.00 1
 43 23.10  9.57  40.00
 46 92.39 28.27  40.00 1
 47 23.10  9.57  20.00
 50 92.39 28.27  20.00 1
 51 23.10  9.57  0.00
 54 92.39 28.27  0.00 1
 55 17.68  17.68 100.00
 58 70.71  70.71 100.00 1
 59 17.68  17.68  80.00
 62 70.71  70.71  80.00 1
```
<table>
<thead>
<tr>
<th>ELEMENT</th>
<th>1 6 7 35 59</th>
<th>1 2 31 55</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>2 7 8 36 35</td>
<td>2 3 32 31</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td>3 8 9 37 36</td>
<td>3 4 33 32</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td>4 9 10 38 37</td>
<td>4 5 34 33</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td>5 11 12 39 63</td>
<td>6 7 35 59</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td>6 12 13 40 39</td>
<td>7 8 36 35</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td>7 13 14 41 40</td>
<td>8 9 37 36</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td>8 14 15 42 41</td>
<td>9 10 38 37</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td>9 16 17 43 67</td>
<td>11 12 39 63</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td>10 17 18 44 43</td>
<td>12 13 40 39</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td>11 18 19 45 44</td>
<td>13 14 41 40</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td>12 19 20 46 45</td>
<td>14 15 42 41</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td>13 21 22 47 71</td>
<td>16 17 43 67</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td>14 22 23 48 47</td>
<td>17 18 44 43</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td>15 23 24 49 48</td>
<td>18 19 45 44</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td>16 24 25 50 49</td>
<td>19 20 46 45</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td>17 26 27 51 75</td>
<td>21 22 47 71</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td>18 27 28 52 51</td>
<td>22 23 48 47</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td>19 28 29 53 52</td>
<td>23 24 49 48</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td>20 29 30 54 53</td>
<td>24 25 50 49</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td>21 35 36 60 59</td>
<td>31 32 56 55</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td>22 36 37 61 60</td>
<td>32 33 57 56</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td>23 37 38 62 61</td>
<td>33 34 58 57</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td>24 39 40 64 63</td>
<td>35 36 60 59</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td>25 40 41 65 64</td>
<td>36 37 61 60</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td>26 41 42 66 65</td>
<td>37 38 62 61</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td>27 43 44 68 67</td>
<td>39 40 64 63</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td>28 44 45 69 68</td>
<td>40 41 65 64</td>
<td>6</td>
</tr>
<tr>
<td>( V )</td>
<td>31</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>( V )</td>
<td>34</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>( V )</td>
<td>55</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>( V )</td>
<td>58</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>( V )</td>
<td>79</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>( V )</td>
<td>82</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>( V )</td>
<td>103</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>( V )</td>
<td>106</td>
<td>3</td>
<td>1</td>
</tr>
</tbody>
</table>

**VELOCITY**

\[
\begin{array}{cccccc}
0.01 & 0.0 & 0.01 & 0.0 & -0.04 & 0.0 \\
\end{array}
\]

**FRICTION**

\[
\begin{array}{c}
1 & 0.3 \\
\end{array}
\]

**PROPERTY**

\[
\begin{array}{cccc}
200.0 & 540.0 & 0.01 & 0.155 \\
\end{array}
\]

**LOAD**

\[
\begin{array}{cc}
C & 13 & 1 & 0.0 \\
\end{array}
\]

ENDMDC
Appendix B

The subroutines ELSETF and FRIC for each of the programs fem3a.f till fem3d.f

Appendix B1

fem3a.f

SUBROUTINE ELSETF

*ELSETF -----. SUBROUTINE ELSETF -------

SUBROUTINE ELSETF(NEL,NFACE)

COMMON /DEBUG/ IDBG(20), IOSP(50)
COMMON /LMVAI/ N1, N2, N3, N4, N5, N6, N7, N8
COMMON /NPVA1/ HEAD(20), IFLAG(20), CLIM, DFACT, ISTIF
* , INTIV, MODEL
COMMON /NDAT/ U(24), XI(8), YJ(8), ZJ(8)
COMMON /FACE/ AVVX(6), AVVy(6), AVVZ(6), CX(6), CY(6), CZ(6),
* , AEV(6), TAUF, AREA(6)
DIMENSION NFACE(6,1)

IF(IFLAG(12).EQ.0) GO TO 400

DO 300 N=1,6
   IF(NFACE(N,NEL).EQ.0) GO TO 300
   GO TO (I10,120,130,140,150,160),N

100 icontinue
   AX=XJ(6)-XJ(1)
   AY=YJ(6)-YJ(1)
   AZ=ZJ(6)-ZJ(1)
   BX=XJ(5)-XJ(2)
   BY=YJ(5)-YJ(2)
   BZ=ZJ(5)-ZJ(2)
   AVVX(N)=0.25*(U(1)+U(4)+U(16)+U(13))
   AVVy(N)=0.25*(U(2)+U(5)+U(17)+U(14))
   AVVZ(N)=0.25*(U(3)+U(6)+U(18)+U(15))
   GO TO 200

200 icontinue
   AX=XJ(7)-XJ(2)
   AY=YJ(7)-YJ(2)
   AZ=ZJ(7)-ZJ(2)
   BX=XJ(6)-XJ(3)
   BY=YJ(6)-YJ(3)
   BZ=ZJ(6)-ZJ(3)
   AVVX(N)=0.25*(U(4)+U(7)+U(19)+U(16))
   AVVy(N)=0.25*(U(5)+U(8)+U(20)+U(17))
   AVVZ(N)=0.25*(U(6)+U(9)+U(21)+U(18))
   GO TO 200

300 icontinue
   AX=XJ(8)-XJ(3)
   AY=YJ(8)-YJ(3)
   AZ=ZJ(8)-ZJ(3)
Improvements on the 3-D RIPLE Program

\[ BX = XJ(7) - XJ(4) \]
\[ BY = YJ(7) - YJ(4) \]
\[ BZ = ZJ(7) - ZJ(4) \]
\[ AVVX(N) = 0.25 \times (U(7) + U(10) + U(22) + U(19)) \]
\[ AVVY(N) = 0.25 \times (U(8) + U(11) + U(23) + U(20)) \]
\[ AVVZ(N) = 0.25 \times (U(9) + U(12) + U(24) + U(21)) \]
GO TO 200

** ** 4-TH FACE

140 CONTINUE
\[ AX = XJ(5) - XJ(4) \]
\[ AY = YJ(5) - YJ(4) \]
\[ AZ = ZJ(5) - ZJ(4) \]
\[ BX = XJ(8) - XJ(1) \]
\[ BY = YJ(8) - YJ(1) \]
\[ BZ = ZJ(8) - ZJ(1) \]
\[ AVVX(N) = 0.25 \times (U(10) + U(1) + U(13) + U(22)) \]
\[ AVVY(N) = 0.25 \times (U(11) + U(2) + U(14) + U(23)) \]
\[ AVVZ(N) = 0.25 \times (U(12) + U(3) + U(15) + U(24)) \]
GO TO 200

** ** 5-TH FACE

150 CONTINUE
\[ AX = XJ(2) - XJ(4) \]
\[ AY = YJ(2) - YJ(4) \]
\[ AZ = ZJ(2) - ZJ(4) \]
\[ BX = XJ(1) - XJ(3) \]
\[ BY = YJ(1) - YJ(3) \]
\[ BZ = ZJ(1) - ZJ(3) \]
\[ AVVX(N) = 0.25 \times (U(10) + U(7) + U(4) + U(1)) \]
\[ AVVY(N) = 0.25 \times (U(11) + U(8) + U(5) + U(2)) \]
\[ AVVZ(N) = 0.25 \times (U(12) + U(9) + U(6) + U(3)) \]
GO TO 200

** ** 6-TH FACE

160 CONTINUE
\[ AX = XJ(7) - XJ(5) \]
\[ AY = YJ(7) - YJ(5) \]
\[ AZ = ZJ(7) - ZJ(5) \]
\[ BX = XJ(8) - XJ(6) \]
\[ BY = YJ(8) - YJ(6) \]
\[ BZ = ZJ(8) - ZJ(6) \]
\[ AVVX(N) = 0.25 \times (U(13) + U(16) + U(19) + U(22)) \]
\[ AVVY(N) = 0.25 \times (U(14) + U(17) + U(20) + U(23)) \]
\[ AVVZ(N) = 0.25 \times (U(15) + U(18) + U(21) + U(24)) \]
GO TO 200

200 CONTINUE
\[ AI = AY \times BZ - AZ \times BY \]
\[ AJ = AZ \times BX - AX \times BZ \]
\[ AK = AX \times BY - AY \times BX \]
\[ AREA(N) = \sqrt{AI \times AI + AJ \times AJ + AK \times AK} \]
\[ CX(N) = AI / AREA(N) \]
\[ CY(N) = AJ / AREA(N) \]
\[ CZ(N) = AK / AREA(N) \]
\[ AREA(N) = AREA(N) \times 0.5 \]
\[ AVEV(N) = \sqrt{AVVX(N) \times AVVX(N) + AVVY(N) \times AVVY(N) + AVVZ(N) \times AVVZ(N)} \]
300 CONTINUE

300 CONTINUE

** ** PRINTS 'EL' ELEMENT INFORMATION.
IF(IOISP(3) .EQ. 0) GO TO 10
* WRITE(15,* ) ' NFACE(1,NEL)-NFACE(4,NEL)', (NFACE(I,NEL), I=1,4)
* WRITE(15,* ) ' VK(1)-VK(4)', (VK(I), I=1,4)
* WRITE(15,* ) ' COST(1)-COST(4)', (COST(I), I=1,4)
* WRITE(15,* ) ' SINT(1)-SINT(4)', (SINT(I), I=1,4)

** ** PRINTS 'EL' ELEMENT INFORMATION.
IF(IOISP(3) .EQ. 0) GO TO 10
* WRITE(15,* ) ' NFACE(1,NEL)-NFACE(4,NEL)', (NFACE(I,NEL), I=1,4)
* WRITE(15,* ) ' VK(1)-VK(4)', (VK(I), I=1,4)
* WRITE(15,* ) ' COST(1)-COST(4)', (COST(I), I=1,4)
* WRITE(15,* ) ' SINT(1)-SINT(4)', (SINT(I), I=1,4)
SUBROUTINE FRIC

*FRECheiten SUBROUTINE FRIC -----

*SUBROUTINE FRIC(ES,H,NFACE,SIGN,EPSEL,SIGZ,NEL,NUMSTA)

COMMON /DEBUG/ IDBG(20), IOSP(50)
COMMON /ELMVA/ N1, N2, N3, N4, N5, N6, N7, N8
COMMON /NPVA/ HEAD(20), IFLAG(20), CLIM, DFACT, ISTIF

COMMON /NNDATI/ U(24),XJ(8),YJ(8),ZJ(8)
COMMON /INPV/ AI HEAD(20), IFLAG(20), CUM, DAFT, ISTIF
COMMON INDDATI
COMMON /FACE/ AVVX(6),AVVY(6),AVVZ(6),CX(6),CY(6),CZ(6),
AVEV(6), TAUF,AREA(6)

DIMENSION ES(2S,2S), H(2S), NX(4),NY(4),NZ(4)
DIMENSION NFACE(6,1)

IF(IFLAG(12).EQ.0) GO TO 600
DO 500 M=1,6
IF(NFACE(M,NEL).EQ.0) GO TO 500
GO TO (310,320),ITAU

310 CONTINUE
SIGNOR=0.0
SIGNOR=SIGN-(SIGEL*EPSEL)*0.666667*SIGZ
IF(RMN*SIGNOR.GT.0.57735*SIGEL) THEN
TAUF=0.57735*SIGEL
ELSE
TAUF=RMN*SIGNOR
END IF

** 2/SQRT(3)=1.1547005
IF(NUMSTA.EQ.1) TAUF=1.1547*RMN*SIGEL
GO TO 330

320 CONTINUE
** 1/SQRT(3)=0.577350
TAUF=0.57735*RMN*SIGEL
330 CONTINUE
TAUF=TAUF*AREA(M)
AVEV2=AVEV(M)*AVEV(M)
AVEV3=AVEV(M)*AVEV2

*** ELEMENT MATRIX MODIFICATION
GO TO (110,120,130,140,150,160), M

1-ST SURFACE
110 CONTINUE
NF1=1
NF2=2
NF3=6
NF4=5
GO TO 170

2-ND SURFACE
120 CONTINUE
NF1=2
NF2=3
NF3=7
NF4=6
GO TO 170
• 3-ED SURFACE
130 CONTINUE
  NFI=3
  NF2=4
  NF3=8
  NF4=7
  GO TO 170
• 4-TH SURFACE
140 CONTINUE
  NFI=4
  NF2=1
  NF3=5
  NF4=8
  GO TO 170
• 5-TH SURFACE
150 CONTINUE
  NFI=4
  NF2=3
  NF3=2
  NF4=1
  GO TO 170
• 6-TH SURFACE
160 CONTINUE
  NFI=5
  NF2=6
  NF3=7
  NF4=8
170 CONTINUE
  NX(I)=NFI*3-2
  NX(2)=NF2*3-2
  NX(3)=NF3*3-2
  NX(4)=NF4*3-2
  NY(I)=NX(1)+1
  NY(2)=NX(2)+1
  NY(3)=NX(3)+1
  NY(4)=NX(4)+1
  NZ(I)=NY(1)+1
  NZ(2)=NY(2)+1
  NZ(3)=NY(3)+1
  NZ(4)=NY(4)+1
• •••
  DO 200 I=1,4
  ES(NX(I),NX(I))=ES(NX(I),NX(I))+TAUFA2/(16*AVEV(M))
  ES(NY(I),NY(I))=ES(NY(I),NY(I))+TAUFA2/(16*AVEV(M))
  ES(NZ(I),NZ(I))=ES(NZ(I),NZ(I))+TAUFA2/(16*AVEV(M))
200 CONTINUE
• •••
  DO 210 I=1,4
  DO 210 J=I,4
  ES(NX(I),NX(J))=ES(NX(I),NX(J))-TAUFA2*AVVX(M)/256*AVEV(M)
  ES(NY(I),NY(J))=ES(NY(I),NY(J))-TAUFA2*AVVY(M)/256*AVEV(M)
  ES(NZ(I),NZ(J))=ES(NZ(I),NZ(J))-TAUFA2*AVVZ(M)/256*AVEV(M)
210 CONTINUE
• •••
  DO 220 I=1,4
  H(NX(I))=H(NX(I))-TAUFA2*AVVX(M)/(4*AVEV(M))
H(NY(I))=H(NY(I))-TAUFA2*AVVY(M)/(4*AVEV(M))
H(NZ(I))=H(NZ(I))-TAUFA2*AVVZ(M)/(4*AVEV(M))
220 CONTINUE

* 
IF(IOSP(7).EQ.0) GO TO 500
WRITE(15,*) ' ELEMENT NO.=',NEL,' * FACE NO.=',M
WRITE(15,*) ' TAV3=',TAV3,' * TAUFA2=',TAUFA2

* 
500 CONTINUE
*
600 CONTINUE
RETURN
END
Appendix B2

SUBROUTINE ELSETF

*ELSETF ------ SUBROUTINE ELSETF ------

SUBROUTINE ELSETF(NEL,NFACE)

COMMON /DEBUG/ IDBG(20) , IOSP(50)
COMMON /ELMVA/ N1 , N2 , N3 , N4 , N5 , N6 , N7 , N8
COMMON /INPVA/ HEAD(20) , IFLAG(20) , CLIM , DFACT , ISTIF
* , INTV , MODEL
COMMON /NDDAT/ U(24),XJ(8),YJ(8),ZJ(8)
COMMON /FACE/ VK1(6), VK2(6), VK3(6), VK4(6), CX(6), CY(6), CZ(6),
* , AVEV(6) , TAUF ,AREA(6)

DIMENSION NFACE(6,1)

IF(IFLAG(12).EQ.0) GO TO 400

* • • • FRICITION MODEL

DO 300 N=1,6
IF(NFACE(N,NEL).EQ.0) GO TO 300
GO TO (I10,120,130,J40,150,160),N

* • • • 1-ST FACE

110 CONTINUE

AX=XJ(6)-XJ(1)
AY=YJ(6)-YJ(1)
AZ=ZJ(6)-ZJ(1)
BX=XJ(5)-XJ(2)
BY=YJ(5)-YJ(2)
BZ=ZJ(5)-ZJ(2)

VK1(N)=SQRT(U(1)*U(1)+U(2)*U(2)+U(3)*U(3))
VK2(N)=SQRT(U(4)*U(4)+U(5)*U(5)+U(6)*U(6))
VK3(N)=SQRT(U(16)*U(16)+U(17)*U(17)+U(18)*U(18))
VK4(N)=SQRT(U(13)*U(13)+U(14)*U(14)+U(15)*U(15))

GO TO 200

* • • • 2-ND FACE

120 CONTINUE

AX=XJ(7)-XJ(2)
AY=YJ(7)-YJ(2)
AZ=ZJ(7)-ZJ(2)
BX=XJ(6)-XJ(3)
BY=YJ(6)-YJ(3)
BZ=ZJ(6)-ZJ(3)

VK1(N)=SQRT(U(4)*U(4)+U(5)*U(5)+U(6)*U(6))
VK2(N)=SQRT(U(7)*U(7)+U(8)*U(8)+U(9)*U(9))
VK3(N)=SQRT(U(19)*U(19)+U(20)*U(20)+U(21)*U(21))
VK4(N)=SQRT(U(16)*U(16)+U(17)*U(17)+U(18)*U(18))

GO TO 200

* • • • 3-RD FACE

130 CONTINUE

AX=XJ(8)-XJ(3)
AY=YJ(8)-YJ(3)
AZ=ZJ(8)-ZJ(3)
BX=XJ(7)-XJ(4)
BY=YJ(7)-YJ(4)
BZ=ZJ(7)-ZJ(4)

VK1(N)=SQRT(U(7)*U(7)+U(8)*U(8)+U(9)*U(9))
VK2(N)=SQRT(U(10)*U(10)+U(11)*U(11)+U(12)*U(12))
VK3(N)=SQRT(U(22)*U(22)+U(23)*U(23)+U(24)*U(24))
IMPORTS ON THE 3-D RIPLE PROGRAM

YK4(N) = SQRT(U(19)*U(19) + U(20)*U(20) + U(21)*U(21))
GO TO 200

* * * 4-TH FACE

140 CONTINUE
AX = XJ(5) - XJ(4)
AY = YJ(5) - YJ(4)
AZ = ZJ(5) - ZJ(4)
BX = XJ(8) - XJ(1)
BY = YJ(8) - YJ(1)
BZ = ZJ(8) - ZJ(1)
VK1(N) = SQRT(U(10)*U(10) + U(11)*U(11) + U(12)*U(12))
VK2(N) = SQRT(U(1)*U(1) + U(2)*U(2) + U(3)*U(3))
VK3(N) = SQRT(U(13)*U(13) + U(14)*U(14) + U(15)*U(15))
VK4(N) = SQRT(U(22)*U(22) + U(23)*U(23) + U(24)*U(24))
GO TO 200

* * * 5-TH FACE

150 CONTINUE
AX = XJ(2) - XJ(4)
AY = YJ(2) - YJ(4)
AZ = ZJ(2) - ZJ(4)
BX = XJ(1) - XJ(3)
BY = YJ(1) - YJ(3)
BZ = ZJ(1) - ZJ(3)
VK1(N) = SQRT(U(10)*U(10) + U(11)*U(11) + U(12)*U(12))
VK2(N) = SQRT(U(7)*U(7) + U(8)*U(8) + U(9)*U(9))
VK3(N) = SQRT(U(4)*U(4) + U(5)*U(5) + U(6)*U(6))
VK4(N) = SQRT(U(1)*U(1) + U(2)*U(2) + U(3)*U(3))
GO TO 200

* * * 6-TH FACE

160 CONTINUE
AX = XJ(7) - XJ(5)
AY = YJ(7) - YJ(5)
AZ = ZJ(7) - ZJ(5)
BX = XJ(8) - XJ(6)
BY = YJ(8) - YJ(6)
BZ = ZJ(8) - ZJ(6)
VK1(N) = SQRT(U(13)*U(13) + U(14)*U(14) + U(15)*U(15))
VK2(N) = SQRT(U(16)*U(16) + U(17)*U(17) + U(18)*U(18))
VK3(N) = SQRT(U(19)*U(19) + U(20)*U(20) + U(21)*U(21))
VK4(N) = SQRT(U(22)*U(22) + U(23)*U(23) + U(24)*U(24))
GO TO 200

200 CONTINUE
AL = AY*AZ - AZ*BY
AJ = AZ*BX - AX*BZ
AK = AX*BY - AY*BX
AREA(N) = SQRT(AL*AL + AJ*AJ + AK*AK)
CX(N) = AL/AREA(N)
CY(N) = AJ/AREA(N)
CZ(N) = AK/AREA(N)
AREA(N) = AREA(N)*0.5
AVEV(N) = SQRT(VK1(N)*VK1(N) + VK2(N)*VK2(N) + VK3(N)*VK3(N) + VK4(N)*
* VK4(N))

300 CONTINUE

400 CONTINUE

* * * PRINTS 'EL' ELEMENT INFORMATION.
IF(ISMP(3).EQ.0) GO TO 10
* WRITE(15,*) ' NFACE(1,NEL)-NFACE(4,NEL)',(NFACE(I,NEL),I=1,4)
* WRITE(15,*) ' VK(1)-VK(4)',(VK(I),I=1,4)
* WRITE(15,*) ' VK(1)-VK(4)',(VK(I),I=1,4)
* WRITE(15,*) ' COST(1)-COST(4)',(COST(I),I=1,4)
* WRITE(15,*) ' SINT(1)-SINT(4)',(SINT(I),I=1,4)
* WRITE(15,*) ' AREA(1)-AREA(6)',(AREA(I),I=1,6)

MEL - Tsukuba - July 1993
Improvements on the 3-D RIPLE Program

SUBROUTINE FRIC

*FREC------SUBROUTINE FRIC------

SUBROUTINE FRIC(ES,H,NFACE,SIGN,EPSEL,SIGEL,SIGZ,NEL,NUMSTA)

COMMON /DEBUG/ IDBG(20),IOSP(50)
COMMON /ELMVA/ N1,N2,N3,N4,N5,N6,N7,N8
COMMON /INPV/ HEAD(20),IFLAG(20),CLIM,DFACT,ISTIF
  ,INITY,MODEL
COMMON /INPA/ U(24),XI(8),YI(8),ZI(8)
COMMON /FRICT/ RMN,ITAU
COMMON /FACE/ VK1(6),VK2(6),VK3(6),VK4(6),CX(6),CY(6),CZ(6),
   AVEV(6),TAUF,AREA(6)
DIMENSION ES(2S,2S),H(2S),NX(4),NY(4),NZ(4)
DIMENSION NFACE(6,1)

IF(IFLAG(12).EQ.0) GO TO 600
DO 500 M=1,6
IF(NFACE(M,NEL).EQ.0) GO TO 500
GO TO (310,320),ITAU
310 CONTINUE
SIGNOR=0.0
SIGNOR=SIGN-(SIGEL*EPSEL)·0.666667·SIGZ
IF(RMN·SIGNOR.GT.0.57735·SIGEL) THEN
   TAUF=0.57735·SIGEL
ELSE
   TAUF=RMN·SIGNOR
END IF
** 2/SQRT(3)=1.1547005
IF(NUMSTA.EQ.1) TAUF=1.1547·RMN·SIGEL
GO TO 330
320 CONTINUE
** 1/SQRT(3)=0.577350
TAUF=0.57735·RMN·SIGEL
330 CONTINUE
TAUFA2=TAUF·AREA(M)
AVEV2=AVEV(M)·AVEV(M)
AVEV3=AVEV(M)·AVEV(M)
TAV3=TAUFA2/AVEV3
*** ELEMENT MATRIX MODIFICATION
GO TO (110,120,130,140,150,160),M
1-ST SURFACE
110 CONTINUE
NF1=1
NF2=2
NF3=6
NF4=5
GO TO 170
2-Nd SURFACE
120 CONTINUE
NF1=2
NF2=3
NF3=7
NF4=6
Improvements on the 3-D RIPLE Program

* 3-ED SURFACE
130 CONTINUE
   NFI=3
   NF2=4
   NF3=8
   NF4=7
   GO TO 170

* 4-TH SURFACE
140 CONTINUE
   NFI=4
   NF2=1
   NF3=5
   NF4=8
   GO TO 170

* 5-TH SURFACE
150 CONTINUE
   NFI=4
   NF2=3
   NF3=2
   NF4=1
   GO TO 170

* 6-TH SURFACE
160 CONTINUE
   NFI=5
   NF2=6
   NF3=7
   NF4=8

170 CONTINUE
   NX(I)=NFI*3-2
   NX(2)=NF2*3-2
   NX(3)=NF3*3-2
   NX(4)=NF4*3-2
   NY(I)=NX(I)+I
   NY(2)=NX(2)+I
   NY(3)=NX(3)+I
   NY(4)=NX(4)+I
   NZ(I)=NY(I)+I
   NZ(2)=NY(2)+I
   NZ(3)=NY(3)+I
   NZ(4)=NY(4)+I

* ***
   DO 200 I=1,4
      ES(NX(I),NX(I))=ES(NX(I),NX(I))+TA V3*AV EV2
      ES(NY(I),NY(I))=ES(NY(I),NY(I))+TAV3*AVEV2
      ES(NZ(I),NZ(I))=ES(NZ(I),NZ(I))+TAV3*AVEV2
   200 CONTINUE
* ***
   DO 210 I=1,4
      DO 210 J=I,4
         ES(NX(I),NX(J))=ES(NX(I),NX(J))-TAV3*U(NX(I))*U(NX(J))
         ES(NY(I),NY(J))=ES(NY(I),NY(J))-TAV3*U(NY(I))*U(NY(J))
         ES(NZ(I),NZ(J))=ES(NZ(I),NZ(J))-TAV3*U(NZ(I))*U(NZ(J))
      210 CONTINUE
* ***
   DO 220 I=1,4
      ES(NX(I),NX(J))=ES(NX(I),NX(J))-TAV3*U(NX(I))*U(NX(J))
      ES(NY(I),NY(J))=ES(NY(I),NY(J))-TAV3*U(NY(I))*U(NY(J))
      ES(NZ(I),NZ(J))=ES(NZ(I),NZ(J))-TAV3*U(NZ(I))*U(NZ(J))
   220 CONTINUE
H(NX(I))=H(NX(I))-TAUFA2*U(NX(I))/AVEV(M)
H(NY(I))=H(NY(I))-TAUFA2*U(NY(I))/AVEV(M)
H(NZ(I))=H(NZ(I))-TAUFA2*U(NZ(I))/AVEV(M)

220 CONTINUE

IF(IOSP(7).EQ.0) GO TO 500
WRITE(15,*) ' * ELEMENT NO. =',NEL,* FACE NO. =',M
WRITE(15,*) ' * TAV3 =',TAV3,* TAUFA2 =',TAUFA2

500 CONTINUE

600 CONTINUE
RETURN
END
Appendix B3

fem3c.f

SUBROUTINE ELSETF

Subroutine ELSETF in fem3c.f is the same as subroutine ELSETF in fem3a.f.

SUBROUTINE FRIC

*FREC  ------- SUBROUTINE FRIC -------

* COMMON /DEBUG/ IDBG(20) , I0SP(50)
COMMON /ELMV/A/ N1 , N2 , N3 , N4 , N5 , N6 , N7 , N8
COMMON /INPV/A/ HEAD(20) , IFLAG(20) , CLIM , DFACT , ISTIF
* , INTV , MODEL
COMMON /NDDAT/ U(24),XI(8),YJ(8),ZJ(8)
COMMON /FRIC/ RMN,ITAU
COMMON /FACE/ AVVX(6),AVYY(6),AVVZ(6),CX(6),CY(6),CZ(6),
* AYEY(6) , TAUF , AREA(6)
DIMENSION ES(2S,2S), H(2S), NX(4), NY(4), NZ(4)
DIMENSION NFACE(6,1)

* IF(IFLAG(12).EQ.0) GO TO 600
DO 500 M=1,6
IF(NFACE(M,NEL).EQ.0) GO TO 500
GO TO (310,320),ITAU
310 CONTINUE
SIGNOR=0.0
SIGNOR=SIGN-(SIGEL*EPSL)-0.666667-SIGZ
IF(RMN-SIGNOR.GT.0.57735-SIGEL) THEN
TAUF=0.57735-SIGEL
ELSE
TAUF=RMN*SIGNOR
END IF
* ** 2/SQRT(3)=1.1547005
IF(NUMSTA.EQ.1) TAUF=1.1547*RMN*SIGEL
GO TO 330
320 CONTINUE
* ** 1/SQRT(3)=0.577350
TAUF=0.57735*RMN*SIGEL
330 CONTINUE
TAUFA2=TAUF*AREA(M)
AVEV2=AVEV(M)*AVEV(M)
AVEV3=AVEV(M)*AVEV2
TAVF3=TAUFA2/AVEV3
* *** ELEMENT MATRIX MODIFICATION
GO TO (110,120,130,140,150,160) , M
* 1-ST SURFACE
110 CONTINUE
NF1=1
NF2=2
NF3=6
NF4=5
GO TO 170
* 2-ND SURFACE

--- MEL - Tsukuba - July 1993 ---
Improvements on the 3-D RIPLE Program

120 CONTINUE
   NF1=2
   NF2=3
   NF3=7
   NF4=6
   GO TO 170

* 3-ED SURFACE
130 CONTINUE
   NF1=3
   NF2=4
   NF3=8
   NF4=7
   GO TO 170

* 4-TH SURFACE
140 CONTINUE
   NF1=4
   NF2=1
   NF3=5
   NF4=8
   GO TO 170

* 5-TH SURFACE
150 CONTINUE
   NF1=4
   NF2=3
   NF3=2
   NF4=1
   GO TO 170

* 6-TH SURFACE
160 CONTINUE
   NF1=5
   NF2=6
   NF3=7
   NF4=8
170 CONTINUE
   NX(I)=NF1*3-2
   NX(2)=NF2*3-2
   NX(3)=NF3*3-2
   NX(4)=NF4*3-2
   NY(I)=NX(I)+1
   NY(2)=NX(2)+1
   NY(3)=NX(3)+1
   NY(4)=NX(4)+1
   NZ(I)=NY(1)+1
   NZ(2)=NY(2)+1
   NZ(3)=NY(3)+1
   NZ(4)=NY(4)+1

   DO 200 I=1,4
      ES(NX(I),NX(I))=ES(NX(I),NX(I))+TA*AV3*AVEV2
      ES(NY(I),NY(I))=ES(NY(I),NY(I))+TA*AV3*AVEV2
      ES(NZ(I),NZ(I))=ES(NZ(I),NZ(I))+TA*AV3*AVEV2
200 CONTINUE

   DO 210 I=1,4
      DO 210 J=I,4
         ES(NX(I),NX(J))=ES(NX(I),NX(J))-TA*AV3*U(NX(I))U(NX(J))
         ES(NY(I),NY(J))=ES(NY(I),NY(J))-TA*AV3*U(NY(I))U(NY(J))
         ES(NZ(I),NZ(J))=ES(NZ(I),NZ(J))-TA*AV3*U(NZ(I))U(NZ(J))
         ES(NX(I),NY(J))=ES(NX(I),NY(J))-TA*AV3*U(NX(I))U(NY(J))
         ES(NY(I),NZ(J))=ES(NY(I),NZ(J))-TA*AV3*U(NY(I))U(NZ(J))
         ES(NZ(I),NX(J))=ES(NZ(I),NX(J))-TA*AV3*U(NZ(I))U(NX(J))
210 CONTINUE
ES(NZ(I),NY(J)) = ES(NZ(I),NY(J)) - TAV3 * U(NZ(I)) * U(NY(J))
ES(NX(I),NZ(J)) = ES(NX(I),NZ(J)) - TAV3 * U(NX(I)) * U(NZ(J))

210 CONTINUE

**
DO 220 I=1,4
H(NX(I)) = H(NX(I)) - TAUFA2 * U(NX(I)) / AVEV(M)
H(NY(I)) = H(NY(I)) - TAUFA2 * U(NY(I)) / AVEV(M)
H(NZ(I)) = H(NZ(I)) - TAUFA2 * U(NZ(I)) / AVEV(M)

220 CONTINUE

IF(ISOP(7).EQ.0) GO TO 500
WRITE(15,*) ' ** ELEMENT NO. =',NEL,' ** FACE NO. =',M
WRITE(15,*) ' ** TAV3 =',TAV3,' ** TAUFA2 =',TAUFA2

500 CONTINUE

600 CONTINUE
RETURN
END
SUBROUTINE ELSETF

SUBROUTINE ELSETF(NEL,NFACE)
COMMON /DE8UGIDBG(20),IOSP(50)
COMMON /ELMVAIM1,N2,N3,N4,N5,N6,N7,N8
COMMON /INPVAIM1HEAD(20),IFLAG(20),CLIM,DFACT,ISTIF
COMMON INITV,MODEL
COMMON INDDATI U(24),XJ(8),YJ(8),ZJ(8)
COMMON IAACEI WX(6),AWY(6),AVVZ(6),CX(6),CY(6),CZ(6),
(*AVEV(6),TAUF,AREA(6)
DIMENSION NFACE(6,1)

IF(IFLAG(12).EQ.0) GO TO 400
**FRICTION MODEL
DO 300 N=1,6
IF(NFACE(N,NEL).EQ.0) GO TO 300
GO TO (I10,120,130,140,150,160),N
**1-ST FACE
110 CONTINUE
AX=XJ(6)-XJ(1)
AY=YJ(6)-YJ(1)
AZ=ZJ(6)-ZJ(1)
BX=XJ(5)-XJ(2)
BY=YJ(5)-YJ(2)
BZ=ZJ(5)-ZJ(2)
AVVX(N)=0.25*(U(1)+U(4)+U(16)+U(13))
AVVY(N)=0.25*(U(2)+U(5)+U(17)+U(14))
AVVZ(N)=0.0
GO TO 200
**2-ND FACE
120 CONTINUE
AX=XJ(7)-XJ(2)
AY=YJ(7)-YJ(2)
AZ=ZJ(7)-ZJ(2)
BX=XJ(6)-XJ(3)
BY=YJ(6)-YJ(3)
BZ=ZJ(6)-ZJ(3)
AVVX(N)=0.25*(U(4)+U(7)+U(19)+U(16))
AVVY(N)=0.25*(U(5)+U(8)+U(20)+U(17))
AVVZ(N)=0.0
GO TO 200
**3-RD FACE
130 CONTINUE
AX=XJ(8)-XJ(3)
AY=YJ(8)-YJ(3)
AZ=ZJ(8)-ZJ(3)
BX=XJ(7)-XJ(4)
BY=YJ(7)-YJ(4)
BZ=ZJ(7)-ZJ(4)
AVVX(N)=0.25*(U(7)+U(10)+U(22)+U(19))
AVVY(N)=0.25*(U(8)+U(11)+U(23)+U(20))
AVVZ(N)=0.0
GO TO 200
**4-TH FACE

MEL - Tsukuba - July 1993
140 CONTINUE
AX=XJ(5)-XJ(4)
AY=YJ(5)-YJ(4)
AZ=ZJ(5)-ZJ(4)
BX=XJ(8)-XJ(1)
BY=YJ(8)-YJ(1)
BZ=ZJ(8)-ZJ(1)
AVVX(N)=0.25*(U(10)+U(1)+U(13)+U(22))
AVVY(N)=0.25*(U(11)+U(2)+U(14)+U(23))
AVVZ(N)=0.0
GO TO 200

* **

5-TH FACE
150 CONTINUE
AX=XJ(2)-XJ(4)
AY=YJ(2)-YJ(4)
AZ=ZJ(2)-ZJ(4)
BX=XJ(1)-XJ(3)
BY=YJ(1)-YJ(3)
BZ=ZJ(1)-ZJ(3)
AVVX(N)=0.25*(U(10)+U(7)+U(4)+U(1))
AVVY(N)=0.25*(U(11)+U(8)+U(5)+U(2))
AVVZ(N)=0.0
GO TO 200

* **

6-TH FACE
160 CONTINUE
AX=XJ(7)-XJ(5)
AY=YJ(7)-YJ(5)
AZ=ZJ(7)-ZJ(5)
BX=XJ(8)-XJ(6)
BY=YJ(8)-YJ(6)
BZ=ZJ(8)-ZJ(6)
AVVX(N)=0.25*(U(13)+U(16)+U(19)+U(22))
AVVY(N)=0.25*(U(14)+U(17)+U(20)+U(23))
AVVZ(N)=0.0
GO TO 200

200 CONTINUE
AI=AY*BZ-AZ*BY
AJ=AZ*BX-AX*BZ
AK=AX*BY-AY*BX
AREA(N)=SQRT(AI*AI+AJ*AJ+AK*AK)
CX(N)=AI/AREA(N)
CY(N)=AJ/AREA(N)
CZ(N)=AK/AREA(N)
ARE(N)=ARE(N)*0.5
AVEV(N)=SQRT(AYVX(N)+AVVY(N))
AVEV(N)=AVEV(N)
300 CONTINUE
400 CONTINUE

* *** PRINTS 'EL' ELEMENT INFORMATION.
IF(0SP(3).EQ.0) GO TO 10
WRITE(15,*) 'NFACE(I,NEL)-NFACE(4,NEL)',(NFACE(I,NEL),I=1,4)
WRITE(15,*) 'VK(I)-VK(4)',(VK(I),I=1,4)
WRITE(15,*) 'VK1(I)-VK1(4)',(VK1(I),I=1,4)
WRITE(15,*) 'COST(I)-COST(4)',(COST(I),I=1,4)
WRITE(15,*) 'SINT(I)-SINT(4)',(SINT(I),I=1,4)
WRITE(15,*) 'AREA(I)-AREA(6)',(AREA(I),I=1,6)
WRITE(15,*) 'AVEV(I)-AVEV(6)',(AVEV(I),I=1,6)
10 CONTINUE
RETURN
END

SUBROUTINE FRIC

---

MEL - Tsukuba - July 1993

69
*FREC ---- SUBROUTINE FRIC ----

SUBROUTINE FRIC(ES,H,NFACE,SIGN,EPSEL,SIGEL,SIGZ,NEL,NUMSTA)

COMMON /DEBUG/ IDBG(20), IOSP(50)
COMMON /ELMVA/ N1, N2, N3, N4, N5, N6, N7, N8
COMMON /INPV/ HEAD(20), IFLAG(20), CLIM, DFACT, ISTIF
    , INTV, MODEL
COMMON /NDDAT/ U(24),XI(8),Y(8),ZI(8)
COMMON /FRICT/ RMN, ITAU
COMMON /FACE/ AVVX(6),AVVY(6),AVVZ(6),CX(6),CY(6),CZ(6),
    AVEV(6), TAU, AREA(6)
DIMENSION ES(2S,2S), H(2S), NX(4), NY(4), NZ(4)
DIMENSION NFACE(6,1)

IF(IFLAG(12).EQ.0) GO TO 600
DO 500 M=1,6
IF(NFACE(M,NEL).EQ.0) GO TO 500
GO TO (310,320), ITAU
310 CONTINUE
SIGNOR=0.0
SIGNOR=SIGN-(SIGEL/EPSEL)*0.666667*SIZ
IF(RMN*SIGNOR.GT.0.57735*SIGEL) THEN
    TAU=0.57735*SIGEL
ELSE
    TAU=RMN*SIGNOR
END IF
** 2/SQRT(3)=1.1547005
IF(NUMSTA,EQ.1) TAU=1.1547*RMN*SIGEL
GO TO 330
320 CONTINUE
** 1/SQRT(3)=0.577350
TAU=0.57735*RMN*SIGEL
330 CONTINUE
TAUF2=TAUF*AREA(M)
AVEV2=AVEV(M)*AVEV(M)
AVEV3=AVEV(M)*AVEV2
TAV=TAUF2/AVEV3
** ** ELEMENT MATRIX MODIFICATION
GO TO (110,120,130,140,150,160), M
** 1-ST SURFACE
110 CONTINUE
NF1=1
NF2=2
NF3=6
NF4=5
GO TO 170
** 2-ND SURFACE
120 CONTINUE
NF1=2
NF2=3
NF3=7
NF4=6
GO TO 170
** 3-ED SURFACE
130 CONTINUE
NF1=3
NF2=4
NF3=8
NF4=7
GO TO 170
** 4-TH SURFACE
Improvements on the 3-D RIPLE Program

140 CONTINUE
   NFI=4
   NF2=1
   NF3=5
   NF4=8
   GO TO 170

* 5-TH SURFACE
150 CONTINUE
   NFI=4
   NF2=3
   NF3=2
   NF4=1
   GO TO 170

* 6-TH SURFACE
160 CONTINUE
   NFI=5
   NF2=6
   NF3=7
   NF4=8

170 CONTINUE
   NX(1)=NFI*3-2
   NX(2)=NF2*3-2
   NX(3)=NF3*3-2
   NX(4)=NF4*3-2
   NY(1)=NX(1)+1
   NY(2)=NX(2)+1
   NY(3)=NX(3)+1
   NY(4)=NX(4)+1
   NZ(1)=NY(1)+1
   NZ(2)=NY(2)+1
   NZ(3)=NY(3)+1
   NZ(4)=NY(4)+1

* •••
   DO 200 I=1,4
      ES(NX(I),NX(I))=ES(NX(I),NX(I))+TAUFA2/(16*AVEV(M))
      ES(NY(I),NY(I))=ES(NY(I),NY(I))+TAUFA2/(16*AVEV(M))
      ES(NZ(I),NZ(I))=ES(NZ(I),NZ(I))
   DO 200 CONTINUE

* •••
   DO 210 I=1,4
      DO 210 J=I,4
         ES(NX(I),NX(J))=ES(NX(I),NX(J))-TAUFA2*AVVX(M)*AVVX(M)/(256*AVEV3)
         ES(NY(I),NY(J))=ES(NY(I),NY(J))-TAUFA2*AVVY(M)*AVVY(M)/(256*AVEV3)
         ES(NZ(I),NZ(J))=ES(NZ(I),NZ(J))
   DO 210 CONTINUE

* •••
   DO 220 I=1,4
      H(NX(I))=H(NX(I))-TAUFA2*AVVX(M)/(4*AVEV(M))
      H(NY(I))=H(NY(I))-TAUFA2*AVVY(M)/(4*AVEV(M))
      H(NZ(I))=H(NZ(I))
   DO 220 CONTINUE

* IF(IOSP(7).EQ.0) GO TO 500
WRITE(15,* ) ' * ELEMENT NO. =',NEL,' * FACE NO. =',M
WRITE(15,* ) ' * TAV3 =',TAV3,' * TAUFA2 =',TAUFA2
*
500 CONTINUE

600 CONTINUE
RETURN
END
Appendix C

SUBROUTINE POWER

*POWER --- SUBROUTINE POWER ----
SUBROUTINE POWER(NEL,NFACE,TPOWER,SIGEL,EPSEL)
COMMON /PAIVA/ PAI
COMMON /DEBUG/ IDBG(20), IOSP(50)
COMMON /ELMV/ N1, N2, N3, N4, N5, N6, N7, N8
COMMON /NPV/ HEAD(20), IFLAG(20), CLIM, DFACT, ISTIF
*    , INITV, MODEL
COMMON /NDA/ U(24), XJ(8), YJ(8), ZJ(8)
COMMON /FACE/ AVVX(6), AVVY(6), AVVZ(6), CX(6), CY(6), CZ(6),
            AVEV(6), TAUF, AREA(6)
*    /FRICT/ RMN, ITAU
DIMENSION NFACE(6,1)

* *** --- COMPUTES ELEMENT VOLUME ----
* ** --- VOLUME TETRAHEDRON 1 ----
X21=XJ(2)-XJ(1)
X51=XJ(5)-XJ(1)
X41=XJ(4)-XJ(1)
Y21=YJ(2)-YJ(1)
Y51=YJ(5)-YJ(1)
Y41=YJ(4)-YJ(1)
Z21=ZJ(2)-ZJ(1)
Z51=ZJ(5)-ZJ(1)
Z41=ZJ(4)-ZJ(1)
D54=Y51*Z41-Z51*Y41
D24=Y21*Z41-Z21*Y41
D25=Y21*Z51-Z21*Y51
VTH1=ABS(0.166667*(X21*D54-X51*D24+X41*D25))

* ** --- VOLUME TETRAHEDRON 2 ----
X43=XJ(4)-XJ(3)
X73=XJ(7)-XJ(3)
X23=XJ(2)-XJ(3)
Y43=YJ(4)-YJ(3)
Y73=YJ(7)-YJ(3)
Y23=YJ(2)-YJ(3)
Z43=ZJ(4)-ZJ(3)
Z73=ZJ(7)-ZJ(3)
Z23=ZJ(2)-ZJ(3)
D72=Y73*Z23-Z73*Y23
D42=Y43*Z23-Z43*Y23
D47=Y43*Z73-Z43*Y73
VTH2=ABS(0.166667*(X43*D72-X73*D42+X23*D47))

* ** --- VOLUME TETRAHEDRON 3 ----
X58=XJ(5)-XJ(8)
X78=XJ(7)-XJ(8)
X48=XJ(4)-XJ(8)
Y58=YJ(5)-YJ(8)
Y78=YJ(7)-YJ(8)
Y48=YJ(4)-YJ(8)
Z58=ZJ(5)-ZJ(8)
Z78=ZJ(7)-ZJ(8)
Z48=ZJ(4)-ZJ(8)
D74=Y78*Z48-Z78*Y48
D54=Y58*Z48-Z58*Y48
D57=Y58*Z78-Z58*Y78
VTH3=ABS(0.166667*(X58*D74-X78*D54+X48*D57))
* * *  
** VOLUME TETRAHEDRON 4 **

\[
\begin{align*}
X76 &= X(7) - X(6) \\
X56 &= X(5) - X(6) \\
X26 &= X(2) - X(6) \\
Y76 &= Y(7) - Y(6) \\
Y56 &= Y(5) - Y(6) \\
Y26 &= Y(2) - Y(6) \\
Z76 &= Z(7) - Z(6) \\
Z56 &= Z(5) - Z(6) \\
Z26 &= Z(2) - Z(6) \\
\end{align*}
\]

\[
\begin{align*}
DS2 &= Y56 * Z26 - Z56 * Y26 \\
D72 &= Y76 * Z26 - Z76 * Y26 \\
D75 &= Y76 * Z56 - Z76 * Y56 \\
VTH4 &= \text{ABS}(0.166667 \times (X76 * DS2 \times X56 - X26 * D72 + X26 * D75))
\end{align*}
\]

* * *  
** VOLUME TETRAHEDRON 5 **

\[
\begin{align*}
X25 &= X(2) - X(5) \\
X75 &= X(7) - X(5) \\
X45 &= X(4) - X(5) \\
Y25 &= Y(2) - Y(5) \\
Y75 &= Y(7) - Y(5) \\
Y45 &= Y(4) - Y(5) \\
Z25 &= Z(2) - Z(5) \\
Z75 &= Z(7) - Z(5) \\
Z45 &= Z(4) - Z(5) \\
\end{align*}
\]

\[
\begin{align*}
D74 &= Y75 * Z45 - Z75 * Y45 \\
D24 &= Y25 * Z45 - Z25 * Y45 \\
D27 &= Y25 * Z75 - Z25 * Y75 \\
VTH5 &= \text{ABS}(0.166667 \times (X25 * D74 - X75 * D24 + X45 * D27))
\end{align*}
\]

* * *  
** VOLUME ELEMENT **

\[
\begin{align*}
VEL &= VTH1 + VTH2 + VTH3 + VTH4 + VTH5
\end{align*}
\]

*  
FRICPN = 0.0
DEFPN = 0.0

* * *  
** COMPUTES FRICTION POWER **

\[
\text{DO 100 M}=1,6 \\
\text{IF(NFACE(M,NEL).EQ.0) GO TO 100} \\
\text{FRICPN} = \text{FRICPN} + \text{TAUF} * \text{AREA(M)} * \text{AVEV(M)}
\]

100 \text{ CONTINUE}

* * *  
** COMPUTES DEFORMATION POWER **

\[
\text{DEFPN} = \text{SIGEL} * \text{EPSEL} * \text{VEL}
\]

* * *  
** COMPUTES TOTAL ELEMENT POWER **

\[
\text{TPOWER} = \text{TPOWER} + \text{DEFPN} + \text{FRICPN}
\]

RETURN

END
Appendix D
Inputfile for fem3d2.f

TITLE
*** RIPLE CODE (CYLINDRICAL ELEMENT - 70) TEST RUN ***

MODEL
  1 126 70
STAGE
  30
CONVERGENCE
  1.0E-5
PRINT
  1
DEBUG
  0 0
ITERATION
  50
DISPLACEMENT
  2.00
STIF
  1
FRICITION
  1
CON1
  0.33330000 50 0.500000
CON2
  18 0.01 0.00001
PUNDIE
  1000.0 500.0 500.0 900.0 7.0 7.0 -4.0
RING
  100.0 150.0 250.0
ENDPC
NODE
  1 0.00 0.00 100.00
  5 100.00 0.00 100.00 1
  6 0.00 0.00 80.00
 10 100.00 0.00 80.00 1
 11 0.00 0.00 60.00
 15 100.00 0.00 60.00 1
 16 0.00 0.00 40.00
 20 100.00 0.00 40.00 1
 21 0.00 0.00 20.00
 25 100.00 0.00 20.00 1
 26 0.00 0.00 0.00
 30 100.00 0.00 0.00 1
 31 23.10 9.57 100.00
 34 92.39 38.27 100.00 1
 35 23.10 9.57 80.00
 38 92.39 38.27 80.00 1
 39 23.10 9.57 60.00
 42 92.39 38.27 60.00 1
 43 23.10 9.57 40.00
 46 92.39 38.27 40.00 1
 47 23.10 9.57 20.00
 50 92.39 38.27 20.00 1
 51 23.10 9.57 0.00
 54 92.39 38.27 0.00 1
 55 17.68 17.68 100.00
<table>
<thead>
<tr>
<th>58</th>
<th>70.71</th>
<th>70.71</th>
<th>100.00</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>59</td>
<td>17.68</td>
<td>17.68</td>
<td>80.00</td>
<td></td>
</tr>
<tr>
<td>62</td>
<td>70.71</td>
<td>70.71</td>
<td>80.00</td>
<td>1</td>
</tr>
<tr>
<td>63</td>
<td>17.68</td>
<td>17.68</td>
<td>60.00</td>
<td></td>
</tr>
<tr>
<td>66</td>
<td>70.71</td>
<td>70.71</td>
<td>60.00</td>
<td>1</td>
</tr>
<tr>
<td>67</td>
<td>17.68</td>
<td>17.68</td>
<td>40.00</td>
<td></td>
</tr>
<tr>
<td>70</td>
<td>70.71</td>
<td>70.71</td>
<td>40.00</td>
<td>1</td>
</tr>
<tr>
<td>71</td>
<td>17.68</td>
<td>17.68</td>
<td>20.00</td>
<td></td>
</tr>
<tr>
<td>74</td>
<td>70.71</td>
<td>70.71</td>
<td>20.00</td>
<td>1</td>
</tr>
<tr>
<td>75</td>
<td>17.68</td>
<td>17.68</td>
<td>0.00</td>
<td></td>
</tr>
<tr>
<td>78</td>
<td>70.71</td>
<td>70.71</td>
<td>0.00</td>
<td>1</td>
</tr>
<tr>
<td>79</td>
<td>9.57</td>
<td>23.10</td>
<td>100.00</td>
<td></td>
</tr>
<tr>
<td>82</td>
<td>38.27</td>
<td>92.39</td>
<td>100.00</td>
<td>1</td>
</tr>
<tr>
<td>83</td>
<td>9.57</td>
<td>23.10</td>
<td>80.00</td>
<td></td>
</tr>
<tr>
<td>86</td>
<td>38.27</td>
<td>92.39</td>
<td>80.00</td>
<td>1</td>
</tr>
<tr>
<td>87</td>
<td>9.57</td>
<td>23.10</td>
<td>60.00</td>
<td></td>
</tr>
<tr>
<td>90</td>
<td>38.27</td>
<td>92.39</td>
<td>60.00</td>
<td>1</td>
</tr>
<tr>
<td>91</td>
<td>9.57</td>
<td>23.10</td>
<td>40.00</td>
<td></td>
</tr>
<tr>
<td>94</td>
<td>38.27</td>
<td>92.39</td>
<td>40.00</td>
<td>1</td>
</tr>
<tr>
<td>95</td>
<td>9.57</td>
<td>23.10</td>
<td>20.00</td>
<td></td>
</tr>
<tr>
<td>98</td>
<td>38.27</td>
<td>92.39</td>
<td>20.00</td>
<td>1</td>
</tr>
<tr>
<td>99</td>
<td>9.57</td>
<td>23.10</td>
<td>0.00</td>
<td></td>
</tr>
</tbody>
</table>

**ELEMENT**

1  6  7  35  59  1  2  31  55
2  7  8  36  35  2  3  32  31
3  8  9  37  36  3  4  33  32
4  9 10 38  37  4  5  34  33
5 11 12 39  63  6  7  35  59
6 12 13 40  39  7  8  36  35
7 13 14 41  40  8  9  37  36
8 14 15 42  41  9 10  38  37
9 15 16 43  67 11 12  39  63
10 17 18 44  43 12 13  40  39
11 18 19 45  44 13 14  41  40
12 19 20 46  45 14 15  42  41
13 21 22 47  71 16 17  43  67
14 22 23 48  47 17 18  44  43
15 23 24 49  48 18 19  45  44
16 24 25 50  49 19 20  46  45
17 25 26 51  75 21 22  47  71
18 27 28 52  51 22 23  48  47
19 28 29 53  52 23 24  49  48
20 29 30 54  53 24 25  50  49
21 30 35 60  59 31 32  56  55
22 36 37 61  60 32 33  57  56
23 37 38 62  61 33 34  58  57
24 39 40 64  63 35 36  60  59
25 40 41 65  64 36 37  61  60

---

**MEL - Tsukuba - July 1993**

76
Improvements on the 3-D RIPEL Program

| 26 | 41 | 42 | 66 | 65 | 37 | 38 | 62 | 61 |
| 27 | 43 | 44 | 68 | 67 | 39 | 40 | 64 | 63 |
| 28 | 44 | 45 | 69 | 68 | 40 | 41 | 65 | 64 |
| 29 | 45 | 46 | 70 | 69 | 41 | 42 | 66 | 65 |
| 30 | 47 | 48 | 72 | 71 | 43 | 44 | 68 | 67 |
| 31 | 48 | 49 | 73 | 72 | 44 | 45 | 69 | 68 |
| 32 | 49 | 50 | 74 | 73 | 45 | 46 | 70 | 69 |
| 33 | 51 | 52 | 76 | 75 | 47 | 48 | 72 | 71 |
| 34 | 52 | 53 | 77 | 76 | 48 | 49 | 73 | 72 |
| 35 | 53 | 54 | 78 | 77 | 49 | 50 | 74 | 73 |
| 36 | 56 | 59 | 83 | 107 | 6 | 59 | 79 | 103 |
| 37 | 59 | 60 | 84 | 83 | 55 | 56 | 80 | 79 |
| 38 | 60 | 61 | 85 | 84 | 56 | 57 | 81 | 80 |
| 39 | 61 | 62 | 86 | 85 | 57 | 58 | 82 | 81 |
| 40 | 61 | 63 | 87 | 111 | 6 | 59 | 83 | 107 |
| 41 | 63 | 64 | 88 | 87 | 59 | 60 | 84 | 83 |
| 42 | 64 | 65 | 89 | 88 | 60 | 61 | 85 | 84 |
| 43 | 65 | 66 | 90 | 89 | 61 | 62 | 86 | 85 |
| 44 | 66 | 67 | 91 | 115 | 11 | 63 | 87 | 111 |
| 45 | 67 | 68 | 92 | 91 | 63 | 64 | 88 | 87 |
| 46 | 68 | 69 | 93 | 92 | 64 | 65 | 89 | 88 |
| 47 | 69 | 70 | 94 | 93 | 65 | 66 | 90 | 89 |
| 48 | 71 | 72 | 95 | 119 | 16 | 67 | 91 | 115 |
| 49 | 72 | 73 | 97 | 96 | 68 | 69 | 93 | 92 |
| 50 | 73 | 74 | 98 | 97 | 69 | 70 | 94 | 93 |
| 51 | 74 | 75 | 99 | 123 | 21 | 71 | 95 | 119 |
| 52 | 75 | 76 | 100 | 99 | 71 | 72 | 96 | 95 |
| 53 | 76 | 77 | 101 | 100 | 72 | 73 | 97 | 96 |
| 54 | 77 | 78 | 102 | 101 | 73 | 74 | 98 | 97 |
| 55 | 78 | 84 | 108 | 107 | 79 | 80 | 104 | 103 |
| 56 | 83 | 84 | 109 | 108 | 80 | 81 | 105 | 104 |
| 57 | 84 | 85 | 110 | 109 | 81 | 82 | 106 | 105 |
| 58 | 85 | 86 | 110 | 110 | 82 | 107 |
| 59 | 87 | 88 | 112 | 111 | 83 | 84 | 108 | 107 |
| 60 | 88 | 89 | 113 | 112 | 84 | 85 | 109 | 108 |
| 61 | 89 | 90 | 114 | 113 | 85 | 86 | 110 | 109 |
| 62 | 91 | 92 | 116 | 115 | 87 | 88 | 112 | 111 |
| 63 | 92 | 93 | 117 | 116 | 88 | 89 | 113 | 112 |
| 64 | 93 | 94 | 118 | 117 | 89 | 90 | 114 | 113 |
| 65 | 95 | 96 | 120 | 119 | 91 | 92 | 116 | 115 |
| 66 | 96 | 97 | 121 | 120 | 92 | 93 | 117 | 116 |
| 67 | 97 | 98 | 122 | 121 | 93 | 94 | 118 | 117 |
| 68 | 99 | 100 | 124 | 123 | 125 | 95 | 96 | 120 | 119 |
| 69 | 100 | 101 | 125 | 124 | 96 | 97 | 121 | 120 |
| 70 | 101 | 102 | 126 | 125 | 97 | 98 | 122 | 121 |

BOUNDARY

| 1  | 2 |
| 30 | 2 |
| 1  | 1 |
| 26 | 1 |
| 103 | 1 |
| 126 | 1 |
| 26 | 3 |
| 30 | 3 |
| 51 | 3 |
| 54 | 3 |
| 75 | 3 |
| 78 | 3 |
| 99 | 3 |
| 102 | 3 |
| 123 | 3 |
126 3 1
V 1 3

VELOCITY 0.001 0.0 0.001 0.0 -0.04 0.0
PROPERTY 200.0 540.0 0.01 0.155
FRICITION 2 0.3
LOAD C 13 1 0.0

ENDMDC
Eindhoven University of Technology
Faculty of Mechanical Engineering
Section of Production technology and Automation
Laboratory of Forming Technology

Improvements on the 3-D RIPLE program

The implementation of friction and power in a forming simulation program

A.P. Vloemans

july 1993
WPA 1561

Guestresearch at the Mechanical Engineering Laboratory
Agency of Industrial Science and Technology
Ministry of International Trade and Industry
Tsukuba, Japan
Listing of the 3-D RIPEL program (fem3d2.f) including friction and power

```
*MAIN
**
** COMMON A(1600000)
**
** MTOT=1600000
** CALL RIPEL(MTOT)
** STOP

*RIPEL
**
** COMMON A(1600000)
**
** COMMON A(I)
** COMMON /RESULT/ RATIO
** COMMON /PARAM/ ALFA, GGG
** COMMON /PRMT/ ALAM
** COMMON /NODE/ CN,MN
** COMMON /STAG/ NSTAGS
** COMMON /CS/ CSF
**
** MTOT=MMTOT
** NSTAGS=0
**
** ISTIF=1 LAGRANGU MATRIX
** ISTIF=2 PENABITY MATRIX
** ISTIF=3 COMPRESS MATRIX
**
** ALFA=200.000
** GGG =0.01
**
** WRITES CARD IMAGE.
** CALL CIMAGE
**
** READS THE PARAMETER CARDS.
** CALL PCINP
**
** READS THE MODEL DEFINITION CARDS.
** CALL MDINP(A)
**
** INITIAL DATA SET.
** CALL INITA
**
** LOOP OF TOTAL STAGE NUMBER.
**```


**Improvements on the 3-D RIPLE Program**

C

CALL FIG
NSTAGE=IFLAG(5)
DO 3000 NSTA=1,NSTAGE
NUMSTA=NSTA
CSF=0.1
* IF(NSTA GT 1) ISTE=3
* *** Computes the global load vector.
* CALL LOAD(A,NUMSTA)
* *** Loop of perturbated iteration.
RATIO=1.0
NUMITE=IFLAG(6)
DO 1000 NITE=1,NUMITE
NITER=NITE
* *** Computes the global stiffness matrix.
CALL STIF(A,NUMSTA,NITER)
* *** Solves the matrix equation.
CALL CALC(A,NITER)
* ***
CALL AFTER(A,NITER,NSTA)
ALAM=1.0
* *** Judgement ( IFLAG(20)=1 ; Iteration is converged )
IF(IFLAG(20).EQ.1) GO TO 2000
1000 CONTINUE
* 2000 CONTINUE
* *** Preparation for the next stage.
CALL NEXT(A,NUMSTA)
* *** Prints the results.
CALL OUTPUT(A,NITER,NUMSTA)
* IFLAG(20)=0
* WRITE(6,*)
WRITE(6,*)
WRITE(6,*) 'STAGE NO.=',NSTA
3000 CONTINUE
* RETURN
END

*BLKD — Block Data —

* BLOCK DATA
* *** COMMON /FILE/ IN, LP, IKF, ISF
COMMON /DEBUG/ IDBG(20), IOSP(50)
COMMON /LINES/ LTOP, LINE, LEND
COMMON /PAIV/ PAI
COMMON /SCALE/ FACTOR
* ***
DATA IN,LP,15/
DATA IKF,ISF,10,20/
DATA IDBG,IOSP,20,50/
DATA LTOP,LEND,11,60/
DATA PAI,14,1593/
DATA FACTOR,1,6/
END

*PAGE — Subroutine PAGE —

SUBROUTINE PAGE(LINE,LD,LEND)
* FUNCTION (SP. NO. =19)
* TOP OF PAGE IN CASE OF (LINE. GE. LEND)
* CALLED BY
* PCINP
* CALLS
* NONE
* INPUT.
* LINE = current line number.
* LD = Incremental line number.
* LEND = Total line number per page.
* OUTPUT.
* LINE   - NEW LINE NUMBER.
* *  COMMON /FILE/ IN, LP, IKF, ISF
* COMMON /LPVA/ HEAD(20), EFLAG(20), CLIM, DFACI, ISTF
* COMMON /INTY,MODEL
* COMMON LINES/ LTOP, LDUMY1, LDUMY2

** LINE=LINE+1
IF(LINE.LE.LEND) GO TO 100
WRITE(LP,6000) HEAD
100 CONTINUE RETURN
6000 FORMAT(IHI,20A4)
END

** ERRR   - SUBROUTINE ERROR1 ---
* SUBROUTINE ERROR1
* FUNCTION(SP. NO. =18)
* STOP IN CASE OF INVALID COMMAND OPTION.
* CALLED BY:
* PCINP
* CALLS
* NONE
* INPUT & OUTPUT
* NONE
** COMMON /FILE/ IN, LP, IKF, ISF
** WRITE(LP,6000)
** RETURN
6000 FORMAT(IHI,20X, '•• ERROR STOP - INVALID COMMAND OPTION ••')
END

** PCINP   - SUBROUTINE PCINP ---
* SUBROUTINE PCINP
* FUNCTION(SP. NO. =17)
* READS THE PARAMETER CARDS
* CALLED BY:
* * CALLS
* * INPUT & OUTPUT (SEE INPUT MANUAL)
* * INPUT P-CARDS ** OUTPUT VARIABLES *
* TITLE   - HEAD(20)
* MODEL   - EFLAG(3)
* RESTART - EFLAG(2)
* STORE   - EFLAG(1)
* PLOT    - EFLAG(4)
* STAGE   - EFLAG(9)
* ITERATION  - EFLAG(6)
* CONVERGE   - EFLAG(7)
* CONTINUE    - EFLAG(9)
* PRINT     - EFLAG(9)
* PROPERTY   - EFLAG(10)
* VELOCITY   - EFLAG(11)
* FRICTION   - EFLAG(8)
* DISPLACE   - EFLAG(12)
* DEBUG     - IDBG
* COMMON /DEBUG/ IDBG(ZO), IOSP(SO)
** COMMON /FILE/ IN, LP, IKF, ISF
** COMMON /LPVA/ HEAD(20), EFLAG(20), CLIM, DFACI, ISTF
** COMMON /INTY,MODEL
** COMMON /BASEV/ NUMEL, NUMNP, NUMDF, NUMID
** COMMON LINES/ LTOP, LINE, LEND
** COMMON /CONV/ PARI, NCONV, PAR2, NCON2, PAR3
** COMMON /CONV2/ NXX, XXX
** COMMON /PUNVEL/ PW2, PPR, D2, DPR, DCR
** COMMON /VELO/ PW2, PPN, PUNVEL, POSIZ
** COMMON /RING/ RINGT, RINGT2, RW1, RW2
** PARAMETER CARDS SET
CHARACTER CTTL*4, CMODE*4, CREST*4, CSTOR*4,
* CPROP*4, CCONP*4, CVELO*4,
* CCON*4, CPROP*4, CCONP*4, CVELO*4,
* CPROP*4, CCONP*4, CVELO*4,
* CCON*4, CPROP*4, CCONP*4, CVELO*4,
* DATA CTTL, CMODE, CREST, CSTOR, "TITLE", "MODE", "REST";
* DATA CTTL, CMODE, CREST, CSTOR, "MODEL", "CONT";
* DATA CTTL, CMODE, CREST, CSTOR, "PLOT", "STAG";
* DATA CTTL, CMODE, CREST, CSTOR, "PRINT", "PROP";
* DATA CTTL, CMODE, CREST, CSTOR, "CONV2";
* DATA CTTL, CMODE, CREST, CSTOR, "RING", "RING2";
* DATA CTTL, CMODE, CREST, CSTOR, "RING", "RING2";
* DATA CHARACTERS SET
** COMMON /PUNVEL/ PW2, PPR, D2, DCR
** COMMON /VELO/ PW2, PPN, PUNVEL, POSIZ
** COMMON /RING/ RINGT, RINGT2, RW1, RW2
** PARAMETER CARDS SET
CHARACTER CTTL*4, CMODE*4, CREST*4, CSTOR*4,
* CPROP*4, CCONP*4, CVELO*4,
* CCON*4, CPROP*4, CCONP*4, CVELO*4,
* CPROP*4, CCONP*4, CVELO*4,
* CCON*4, CPROP*4, CCONP*4, CVELO*4,
* DATA CTTL, CMODE, CREST, CSTOR, "TITLE", "MODE", "REST";
* DATA CTTL, CMODE, CREST, CSTOR, "MODEL", "CONT";
* DATA CTTL, CMODE, CREST, CSTOR, "PLOT", "STAG";
* DATA CTTL, CMODE, CREST, CSTOR, "PRINT", "PROP";
* DATA CTTL, CMODE, CREST, CSTOR, "CONV2";
* DATA CTTL, CMODE, CREST, CSTOR, "RING", "RING2";
* DATA CTTL, CMODE, CREST, CSTOR, "RING", "RING2";
* DATA CHARACTERS SET
** COMMON /PUNVEL/ PW2, PPR, D2, DCR
** COMMON /VELO/ PW2, PPN, PUNVEL, POSIZ
** COMMON /RING/ RINGT, RINGT2, RW1, RW2
** PARAMETER CARDS SET
CHARACTER CTTL*4, CMODE*4, CREST*4, CSTOR*4,
* CPROP*4, CCONP*4, CVELO*4,
* CCON*4, CPROP*4, CCONP*4, CVELO*4,
* CPROP*4, CCONP*4, CVELO*4,
* CCON*4, CPROP*4, CCONP*4, CVELO*4,
* DATA CTTL, CMODE, CREST, CSTOR, "TITLE", "MODE", "REST";
* DATA CTTL, CMODE, CREST, CSTOR, "MODEL", "CONT";
* DATA CTTL, CMODE, CREST, CSTOR, "PLOT", "STAG";
* DATA CTTL, CMODE, CREST, CSTOR, "PRINT", "PROP";
* DATA CTTL, CMODE, CREST, CSTOR, "CONV2";
* DATA CTTL, CMODE, CREST, CSTOR, "RING", "RING2";
* DATA CTTL, CMODE, CREST, CSTOR, "RING", "RING2";
* DATA CHARACTERS SET
** COMMON /PUNVEL/ PW2, PPR, D2, DCR
** COMMON /VELO/ PW2, PPN, PUNVEL, POSIZ
** COMMON /RING/ RINGT, RINGT2, RW1, RW2
** PARAMETER CARDS SET
CHARACTER CTTL*4, CMODE*4, CREST*4, CSTOR*4,
* CPROP*4, CCONP*4, CVELO*4,
* CCON*4, CPROP*4, CCONP*4, CVELO*4,
* CPROP*4, CCONP*4, CVELO*4,
* CCON*4, CPROP*4, CCONP*4, CVELO*4,
* DATA CTTL, CMODE, CREST, CSTOR, "TITLE", "MODE", "REST";
* DATA CTTL, CMODE, CREST, CSTOR, "MODEL", "CONT";
* DATA CTTL, CMODE, CREST, CSTOR, "PLOT", "STAG";
* DATA CTTL, CMODE, CREST, CSTOR, "PRINT", "PROP";
* DATA CTTL, CMODE, CREST, CSTOR, "CONV2";
* DATA CTTL, CMODE, CREST, CSTOR, "RING", "RING2";
* DATA CTTL, CMODE, CREST, CSTOR, "RING", "RING2";
* DATA CHARACTERS SET
** COMMON /PUNVEL/ PW2, PPR, D2, DCR
** COMMON /VELO/ PW2, PPN, PUNVEL, POSIZ
** COMMON /RING/ RINGT, RINGT2, RW1, RW2
** PARAMETER CARDS SET
CHARACTER CTTL*4, CMODE*4, CREST*4, CSTOR*4,
* CPROP*4, CCONP*4, CVELO*4,
* CCON*4, CPROP*4, CCONP*4, CVELO*4,
** Improvements on the 3-D RIPLE Program **

```plaintext
** ** INITIAL DATA SET
LINE=4,TOP
CALL ICLEAR(20,DBG)
CALL ICLEAR(20,IFLAG)
**C CALL ICLEAR(90,IOSP)
** ** DEFAULT VALUE SET.
NUMNP=1000
NUMEL=100
CLIM=.5E-4
** ** READS THE PARAMETER CARDS.
100 READ(N,4000) CHAR
   IF(CHAR.EQ.CENDP) GO TO 300
   IF(CHAR.NE.CTITL) GO TO 110
** ** TITLE
READ(N,5000) HEAD
WRITE(LP,6000) HEAD
WRITE(3I,6001) HEAD
GO TO 100
**
110 CONTINUE
IF(CHAR.NE.CMODE) GO TO 120
** ** MODEL
READ(N,5010) IFLAG(I), NUMNP, NUMEL
IF(NUMNP.LE.O) NUMNP=1000
IF(NUMEL.LE.O) NUMEL=100
CALL PAGE(LINE,2,LEND)
WRITE(LP,6010) IFLAG(I), NUMNP, NUMEL
IF(IFLAG(I).NE.I) CALL ERROR1
GO TO 100
**
120 CONTINUE
IF(CHAR.NE.CREST) GO TO 130
** ** RESTART
READ(N,5010) IFLAG(I)
WRITE(LP,6020) IFLAG(I)
IF(IFLAG(I).NE.0) CALL ERROR1
GO TO 100
**
130 CONTINUE
IF(CHAR.NE.CSTOR) GO TO 140
** ** STORE
CALL PAGE(LINE,5,LEND)
IFLAG(3)=1
WRITE(LP,6030) IFLAG(3)
CALL ERROR1
GO TO 100
**
140 CONTINUE
IF(CHAR.NE.CPLOT) GO TO 150
** ** PLOT
CALL PAGE(LINE,5,LEND)
IFLAG(4)=1
WRITE(LP,6040) IFLAG(4)
CALL ERROR1
GO TO 100
**
150 CONTINUE
IF(CHAR.NE.CSTAG) GO TO 160
** ** STAGE
READ(N,5010) IFLAG(I)
CALL PAGE(LINE,4,LEND)
WRITE(LP,6050) IFLAG(I)
GO TO 100
**
160 CONTINUE
IF(CHAR.NE.CITER) GO TO 170
** ** ITERATION
READ(N,5010) IFLAG(I)
CALL PAGE(LINE,4,LEND)
WRITE(LP,6060) IFLAG(I)
GO TO 100
**
170 CONTINUE
IF(CHAR.NE.CCONV) GO TO 180
** ** CONVERGENCE
READ(N,5020) CLIM
IFLAG(7)=1
CALL PAGE(LINE,4,LEND)
WRITE(LP,6070) CLIM
GO TO 100
**
180 CONTINUE
```
Improvements on the 3-D RIPLE Program

IF(CHAR,NE,CON1) GO TO 190
*** CONTINUE
  **.Flag**(1) = 1
  CALL PAGE(LINE,7,LEND)
  WRITE(LP,6080) **Flag**(8)
  GO TO 100
***
190 CONTINUE
IF(CHAR,NE,CPRIN) GO TO 200
*** PRINT
READ(IN,5010) **Flag**(9)
CALL PAGE(LINE,7,LEND)
WRITE(LP,6090) **Flag**(9)
GO TO 100
***
200 CONTINUE
IF(CHAR,NE,CPROP) GO TO 210
*** PROPERTY
**Flag**(10) = 1
CALL PAGE(LINE,7,LEND)
WRITE(LP,6100) **Flag**(10)
GO TO 100
***
210 CONTINUE
IF(CHAR,NE,CVELO) GO TO 220
*** VELOCITY
**Flag**(11) = 1
CALL PAGE(LINE,7,LEND)
WRITE(LP,6110) **Flag**(11)
GO TO 100
***
220 CONTINUE
IF(CHAR,NE,CFRIC) GO TO 230
*** FRICTION
READ(IN,5010) **Flag**(12)
IF (**Flag**(12),EQ.2) **Flag**(12) = 1
CALL PAGE(LINE,6,LEND)
WRITE(LP,6120) **Flag**(12)
IF (**Flag**(12),NE,0) CALL ERROR I
GO TO 100
***
230 CONTINUE
IF(CHAR,NE,CDISP) GO TO 240
*** DISPLACEMENT
READ(IN,5020) DFACT
**Flag**(13) = 1
CALL PAGE(LINE,4,LEND)
WRITE(LP,6130) DFACT
GO TO 100
***
240 CONTINUE
IF(CHAR,NE,CDEBU) GO TO 250
*** DEBUG
READ(IN,5030) IDBG(1), IDBG(2)
CALL PAGE(LINE,4,LEND)
WRITE(LP,6140) IDBG(1), IDBG(2)
GO TO 100
250 CONTINUE
IF(CHAR,NE,CSTIF) GO TO 260
*** STIF
READ(IN,5040) ISTIF
CALL PAGE(LINE,4,LEND)
WRITE(LP,6150) ISTIF
GO TO 100
260 CONTINUE
IF(CHAR,NE,CON2) GO TO 265
*** CONV
READ(IN,5050) PAR1,NCON1,PAR2,NCON2,PAR3
CALL PAGE(LINE,4,LEND)
WRITE(LP,6160) PAR1,NCON1,PAR2,NCON2,PAR3
GO TO 100
265 CONTINUE
IF(CHAR,NE,CPUND) GO TO 270
*** ITERATION
READ(IN,5060) PW2,PR2,DW2,DR2,DF2
CALL PAGE(LINE,4,LEND)
WRITE(LP,6170) PW2,PR2,DW2,DR2,DF2
GO TO 100
***
270 CONTINUE
IF(CHAR,NE,CON2) GO TO 275
*** MODEL
READ(IN,5060) NX,XX,XXX
CALL PAGE(LINE,4,LEND)
WRITE(LP,6180) NX,XX,XXX
GO TO 100
275 CONTINUE
IF(CHAR.NE.CRlNG) GO TO 280
*** RING, BLANK AND BLANKHOLDER
READ(IN,5070) BL1H,RlNGTH,BHWI
CALL PAGE(LINE,4,LEND)
WRITE(LP,6066) BL1H,RlNGTH,BHWI
IF(BHWI.EQ.0.0) THEN
  BH=0
ELSE
  BH=1
END IF
GO TO 100
280 CONTINUE
*** INVALID COMMAND DATA
WRITE(LP,6150) CHAR
STOP
*** DEFAULT COMMAND DATA SET
300 CONTINUE
CALL PAGE(LINE,6,LEND)
WRITE(LP,6200)
*** MODEL
IF(IFLAG(1).NE.0) GO TO 310
IFLAG(1)=4
CALL PAGE(LINE,6,LEND)
WRITE(LP,6010) IFLAG(1) , NUMNP , NUMEL
310 CONTINUE
*** RESTART
IF(IFLAG(2).NE.0) GO TO 320
CALL PAGE(LINE,6,LEND)
WRITE(LP,6020) IFLAG(2)
320 CONTINUE
*** STORE
IF(IFLAG(3).NE.0) GO TO 330
CALL PAGE(LINE,6,LEND)
WRITE(LP,6030) IFLAG(3)
330 CONTINUE
*** PLOT
IF(IFLAG(4).NE.0) GO TO 340
CALL PAGE(LINE,6,LEND)
WRITE(LP,6040) IFLAG(4)
340 CONTINUE
*** STAGE
IF(IFLAG(5).NE.0) GO TO 350
IFLAG(5)=50
CALL PAGE(LINE,6,LEND)
WRITE(LP,6050) IFLAG(5)
350 CONTINUE
*** ITERATION
IF(IFLAG(6).NE.0) GO TO 360
IFLAG(6)=20
CALL PAGE(LINE,6,LEND)
WRITE(LP,6060) IFLAG(6)
360 CONTINUE
*** CONVERGE
IF(IFLAG(7).NE.0) GO TO 370
CALL PAGE(LINE,6,LEND)
WRITE(LP,6070) CLiM
370 CONTINUE
*** CONTINUE
IF(IFLAG(8).NE.0) GO TO 380
CALL PAGE(LINE,7,LEND)
WRITE(LP,6080) IFLAG(8)
380 CONTINUE
*** PRINT
IF(IFLAG(9).NE.0) GO TO 390
CALL PAGE(LINE,7,LEND)
WRITE(LP,6090) IFLAG(9)
390 CONTINUE
*** PROPERTY
IF(IFLAG(10).NE.0) GO TO 400
CALL PAGE(LINE,7,LEND)
WRITE(LP,6100) IFLAG(10)
400 CONTINUE
*** VELOCITY
IF(IFLAG(11).NE.0) GO TO 410
CALL PAGE(LINE,7,LEND)
WRITE(LP,6110) IFLAG(11)
410 CONTINUE
*** FRICTION
**Improvements on the 3-D RIPLE Program**

```fortran
*  IF(LFLAG(12).NE.0) GO TO 420
   CALL PAGE(LINE,6,LEND)
   WRITE(LP,6120) IFLAG(12)
420 CONTINUE

*  DISPLACE
   IF(LFLAG(IJ).NE.O) GO TO 430
   CALL PAGE(LINE, 4 ,LEND)
   DFACT=0
   WRITE(LP,61J0) DFACT
430 CONTINUE

END OF PARAMETER CARDS
   WRITE(LP,6300)

OUTPUT DATA FOR POST.
   DO 10 K = 1,36
      ICNTR(K)=0
10 CONTINUE
   ICNTR(I)=3
   ICNTR(2)=NUMNP
   IFLAG9=IFLAG(9)
   IF(FLAG9.EQ.0) THEN
      ICNTR(4)=IFLAGS
   ELSE
      ICNTR(4) = IFLAG(9)
   END IF
   ICNTR(5)=3
   ICNTR(6)=1
   ICNTR(7)=1
   ICNTR(19)=NUMEL
   ICNTR(20)=1
   ICNTR(26)=1
   WRITE(J1,6002) (ICNTR(1),I=1,20)
   WRITE(J1,6003) (ICNTR(1),I=21,36)
   RETURN

*** READ FORMAT
3000 FORMAT(H1,8X,9X)
   * " **** PARAMETER CARDS ****">
   * "9X, --- INPUT PARAMETER CARDS ----")
6000 FORMAT(H1,8X,„ TITLE‘,5X,20A4)
6001 FORMAT(20A4)
6002 FORMAT(20A4)
6003 FORMAT(20A4)
6010 FORMAT(H10,8X,* MODEL»,8X,* ‘15/I3X,EQ.1 , AXISYMMETR’
   * ’,C‘,I3X,EQ.2 , PLANE STRESS’,I3X,EQ.3 , PLANE STRAIN’,
   * ‘,I3X,TOTAL NUMBER OF NODE =’,15,
   * ‘,I3X,TOTAL NUMBER OF ELEMENT =’)
6020 FORMAT(H10,8X,* ‘RESTART’,6X,* ,15/I3X,EQ.0 , START STAGE NO.’)
6030 FORMAT(H10,8X,* ‘STORE’,6X,* ‘0 , NO DATA STORE’)
   * ‘,I3X,EQ.1 , DATA STORE (SEE H.D. CARDS )’)
6040 FORMAT(H10,8X,* ‘PLOT’,5X,* ,15/I3X,EQ.0 , NO PLOT’)
   * ‘,I3X,EQ.1 , PLOT (SEE H.D. CARDS )’)
6050 FORMAT(H10,8X,* ‘STAGE’,5X,* ,15/I3X,EQ.0 , START STAGE’)
   * ‘,TOTAL NUMBER OF CALCUATION STAGE’)
6060 FORMAT(H10,8X,* ‘ITERATION’,4X,* ,15/I3X,EQ.1 , ITERATION FOR EACH STAGE’)
6065 FORMAT(H10,8X,* ‘PUNCH WIDTH’,2X,* ,15/F6.2X,’PUNCH PROFILE’
   * ‘,RADIUS’,2X,* ,15/F6.2X,’ DIE WIDTH’,2X,* ,15/F6.2X,’ DIF PROFILE’
   * ‘,RADIUS’,2X,* ,15/F6.2X,’ DIE CORNER RADIUS’,2X,’
   * ‘,PUNCH CORNER RADIUS’,2X,’
   * ‘,PUNCH CORNER RADIUS’,2X,’
   * ‘,RADIUS’,2X,* ,15/F6.2X,’ DIE CORNER RADIUS’,2X,’
   * ‘,PUNCH CORNER RADIUS’,2X,’
6066 FORMAT(H10,8X,* ‘Blank thickness’ =’,F6.2,’Ring thickness =’
   * ‘,F6.2,’Blankholder width =’,F6.2)
6070 FORMAT(H10,8X,* ‘CONVERGE’,5X,* ,15/I1X,EQ.1 , CONVERGE (SEE H.D. CARDS )’)
6080 FORMAT(H10,8X,* ‘TOLERANCE OF ITERATION FOR EACH STAGE’)
6085 FORMAT(H10,8X,* ‘FLAG FOR CONTINUATION IN CASE OF NOT-CONVERGED’,
   * ‘,I1X,EQ.0 , STOP’,
   * ‘,I1X,EQ.1 , CONTINUE THE NEXT STAGE’)
6090 FORMAT(H10,8X,* ‘PRINT’,8X,* ,15/I1X,EQ.1 , PRINT THE RESULTS’,
   * ‘,I1X,EQ.0 , FINAL STAGE ONLY’,
   * ‘,I1X,NEQ.0 , PRINT INTERVAL’)
```

— MEL - Tsukuba - July 1993 —

8
Improvements on the 3-D RIPLE Program

6100 FORMAT(100,8X)** PROPERTY**: SX='J5./,13X,
* DEFINES THE EQUIVALENT STRESS - EQUIVALENT STRAIN RELATION,
* (/13X,'EQUVALENT STRESS - EQUIVALENT STRAIN RELATION',
* (/13X,'EQ.1, USERS SUBROUTINE')
6110 FORMAT(100,8X)** VELOCITY**: SX='J5./,13X,
* DEFINES THE INITIAL VELOCITY FIELD,
* (/13X,'EQUVALENT STRESS - EQUIVALENT STRAIN RELATION',
* (/13X,'EQ.1, USERS SUBROUTINE')
6120 FORMAT(100,8X)** FRICTION**: SX='J5./,13X,
* 'DEFINES SIMPLE FRICTION MODEL',
* (/13X,'EQ.0. SLIP MODEL',13X,'EQ.1, SIMPLE FRICTION MODEL',
* (/13X,'EQ.2. DETAILED FRICTION MODEL')
6130 FORMAT(100,8X)** DISPLACE**: SX='J5./,13X,
* 'MAXIMUM DISPLACEMENT RATIO FOR EACH STAGE',
6140 FORMAT(100,8X)** DEBUG**: SX='J5./,13X,
* 'FLAG FOR DEBUG PRINT AS FOLLOWS
* 'J5./,13X,'SOLVER INFORMATION',
* 'J5./,13X,'COMMON AREA INFORMATION',
* 'J5./,13X,'NORMAL DISPLACEMENTS & COMMONS'
6150 FORMAT(100,8X)** ERROR STOP - INVALID COMMAND DATA =',SX,A4,
* SX,
6160 FORMAT(100,8X)** STIFF MATRIX=',IS,
6170 FORMAT(100,10X)** PAR1=',F10.5,' PAR2=',F10.5,
6180 FORMAT(100,10X)** COMMON AREA INFORMATION',
6190 FORMAT(100,10X)** COMMON AREA INFORMATION',
6200 FORMAT(100,10X)** DEFAULT PARAMETER CARDS ---**
6200 FORMAT(100,10X)** END OF P.C. ---**
END

**ICLR** -- SUBROUTINE ICLEAR ---

SUBROUTINE ICLEAR(MAX,IA)
* FUNCTION(SEP. NO. =20)
* CLEAR THE AREA IA(1)-IA(MAX)
* CALLED BY
* PCINP
* CALLS
* NONE
* INPUT.
* IA(1) - IA(MAX)
* OUTPUT.
* IA(1) = 0
* IA(MAX) = 0
* DIMENSION IA(1)
** CLEAR
DO 10 I=1,IA
IA(I)=0
10 CONTINUE
RETURN
END

**CHECK** -- SUBROUTINE CHECK ---

SUBROUTINE CHECK(MTOT,MEND,CHAR1,CHAR2)
* FUNCTION(SEP. NO. =22)
* CHECK THE BLANK COMMON SIZE.
* CALLED BY
* PCINP
* CALLS
* NONE
* INPUT.
* MTOT=MEND=COMMON SIZE OF BLANK COMMON,
* CHAR1 = FIRST TERM OF THE TITLE
* CHAR2 = SECOND TERM OF THE TITLE
* OUTPUT.
* COMMON FILE IN .LP, ISF
** COMMON FILE IN .LP
* WRITE(LP,6000) CHAR1,CHAR2,MTOT,MEND
STOP
10 CONTINUE
RETURN
END

--- MEL - Tsukuba - July 1993 ---
Improvements on the 3-D RIPLE Program

*MDIN — SUBROUTINE MDINP —

SUBROUTINE MDINP(A)
*FUNCTION(SP.NO.=-21)
*READS THE MODEL DEFINITION CARDS.
*CALLED BY:
*CALLS:
*CHECK,NODGEN,ELEGEN,BONGEN

*INPUT & OUTPUT:
* * INPUT M.D.-CARDS * - * OUTPUT VARIABLES*
* NODE — RN(JN(JMP)) AND Z(JN(JMP))
* ELEMENT — IX(JN(JMEL))
* BOUNDARY — IFIX(JUMDF)
* LOAD — PLOAD(JMDF,J)
* PROPERTY — Y(J), CK, CM, CN
* VELOCITY — A(J), B(J), A(J), B(J), A(J), B(J)
* FRICTION — IJTAU, RMV

*** COMMON /FILE INCDF, ISP, ISF
COMMON /DEBUG IDBG(20), IDSP(50)
COMMON /COEFF Y(J), CM, CN, A(J), B(J), A(J), B(J), A(J), B(J)
COMMON /ADDRESS MTOT, MMAX, MMNS, MMDF, MMNE, MMES,
* MMEE, MMBS, MMBE, MMLS, MMML, MMMLM2, MMLE,
* MMVS, MMVW, MMVE, MMIS, MMEM, MMIE, MMIM,
* MMNU, MMNV, MMBS1, MMNW
COMMON /BASAYA NUMEL, NUMNP, NUMDF, NUMLD
COMMON /LINES LTOP, LINE, LEND
COMMON /IFRICT RMV, ITAU
COMMON /INPV HEAD(20), IFLAG(20), CUM, DFACT, ISTIF
* , INITV, MODEL
COMMON /GTI TIGRI, TIGR2, TIGZ1, TIGZ2, DEGI, DEG2
DIMENSION A(I)

MODEL DEFINITION CARDS SET.
CHARACTER CNODE'4,CELEM'4,CBOUN'4,CLOAD'I
CPROP'I, CVELO'I, CFRIC'I, CENDM'I
DATA CNODE,CELEM,CBOUN,CLOAD'I
DATA CPROP, CVELO, CFRIC, CENDM'I
DATA CTIG'I
WRITE HEADER PRINT.
WRITE(LP,6000)
*** INITIAL DATA SET.
LINE=1
MMAX=0
*** READS THE MODEL DEFINITION CARDS
100 READ(IN,5000) CHAR
IF(CHAR.EQ.CENDM) GO TO 300
IF(CHAR.NE.CNODE) GO TO 110

*** NODE
MMNS=MMAX+1
MMNS=MMNS+NUMNP
MMNN=MMNS+NUMNP
MMNE=MMNN+NUMNP-1
CALL PAGE(LINE, LEND)
CALL CHECK(MTOT,MMNE,ELEM)
*** GENERATES THE NODEAL POINT DATA
CALL NODGEN(NUMNP,LINEL,ELEND, 0, AMMNS, AMMNS, AMMNS)
MMAX=MMNE
**

110 CONTINUE
IF(CHAR.NE.CELEM) GO TO 120

*** ELEMENT
MMES=MMAX+1
MMEM=MMES+NUMEL
MMME=MMEM+NUMEL
MMEE=MMME+NUMEL-1
CALL PAGE(LINE, LEND)
CALL CHECK(MTOT,MMEE,ELEM,ELEM)
*** GENERATES THE ELEMENT DATA
CALL ELEGEN(NUMEL,LINEL,ELEND, 0, AMMME, AMMME)
MMAX=MMEE
**

120 CONTINUE
IF(CHAR.NE.CBOUN) GO TO 130

*** BOUNDARY
IF(ISTIF.EQ.1) THEN
END IF
MMBS=MMAX+1
MMBS1+MMBS+NUMDF

MEL - Tsukuba - July 1993
Improvements on the 3-D RIPLE Program

MMUW = MMBS1 + NDF
MMBE = MMBS1 + NDF - 1
CALL PAGE(LINE, LEND)
CALL CHECK(MTOT, MMBE, 'BOUNDARY')

*** Generates the Boundary Condition Data.
CALL BONGEN(NUMBC, NDF, LINE, LEND, 0, A(MMBS), A(MMBS1))
MAX = MMBE
GO TO 100

130 CONTINUE
IF (CHAR .NE. 'LOAD') GO TO 140

*** Load
NUMNP = NUMNP + 3
MMLE = MMAX + 1
MMLM = MMLM + NUMNP

CALL PAGE(LINE, LEND)
CALL CHECK(MTOT, MMLE, 'LOAD')

Generates the Boundary Condition Data.
CALL BONGEN(NUMBC, NDF, LINE, LEND, 0, A(MMBS), A(MMBS1))
MAX = MMBE
GO TO 100

140 CONTINUE

*** Property
READ(IN, 5010) YO, CK, CM, CN
CALL PAGE(LINE, LEND)
WRITE(LP, 6040) CK, CM, CN, YO

GO TO 100

*** Velocity
READ(IN, 5010) AI, BI, A2, B2, AJ, BJ
CALL PAGE(LINE, LEND)
WRITE(LP, 6050) AI, BI, A2, B2, AJ, BJ

GO TO 100

*** Friction
READ(IN, 5020) ITAU, RMN
CALL PAGE(LINE, LEND)
WRITE(LP, 6080) ITAU
IF (ITAU.EQ.1) WRITE(LP, 6090) RMN
IF (ITAU.EQ.2) WRITE(LP, 6100) RMN

GO TO 100

*** Fix
READ(IN, 5010) TIG1, TIG2, TIG3, TIG4, DEG1, DEG2

GO TO 100

*** Invalid Command Data
WRITE(LP, 6070) CHAR
STOP

*** End of Model Definition Cards.
WRITE(LP, 6000) CHAR
RETURN

*** Read Format
5000 FORMAT(A4)
5010 FORMAT(FI0.0)
5020 FORMAT(I0, I0)

*** Write Format
6000 FORMAT(IHI, HI, 9X, 'MODEL DEFINITION')
   'CARS ***',
   'INPUT MODEL DEFINITION CARDS ---')
6040 FORMAT(HI, 9X, 'PROPERTY', HI, 'DEFINES THE EQUIVALENT STRESS')
   'EQUIVALENT STRAIN RELATION', HI, 'K * (EPS)** M * (EPS)** N + YO',
   '/11x', 'WHERE K = ', IPEI0.3, 'M = ', IPEI0.3,
   'N = ', IPEI0.3, '20x', 'Y0 = ', IPEI0.3)

--- MEL - Tsukuba - July 1993 ---

11
Improvements on the 3-D RIPLE Program

6050 FORMAT(1H0,8X, 'VELOCITY',
       * I13X,'DEFINES THE INITIAL VELOCITY FIELD',
       * I13X,'( VX ) = A1 ( X ) + B1',
       * I13X,'( VY ) = A2 ( Y ) + B2',
       * I13X,'( VZ ) = A3 ( Z ) + B3',
       * I13X,'WHERE A1 = 'P103, B1 = 'P103,
       * I13X,'A2 = 'P103, B2 = 'P103,
       * I13X,'A3 = 'P103, B3 = 'P103)

6070 FORMAT(1H0,8X, 'ERROR STOP - INVALID COMMAND DATA=', 5X, A4, * 5X, '***')

6080 FORMAT(1H0,8X, 'FRICTION',
       * I13X,'DEFINES THE FRICTION MODEL',
       * I13X,'ITAU = 'P103, 'SIG ')

6090 FORMAT(1H0,13X, 'TAU = 'P103, 'SIG ')

6300 FORMAT(1H0,11X, 'END OF M.D.C. ***')

* ELGN  -- SUBROUTINE ELEGEN  --

* SUBROUTINE ELEGEN(NUMEL,LLINE,LLEND,DUMY,IX,NFACE)
* FUNCTION (SP. NO. = 24)
* GENERATES THE ELEMENT DATA.
* CALLED BY
* MDINP
* CALLS
* PAGE
* INPUT
* NUMEL  = TEMPORARY NUMBER OF ELEMENTS.
* LLINE  = CURRENT LINE NUMBER.
* LLEND  = TOTAL LINE NUMBER PER PAGE.
* OUTPUT
* NUMEL  = TOTAL NUMBER OF ELEMENTS
* IX  = NODE NUMBER COMPOSED THE ELEMENT.
* NFACE  = SURFACE NUMBER OF FRICTION
* *** INITIAL DATA SET.
* DUMY=0.0
* LINE=LLINE
* LEND=LLEND
* NUME=0
* *** READS THE ELEMENT DATA.
* READ(5,5000) N, (IX(J,N),1=I,S) , NF • KE
* DO 10 I=1,6
* NF=NF+I
* CONTINUE
* READ(5,5000) N, IXN , NF , KE
* IF(N.EQ.0) GO TO 500
* DO 150 I=1,8
* IX(I)=IXNO)
* CONTINUE
* DO 151 1=1,6
* NFACE(1)=NFACE(1)
* CONTINUE
* IF(K.EQ.0) GO TO 400
* NUME=NUME+I
* CONTINUE
* IF(K.EQ.0) GO TO 400
* *** GENERATES THE ELEMENT DATA.
* NUME=NUME+K
* NUME=ABS(NUME)
* IF(NUME.EQ.0) GO TO 400
* K=1
* CONTINUE
* DO 300 I=1,NUM
* K=K+1
* CONTINUE
* DO 200 I=1,6
* NFACE(K)=NFACE(K)
* CONTINUE
* *** MEL - Tsukuba - July 1993
Improvements on the 3-D RIPPLE Program

400 CONTINUE
CALL PAGE(LINE,1,LEND)
WRITE(LP,6010) N , (DX(NJ),J=1,3) , (DFACE(NJ),J=1,6) , KE
NOLD=N
GO TO 100
*** CALCULATES NUMEL.
500 CONTINUE
NUMEL=NUME
CALL PAGE(LINE,1,LEND)
WRITE(LP,6030) NUMEL
LLINE=LINE
DO 600 K = 1,NUMEL
WRITE(10,6020) K , X(K), Y(K), Z(K)
600 CONTINUE
RETURN
**** READ FORMAT
5000 FORMAT(I6,B5,I4,X,6I1,35)
**** WRITE FORMAT
6000 FORMAT(H0,X,' • ELEMENT 1',/9X,
  • ELEMENT DATA -/11X,
  • ELEMENT NODE I NODE-J NODE-K NODE-L,
  • NODE-NN NODE-NO NODE-P,IX,YFACE,
  • AX,KE,5X,3X,6I1,1X,35)
6002 FORMAT(6I5)
6010 FORMAT(H ,5X,3X,6I1,1X,35)
6020 FORMAT(H ,5X,3X,6I1,5X,'G')
6030 FORMAT(H0,X,TOTAL NUMBER OF ELEMENTS =',IS)
END
*NOGN — SUBROUTINE NOGEND —
* FUNCTION(SP. NO. =23)
* GENERATES THE NODAL POINT DATA.
* CALLED BY:
* MDINP
* CALLS:
* PAGE
* INPUT:
* NUMNP = TEMPORARY NUMBER OF NODAL POINTS.
* LLINE - CURRENT LINE NUMBER.
* LLEND - TOTAL LINE NUMBER PER PAGE.
* OUTPUT:
* NUMNP = TOTAL NUMBER OF NODAL POINTS.
* X - RADIAL COORDINATE OF NODAL POINT.
* Y -
* Z = AXIAL COORDINATE OF NODAL POINT.
*** COMMON /FL/ IN, LP, IKF, ISF
COMMON /XYZCOR/ XCOR(IO00), YCOR(IO00), ZCOR(IO00)
DIMENSION X(I), Y(I), Z(I)
*** INITIAL DATA SET.
DUMY=0.0
LINE=LLINE
LEND=LLEND
NUMND=0
*** READS THE NODAL POINT DATA.
READ(NK,1000) N , X(N), Y(N), Z(N), KN
CALL PAGE(LINE,LEND)
WRITE(LP,6000) N , X(N), Y(N), Z(N), KN
NOLD=N
NUMND+NUMND+1
100 READ(NK,1000) N , X(N), Y(N), Z(N), KN
IF(NK.EQ.0) GO TO 400
X(N)=X(N)
Y(N)=Y(N)
Z(N)=Z(N)
IF(NK.EQ.0) GO TO 100
*** GENERATES THE NODAL POINT DATA.
NUM=NEW-NOLD
NUMN=NUM-1
DNUM=FLOAT(NUM)
DY=(Y(N)-YN)/DNUM
DZ=(Z(N)-ZN)/DNUM
K=NOLD
DO 200 J=1,NUMN
X(K)=DX
Y(K)=DY
Z(K)=DZ
CALL PAGE(LINE,LEND)
WRITE(LP,6020) K , X(K), Y(K), Z(K)
NUMND=NUMND+1

MEL - Tsukuba - July 1993

13
*** WRITE FORMAT
6000 FORMAT(1H10,1X,* NODE NO ,19X,
   * NODE NO. X-COORDINATE Y-COORDINATE Z-COORDINATE,
   * Z-COORDINATE,10X,KN,
   * 10X,1SPE10.3,2X,1PE10.3,2X,1PE10.3,3X,1S)
6002 FORMAT(3,IE15.0)
6010 FORMAT(1H8,1S,1X,5,1S,1PE10.3,4X,1PE10.3,4X,1PE10.3,4X,1PE10.3,4X)
6015 FORMAT(1H8,1S,1X,5,1S,1PE10.3,4X,1PE10.3,4X,1PE10.3,4X,1PE10.3,4X,1S)
6016 FORMAT(1H8,1S,1PE10.3,3X,1PE10.3,3X,1PE10.3,3X,1PE10.3,3X,1S)

--- MEL - Tsukuba - July 1993 ---
IMPROVEMENTS ON THE 3-D RIPLE PROGRAM

```plaintext
IFIX(NUMY) = -1
IF(CHAR.EQ.CV) IFIX(NUMY) = -2
NUMBC = NUMBC + 1
CALL PAGE(LINE,LEND)
WRITE(LP,6010) CHAR, NUMBC, N, IC(I), KB
GO TO 30
20 CONTINUE
IFIX(NUMX) = -1
IF(CHAR.EQ.CV) IFIX(NUMX) = -2
NUMBC = NUMBC + 1
CALL PAGE(LINE,LEND)
WRITE(LP,6010) CHAR, NUMBC, N, IC(I), KB
GO TO 30
40 CONTINUE
IFIX(NUMX) = 4
IFIX(NUMY) = 4
IFIX(NUMZ) = 4
IF(CHAR.EQ.CV) IFIX(NUMZ) = -3
SCN(NUMX) = AF*1.14159265/180.0
SCN(NUMY) = BT*1.14159265/180.0
NUMBC = NUMBC + 1
CALL PAGE(LINE,LEND)
WRITE(LP,6010) CHAR, NUMBC, N, IC(I), KB, AF, BT
30 CONTINUE
NOLD = N
NXOLD = NUMX
NYOLD = NUMY
NZOLD = NUMZ

**** READS THE NEXT BOUNDARY CONDITION DATA.
300 READ(IN,5000) CHAR, N, IC, KB, AF, BT
IF(N.EQ.0) GO TO 500
IF(KB.EQ.0) GO TO 200

**** GENERATES THE BOUNDARY CONDITIONS.
NUM = (N-NOLD)/KB - 1
NUM = ABS(NUM)
IF(NUM.EQ.0) GO TO 200
K = NOLD
KXOLD = NXOLD
KYOLD = NYOLD
KZOLD = NZOLD
DO 400 J = 1, NUM
K = K + KB
KXOLD = KXOLD + KB
KYOLD = KYOLD + KB
KZOLD = KZOLD + KB
DO 330 I = 1, 3
ICI = ICO + I
GO TO (330, 320, 310, 305, 340), ICI
310 CONTINUE
IFIX(KZOLD) = -1
IF(CHAR.EQ.CV) IFIX(KZOLD) = -2
NUMBC = NUMBC + 1
CALL PAGE(LINE,LEND)
WRITE(LP,6030) CHAR, NUMBC, K, IC(I)
GO TO 330
305 CONTINUE
IFIX(KYOLD) = -1
IF(CHAR.EQ.CV) IFIX(KYOLD) = -2
NUMBC = NUMBC + 1
CALL PAGE(LINE,LEND)
WRITE(LP,6030) CHAR, NUMBC, K, IC(I)
GO TO 330
320 CONTINUE
IFIX(KXOLD) = -1
IF(CHAR.EQ.CV) IFIX(KXOLD) = -2
NUMBC = NUMBC + 1
CALL PAGE(LINE,LEND)
WRITE(LP,6030) CHAR, NUMBC, K, IC(I)
GO TO 330
340 CONTINUE
IFIX(KXOLD) = 4
IFIX(KYOLD) = 4
IFIX(KZOLD) = -4
IF(CHAR.EQ.CV) IFIX(KZOLD) = -3
SCN(KXOLD) = AF*1.14159265/180.0
SCN(KYOLD) = BT*1.14159265/180.0
NUMBC = NUMBC + 1
CALL PAGE(LINE,LEND)
WRITE(LP,6030) CHAR, NUMBC, K, IC(I), KB, AF, BT
330 CONTINUE
400 CONTINUE
GO TO 200

**** END OF B.C. DATA SET.
```

---

MEL - Tsukuba - July 1993

15
Improvements on the 3-D RIPLE Program

500 CONTINUE
CALL PAGE(LINE,3,LEND)
WRITE(LP,6030) NUMBC
LLINE=LINE
*** DEBUG PRINT
IF(ISP(55).EQ.0) GO TO 600
WRITE(5,*) SP,BONDEN INFORMATION
WRITE(5,*) IFIX(NUMDF)
WRITE(5,*) (ISP(D),I=1,NUMDF)
600 CONTINUE
RETURN
*** READ FORMAT
500 FORMAT(A1,8,41I0,2F10.0)
*** WRITE FORMAT
600 FORMAT(H10,IS,* BOUNDARY',/8X,
 ' B. C. NODE NO. DEGREE OF',/8X,'KB',
 'B.C. NO. NODE NO. DEGREE OF',/8X,'KB',
 ' /,5X,'/5X,1,30X,'FREEDOM')
6010 FORMAT(I4,8X,A8,14,4X,IS,SX,IS,IX,IS,2FI0.0)
6020 FORMAT(I4,8X,A8,14,4X,IS,SX,SX,'G',2FI0.0)
6030 FORMAT(I20,SX,TOTAL NUMBER OF BOUNDARY CONDITIONS =',IS)
END
*LDGN  — — SUBROUTINE LODGEN — —
* SUBROUTINE LODGEN(NUMNP,LLINE,LLEND,DUMY,CLOAD,DLOAD,I0SET,NUMLD)
* FUNCTION (SP. NO. =26)
* GENERATES THE LOAD DATA.
* CALLED BY:
* MDINP
* CALLS:
* PAGE
* INPUT
* NUMNP  = TOTAL NUMBER OF NODAL POINTS.
* LLINE  = CURRENT LINE NUMBER.
* LLEND  = TOTAL LINE NUMBER PER PAGE.
* OUTPUT
* CLOAD  = CONCENTRATED LOAD VECTOR.
* DLOAD  = DISTRIBUTED LOAD VECTOR.
* I0SET  = TABLE OF DISTRIBUTED LOAD CASE.
* NUMLD  = NUMBER OF DISTRIBUTED LOAD CASE.
* COMMON /FILE/N , LP , IKF , ISF
COMMON /DEBUG1/ IDBG(20) , IOSP(SO)
DIMENSION CLOAD(I) , DLOAD(I) , I0SET(4,1) , ND1(3)
CHARACTER CCON,CDIS,CHAR
DATA CCON,CDISI'C','D'!
INITIAL DATA SET.
DUMY:='O.O
CHAR=CCON
NUMC=0
NUMLD=0
LINE=LLINE
LEND=LLEND
NUMNP3=NUMNP+3
CALL CLEAR(CLOAD,NUMNP3)
CALL CLEAR(DLOAD,NUMNP)
*** HEADER PRINT.
CALL PAGE(LINE,1,LEND)
WRITE(LP,6000)
*** READS THE LOAD DATA
100 READ(N,5000) CHAR , NI , NDJ , VALUE
IF(NEQ.0) GO TO 400
IF(CHAR.EQ.CDIS) GO TO 300
CONCENTRATED LOAD
NUMC=NUMC+1
CALL PAGE(LINE,1,LEND)
WRITE(LP,6020) NUMC , NI , NDJ , VALUE
*** COMPUTES CLOAD.
N1=NI+NDJ+1
CLOAD(N1)=VALUE
GO TO 100
DISTRIBUTED LOAD
300 CONTINUE
CALL PAGE(LINE,1,LEND)
NUMLDM=NUMLD+1
WRITE(LP,6030) NUMLDM , NI , NDJ , VALUE
*** COMPUTES DISTRIBUTED LOAD
LDSET(N1,NUMLDM)=NI
LDSET(2,NUMLDM)=NDJ
LDSET(0,NUMLDM)=NDJ
LDSET(DUMY)=NDJ
DLOAD(NUMLDM)=VALUE
----------------- MEL - Tsukuba - July 1993 -----------------
Improvements on the 3-D RIPLE Program

GO TO 100

*** END OF LOAD DATA SET.
400 CONTINUE
CALL PAGE(LINE,3,LEND)
WRITE(6,6040) NUMC, NUMLD
RETURN

*** DEBUG PRINT
IF(ISP(26).EQ.0) GO TO 500
WRITE(15,6010)
WRITE(15,6020)
WRITE(15,6030)
GO TO 500

500 CONTINUE
RETURN

*** READ FORMAT
5000 FORMAT(A1,19,310,F10.0)
*** WRITE FORMAT
6000 FORMAT(16X,'LOAD',/12X,'LOAD CASE NO. DEGREE OF FREEDOM VALUE',/12X,'TOTAL NUMBER OF CONCENTRATED LOAD CASE =',15,
       'TOTAL NUMBER OF DISTRIBUTED LOAD CASE =',IS)

END

CIMG - SUBROUTINE CIMAGE -
SUBROTVNE CIMAGE
FUNCTION.(SP. NO. =21)
PRINTS CARD IMAGE.
CALLED BY.
COMMON /FILEJ IN , LP , IKF , ISF
DIMENSION A(20)
DATA LINE,LEND/O,SSI
100 READ(IN,5000,END=300) A
IF(MOD(LINE,LEND).NE.0) GO TO 200
WRITE(LP,6000)
200 CONTINUE
LINE=LINE+1
WRITE(LP,6010) LINE, A
GO TO 100

300 CONTINUE
REWIND IN
RETURN

*** READ FORMAT
5000 FORMAT(20A4)
*** WRITE FORMAT
6000 FORMAT(16X,'CARD IMAGE',/20X,'[1234567890]
       20X,20A4)

END

INIT - SUBROUTINE INIT -
SUBROUTINE INIT(A)
FUNCTION.(SP. NO. =28)
COMPUTES THE INITIAL VELOCITY AND EQUATION TABLE.
CALLED BY.
MAIN
CALLS.
VELCAL,CHECK,PAGE,UINIV
INPUT.
IFLAG(1) = FLAG FOR CALCULATION OF INITIAL VELOCITY.
R = RADIAL COORDINATE OF NODAL POINT.
Z = AXIAL COORDINATE OF NODAL POINT.
IX = NODE NO. COMPOSED THE ELEMENT.
NUMNP = TOTAL NUMBER OF NODAL POINTS.
NUMEL = TOTAL NUMBER OF ELEMENTS.
OUTPUT.
VR = NODAL POINT VELOCITY IN R-DIRECTION.
VZ = NODAL POINT VELOCITY IN Z-DIRECTION.
INDEXN = EQUATION TABLE OF NODAL POINT.
INDEXE = EQUATION TABLE OF ELEMENT.
*** COMMON (INPV, HEADQ) , IFLAG(20) , CLIM , DFACT , ISTE
   , INITV , MODEL

MEL - Tsukuba - July 1993

17
**Improvements on the 3-D RIPLE Program**

**COMMON IBASV**

**NUMEL, NUMNP, NUMDF, NUMLD**

**COMMON ADDRESS, MTOT, MAX, MINS, MNUM, MGE, MMES, MMBE, MMLS, MMLMI, MMLM2, MMLE, MMLM4, MMLM5, MMLM6, MMLM7, MMLM8, MMLS, MMLMI, MMLM2, MMLE, MMLM4, MMLM5, MMLM6, MMLM7, MMLM8, MMLM9**

**COMMON /LINES/ LTOP, LINE, LEND**

**COMMON /DEBUG/ IDBG(20), IOSP(50)**

**DIMENSION A(I)**

*** COMPUTES THE INITIAL VELOCITY FIELD.

MMVS=MMAX+1

MMVU=MMVS+NUMNP

MMVE=MMVU+NUMNP

IF(IDBG(2).EQ.0) GO TO 50

CALL PAGE(LINE,6,LEND)

CALL CHECK(MTOT,MMVE,'VELO', 'CITY')

50 CONTINUE

MMAX-MMVE

IF(IFLAG(II).NE.I) GO TO 100

*** USER'S SUBROUTINE:

CALL UlNIV(A(MMNS),A(MMNU),A(MMNM),NUMNP,A(MMVS),A(MMVU),A(MMVM))

GO TO 200

*** STANDARD RELATION.

100 CONTINUE

CALL VELCAL(A(MMNS),A(MMNU),A(MMNM),A(MMBS),NUMNP, 0, A(MMVS),A(MMVU),A(MMVM))

200 CONTINUE

*** COMPUTES THE EQUATION TABLE ( INDEXN & INDEXE ).

MMIS=MMAX+1

MMIM=MMIS+NUMNP

MMIE=MMIM+NUMEL-I

IF(1DBG(2).EQ.O) GO TO 250

CALL PAGE(LINE,6,LEND)

CALL CHECK(MTOT,MMIE,' IND','EX ') 

250 CONTINUE

MMAX=MMIE

CALL INDEX(A(MMES),NUMEL,NUMNP, 0 ,A(MMIS),A(MMIM))

RETURN

END

**SUBROUTINE VELCAL(X,Y,Z,IFIX,NUMNP, DUMY, VX,Y,VZ )**

**FUNCTlON.(SP. NO. =29)**

**COMPUTES THE INITIAL VELOCITY BY MEANS OF STANDARD RELATION.**

**CALLED BY.**

**INIT**

**CLEAR**

**INPUT.**

X = RADIAL COORDINATE OF NODE.

Y = RADIAL COORDINATE OF NODE.

Z = AXIAL COORDINATE OF NODE.

IFIX = BOUNDARY CONDITION.

NUMNP = TOTAL NUMBER OF NODE.

**OUTPUT.**

VX = NODAL POINT VELOCITY IN R-DIRECTION.

VY = NODAL POINT VELOCITY IN R-DIRECTION.

VZ = NODAL POINT VELOCITY IN Z-DIRECTION.

*** COMMON /COEFF/ Y0, C1, CM, CN, A1, B1, A2, B2, A3, B3

COMMON /DEBUG/ IDBG(20), IOSP(50)

**DIMENSION X(NUMNP), Y(NUMNP), Z(NUMNP), IFIX(I),

VX(NUMNP), VY(NUMNP), VZ(NUMNP)**

*** INITIAL DATA SET.

NUMN=NUMNP

CALL CLEARX(X,NUMN)

CALL CLEARY(Y,NUMN)

CALL CLEARZ(VZ,NUMN)

*** DO 200 N=1,NUMNP

**NO=N+1**

**NY=NY+1

NZ=NZ+1

*** EXCEPT THE FIXED COMPONENT OF THE NODE.

IF(IFIX(X(N))<1) GO TO 100

VX(N)=A*X(N)+B1

100 CONTINUE

IF(IFIX(Y(N)).EQ.1 OR IFIX(Z(N)).EQ.4) GO TO 150

VY(N)=A*Y(N)+B2

150 CONTINUE

IF(IFIX(Z(N)).EQ.1 OR IFIX(Z(N)).EQ.4) GO TO 200

VZ(N)=A*Z(N)+B3

---

MEL - Tsukuba - July 1993
** Improvements on the 3-D RIPLE Program **

*INDEX* — SUBROUTINE INDEX —

SUBROUTINE INDEX(IX, NUMEL, NUMNP, DUMY, INDEXN, INDEXE)
* FUNCTION (SP. NO. = 8)*
* COMPUTES BAND WIDTH AND EQUATION TABLE.
* CALLED BY.*
* INIT
* CALLS.
* NONE
* INPUT.
* IX = NODE NUMBER OF COMPOSED THE ELEMENT.
* NUMEL = NUMBER OF ELEMENTS.
* NUMNP = NUMBER OF NODAL POINTS.
* OUTPUT.
* NBAND = BAND WIDTH.
* INDEXN = EQUATION TABLE OF NODAL POINTS.
* INDEXE = EQUATION TABLE OF ELEMENTS.
* DIMENSION IX(S,1), INDEXN(NUMNP), INDEXE(NUMEL)
** COMMON DEBUG, IDBG(20), IOSP(50)
** COMMON ISOLV, NBAND, NBND
** COMMON DIFVA, HEAD(20), IFLAG(20), CLIM, DFACT, ISTIF
** INITV, MODEL
** ** INITIAL DATA SET.
** DUMY=0.0
** NBAND=0
** DO 10 IX=1, NUMNP
** INDEXN(IX)=IX+2
** 10 CONTINUE
** COMPUTES THE EQUATION TABLE.
** DO 50 IX=1, NUMEL
** N1=IX(1,1)
** N2=IX(2,1)
** N3=IX(3,1)
** N4=IX(4,1)
** N5=IX(5,1)
** N6=IX(6,1)
** N7=IX(7,1)
** N8=IX(8,1)
** NMAX=MAX0(N1,N2,N3,N4,N5,N6,N7,N8)
** NMIN=MIN0(N1,N2,N3,N4,N5,N6,N7,N8)
** COMPUTES THE BAND WIDTH OF ELEMENT.
** BAND=(NMAX-NMIN)+1
** IF(NBAND.GT.0) GO TO 20
** NBAND=BAND
** NMIN=SIN+1
** NMAX=SIN
** *** SEARCH THE NODE NO. OF POSITIVE INDEXN.
** 20 CONTINUE
** IF(ISTIF.GT.1) GO TO 50
** IF(INDEXN(MAX),GT.0) GO TO 30
** MAX=MAX+1
** IF(INDEXN.GT.0) GO TO 20
** WRITE(15,*'*** INDEX DATA ( IX ) ERROR ***
** WRITE(15,*'N1=',N1,' N2=',N2,' N3=',N3,' N4=',N4,
** N5=',N5,' N6=',N6,' N7=',N7,' N8=',N8,' MAX=',MAX
** STOP
** 30 CONTINUE
** *** INDEX AND INDEXE SET.
** DO 40 IX=1, NUMNP
** IF(INDEXN(IX).LT.0) INDEXN(IX)=INDEXN(IX)+1
** IF(INDEXE(IX).LT.0) INDEXE(IX)=INDEXE(IX)+1
** 40 CONTINUE
** INDEXE(INDEX)(MAX)=INDEXE(INDEX)
** INDEXE(INDEX)=INDEXE(INDEX)(MAX)
** 50 CONTINUE
** *** COMPUTES THE NEW BAND WIDTH.
** IF(INSTIF.GT.1) GO TO 75
** DO IX=1, NUMNP
** IF(INDEXE(IX).LT.0) NBAND=NBAND+1
** 60 CONTINUE
** *** SETS THE INDEXN TO POSITIVE VALUE.
DO 70 I=1,NUMNP
INDEXN(I)=ABS(INDEXN(I))
70 CONTINUE
75 CONTINUE
** PRINTS EQUATION TABLE AND BAND WIDTH.
IF(ISP(EQ.0)) GO TO 80
WRITE(15,*) "***************"**
WRITE(15,*) ** SP INDEX INFORMATION **
WRITE(15,*) "***************"
WRITE(15,*) "INDEX "
WRITE(15,*) (INDEXN(J),J=1,NUMNP)
WRITE(15,*) "INDEX "
WRITE(15,*) (INDEXE(J),J=1,NUMEL)
WRITE(15,*) "NBAND ",NBAND
80 CONTINUE
RETURN
END

*LOAD*  SUBROUTINE LOAD(A,NSTA)
* FUNCTION(SP. NO. =31)*
* COMPUTES THE GLOBAL LOAD VECTOR.*
* CALLED BY.*
* RIPLE*
* CALLS.*
* LOCAL,CHECK,PAGE*
* INPUT. *
MTOT = MAXIMUM ADDRESS NUMBER IN BLANK COMMON.
MMAX = CURRENT ADDRESS NUMBER IN USED BLANK COMMON.
NSTA = STAGE NUMBER.
* OUTPUT. *
NMAX = CURRENT ADDRESS NUMBER IN USED BLANK COMMON.

*** COMMON MBSVN NUMEL , NUMNP , NUMDF , NUMLD
COMMON ADDRESS MTOT , MMAX , MMIN , MMAX , MMNE , MMES ,
* MMEE , MMBE , MMBE , MMIL , MMIL , MMGE ,
* MMGE , MMGE , MMGE , MMGE , MMGE ,
* MMGE , MMGE , MMGE , MMGE , MMGE ,
COMMON ADDRESS/ NMAX , NNPS , NNPM , NNPN , NNPE , NNEE , NNES ,
* NNAS , NNAY , NNAS , NNAS , NNAS , NNAS ,
COMMON (INDEXE LTOP , LINE , LEND
COMMON /DEBUG/ IDB0(20) , 10SP(SO)
DIMENSION A(I)
NUMSTA-NSTA
***
IF(NUMSTA.GT.1 .AND. NUMLD.EQ.0) GO TO 200
NNPS=MMAH+1
NNPM=NNPS+NUMDF*2
NNPM=NNPM+NUMDF
NNPE=NNPM+NUMDF-1
*** WORKING AREA — JFIX(NUMDF)
NNWS=NNPE+1
NNWS=NNWS+NUMDF-1
IF(IDBG(2).EQ.0) GO TO 100
CALL PAGE(LINE,LEND)
CALL CHECK(MTOT,NNWE,' PLO','AD ')
100 CONTINUE
NMAX=NNPE
*** COMPUTES THE GLOBAL LOAD VECTOR.
CALL LDCAL(A(MMNS),A(MMNU),A(MMNM),A(MMLS),A(MMLM1),A(MMLM2),A(NNPS,
*, NUMDF , 0 , ANNPS))
***
IF(NUMSTA.GT.1) GO TO 200
*** COMPUTES THE BOUNDARY CONDITIONS.
CALL Bonset(A(MMNS),ANUMEL,A(MMNS),ANUMNP,A(MMNS),A(MMNS),A(NNWS,
*, ANNPS),NUMDF,A(MMNS))
***
CALL TOOLS(A(MMNS),A(MMNS),A(MMNS),NUMNP,A(MMNS),A(MMNS))
200 CONTINUE
***
RETURN
END

LDCL  SUBROUTINE LDCL ***
* FUNCTION(SP. NO. =30)*
* COMPUTES THE GLOBAL LOAD VECTOR.*
* CALLED BY.*
* LOAD*
* CALLS.*
* INPUT.*

----------------- MEL - Tsukuba - July 1993 -----------------
Improvements on the 3-D RIPPLE Program

* CLOAD = CONCENTRATED LOAD VECTOR.
* DLOAD = DISTRIBUTED LOAD VECTOR.
* NUMDF = TOTAL NUMBER OF DEGREE OF FREEDOM.
* OUTPUT.
* PLOAD = GLOBAL LOAD VECTOR.

** COMMON /BASEV/ NUMEL , NUMNP , NDUMY , NUMLD
** COMMON /DEBUG/ IDBG(20) , IOSP(50)
** DIMENSION X(I) , Y(I) , Z(I) , CLOAD(I) , DLOAD(I) ,
** LDSET(G-I) , PLOAD(NUMDF,2)
** ** INITIAL DATA SET.
** NUMDF=NUMDF2
** CALL CLEAR(PLOAD,NUMDF2)
** NUMNP3=NUMNP3
** ** COMPUTES THE GLOBAL LOAD VECTOR.
** DO 100 N=1,NUMNP3
** PLOAD(N,1)=CLOAD(N)
** 100 CONTINUE
** ** DISTRIBUTED LOAD SET.
** IF(NUMLD.EQ.0) GO TO 300
** DO 200 N=1,NUMLD
** N=LDSET(N)
** N=LDSET(2,N)
** N=LDSET(3,N)
** N=LDSET(4,N)
** VALUE=DLAOD(N)
** AX=X(NI)-X(NI)
** AY=Y(NI)-Y(NI)
** AZ=Z(NI)-Z(NI)
** BB=0.25
** COMPUTES THE DISTRIBUTED LOAD.
** PLOAD(NX,1)=PLOAD(NX,1)-BB*FX
** PLOAD(NY,1)=PLOAD(NY,1)-BB*FY
** PLOAD(NIZ,1)=PLOAD(NIZ,1)-BB*FZ
** PLOAD(NX,1)=PLOAD(NX,1)-BB*FX
** PLOAD(NY,1)=PLOAD(NY,1)-BB*FY
** PLOAD(NIZ,1)=PLOAD(NIZ,1)-BB*FZ
** PLOAD(NX,1)=PLOAD(NX,1)-BB*FX
** PLOAD(NY,1)=PLOAD(NY,1)-BB*FY
** PLOAD(NIZ,1)=PLOAD(NIZ,1)-BB*FZ
** 200 CONTINUE
** 300 CONTINUE
** DEBUG PRINTS.
** IF(IOSP(30).EQ.0) GO TO 500
** WRITE(15,') 'CLOAD(NUMNP3)
** WRITE(15,') (N,CLOAD(N),N=1,NUMNP3)
** WRITE(15,') , ,NUMNP3,NUMNP3
** IF(NUMLD.EQ.0) GO TO 400
** WRITE(15,') , 'LDSET(NUMLD)
** WRITE(15,') (N,(LDSET(J,N),J=1,4),N=1,NUMLD)
** WRITE(15,') , 'DLAOD(NUMLD)
** WRITE(15,') (N,DLOAD(N),N=1,NUMLD)
** 400 CONTINUE
** WRITE(15,') , PLOAD(NUMDF,1)
** WRITE(15,') (N,PLOAD(N,1),N=1,NUMDF)
** 500 CONTINUE
** RETURN
** END

*BNST — SUBROUTINE BONSET —
SUBROUTINE BONSET(INDEXE, NUMEL, INDEXN, NUMNP, IFIX, JFIX, PLOAD, NUMDF, SCN)

* FUNCTION (SP. NO. = 15)
* COMPUTES THE BOUNDARY CONDITION.
* CALLED BY.
* LOAD
* CALLS.
* NONE
* INPUT.
* INDEXE = EQUATION TABLE OF ELEMENTS.
* NUMEL = NUMBER OF ELEMENTS.
* INDEXN = EQUATION TABLE OF NODES.
* NUMNP = NUMBER OF NODES.
* IFIX = BOUNDARY CONDITION.
* JFIX = BOUNDARY CONDITION (WORKING AREA).
* NUMDF = NUMBER OF D.O.F.
* PLOAD = EXTERNAL LOAD VECTOR.

* OUTPUT.
* IFIX = BOUNDARY CONDITION CONSIDERED THE 'INDEXE'.
* PLOAD = EXTERNAL LOAD VECTOR CONSIDERED THE 'INDEXE'.

***
COMMON /DEBUG/ IDBG(20), IOSP(50)
COMMON /FILE/ IN, LP, IIF, ISF
COMMON (IFVAR, HEAD(20), IFLAG(20), CLM, DFAC, ISTIF)
* INTY, MODEL
DIMENSION INDEXE(NUMEL), INDEXN(NUMNP), IFIX(NUMDF)
* PLOAD(NUMDF,2), JFIX(NUMDF), SCN(NUMDF)
***
DO 100 N=1, NUMNP
NX=NX-2
NY=NX+1
NZ=NY+1
IX=INDEXN(N)
IY=IX+1
IZ=IY+1
IFIX(IX)=IFIX(NX)
PLOAD(IX,2)=PLOAD(NX,1)
IFIX(IY)=IFIX(NY)
PLOAD(IY,2)=PLOAD(NY,1)
IFIX(IZ)=IFIX(NZ)
PLOAD(IZ,2)=PLOAD(NZ,1)
100 CONTINUE

IF(ISTIF.EQ.1) THEN
DO 200 N=1, NUMEL
IE=INDEXE(N)
IEN=INDEXN(IE)-1
IFIX(IEN)=1
PLOAD(IEN,2)=0.0
200 CONTINUE
*** IFIX(NUMDF) & PLOAD(NUMDF,1) RESET.
DO 300 N=1, NUMDF
IFIX(N)=IFIX(N)
PLOAD(N,2)=PLOAD(N,2)
300 CONTINUE
***
DO 500 I=1, NUMNP
N=NUMNP+1
NX= 1*N-2
NY= NX+1
NZ= NY+1
IX= INDEXN(N)
IY= IX+1
IZ= IY+1
SCN(IX)=SCN(NX)
SCNY=SCN(NY)
500 CONTINUE
END IF
***
*** WRITES THE GLOBAL LOAD VECTOR.
REWIND ISF
WRITE(ISF) PLOAD
***
IF(ISPS(15).EQ.0) GO TO 400
WRITE(15,1) '********************
WRITE(15,1) ' SP BONSET INFORMATION *
WRITE(15,1) '********************
WRITE(15,2) ' IFIX *
WRITE(15,2) (IIFIX(I)=1,NUMDF)
WRITE(15,2) ' PLOAD *
WRITE(15,2) (ILOAD(I,1)=1,NUMDF)
400 CONTINUE

--- MEL - Tsukuba - July 1993 ---
RETURN
END

*TOOLS — SUBROUTINE TOOLS —
SUBROUTINE TOOLS(X, Y, Z, NUMNP, IFIX, INDEXN)
* SETS PUNCH AND DIE BOUNDARY CONDITION
* CALLED BY:
* LOAD
* CALLS:
* NONE
*
** COMMON (NPVA, HEAD(20), IFLAG(20), CLIM, DFACI, ISTIF
* , INTV, MODEL
COMMON (PUNDE(2), PW2, PPR, PCL, DW2, DFR, DCR
COMMON (PANG/ XPP(7,7), YPP(7,7), ZPP(7,7), XNOS(11), XNOTT(11)
COMMON (PANG/ XPD(7,7), YPD(7,7), ZPD(7,7), XNOTS(11), XNOTT(11)
COMMON (XNOTV, XNM
COMMON (BLRNG/ ELTH, RINGTH, BHWD, BHWE
** DIMENSION X(I), Y(I), Z(I), IFIX(I), INDEXN(I)
**
*** FIND OUT A NODE WHICH WILL CONTACT PUNCH ****
IF(pw2.1e.0.00001) goto 800
PPXE=PWW*1.1
PPYE=PW2+1.1
PPPE=PW2+1.1
DO 100=1,NUMNP
IF[(X(I).le.PPX AND Y(I).le.PPYE AND Z(I).GT.BLTH*0.9
AND IFIX(INDEXN(I)+2).GT.0) THEN
IF(X(I).GT.PPXs)
IFIX(INDEXN(I)=5
ELSE IF(Y(I).GT.PPYS)
IFIX(INDEXN(I)+1)=5
END IF
100 CONTINUE

*** FIND OUT A NODE WHICH WILL CONTACT DIE ****
DPXS=DW2
DPY£=DW2
DPPE=DPW*2+1.1
DO 200=1,NUMNP
IF[(IFIX(INDEXN(I)=2).GT.0 AND Z(I).LT.BLTH*0.1)
AND IFIX(INDEXN(I)+2).GT.0) THEN
IFIX(INDEXN(I)+1)=6
IFIX(INDEXN(I)+1)=6
END IF
200 CONTINUE

*** FIND OUT A NODE WHICH WILL CONTACT THE BLANKHOLDER ****
IF(BHWE.GE.1000 TO 500
DO 500=1,NUMNP
RING2=RHNG TH*2
IF[(X(I).LT.RING2) AND Z(I).GT.BLTH*0.1)
AND IFIX(INDEXN(I)+1).GT.0) THEN
IFIX(INDEXN(I)+1)=7
IFIX(INDEXN(I)+1)=7
ELSE IF(X(I).LE.BHW1 AND Y(I).LE.BHWE AND Z(I).GE.BHW12) THEN
IFIX(INDEXN(I)+1)=7
END IF
500 CONTINUE

*** CALCULATES POLYGON VECTOR FOR PUNCH AND DIE ****
XPP(1,1) = 0.0
ZPP(1,1) = BLTH
XPP(6,1) = PW2+1.145*PPR
ZPP(6,1) = BLTH
XPP(1,1) = PW2+0.055*PPR
ZPP(1,1) = BLTH+0.0333*PPR
XPP(4,1) = PW2+0.25*PPR
ZPP(4,1) = BLTH+0.25*PPR
XPP(3,1) = PW2+0.0333*PPR
ZPP(3,1) = BLTH+0.055*PPR
XPP(2,1) = PW2
ZPP(2,1) = BLTH+1.145*PPR
XPP(1,1) = PW2
ZPP(1,1) = BLTH+60.0

--- MEL - Tsukuba - July 1993 ---

23
Improvements on the 3-D RIPLE Program

YPP(I,1) = 0.0
XPP(I,2) = PW2
YPP(I,2) = PW2-1.145*PCR
XPP(I,3) = PW2-0.033*PCR
YPP(I,3) = PW2-0.805*PCR
XPP(I,4) = PW2-0.247*PCR
YPP(I,4) = PW2-0.247*PCR
XPP(I,5) = PW2+0.033*PCR
YPP(I,5) = PW2-1.145*PCR
XPP(I,6) = PW2-0.805*PCR
YPP(I,6) = PW2-0.805*PCR
XPP(I,7) = 0.0
YPP(I,7) = PW2
XPD(1,1) = DW2+3.0
ZPD(1,1) = 0.0
XPD(2,1) = DW2+0.333*DPD
ZPD(2,1) = 0.0
XPD(3,1) = DW2+0.805*DPD
ZPD(3,1) = 0.0
XPD(4,1) = DW2+0.247*DPD
ZPD(4,1) = 0.0
XPD(5,1) = DW2+0.247*DPD
ZPD(5,1) = 0.0
XPD(6,1) = DW2+0.033*DPD
ZPD(6,1) = 0.0
XPD(7,1) = DW2
ZPD(7,1) = 0.0

DO 300 I=1,7
ZPP(I) = ZPP(I,1)
300 CONTINUE

DO 400 I=1,7
ZPD(I) = ZPD(I,1)
400 CONTINUE

DO 500 J=2,7
XPP(I,J) = XPP(I,1)·XPP(I,J-1)
YPP(I,J) = YPP(I,1)·XPP(I,J-1)
XPD(I,J) = XPD(I,1)·XPD(I,J-1)
YPD(I,J) = YPD(I,1)·XPD(I,J-1)
500 CONTINUE

DO 432 J=1,6
XPD(I,J) = XPD(I,1)·XPD(I,J-1)
YPD(I,J) = YPD(I,1)·XPD(I,J-1)
432 CONTINUE

DO 440 I=1,4
NXNOTSP(I) = 0.0
NXNOTTP(I) = 0.0
NXNOTSD(I) = 0.0
NXNOTTD(I) = 0.0
440 CONTINUE

NM4 = 3
DO 450 I=1,NM4
NXNOTSP(I) = REAL(I)
NXNOTTP(I) = REAL(I)
NXNOTSD(I) = REAL(I)
NXNOTTD(I) = REAL(I)
450 CONTINUE

NM3 = NM4 + 1
DO 460 I=1,NM3
NF1 = 7 + 1
NXNOTSP(NF1) = REAL(NM3)
NXNOTTP(NF1) = REAL(NM3)
**Improvements on the 3-D RIPLE Program**

XKNOTSD(NPI) = REAL(NMJ)
XKNOTTD(NP1) = REAL(NMJ)
460 CONTINUE
800 CONTINUE
RETURN
END

*STIF* — SUBROUTINE STIF —

*SUBROUTINE STIF(A,NSTA,NUMITE)*

*FUNCTION(SP. NO. = 32)*

*COMPUTES THE GLOBAL STIFFNESS MATRIX.*

*CALLED BY.*

*MAIN*

*CALLS GSCAL,CHECK,PAGE*

*INPUT.*

MTOT = MAXIMUM ADDRESS NUMBER IN USED BLANK COMMON.

NITE = INNER ITERATION NUMBER.

NSTA = STAGE NUMBER.

*OUTPUT.*

NMAX = CURRENT ADDRESS NUMBER IN USED BLANK COMMON.

*COMMON /BASEV/ NUMEL , NUMNP , NUMDF , NUMLD*

*COMMON /SOLVE/ NBAND , NBNDF*

*COMMON /ADDRESS/ MTOT , NMAX , MMNS , MMNM , MMNE , MMES ,*

MMEE , MMBS , MMSE , MMWS , MMWS , MMWE , MMSE , MMWS ,*

MMVS , MMVM , MMVE , MMUE , MMUE , MMVE , MMUE ,*

MMNU , MMVU , MMBS1 , MMWU*

*COMMON /ADDRESS/ NMAX , NNPS , NNPM , NNPN , NNPE , NNES , NNNE ,*

NNSS , NNSE , NNGS , NNGE , NNRS , NNS , NNST , NGET , NGSD*

*COMMON (DEBUG/ IDBG(20) , IOPS(10)*

*COMMON (NPVA/ HEAD(20) , IFLAG(20) , CUM , DFACT , ISTIF*

*... INTIV , MODEL*

**DIMENSION A(1)**

*... INITIAL DATA SET.*

NITE = NUMITE

NUMSTA = NSTA

**... EPS(NUMEL)**

NNES = NNPE + 1

NNEE = NNES + NUMEL - 1

(NUMSTA EQ 1) CALL CLEAR(A(NNES), NUMEL)

**... SG(NUMEL)**

NNSS = NNNE + 1

NNSE = NNSS + NUMEL - 1

**... GS(NBNDF)**

NNGS = NNGE + 1

NBNDF = NBAND * NUMDF

NNGE = NNGS + NBNDF - 1

IF(NITE .GT. 1) GO TO 100

IF(IDBG(2).EQ.0) GO TO 100

CALL PAGE(LINE,6,LEND)

CALL CHECK(MTOT,NNGE,' ST','IF .

100 CONTINUE

MMAX = NNGE

**... COMPUTES THE GLOBAL STIFFNESS MATRIX**

IF(ISTIF.EQ.1) THEN

CALL GSCAL(NUMSTA,NITE,A(MMES),A(MMNS),A(MMNM),A(MMNE),A(MMWS),*

A(MMWE),A(MMWS),A(MMWE),A(MMWS),*

A(MMVS),A(MMWM,A(MMWS),A(NNES),A(NNEE),A(NNPS),A(NNPD),A(MMNS),A(MMME),*

A(MMBS),A(NNGS),A(NNPS),A(NNST),A(MMBS),A(MMBS1),A(MMWU))

END IF

**... RETURN**

END

**MATMAT** — SUBROUTINE MATMAT —

*SUBROUTINE MATMAT(A,IA,IAB,B,IB,IOPT, DUMY ,C)*

*FUNCTION(SP. NO. = 4)*

MULTIPLY MATRIX C = A * B ( IOPT = 1 )

C = A * B ( IOPT = 0 )

C = C + A * B ( IOPT = 1 )

**CALLED BY**

GSCAL , PMCAL

**CALLS**

NONE

**DIMENSION A(IA,IB) , B(IB,IB) , C(A,IB)**
Improvements on the 3-D RIPLE Program

DIMY=0.0
IF(IOPT) 100,50,100
50 CONTINUE
***  C = A * B
DO 20 I=1,JA
DO 20 J=1,JB
CU=0.0
DO 10 K=1,JK
CU=CU+A(I,K)*B(K,J)
10 CONTINUE
C(I,J)=CU
20 CONTINUE
GO TO 200
***
100 CONTINUE
SIGN=FLOAT(IOPT)
DO 40 I=1,IA
DO 40 J=1,JB
CU=0.0
DO 30 K=1,JK
C(I,J)=C(I,J)+SIGN*CU
30 CONTINUE
40 CONTINUE
200 CONTINUE
RETURN
END

**MTMT** — SUBROUTINE MATMA1 —

SUBROUTINE MATMA1(A,IA,IB,B,IOPT, DUMY,C)
*FUNCTION(SP. NO. = 4)*
* CALLED BY.*
* PMCAL*
* CALLS.*
* NONE*
***
DIMENSION A(24), B(24), C(24,24)
***
DIMY=0.0
***
C = A * B
DO 20 I=1,24
DO 20 J=I,24
C(I,J)=A(I)*B(J)
20 CONTINUE
RETURN
END

**MTMT** — SUBROUTINE MATMA2 —

SUBROUTINE MATMA2(A,IA,IAB,B,m,IOPT, DUMY,C)
*FUNCTION(SP. NO. = 4)*
* CALLED BY.*
* GSCAL*
* CALLS.*
* NONE*
***
DIMENSION A(24), B(24)
***
DIMY=0.0
***
C = A * B
C=0.0
DO 10 K=1,24
C=C+A(K)*B(K)
10 CONTINUE
RETURN
END

**MTMT** — SUBROUTINE MATMA3 —

SUBROUTINE MATMA3(A,IA,IB,B,m,IOPT, DUMY,C)
*FUNCTION(SP. NO. = 4)*
* CALLED BY.*
* GSCAL*
* CALLS.*
* NONE*
***
DIMENSION A(24,24), B(24), C(24)
***
DIMY=0.0
***
C = A * B
DO 20 I=1,24
C=0.0
20 CONTINUE
RETURN
END

MTMT - SUBROUTINE MATMA1

MTMT - SUBROUTINE MATMA2

MTMT - SUBROUTINE MATMA3

MEL - Tsukuba - July 1993
DO 10 K=1,24
   CU=CU+AL(K)*B(K)
10 CONTINUE
C(J)=CU
20 CONTINUE
RETURN
END

SUBROUTINE NDSET

SUBROUTINE NDSET(NEL,X,Y,Z,VX,VY,VZ)
COMMON /DEBUGI/ IOBG(20), IOSP(50)
COMMON /DATA/ HEAD(20), IFLAG(20), CUM, DFACT, ISTIF
COMMON /NDATY/ U(I),V(I),W(I)
COMMON /AREA/ AREA, AREAU
DIMENSION X(I), Y(I), Z(I), VX(I), VY(I), VZ(I)

X(I)=X(NI)
X(2)=X(N2)
X(J)=X(NJ)
X(4)=X(N4)
X(5)=X(N5)
X(6)=X(N6)
X(7)=X(N7)
X(8)=X(N8)

Y(I)=Y(NI)
Y(2)=Y(N2)
Y(J)=Y(NJ)
Y(4)=Y(N4)
Y(5)=Y(N5)
Y(6)=Y(N6)
Y(7)=Y(N7)
Y(8)=Y(N8)

Z(I)=Z(NI)
Z(2)=Z(N2)
Z(J)=Z(NJ)
Z(4)=Z(N4)
Z(5)=Z(N5)
Z(6)=Z(N6)
Z(7)=Z(N7)
Z(8)=Z(N8)

U(I)=VX(NI)
U(2)=VY(N1)
U(3)=VZ(N1)
U(4)=VX(N2)
U(5)=VY(N2)
U(6)=VZ(N2)
U(7)=VX(N3)
U(8)=VY(N3)
U(9)=VZ(N3)
U(10)=VX(N4)
U(11)=VY(N4)
U(12)=VZ(N4)
U(13)=VX(N5)
U(14)=VY(N5)
U(15)=VZ(N5)
U(16)=VX(N6)
U(17)=VY(N6)
U(18)=VZ(N6)
U(19)=VX(N7)
U(20)=VY(N7)
U(21)=VZ(N7)
U(22)=VX(N8)
U(23)=VY(N8)
U(24)=VZ(N8)

* CALCULATES SQUARE AREA AT ELEMENT
AX1=X(N2)-X(N1)
AY1=Y(N2)-Y(N1)
AZ1=Z(N2)-Z(N1)
BX1=X(N4)-X(N1)
BY1=Y(N4)-Y(N1)
BZ1=Z(N4)-Z(N1)
AX2=X(N3)-X(N1)
AY2=Y(N3)-Y(N1)
AZ2=Z(N3)-Z(N1)
BX2=X(N4)-X(N1)
BY2=Y(N4)-Y(N1)
BZ2=Z(N4)-Z(N1)
AX3=X(N6)-X(N1)
AY3=Y(N6)-Y(N1)
AZ3=Z(N6)-Z(N1)
AX4=X(N8)-X(N1)
AY4=Y(N8)-Y(N1)
AZ4=Z(N8)-Z(N1)

-----------------
MEL - Tsukuba - July 1993
-----------------
AX4=XT(N6)-XT(N7)
AY4=YT(N6)-YT(N7)
AZ4=ZT(N6)-ZT(N7)
BX4=XT(N8)-XT(N7)
BY4=YT(N8)-YT(N7)
BZ4=ZT(N8)-ZT(N7)

AREA1=SQRT((AY4*AZ4*BY4)**2+(AZ4*BX4*AZ4)**2+(AZ4*BY4*AZ4)**2)

AREA2=SQRT((AY2*AZ2*BY2)**2+(AZ2*BX2*AZ2)**2+(AZ2*BY2*AZ2)**2)

AREA3=AREA1+AREA2

AREA4=SQRT((AY4*AZ4*BY4)**2+(AZ4*BX4*AZ4)**2+(AZ4*BY4*AZ4)**2)

AREAO=(AREA3+AREA4)/2.0

RETURN
END

*ELST — SUBROUTINE ELSET —

SUBROUTINE ELSET(NEL,AI,AJ,AK)

FUNCTION (SP. NO. = 3)

COMPUTES THE 'NEL' ELEMENT INFORMATION.

CALLED BY.
* GSCAL

CALLS.
* NONE

INPUT.
* NEL = ELEMENT NO.
* IX = NODE NO. COMPOSED THE 'NEL' ELEMENT
* X = X AXIAL COORDINATION
* Y = Y AXIAL COORDINATION
* Z = Z AXIAL COORDINATION
* VX = X AXIAL NODE VELOCITY
* VY = Y AXIAL NODE VELOCITY
* VZ = Z AXIAL NODE VELOCITY
* AI =
* AJ =
* AK =
* NFACE = FRICTION SURFACE NUMBER

OUTPUT.
* U = NODE VELOCITY VECTOR
* COMMON/RZ1I AND /PAIV/A

COMMON (XVJU, DJ), (X(8),Y(8),Z(8))
COMMON /DEBUG/ IDBG(20), IOSP(50)
COMMON RELMV/ N1, N2, N3, N4, N5, N6, N7, N8
COMMON /NPVA/ HEADQO, DFLAG(20), CLIM, DFACT, ISTIF

COMMON /DEBUG/ JVT, MODEL
COMMON (XVJU, DJ), (X(8),Y(8),Z(8))
COMMON RZ1I, RELMV/ N1, N2, N3, N4, N5, N6, N7, N8
COMMON /NPVA/ HEADQO, DFLAG(20), CLIM, DFACT, ISTIF

COMMON /DEBUG/ JVT, MODEL
COMMON RZ1I, RELMV/ N1, N2, N3, N4, N5, N6, N7, N8
COMMON /NPVA/ HEADQO, DFLAG(20), CLIM, DFACT, ISTIF

DIMENSION GN(8),EN(8),ZN(8)

*...
* GN(I),EN(I),ZN(I)
*...

AA=1.0/8.0

GN(1)=AA*(1.0-AK)*(1.0-AK)
GN(2)=AA*(1.0-AK)*(1.0-AK)
GN(3)=AA*(1.0-AK)*(1.0-AK)
GN(4)=AA*(1.0-AK)*(1.0-AK)
GN(5)=AA*(1.0-AK)*(1.0-AK)
GN(6)=AA*(1.0-AK)*(1.0-AK)
GN(7)=AA*(1.0-AK)*(1.0-AK)
GN(8)=AA*(1.0-AK)*(1.0-AK)
EN(1)=AA*(1.0-AK)*(1.0-AK)
EN(2)=AA*(1.0-AK)*(1.0-AK)
EN(3)=AA*(1.0-AK)*(1.0-AK)
EN(4)=AA*(1.0-AK)*(1.0-AK)
EN(5)=AA*(1.0-AK)*(1.0-AK)
EN(6)=AA*(1.0-AK)*(1.0-AK)
EN(7)=AA*(1.0-AK)*(1.0-AK)
EN(8)=AA*(1.0-AK)*(1.0-AK)
ZN(1)=AA*(1.0-AK)*(1.0-AK)
ZN(2)=AA*(1.0-AK)*(1.0-AK)
ZN(3)=AA*(1.0-AK)*(1.0-AK)
ZN(4)=AA*(1.0-AK)*(1.0-AK)
ZN(5)=AA*(1.0-AK)*(1.0-AK)
ZN(6)=AA*(1.0-AK)*(1.0-AK)
ZN(7)=AA*(1.0-AK)*(1.0-AK)
ZN(8)=AA*(1.0-AK)*(1.0-AK)
XG=0.0
YG=0.0
ZG=0.0
XZ=0.0
YZ=0.0
ZZ=0.0

MEL - Tsukuba - July 1993
Improvements on the 3-D RIPLE Program

XE=0.0
YE=0.0
ZE=0.0
DO 30 I=1,8
XG=XG+<IN(I)*XG
YG=YG+<IN(I)*YG
ZG=ZG+<IN(I)*ZG
XZ=XZ+<IN(I)*XZ
YZ=YZ+<IN(I)*YZ
ZZ=ZZ+<IN(I)*ZZ
XE=XE+<IN(I)*XE
YE=YE+<IN(I)*YE
ZE=ZE+<IN(I)*ZE
30 CONTINUE

DJ=(XG*YE*ZZ+YG*ZE*XZ+XE*YZ*ZG)/DIV
DJV=1.0/DJ
DO 20 I=1,8
XI(I)=(YE*ZZ-ZE*YZ)*<IN(I)+{YZ*ZZ-ZG*YD)*EN(I)+
(XE*ZG-ZE*XG)*<IN(I))*DJV
YI(I)=(ZE*XZ-XE*ZZ)*<IN(I)+{XG*ZZ-ZG*XZ)*EN(I)+
(ZE*XZ-XE*ZZ)*<IN(I))*DJV
20 CONTINUE
IF (IOSP(3).EQ.0) GO TO 10
WRITE(15,')','ELEMENT NO.',NEL,':''
WRITE(15,')','DJ=',DJ
WRITE(15,')','XI(I),YI(I),ZI(I)=
WRJ1E(15,') (XI(I),I=I,8)
WRJ1E(15,') (YI(I),I=I,8)
WRJ1E(15,') (ZI(I),I=I,8)
WRITE(15,')','U(I)=U(24)
WRJ1E(15,') (U(I),I=I,6)
WRJ1E(15,') (U(I),I=I,12)
WRJ1E(15,') (U(I),I=I,18)
10 CONTINUE
RETURN
END

SUBROUTINE ELSEF(NEL,NFACE,lFIX,lINDEXN,epsx,epsy,epsz,sigel,sign,
epsel,uw,nsla)
COMMON /DEBUG/ IDBG(20) , IOSP(50)
COMMON /ELMV/ N1, N2, N3, N4, N5, N6, N7, N8
COMMON /nPVAI/ HEAD(20), IFLAG(20), CLIM, DFACT, ISTIF
COMMON /INDDATI/U(24),XJ(8),YJ(8),ZJ(8)
COMMON ./FACFJ/VKx(6), VKy(6),VKz(6),CX(6),CY(6),CZ(6),
AVEV(6), TAUF(6), AREA(6)
COMMON /NODEF/ NODEF(6,4),signor(6,4)
COMMON /IPUND/ PW2,PPR,PCR,DW2,DPR,DCR
DIMENSION NFACE(6,1),IFIX(I),INDEXN(I),uw(I),tmat(3,3)
IF(IFLAG(12).EQ.0) GO TO 400
DO 10 I=1,4
DO 10 J=1,6
NODEF(J,0)=0
signor(J,0)=0
10 CONTINUE
IF(IFIX(NFACE(1)+2).LT.0) NFACE(1)=6
IF(IFIX(NFACE(2)+2).LT.0) NFACE(2)=6
IF(IFIX(NFACE(3)+2).LT.0) NFACE(3)=6
IF(IFIX(NFACE(4)+2).LT.0) NFACE(4)=6
IF(IFIX(NFACE(5)+2).LT.0) NFACE(5)=6
IF(IFIX(NFACE(6)+2).LT.0) NFACE(6)=6
IF(IFIX(NFACE(7)+2).LT.0) NFACE(7)=6
IF(IFIX(NFACE(8)+2).LT.0) NFACE(8)=6
DO 300 N=1,6
IF(NFACE(N).EQ.0) GO TO 300
GO TO (I10,120,130,140,150,160), N
I10 CONTINUE

* *** computes stress in x,y and z direction
**
** SIGX=SIGX*(SIGN(0.66667*SIGX/EPSE1*EPSY)
** SIGY=SIGY*(SIGN(0.66667*SIGY/EPSE1*EPSY)
** SIGZ=SIGZ*(SIGN(0.66667*SIGZ/EPSE1*EPSY)
**
** DO 300 N=1,6
** IF(NFACE(N).EQ.0) GO TO 300
** GO TO (110,120,130,140,150,160), N
**
** T-EST FACE
** AX=X(6-X(1))
Improvements on the 3-D RIPLE Program

\[ \begin{align*}
AY & = Y_I - Y_K \\
AZ & = Z_I - Z_K \\
BX & = X_I - X_K \\
BY & = Y_I - Y_K \\
BZ & = Z_I - Z_K \\
YK_x(N) & = U(1)'U(4)'U(16)'U(13) \\
YK_y(N) & = U(2)'U(5)'U(17)'U(14) \\
\end{align*} \]

IF (IFIX(INDEXN(N1)'I).LT.0) NODEF(I,1)=1 \\
IF (IFIX(INDEXN(N2)'I).LT.0) NODEF(I,2)=2 \\
IF (IFIX(INDEXN(N6)'I).LT.0) NODEF(I,3)=3 \\
IF (IFIX(INDEXN(N8)'I).LT.0) NODEF(I,4)=4 \\
GO TO 200

** 2-N D FACE 

120 CONTINUE 

\[ \begin{align*}
AX & = X_I - X_K \\
AY & = Y_I - Y_K \\
AZ & = Z_I - Z_K \\
BX & = X_I - X_K \\
BY & = Y_I - Y_K \\
BZ & = Z_I - Z_K \\
V_K(N)-U(0)+U(7)+U(14)+U(16) \\
V_Ky(N)-U(1)+U(8)+U(15)+U(17) \\
\end{align*} \]

IF (IFIX(INDEXN(N1)'I).LT.0) NODEF(2,1)=1 \\
IF (IFIX(INDEXN(N2)'I).LT.0) NODEF(2,2)=2 \\
IF (IFIX(INDEXN(N6)'I).LT.0) NODEF(2,3)=3 \\
IF (IFIX(INDEXN(N8)'I).LT.0) NODEF(2,4)=4 \\
GO TO 200

** 3-R D FACE 

130 CONTINUE 

\[ \begin{align*}
AX & = X_I - X_K \\
AY & = Y_I - Y_K \\
AZ & = Z_I - Z_K \\
BX & = X_I - X_K \\
BY & = Y_I - Y_K \\
BZ & = Z_I - Z_K \\
V_K(N)-U(1)+U(4)+U(16)+U(19) \\
V_Ky(N)-U(1)+U(4)+U(16)+U(17) \\
\end{align*} \]

IF (IFIX(INDEXN(N1)'I).LT.0) NODEF(3,1)=1 \\
IF (IFIX(INDEXN(N2)'I).LT.0) NODEF(3,2)=2 \\
IF (IFIX(INDEXN(N6)'I).LT.0) NODEF(3,3)=3 \\
IF (IFIX(INDEXN(N8)'I).LT.0) NODEF(3,4)=4 \\
GO TO 200

** 4-TH FACE 

140 CONTINUE 

\[ \begin{align*}
AX & = X_I - X_K \\
AY & = Y_I - Y_K \\
AZ & = Z_I - Z_K \\
BX & = X_I - X_K \\
BY & = Y_I - Y_K \\
BZ & = Z_I - Z_K \\
V_K(N)-U(1)+U(4)+U(16)+U(19) \\
V_Ky(N)-U(1)+U(4)+U(16)+U(17) \\
\end{align*} \]

IF (IFIX(INDEXN(N1)'I).LT.0) NODEF(4,1)=1 \\
IF (IFIX(INDEXN(N2)'I).LT.0) NODEF(4,2)=2 \\
IF (IFIX(INDEXN(N6)'I).LT.0) NODEF(4,3)=3 \\
IF (IFIX(INDEXN(N8)'I).LT.0) NODEF(4,4)=4 \\
GO TO 200

** 5-TH FACE 

150 CONTINUE 

\[ \begin{align*}
AX & = X_I - X_K \\
AY & = Y_I - Y_K \\
AZ & = Z_I - Z_K \\
BX & = X_I - X_K \\
BY & = Y_I - Y_K \\
BZ & = Z_I - Z_K \\
V_K(N)+U(1)+U(4)+U(16)+U(19) \\
V_Ky(N)+U(1)+U(4)+U(16)+U(17) \\
\end{align*} \]

IF (IFIX(INDEXN(N1)'I).LT.0) NODEF(5,1)=1 \\
IF (IFIX(INDEXN(N2)'I).LT.0) NODEF(5,2)=2 \\
IF (IFIX(INDEXN(N6)'I).LT.0) NODEF(5,3)=3 \\
IF (IFIX(INDEXN(N8)'I).LT.0) NODEF(5,4)=4 \\
GO TO 200

if (ifa.eq.2) goto 180 
if (ifsta.eq.1) goto 200

ipd=2 
spix=0 
if(ifix(indexn(n1)=2),eq.-6) then 
ud=uu(indexn(n1) ) 
w=uu(indexn(n1) ) 
call transpose(mat,ud,w,ipd) 
sign=mat.(1,3)*spix+mat.(2,3)*spix+mat.(3,3)*spix 
endif 
if (sign .LT. 0.57735*spix) sign=0.57735*spix 
if (sign .GT. 0.57735*spix) sign=0.57735*spix 
Vxd=1*mat.(1,1)*spix+mat.(2,1)*spix+mat.(3,1)*spix 
Vyd=1*mat.(1,2)*spix+mat.(2,2)*spix+mat.(3,2)*spix 
Vzd=1*mat.(1,3)*spix+mat.(2,3)*spix+mat.(3,3)*spix 
endif 
Vxd=1*sign 
Vyd=1*sign 
Vzd=1*sign
Improvements on the 3-D RIPLE Program

Vydl=U(2)
end if
if(ifix(indexn(n2)+2).eq.-6)
ud=ww(indexn(n2))
wd=ww(indexn(n2)+1)
call transm(mat,ud,wd,ipd)
signz=mat(1,3)*sign+mat(2,3)*sign+mat(3,3)*signz
if(signz.gt.0.5775*signel)
Vxd1=mat(1,1)*U(4)+mat(2,1)*U(5)+mat(3,1)*U(6)
Vyd1=mat(1,2)*U(4)+mat(2,2)*U(5)+mat(3,2)*U(6)
signor{S,1)=signz
else
signor{S,1)=signz
Vxd1=U(4)
Vyd1=U(5)
end if
if(ifix(indexn(n4)+2).eq.-6)
ud=ww(indexn(n4))
wd=ww(indexn(n4)+1)
call transm(mat,ud,wd,ipd)
signz=mat(1,3)*sign+mat(2,3)*sign+mat(3,3)*signz
if(signz.gt.0.5775*signel)
Vxd4=mat(1,1)*U(10)+mat(2,1)*U(11)+mat(3,1)*U(12)
Vyd4=mat(1,2)*U(10)+mat(2,2)*U(11)+mat(3,2)*U(12)
signor{S,4)=signz
else
signor{S,4)=signz
Vxd4=U(10)
Vyd4=U(11)
end if
VKx(n)=Vydl+Vxd2+VxdJ+Vxd4
VKy(n)=Vydl+Vyd2+VydJ+Vyd4
GO TO 200
180 CONTINUE
AX=XJ(7)-XJ(5)
AY=YJ(7)-YJ(5)
AZ=ZJ(7)-ZJ(5)
BX=XJ(8)-XJ(6)
BY=YJ(8)-YJ(6)
BZ=ZJ(8)-ZJ(6)
IF(IFIX(INDEXN(N5)+2).LT.0) NODEF(6,1)=1
IF(IFIX(INDEXN(N6)+2).LT.0) NODEF(6,2)=2
IF(IFIX(INDEXN(N7)+2).LT.0) NODEF(6,J)=J
IF(IFIX(INDEXN(N8)+2).LT.0) NODEF(6,4)=4
if(ifix(eq=2) goto 170
if(fixa.eq.1) goto 200
ipd=1
sign=0.0
if(ifix(indexn(n5)+2).eq.-5)
upww=indexn(n5)
w=ww(indexn(n5)+1)
call transm(mat,up,w,ipd)
signz=mat(1,3)*sign+mat(2,3)*sign+mat(3,3)*signz
if(signz.gt.0.5775*signel)
Vxpl=mat(1,1)*U(4)+mat(2,1)*U(5)+mat(3,1)*U(6)
Vyp1=mat(1,2)*U(4)+mat(2,2)*U(5)+mat(3,2)*U(6)
signor{S,1)=signz
else
signor{S,1)=signz
Vxpl=U(4)
Vyp1=U(5)
end if
if(if(index(n+2).eq.-5) then
  up=us(index(n+6))
  wp=us(index(n)+1)
call transm(mat,u,p,wp)
sign=sign(1,3)*sign(mat(2,3))*sign(mat(3,3))*sign
if(sign.gt.0.57735*signel) sign=0.57735*signel
Vxp=mat(1,1)*u(16)+mat(2,1)*u(17)+mat(3,1)*u(18)
Vxp3=mat(1,2)*u(19)+mat(2,2)*u(20)+mat(3,2)*u(21)
  Vxp3=0.57735*signel
else
  Vxp2=mat(1,1)*u(16)+mat(2,1)*u(17)+mat(3,1)*u(18)
  Vxp3=mat(1,2)*u(19)+mat(2,2)*u(20)+mat(3,2)*u(21)
  Vxp3=0.57735*signel
endif
if(if(index(n+2).eq.-5) then
  up=us(index(n)+7)
  wp=us(index(n)+1)
call transm(mat,u,p,wp)
sign=sign(1,3)*sign(mat(2,3))*sign(mat(3,3))*sign
if(sign.gt.0.57735*signel) sign=0.57735*signel
Vxp=mat(1,1)*u(19)+mat(2,1)*u(20)+mat(3,1)*u(21)
Vxp4=mat(1,2)*u(22)+mat(2,2)*u(23)+mat(3,2)*u(24)
  Vxp4=0.57735*signel
else
  Vxp4=0.57735*signel
endif
if(if(index(n+2).eq.-5) then
  up=us(index(n)+8)
  wp=us(index(n)+1)
call transm(mat,u,p,wp)
sign=sign(1,3)*sign(mat(2,3))*sign(mat(3,3))*sign
if(sign.gt.0.57735*signel) sign=0.57735*signel
Vxp=mat(1,1)*u(22)+mat(2,1)*u(23)+mat(3,1)*u(24)
Vxp5=mat(1,2)*u(26)+mat(2,2)*u(27)+mat(3,2)*u(28)
  Vxp5=0.57735*signel
else
  Vxp5=0.57735*signel
endif
VKx(n)=Vxp1+Vxp2+Vxp3+Vxp4
VKy(n)=Vxp1+Vxp2+Vxp3+Vxp5
GO TO 200
170 continue
VKx(N)=U(13)+U(16)+U(19)+U(22)
VKy(N)=U(14)+U(17)+U(20)+U(23)
  Vxp5=0.57735*signel
signor(6,1)=0.57735*signel
signor(6,2)=0.57735*signel
signor(6,3)=0.57735*signel
signor(6,4)=0.57735*signel
200 CONTINUE
AI=AY*BZ-AZ*BY
AJ=AZ*BX-AX*BZ
AK=AX*BY-AY*BX
AREA(N)=SQRT(AI*AI+AJ*AJ+AK*AK)
CXP=N/AREA(N)
CYN=N/AREA(N)
CZN=N/AREA(N)
AIVITY(N)=V(0.025*SQRT(VKa(N)*VKa(N))+VKy(N)*VKy(N)+0.0001)
if(fabsa(eq.1 and eq.2 and eq.3) then
  signor(6,1)=1.1547*signel
signor(6,2)=1.1547*signel
signor(6,3)=1.1547*signel
signor(6,4)=1.1547*signel
endif
300 CONTINUE
400 CONTINUE
1000 CONTINUE
RETURN
END

*QVCL — SUBROUTINE QVCAL —

* SUBROUTINE QVCAL(Q)
* FUNCTION (SP. NO. = 2)
* Computes Q-VECTOR OF EACH ELEMENT
* CALLED BY
* QSCAL
* CALLS
* NONE
* INPUT
* COMMON/R2I/
**OUTPUT**
- Q = Q-VECTOR OF EACH ELEMENT.

**Q**

**COMMON** /XYZUI/ DLX(I),Y(I),Z(I)
**COMMON** /DEBUG/ IDBG(20), ISP(50)
**DIMENSION** Q(24)

**Q**

**DO** 10 I=1,8
Q(I) = Q(I) + 4.0*XI(I)*DJ
Q(3*(I-1)+4) = 0.0*Y(I)*DJ
Q(3*(I-1)+8) = 0.0*Z(I)*DJ
10 **CONTINUE**

**PRINTS Q-VECTOR.**
IF(ISP(2).EQ.0) GO TO 20
WRITE(IS,6) **"Q-VECTOR "**
WRITE(IS,6) (Q(I),I=1,24)
20 **CONTINUE**

**RETURN**

**END**

**BVCL**
--- SUBROUTINE BVCL ---

**SUBROUTINE** BVCL(BV)

**FUNCTION** (SP. NO. = 2)

**COMPUTES B-VECTOR OF EACH ELEMENT.**

**CALLED BY.**
- GSCAL
- **CALLS.**
- NONE

**INPUT.**
- **COMMON** /XYZUI/

**OUTPUT.**

**Q**

**COMMON** /XYZUI/ DLX(I),Y(I),Z(I)
**COMMON** /DEBUG/ IDBG(20), ISP(50)
**DIMENSION** BV(6,24)

**COMPUTES B-VECTOR.**

**DO** 100 I=1,6
**DO** 100 J=1,24
BV(I,J) = 0.0
100 **CONTINUE**

**DO** 200 I=1,8
BV(3*(I-1)+2) = XI(I)
BV(3*(I-1)+3) = YI(I)
BV(3*(I-1)+4) = ZI(I)
BV(3*(I-1)+5) = XI(I)
BV(3*(I-1)+6) = YI(I)
BV(3*(I-1)+7) = ZI(I)
200 **CONTINUE**

**RETURN**

**END**

**KMCAL**
--- SUBROUTINE KMCAL ---

**SUBROUTINE** KMCAL(NITER, DUMY, EK)

**FUNCTION** (SP. NO. = 1)

**COMPUTES K-MATRIX OF EACH ELEMENT.(COMPRESS)**

**CALLED BY.**
- **CALLS.**
- NONE

**INPUT.**
- NITER = INNER ITERATION NO.
- **COMMON** /PAIV/ AND /ZUI/

**OUTPUT.**

**Q**

**COMMON** /XYZUI/ DLX(I),Y(I),Z(I)
**COMMON** /DEBUG/ IDBG(20), ISP(50)
**DIMENSION** EK(24,24)

**DUMY** = 0.0
1**F** (NITER.GT.1) GO TO 50

**K**

**DIMENSION** EK(24,24)

**DUMY** = 0.0
1**F** (NITER.GT.1) GO TO 50

**FIRST INNER ITERATION ONLY.**
A6 = 0.66666666
A3 = 0.33333333
**D**

**2**

**DO** 100 J=1,8
**DO** 100 I=1,8
EK(I,J) = 5**A6*X(I)*X(J)+A3**Y(I)*Y(J)+Z(I)*Z(J)
100 **CONTINUE**

**RETURN**

**END**
Improvements on the 3-D RIPLE Program

EK(I,J)* = 6*Y(J)*Y(J)+A*I*X(I)*X(J)+Z(I)*Z(J)
EK(I,J)* = 6*Z(I)*Z(J)+A*I*X(I)*X(J)+Y(J)*Y(J)
EK(I,J,I,J) = 6*(Y(J)*X(I)+A*I*X(I)*X(J)+Y(J)*Y(J)
EK(I,J,J,J) = 6*(Z(I)*Z(J)+A*I*X(I)*X(J)+Y(J)*Y(J)
EK(I,J,J,J,I) = 6*(X(I)*Y(J)+A*I*X(I)*X(J)+Y(J)*Y(J)
EK(I,J,J,J) = 6*(Z(I)*Z(J)+A*I*X(I)*X(J)+Y(J)*Y(J)

I 00 CONTINUE
DO 200 I=J+1,24
EK(I,I)=EK(J,I)
200 CONTINUE
WRFFfiE K-MATRIX TO IKF FILE.
WRITE(IKF) EK
PRINTS K-MATRIX.
IF(lOSP(I).EQ.O) GO TO 40
WRITE(I5") , "K-MATRIX",
DO 50 I
WRITE(I5") (EK(I,J),I,J=1,24)
50 CONTINUE
40 CONTINUE
GO TO 60
READS K-MATRIX FROM IKF FILE.
50 CONTINUE
READ(IKF) EK
60 CONTINUE
RETURN
END

SUBROUTINE GSCON
SUBROUTINE GSCON(NEL,NUMDF,INDEXN,INDEXE,ES,H,DUMY,GS,PLOAD)
.* FUNCTION (SP. NO. = 6)
. * COMPUTES THE GLOBAL STIFFNESS MATRIX AND LOAD VECTOR.
. * CALLED BY.
. * GSCAL.
. * CALLS.
. * NONE.
.* INPUT
. * NEL = ELEMENT NUMBER.
. * NUMDF = TOTAL DEGREE OF FREEDOM.
. * INDEXN = EQUATION TABLE OF NODAL POINTS.
. * INDEXE = EQUATION TABLE OF ELEMENTS.
. * NBAND = BAND WIDTH.
. * ES(24,24)= ELEMENT STIFFNESS MATRIX.
. * H(25)= ELEMENT LOAD VECTOR.
.* OUTPUT
. * GS = GLOBAL STIFFNESS MATRIX.
. * PLOAD = GLOBAL LOAD VECTOR.
.* LOCAL EQUATION TABLE 'NG' OF THE 'NEL' ELEMENT.
NG(1)=INDEXN(N1)
NG(4)=INDEXN(N4)
NG(7)=INDEXN(N7)
NG(10)=INDEXN(N10)
NG(13)=INDEXN(N13)
NG(16)=INDEXN(N16)
NG(19)=INDEXN(N19)
NG(22)=INDEXN(N22)
DO 10 I=2,25
NG(I)=NG(I)+1
10 CONTINUE
NG(25)=INDEXN(N25)-1
ES(I-ROW,I-COLUMN) TO GS(II-ROW,II-COLUMN)
IF(ISTIF.EQ.I) THEN
DO 20 I=1,25
IF(I.EQ.7 .OR. I.EQ.8) GO TO 20
II=NG(I)
LO=(II-I )*NBAND-II+1
20 CONTINUE
RETURN
END

GSCN - SUBROUTINE GSCON

SUBROUTINE GSCON(NEL,NUMDF,INDEXN,INDEXE,ES,H,DUMY,GS,PLOAD)
.* FUNCTION (SP. NO. = 6)
. * COMPUTES THE GLOBAL STIFFNESS MATRIX AND LOAD VECTOR.
. * CALLED BY.
. * GSCAL.
. * CALLS.
. * NONE.
.* INPUT
. * NEL = ELEMENT NUMBER.
. * NUMDF = TOTAL DEGREE OF FREEDOM.
. * INDEXN = EQUATION TABLE OF NODAL POINTS.
. * INDEXE = EQUATION TABLE OF ELEMENTS.
. * NBAND = BAND WIDTH.
. * ES(24,24)= ELEMENT STIFFNESS MATRIX.
. * H(25)= ELEMENT LOAD VECTOR.
.* OUTPUT
. * GS = GLOBAL STIFFNESS MATRIX.
. * PLOAD = GLOBAL LOAD VECTOR.
.* LOCAL EQUATION TABLE 'NG' OF THE 'NEL' ELEMENT.
NG(1)=INDEXN(N1)
NG(4)=INDEXN(N4)
NG(7)=INDEXN(N7)
NG(10)=INDEXN(N10)
NG(13)=INDEXN(N13)
NG(16)=INDEXN(N16)
NG(19)=INDEXN(N19)
NG(22)=INDEXN(N22)
DO 10 I=2,25
NG(I)=NG(I)+1
10 CONTINUE
NG(25)=INDEXN(N25)-1
ES(I-ROW,I-COLUMN) TO GS(II-ROW,II-COLUMN)
IF(ISTIF.EQ.I) THEN
DO 20 I=1,25
IF(I.EQ.7 .OR. I.EQ.8) GO TO 20
II=NG(I)
LO=(II-I )*NBAND-II+1
20 CONTINUE
RETURN
END

MEL - Tsukuba - July 1993

---
Improvements on the 3-D RlPLE Program

SUBROUTINE PMCAL(B,EPSEL,SIGEL,EK,ALAM)

FUNCTION. (SP. NO. ~ S)

COMPUTES Y·P·MATRIX

CALLED BY.

GSCAL

CALLS.

MATMAT

INPUT.

B = B-VECTOR

EPSEL = EQUIVALENT STRAIN VELOCITY

SIGEL = EQUIVALENT STRESS

EK = K-MATRIX

OUTPUT.

Y·P·MATRIX

COMMON /DEBUG, IOBG(20) , 10SP(SO)

COMMON IINPVN HEAD(20) ,IFLAG(20) , CUM, DFACT, ISTIF

INITV , MODEL

DIMENSION B(24), EK(24,24),BTB(24,24)

COMPUTES B’-VECTOR. ( B’=SQR(2J) * EPSEL * B )

F=1,0*EPSEL

DO 10 1=1,24

B(1)=F

10 CONTINUE

COMPUTES P’-MATRIX ( P’= K - B’·B’T )

CALL MATMAT(B,24;B,24,0,0,BTB)

COMPUTES Y·P·MATRIX ( Y·P =2J * SIGEL / EPSEL * P’ )

F=SIGEL/EPSEL

DO 20 1=1,24

DO 20 1=1,24

EK(I,J)=F*EK(I,J)+BTB(I,J)*ALAM

20 CONTINUE

PRINTS Y·P·MATRIX.

IF(LOSP(S).EQ.0) GO TO 40

WRITE(1S,,)

Y·P·MATRIX

DO 30 1=1,24

WRITE(IS,,) (EK(I,I),I=1,24)

30 CONTINUE

40 CONTINUE

RETURN

END

SUBROUTINE FRIC(ES,H,NF ACE,SIGN,EPSEL,SIGEL,NEL)

COMMON /DEBUG/ IDR(20) , 10SP(S0)

COMMON INVPA, HEAD(20) , IFLAG(20) , CUM, DFACT, ISTIF

INITV, MODEL

DIMENSION B(24), EK(24,24),BTB(24,24)

COMPUTES P’-VECTOR. ( P’ = SQR(2J) / EPSEL * B )

F=1,0*EPSEL

DO 10 1=1,24

B(1)=F

10 CONTINUE

COMPUTES P’-MATRIX. ( P’ = K - B’·B’T )

CALL MATMAT(B,24;B,24,0,0,BTB)

COMPUTES Y·P·MATRIX. ( Y·P = 2J * SIGEL / EPSEL * P’ )

F=SIGEL/EPSEL

DO 20 1=1,24

DO 20 1=1,24

EK(I,J)=F*EK(I,J)+BTB(I,J)*ALAM

20 CONTINUE

PRINTS Y·P·MATRIX.

IF(LOSP(S).EQ.0) GO TO 40

WRITE(1S,,)

Y·P·MATRIX

DO 30 1=1,24

WRITE(IS,,) (EK(I,I),I=1,24)

30 CONTINUE

40 CONTINUE

RETURN

END

SUBROUTINE FREC(SH,NFACE,SIGN,EPSEL,SIGEL,NEL)
COMMON DEBUG1,IDBG(20),IOSP(10)
COMMON /LMV/ N1, N2, N3, N4, N5, N6, N7, N8
COMMON /NPV/ HEAD(20),IFLAG(20),CLM,DFACT,ISTIF
* INITV, MODEL
COMMON ADDDATU,X(I),Y(I),Z(I)
COMMON /FACE/ VKx(6),VKy(6),VKz(6),CX(6),CY(6),CZ(6),
** AVEV(6), TAUF(6), AREA(6)
COMMON /NODE/ NODEF(6,4),signor(6,4)
DIMENSION ER(25,25), H(25), NX(4),NY(4),NZ(4)
DIMENSION NFACE(6,1)

IF(IFLAG(12).EQ.0) GO TO 600
DO 500 M=1,6
IF(NFACE(M,IFLAG)EQ.0) GO TO 500
** ELEMENT MATRIX MODIFICATION
GO TO (110,120,130,140,150,160) , M

1-ST SURFACE
110 CONTINUE
NF1=1
NF2=2
NF3=6
NF4=5
GO TO 170

2-ND SURFACE
120 CONTINUE
NF1=2
NF2=3
NF3=7
NF4=6
GO TO 170

3-ED SURFACE
130 CONTINUE
NF1=3
NF2=4
NF3=8
NF4=7
GO TO 170

4-TH SURFACE
140 CONTINUE
NF1=4
NF2=1
NF3=5
NF4=8
GO TO 170

5-TH SURFACE
150 CONTINUE
NF1=5
NF2=3
NF3=2
NF4=1
GO TO 170

6-TH SURFACE
160 CONTINUE
NF1=6
NF2=7
NF3=8
NF4=8

170 CONTINUE
NX(1)=NF1*3-2
NX(2)=NF2*3-2
NX(3)=NF3*3-2
NX(4)=NF4*3-2
NY(1)=NX(1)+1
NY(2)=NX(2)+1
NY(3)=NX(3)+1
NY(4)=NX(4)+1
NZ(1)=NY(1)+1
NZ(2)=NY(2)+1
NZ(3)=NY(3)+1
NZ(4)=NY(4)+1

** DO 200 I=1,4
IF(NODEF(I).EQ.0) GO TO 200
ES(NX(I),NY(I))=ES(NX(I),NY(I))+signor(m,i)*area(m)(16.0*avev(m))
ES(NY(I),NZ(I))=ES(NY(I),NZ(I))+signor(m,i)*area(m)(16.0*avev(m))
ES(NZ(I),NX(I))=ES(NZ(I),NX(I))+signor(m,i)*area(m)(16.0*avev(m))

** DO 210 J=1,4
IF(I.EQ.0) GO TO 210
ES(NX(I),NY(I))=ES(NX(I),NY(I))+signor(m,i)*area(m)
ES(NX(I),NZ(I))=ES(NX(I),NZ(I))+signor(m,i)*area(m)
ES(NY(I),NX(I))=ES(NY(I),NX(I))+signor(m,i)*area(m)
ES(NY(I),NZ(I))=ES(NY(I),NZ(I))+signor(m,i)*area(m)
ES(NZ(I),NX(I))=ES(NZ(I),NX(I))+signor(m,i)*area(m)
ES(NZ(I),NY(I))=ES(NZ(I),NY(I))+signor(m,i)*area(m)
Improvements on the 3-D RIPLE Program

\[ ES(NZ(I),NZ(J)) = ES(NZ(I),NZ(J)) - \text{signor}(m,i)'area(m) \cdot U(NZ(I) \cdot U(NZ(J)))/(512.0 \cdot \text{avev3}) \]

\[ ES(NX(I),NY(J)) = ES(NX(I),NY(J)) - \text{signor}(m,i)'area(m) \cdot U(NX(I) \cdot U(NY(J)))/(512.0 \cdot \text{avev3}) \]

\[ ES(NY(I),NZ(J)) = ES(NY(I),NZ(J)) - \text{signor}(m,i)'area(m) \cdot U(NY(I) \cdot U(NZ(J)))/(512.0 \cdot \text{avev3}) \]

\[ ES(NX(I),NZ(J)) = ES(NX(I),NZ(J)) - \text{signor}(m,i)'area(m) \cdot U(NX(I) \cdot U(NZ(J)))/(512.0 \cdot \text{avev3}) \]

\[ ES(NX(I),NX(J)) = ES(NX(I),NX(J)) - \text{signor}(m,i)'area(m) \cdot U(NX(I) \cdot U(NX(J)))/(256.0 \cdot \text{avev3}) \]

\[ ES(NY(I),NY(I)) = ES(NY(I),NY(I)) - \text{signor}(m,i)'area(m) \cdot U(NY(I) \cdot U(NY(I)))/(256.0 \cdot \text{avev3}) \]

\[ ES(NZ(I),NZ(I)) = ES(NZ(I),NZ(I)) - \text{signor}(m,i)'area(m) \cdot U(NZ(I) \cdot U(NZ(I)))/(256.0 \cdot \text{avev3}) \]

\[ ES(NX(J),NY(J)) = ES(NX(J),NY(J)) - \text{signor}(m,i)'area(m) \cdot U(NX(J) \cdot U(NY(J)))/(256.0 \cdot \text{avev3}) \]

\[ ES(NY(I),NZ(J)) = ES(NY(I),NZ(J)) - \text{signor}(m,i)'area(m) \cdot U(NY(I) \cdot U(NZ(J)))/(256.0 \cdot \text{avev3}) \]

\[ ES(NZ(I),NX(J)) = ES(NZ(I),NX(J)) - \text{signor}(m,i)'area(m) \cdot U(NZ(I) \cdot U(NX(J)))/(256.0 \cdot \text{avev3}) \]

\[ ES(NX(I),NY(J)) = ES(NX(I),NY(J)) - \text{signor}(m,i)'area(m) \cdot U(NX(I) \cdot U(NY(J)))/(256.0 \cdot \text{avev3}) \]

\[ ES(NY(I),NZ(J)) = ES(NY(I),NZ(J)) - \text{signor}(m,i)'area(m) \cdot U(NY(I) \cdot U(NZ(J)))/(256.0 \cdot \text{avev3}) \]

\[ ES(NZ(I),NX(J)) = ES(NZ(I),NX(J)) - \text{signor}(m,i)'area(m) \cdot U(NZ(I) \cdot U(NX(J)))/(256.0 \cdot \text{avev3}) \]

\[ H(NX(I)) = H(NX(I)) - \text{signor}(m,i)'area(m) \cdot U(NX(I))/(4.0 \cdot \text{AVEV} M) \]

\[ H(NY(I)) = H(NY(I)) - \text{signor}(m,i)'area(m) \cdot U(NY(I))/(4.0 \cdot \text{AVEV} M) \]

\[ H(NZ(I)) = H(NZ(I)) - \text{signor}(m,i)'area(m) \cdot U(NZ(I))/(4.0 \cdot \text{AVEV} M) \]

210 CONTINUE

37
Improvements on the 3-D RIPLE Program

X48 = X(4) - X(8)
Y48 = Y(4) - Y(8)
Z48 = Z(4) - Z(8)

D74 = Y*Z48 - Z74*Y48
DS4 = Z*Z48 - ZS4*Z48
D75 = Y*Z56 - Z75*Y56

VTH4 = ABS(0.166667*(X76*DS2 + X26*D72 + X4S*D24 + X7S*D74))

X25 = X(2) - X(5)
X75 = X(7) - X(5)
X45 = X(4) - X(5)
Y25 = Y(2) - Y(5)
Y75 = Y(7) - Y(5)
Y45 = Y(4) - Y(5)
Z25 = Z(2) - Z(5)
Z75 = Z(7) - Z(5)
Z45 = Z(4) - Z(5)

D74 = Y*Z45 - Z74*Y45
D24 = Z*Z45 - ZS4*Z45
D75 = Y*Z56 - Z75*Y56

VTH5 = ABS(0.166667*(X2S*D74 + X7S*D24 + X4S*D27))

FRICPN = 0.0
DEFPN = 0.0

COMPUTES FRICTION POWER

DO 100 M=1,6
IF(NFACE(M,NEL).EQ.O) GO TO 100
FRICPN = FRICPN + TAU(M) * AREA(M) * AVEV(M)
100 CONTINUE

COMPUTES DEFORMATION POWER

DEFPN = SIGEL * EPSG*VEL

COMPUTES TOTAL ELEMENT POWER

TPOWER = TPOWER + DEFPN + FRICPN

END

GSCAL - SUBROUTINE GSCAL

SUBROUTINE GSCAL(NSTA, NITE, IX, X, Y, Z, VX, VY, VZ, EPS, SIGE, NDF, INDEXN, INDEXE, NFACE, DUMY, GS, PLOAD, SIGELV, IFIX, SCN, UW)

FUNCTION (SP. NO. = 7)

COMPUTES GLOBAL STIFFNESS MATRIX AND LOAD VECTOR

CALLED BY

STIF

CALLS

KMCAL, ESLSET, MATMAT, ESMET, ESCAL, GS, GCON, VEXC

INPUT

NSTA = STAGE NUMBER
NITE = INNER ITERATION NO.
IX = NODE NO. COMPOSED THE ELEMENT
X = X AXIAL COORDINATES
Y = Y AXIAL COORDINATES
Z = Z AXIAL COORDINATES
VX = X AXIAL NODE VELOCITY
VY = Y AXIAL NODE VELOCITY
VZ = Z AXIAL NODE VELOCITY
EPS = EQUIVALENT STRAIN VELOCITY AND STRAIN
SIGE = AVERAGED ELEMENT STRESS
NDF = TOTAL DEGREE OF FREEDOM
INDEXN = EQUATION TABLE OF NODAL POINTS.
INDEXE = EQUATION TABLE OF ELEMENTS.
NFACE = FRICITION SURFACE NUMBER
UW = B-SPLINE PATCH PARAMETER U AND W AT CONTACT SURFACE.

OUTPUT

MEL - Tsukuba - July 1993
Improvements on the 3-D RlPLE Program

* GS = I-DIMENSIONAL GLOBAL STIFFNESS MATRIX
* PLOAD = GLOBAL LOAD VECTOR
* FLOAD = GLOBAL LOAD VECTOR
* SIGELV = YIELD ELEMENT STRESS

** COMMON PAIVAV# PAI
COMMON X(Y(24),X(24),Y(24),Z(24)
COMMON #ELMVA# N1, N2, N3, N4, N5, N6, N7, N8
COMMON #OLMVA# NBAND, NBNDF
COMMON FILE/N, LP, MK, ISF
COMMON #ASIVAV# NUMEL, NUMDF, NUMNL, NUMFD
COMMON #BEVUG# EBSC(20), IOS(20)
COMMON SCALE/ FACTOR
COMMON #FACT# RMN, JTAU
COMMON #ACIE# VX(6), VX(6), CY(6), CY(6), C2(6),
* AVEV(6),tau(6), AREA(6)
COMMON #INVP# HEAD(20), IFLAG(20), CLIM, DFACT, IISTIF
* , INTIV. MODEL
COMMON #PARAM# ALFA, GOG
COMMON #NAND# U(24),X(24),Y(24),Z(24)
COMMON #PAR# IMIT/ ALAM
COMMON #FRU# TVP WER
DIMENSION IX(1,1), Y(1,1), Z(1,1), VX(1,1),VY(1,1),VZ(1,1),O(24)
* , EK(24,24), B(24), EPS(9,1), SIGE(1) , H(25), ES(25,25)
* , E(15), LOAD(NDF,2) , INDEX(1), INDEX(1)
* , SIGELV(15), TVP(6,24), W(NDI), FPAC(6,1),FPAC(1,1), SCN(1)

** DUMY=0.0
DUM1=SIGE(1)
NITER=NITE
NUMSTA=NSTA
NUMDF=NDF

* **** READS THE GLOBAL LOAD VECTOR.
REWIND ISF
** DEBUG PRINT
* IF(ISF(7).EQ.0) GO TO 1
WRITE(15,'(12,2F10.4)') I, PLOAD(I,1)
1 CONTINUE
REWIND IKF
** MULTIPLIES THE SCALE FACTOR.
DO 3 N=1,NUMDF
PLOAD(1,N)=PLOAD(1,N)*FACT R
3 CONTINUE
** --- CLEAR POWER ---
TPOWER=0.0

* **** GLOBAL STIFFNESS AREA CLEAR.
**
CALL CLEAR(GS,NBNDF)
DO 100 N=1,NUMEL
NEL=N
N1=IX(1,NEL)
N2=IX(2,NEL)
N3=IX(3,NEL)
N4=IX(4,NEL)
N5=IX(5,NEL)
N6=IX(6,NEL)
N7=IX(7,NEL)
N8=IX(8,NEL)

* SIGELV(VN)-0.0
EPS2(N)-0.0
EPSM0=0.0
DO 1001 J=1,25
DO 1002 J=1,25
ES(J)=0.0
1002 CONTINUE
H(0)=0.0
1001 CONTINUE
CALL MSET(NEL,X,Y,Z,VX,VY,VZ)
** Computes element information.
CALL ESETF(NEL,0,0,0,0,0)
* ** Computes Q-vector.
** CALL QVCAL(0)
** Computes BV-vector.

CALL BVCAL(BV)

EPS(I,N)=0.0
EPS(2,N)=0.0
EPS(3,N)=0.0
EPS(4,N)=Q.0
EPS(5,N)=Q.0
EPS(6,N)=0.0
DO 8 I=1,6
DO 8 J=I,24
EPS(I,N)=EPS(I,N)+BV(I,J)'U(J)
8 CONTINUE
EPS(9,N)=EPS(1,N)+EPS(2,N)+EPS(3,N)
CONPlITES Q'U-VALUE.
IF(ISTIF.GT.1) GO TO 50
CALL MATMA2(Q,I,24,U,I,O,O,QTU)
IF(ISOP(7).EQ.0) GO TO 6
WRITE(15,') .. QTU .. QTU ..'
6 CONTINUE
H(25)=QTU
50 CONTINUE
DO 30 I=1,24
IF(ISTIF.EQ.I) THEN
ES(25,25)=-Q(1)
END IF
30 CONTINUE
IF(ISTIF.EQ.I) THEN
ES(25,25)=0.0
END IF
DO 1000 I=1,2
DO 1000 J=1,2
DO 1000 K=1,2
IF(I.IEQ.I) THEN
AI=0.5
ELSE
AI=-0.5
END IF
IF(J.IEQ.I) THEN
AJ=0.5
ELSE
AJ=-0.5
END IF
IF(K.IEQ.I) THEN
AK=0.5
ELSE
AK=-0.5
END IF
CALL ELSET(NEL,AI,AJ,AK)
COMPlITES K-MATRIX.
CALL KMCAL(NITER, 0 ,EK )
COMPUTES B-VECTOR.
CALL MATMA3(EK,24,24,U,I,O,0,B)
COMPUTES EQUIVALENT STRAIN VELOCITY.
CALL MATMA2(U,I,24,B,I,O,0,EPS2)
IF(EPS2.LT.0.0) EPS2=0.0
EPSEL=SQRT(EPS2)
EPS(1,NEL)=EPS2
EPSDM=EPSEL*0.125+EPSDM
IF(ISOP(7).EQ.0) GO TO 2
WRITE(15,') .. EPS2 .. EPS2 ..
2 CONTINUE
COMPUTES THE EQUIVALENT STRESS.
CALL ESCAL(NUMSTA,NEL,EPS, 0 ,SIGEL)
SIGELV(N)=SIGEL '0 12S+SIGEL V(N)
COMPUTES Y'H-VECTOR.
F=-DJ'SIGELJEPSEL 'FACTOR
DO 101 I=1,24
H(I)=F(I)+HI
10 CONTINUE
IF(ISOP(7).EQ.0) GO TO 4
WRITE(15,') .. H-VECTOR .. F
4 CONTINUE
**Improvements on the 3-D RIPLE Program**

WRITE(15,*) (I,H(I),I=1,24)
4 CONTINUE

**** COMPUTES $Y^P$-MATRIX.

CALL PMCAL(0,EPSEL,SIGEL,EX,ALAM)

**** COMPUTES ELEMENT STIFFNESS MATRIX AND MULTIPLIES THE FACTOR.

DO 20 I=1,24
DO 20 J=1,24
ES(J,I) = EK(J)*FACTOR*ES(I,J)
20 CONTINUE
100 CONTINUE

EPS(7,NEL) = EPSDM
EPSEL = EPS(7,NEL)
SIGEL = SIGE(NEL)
epsx = eps(1,NEL)
epsy = eps(2,NEL)
epsz = eps(3,NEL)
SIGN = SIGE(NEL)

CALL ELSETF(NEL,NFACE,IFIX,INDEXN,epsx,epsy,epsz,sigel,sigel)

** FRICTION MODEL

CALL FRIC(EV,N,EPSEL,SIGE)

CONSTRUCTS THE GLOBAL STIFFNESS MATRIX AND LOAD VECTOR.

CALL GSCON(NEL,NUMDF,INDEXE,ES,H,GS,PLOAD)

**

100 CONTINUE

EPS(7,NEL) = EPSDM
EPSEL = EPS(7,NEL)
SIGEL = SIGE(NEL)
epsx = eps(1,NEL)
epsy = eps(2,NEL)
epsz = eps(3,NEL)
SIGN = SIGE(NEL)

CALL ELSETF(NEL,NFACE,IFIX,INDEXN,epsx,epsy,epsz,sigel,sigel)

** FRICTION MODEL

CALL FRIC(EV,N,EPSEL,SIGE)

CONSTRUCTS THE GLOBAL STIFFNESS MATRIX AND LOAD VECTOR.

CALL GSCON(NEL,NUMDF,INDEXE,ES,H,GS,PLOAD)

**

100 CONTINUE

EPS(7,NEL) = EPSDM
EPSEL = EPS(7,NEL)
SIGEL = SIGE(NEL)
epsx = eps(1,NEL)
epsy = eps(2,NEL)
epsz = eps(3,NEL)
SIGN = SIGE(NEL)

CALL ELSETF(NEL,NFACE,IFIX,INDEXN,epsx,epsy,epsz,sigel,sigel)

**

100 CONTINUE

CALL VEXC(NUMDF,NBAND,GS,PLOAD,IFIX,SCN,INDEXN,UW)

**

PRINTS THE GLOBAL STIFFNESS MATRIX AND LOAD VECTOR.

IF(IFIX(7) .EQ. 0) GO TO 300
WRITE(15,*) '********** SP.GSCAL INFORMATION **********
WRITE(15,*) 'GLOBAL STIFFNESS MATRIX '*
ISTR=1
200 CONTINUE

END=ISTR+NBAND-1
WRITE(15,*) (LGO(I),I=ISTR,END)
ISTR=ISTR+1
IF(ISTR.LT.NBAND) GO TO 200
WRITE(15,*) 'GLOBAL LOAD VECTOR '*
ISTR=ISTR+1
300 CONTINUE
RETURN

**SUBROUTINE VEXC** —

**SUBROUTINE VEXC(ND,NBAND,GS,PLOAD,IFIX,SCN,INDEXN,UW)**

COMMON /ASVA/ NUMEL, NUMNP, NUMDF, NUMLD
DIMENSION GS(1),PLOAD(ND,2),IFIX(ND),SCN(ND),INDEXE(1)

U = (UW(3,1))
W = UW(3,2)
CALL TRANS(TMAT,U,W,IPD)
DO 7000 I=1,ND
IF(IFIX(I) .EQ. 6) OR IFIX(I) .EQ. 5 OR IFIX(I) .EQ. 4 OR
IF(IFIX(I) .EQ. 3) THEN
IF(IFIX(I) .EQ. 5) IPD = 1
IF(IFIX(I) .EQ. 4) IPD = 2
U = UW(I-2)
W = UW(I-1)
CALL TRANS(TMAT,U,W,IPD)
DO 6000 I=2,NBAND-2
L3 = I-2+NBAND+1
L2 = L1-NBAND+1
L1 = L2-NBAND+1
A= GS(L3)
A+= GS(L2)
AK = GS(L1)
GS(L1) = TMAT(1,1)*A+TMAT(2,1)*A+TMAT(3,1)*AK

**MEL - Tsukuba - July 1993**

41
GS(L2) = TMAT(1,2)*AI+TMAT(2,2)*AJ+TMAT(3,2)*AK
GS(L3) = TMAT(1,3)*AI+TMAT(2,3)*AJ+TMAT(3,3)*AK

END IF
7000 CONTINUE
RETURN
* ESCL  -- SUBROUTINE ESCAL --
* SUBROUTINE ESCAL(NSTA,NEL,EPS, DUMY,SIGEL)
* FUNCTION (SP. NO. -34)
* COMPUTES THE EQUIVALENT STRESS.
* CALLED BY.
* GSCAL
* CALLS.
* USTR
* INPUT.
* NSTA = STAGE NUMBER
* NEL = ELEMENT NUMBER.
* EPS7(NEL)=EQUIVALENT STRAIN VELOCITY.
* EPS8(NEL)=EQUIVALENT STRAIN
* Y0,CK,CM,CN = COEFFICIENT IN THE STANDARD RELATION.
* OUTPUT.
* SIGEL = EQUIVALENT STRESS.
* COMMON (COEFF,Y0,CK,CM,CN, A1, B1, A2, B2, A3, B3
* COMMON ANPVYA HEAD20, IFLAG20, CLIM, DFAC,T ISTIF
* INTIV, MODEL

--- MEL - Tsukuba - July 1993 ---
COMMON /DEBUG/ IDBG(20), IOEP(30)
DIMENSION EPS(0,1)
***
DUMMY=0.0
NDUM1=NEL
IF(DUM1.EQ.0) GO TO 100
*** USER'S SUBROUTINE
CALL USTR(EPS,SIGEL,NEL)
GO TO 100
*** STANDARD RELATION
100 CONTINUE
** CM=0.0
** CN=0.0
** SIGEL=CK*EPS(7,NEL)**CM*EPS(8,NEL)**CN
***
200 CONTINUE
*** DEBUG PRINT
IF(IOSP(34).EQ.0) GO TO 300
WRITE(IS,'** SPESCAL INFORMATION **')
WRITE(IS,'** EPS(7,NEL) **EPS(8,NEL) &
** EPS(9,NEL) **YO,CK,CM,CN**Y0,CK,CM,CN
WRITE(IS,'** SIGEL **SIGEL
300 CONTINUE
***
RETURN
END
SUBROUTINE CLEAR(A,IMAX)
FUNCTION(SP. NO. = 9)
CLEAR TIiE AREA A(I)-A(IMAX)
** CALLED BY.
** GSCAL
** CALLS.
** NONE
** INPUT.
** A(I) - A(IMAX)
** OUTPUT.
** A(I) = 0.0
** A(IMAX) = 0.0
***
DIMENSION A(I)
DO 10 I=I,IMAX
A(IFO.O
10 CONTINUE
RETURN
END
SUBROUTINE JUDGE(NUMITE,INDEXN,NUMNP,INDEXE,NUMEL,PLOAD,NUMDF,
VX,VY,VZ,ASIG,IFIX,SCN,UW)
** COMPUTES THE NEW NODAL POINT VELOCITY
** AND JUDGES THE CONVERGENCE.
** CALLED BY.
** CALC
** CALLS.
** NONE
** INPUT.
** NUMITE = INNER ITERATION NUMBER.
** INDEEQ = EQUATION TABLE OF NODAL POINTS.
** NUMNP = NUMBER OF NODAL POINTS.
** INDEXE = EQUATION TABLE OF ELEMENTS.
** NUMEL = NUMBER OF ELEMENTS.
** PLOAD(I) = INCREMENTAL VELOCITY OF THE NODAL POINTS.
** NUMDF = TOTAL NUMBER OF D.O.F.
** VX = NODAL POINT VELOCITY IN R-DIRECTION.
** VY = NODAL POINT VELOCITY IN Z-DIRECTION.
** VZ = NODAL POINT VELOCITY IN Z-DIRECTION.
** ASIG = AVERAGED STRESS OF THE ELEMENT.
***
COMMON /DEBUG/ IDBG(20), IOEP(30)
COMMON /NPVA/ HEAD(20), SLIM, DFAC, ISTIF
** CM=0.0
** INTIV . MODEL
COMMON /FILE/ IN, LP, NKF, ISF
COMMON /RESULT/ RATIO

--- Improvements on the 3-D RIPLE Program ---

--- MEL - Tsukuba - July 1993 ---

43
DIMENSION VX(NUMNP), VY(NUMNP), INDEXN(NUMNP), INDEXE(NUMEL),
    PLOAD(NUMDF,2), ASIG(NUMEL), VZ(NUMNP), IFIX(NUMDF),
    SCN(NUMDF), UW(NUMDF), TMAT(3,3)

***

VNORM=0.0
DVNORM=0.0
DO 600 N=I,NUMNP
   ID=INDEXN(N)
   IF(IFIX(ID+2).EQ.-6.0R.IF1X(ID+2).EQ.-5.OR.IFIX(ID).EQ.4)THEN
      IF(IFIX(ID+2).EQ.-6) IPD = 3
      IF(IFIX(ID+2).EQ.-5) IPD = 2
      U = UW(ID)
      W = UW(ID+1)
      CALL TRANSM(TMAT, U, W, IPD)
      DVX = TMAT(1,1)*PLOAD(ID,1)+TMAT(1,2)*PLOAD(ID+1,1)
      DVY = TMAT(2,1)*PLOAD(ID,1)+TMAT(2,2)*PLOAD(ID+1,1)
      DVZ = TMAT(3,1)*PLOAD(ID,1)+TMAT(3,2)*PLOAD(ID+1,1)
   ELSE
      DVX=PLOAD(ID,1)
      DVY=PLOAD(ID+1,1)
      DVZ=PLOAD(ID+2,1)
   END IF
   COMPUTES THE NORM.
   VNORM=VNORM+VX(N)'VX(N)+VY(N)'VY(N)+VZ(N)'VZ(N)
   DVNORM=DVNORM+DVX'DVX+DVY'DVY+DVZ'DVZ
600 CONTINUE
   RATIO=SQRT(DVNORM/VNORM)
***
   COMPUTES THE AVERAGED STRESS OF ELEMENT.
   IF(IFIX(ID).EQ.1) THEN
      DO 120 N=1,NUMEL
         NE=INDEXE(N)
         NEP=INDEXN(NE)-1
         ASIG(N)=PLOAD(NEP,1)
         WRITE(6,') ASIG(N)
      120 CONTINUE
   IF(RATIO.GT.CLIM .AND. NUM1TE.LT.IFLAG(6)) GO TO ISO
   ITERATION IS CONVERGED.
   IFLAG(20)=1
   END IF
ISO CONTINUE
   WRITE(6,') 'ITERATION NO.=',NUM1TE,
   WRITE(LP,') 'ITERATION NO.=',NUM1TE,
   'RATIO=',RATIO
   'RATIO=',RATIO
200 CONTINUE
RETURN
END

SUBROUTINE CALC(A,NUMITE)
    'FUNCTION(SF NO. =33)
    SOLVES THE MATRIX EQUATION.
    CALLED BY
    MAIN
    CALLS
    CHECK, PAGE, SOLVE,JUDGE
    INPUT:
    MTOT = MAXIMUM ADDRESS NUMBER IN BLANK COMMON.
    NMAX = CURRENT ADDRESS NUMBER IN USED BLANK COMMON.
    NUMITE = INNER ITERATION NUMBER.
    OUTPUT:
    NMAX = CURRENT ADDRESS NUMBER IN USED BLANK COMMON.
    COMMON BASAV NUMEL, NUMNP, NUMDF, NUMLD
    COMMON (LINES/ LTOP, LINE, LEND)
    COMMON (ADDRESS M'TOT, M'MAX, M'MNS, M'MAM, M'MBE, M'MES, M'MBE ,
    M'MBS, M'MBE, M'MBS, M'MLS, M'MAM, M'MBE, M'MES, M'MBE, M'MBS, M'MBE, M'MES )

---

MEL - Tsukuba - July 1993
Improvements on the 3-D RIPLE Program

**NMAX, NMV, MBRS, MNUW**
**COMMON ADDRESSM NMAX, NMP, NIPF, NPE, NNAS, NNEE**
**NNS, NNS, NNG, NN, NNN, NNN, NNN, NNN, NNN, NNN**
**COMMON SOLV/ NITM, NITM**
**COMMON DEBUGF IDBG(20), ISOF(10)**

**DIMENSION A(1)**

**NITE=NITM**

**NMAX=NMAX+1**

**NMAX=NNAS+NUMEL**

IF(NITE.GT.1) GO TO 100

IF,IDBG, IEQ.0. GO TO 100
CALL PAGE,LINE,6,LEND
CALL CHECK(MOTOT,NMAX,' CA','LC ')
100 CONTINUE

NMAX=NNAS

SOLVES THE MATRIX EQUATION.
CALL SOLVE(A(NMM),NBNDF,A(NNP),NUMDF,A(MMBS)

**COMPUTES THE NEW NODAL VELOCITY AND JUDGES THE TOLERANCE.**
CALL JUDGE(NITM,A(MMIS),NUMNP,A(MMIM),NUMEL,A(NNP),NUMDF,

* A(MMIS),A(MMIV),A(MMVI),A(NMM),A(NNP),A(MMBS),A(MMBS)

RETURN

END

**SOLVE** --- SUBROUTINE SOLVE ---

**SUBROUTINE SOLVE(GS,NBNDF,PLOAD,NDF,IFIX)**
**FUNCTION (SP. NO. = 10)**
**SOLVES THE EQUATION BY MEANS OF CHOLESKY DECOMPOSITION.**
**CALLED BY.**
**MAIN**
**CALLS**
* DCOMP, DWD, BAKWD, MULTI, SOLOUT
**INPUT.**
* GS = GLOBAL STIFFNESS MATRIX.
* PLOAD(,1) = GLOBAL LOAD VECTOR.
* NUMDF = TOTAL NUMBER OF DEGREE.
* IFIX = INDEX OF FIXED NODE. (IFIX.LT.O = FIX,

**OUTPUT.**
* PLOAD(,2) = SOLUTION VECTOR.
**DIMENSION GS(NBNDF), PLOAD(NDF,2), IFIX(I)
**COMMON SOLV/ NITM, NITM**
**COMMON DEBUGF IDBG(20), ISOF(10)**

READS GLOBAL STIFFNESS MATRIX.
IF(NITY.LT.1) THEN
DO 100 N=1, NUMDF
IF(IFIX(N).LT.0) GO TO 100
PLOAD(N,2)=0.0
100 CONTINUE
END IF

**READS GLOBAL STIFFNESS MATRIX**
**REWIND ISF**
**READ(ISF) GS**

**IF(IDBG(I).EQ.0) GO TO 200**
CALL SOLOUT(GS,IFIX,NUMDF,NBNDF,' ST','IFF', LP)
200 CONTINUE

**COMPUTES (PI-KU2) VECTOR.**
* LD=1
* IDX=1
* IDY=1
* IFP=1
CALL MULTI(GS,IFIX,NUMDF,LD,IDX,IDY,IFP, 0,PLOAD)

**COMPUTES (PI-KU2) VECTOR.**
* IF(IFBG(I).EQ.0) GO TO 300
CALL SOLOUT(PLOAD,IFIX,NUMDF,'PI-K', 12U2', LP)
300 CONTINUE

**CHOLESKY DECOMPOSITION**
* CALL DCOMP(GS,IFIX,NUMDF,NBNDF,LP)

**IF(IFBG(I).EQ.0) GO TO 400**
CALL SOLOUT(GS,IFIX,NUMDF,NBNDF,' DEC', LP)
400 CONTINUE
**FORWARD REDUCTION.**

```fortran
CALL FORWD(GS,IFLX,NUMDF,NBAND,PLOAD,NUMDF,1)

IF(IDBG(I).EQ.0) GO TO 500
CALL SOLOUT(PLOAD,IFLX,NUMDF,I,'FORWARD',LP)

500 CONTINUE
```

**BACKWARD SUBSTITUTION.**

```fortran
CALL BAKWD(GS,IFLX,NUMDF,NBAND,PLOAD,NUMDF,1)

IF(IDBG(I).EQ.0) GO TO 600
CALL SOLOUT(PLOAD,IFLX,NUMDF,I,'BACKWARD',LP)

600 CONTINUE
```

**RETURN**

END

**SLOT** — SUBROUTINE SOLOUT —

```fortran
SUBROUTINE SOLOUT(A,IND,ND,NBW,CH1,CH2,ITO)

FUNCTION.
PRINT ROUTINE FOR SOLVER.

DIMENSION A(I),IND(I)

WRITE(ITO,1009) CH1,CH2

1009 FORMAT(120X,'M',32A4,20X,'L')

DO 3000 I=1,ND
   JBEG=I-1,NBW+1
   JEND=JBEG+NBW-1
   WRITE(ITO,3919) IND(I-1),A(J),J=JBEG,JEND

3000 CONTINUE

3919 FORMAT(' ',16,I8,P10,12.4/(7X,IO,E12.4')

RETURN
END
```

**DCMP** — SUBROUTINE DCMP —

```fortran
SUBROUTINE DCMP(A,IND,ND,NBW,ITO)

FUNCTION.(SP. NO. = 14)
SUB FOR CHOLESKI DECOMPOSITION OF SYMMETRIC POSITIVE DEFINITE
MATRIX PACKED ROWWISE IN ONE ARRAY

* INPUTS

  A MATRIX TO BE DECOMPOSED
  IIND INDEX OF A, IF NEGATIVE, CORRESPONDING ROW OF A
  IS PRESCRIBED BOUNDARY CONDITIONS.
  ND LENGTH OF A (COLUMN)
  NBW HALF BAND WIDTH OF A
  ITO OUTPUT FILE

  DIMENSION A(I),IND(I)

  DO 7000 I=2,ND
     IF(IND(I).LT.0) GO TO 7000
     JEND=I+NBW-1
     IF(JEND.GT.ND) JEND=ND
     DO 5000 J=I,JEND
        IF(IND(J).LT.0) GO TO 5000
        KBEG=J-NBW+I
        IF(KBEG.LT.I) KBEG=I
        KEND=I-I
        IF(KBEG.GT.KEND) GO TO 5000
        LIJ=(I-I)*NBW+I-I+J
        DO 3000 K=KBEG,KEND
           IF(IND(K).LT.0) GO TO 3000
           LC=(K-I)*NBW+I-K
           L=LC+1
           LK=LC+J
           LK=LC+K
           IF(ALKK) 2000,8000,2000

2000 CONTINUE

8000 CONTINUE

```

---

**MEL - Tsukuba - July 1993**

46
A(LU)=A(LU)-A(LKI)' A(LKJ)/ A(LKK)

3000 CONTINUE
5000 CONTINUE
7000 CONTINUE

*8000 WRITE(IT0;8009) K
*8009 FORMAT(***ERROR*** DIAGONAL OF MATRIX BECOME ZERO AT=' ', INTEGER)

STOP

9000 RETURN

END

*FRWD — SUBROUTINE FORWD —

SUBROUTINE FORWD(A,IND,ND,NBW,Y,NY,LD)

*FUNCTION.(SP. NO. = 12)*

*SUB FOR CHOLESKI FORWARD SUBSTITUTION

* INPUT
  * A  UPPER DECOMPOSED MATRIX
  * IND INDEX OF A (IF NEG, SUCH ROW IS PRESCRIBED B.C.
  * ND ORDER OF A
  * NBW HALF BANDWIDTH OF A
  * Y  RIGHT HAND VECTOR
  * NY FIRST DIM. OF Y
  * LD LOADCN

* OUTPUT

  * Y  SOLUTION

DIMENSION A(I),IND(I),Y(NY,I)

DO 4000 I=1,ND
  IF(IND(I).LT.0) GO TO 4000
  JBEG=I+1-NBW
  JEND=I-I
  IF(JBEG.ILT.I) JBEG=1
  IF(JBEG.GT.JEND) GO TO 3500
  DO 3000 J=JBEG,JEND
    IF(IND(J).LT.0) GO TO 3000
    L=(1-I)*NBW+I-J+1
  2900 Y(I,L)=Y(I,L)-A(L)'Y(J,L)
  3000 CONTINUE

3500 L=(I-I)*NBW+J+1

DO 3600 IL=I-LD
  3600 Y(I,IL)=Y(I,IL)/A(L)

4000 CONTINUE
RETURN

END

*BKWD — SUBROUTINE BAKWD —

SUBROUTINE BAKWD(A,IND,ND,NBW,Y,NY,LD)

*FUNCTION.(SP. NO. = 13)*

* SUB FOR CHOLESKI BACKWARD SUBSTITUTION

* INPUT

  * A  UPPER DECOMPOSED MATRIX
  * IND INDEX OF A
  * ND ORDER OF A
  * NBW HALF BANDWIDTH OF A
  * Y  RIGHT HAND VECTOR
  * NY FIRST DIM. OF Y
  * LD LOADCN
  * WS WORKING VECTOR (LENGTH=LD)

* OUTPUT

  * Y  SOLUTION

DIMENSION A(I),IND(I),Y(NY,I),WS(100)
DO 4000 IC=2,ND
I-ND-IC+1
IF(IND(I).LT.0) GO TO 4000

DO 1800 IL=1,LD
1800 WS(IL)=0.0
BEGIN=I+1
JEND=I+NBW-1
IF(BEGIN.GT.ND) BEGIN=ND
IF(END .GT.ND) END = ND
DO 3000 J=BEGIN,JEND
IF(IND(J).LT.0) GO TO 3000
L=(I-J)*NBW+I-J
DO 3800 D..=I,LD
3800 Y(L,L)=Y(L,L)-WS(L,L)/A(L)
3000 CONTINUE
DO 3800
3800 Y(L,D..)=Y(L,D..)-WS(L,L)/A(L)
4000 CONTINUE
RETURN
END

MULT — SUBROUTINE MULTI —

SUBROUTINE MULTI(A,IND,X,NY,LD,IDX,IDY,IOP, DUMY,Y)
* FUNCTION (SP. NO. -11)
* SUB FOR MULTIPLY MATRIX Y = Y - A* X (IOP=0)
* Y = A* X (IOP=1)
* Y = Y + A* X (IOP=2)
* INPUT
* A MATRIX TO BE MULTIPLIED
* IND INDEX OF A
* NY ORDER OF A
* NBAND HALF BANDWIDTH OF A
* X VECTOR TO BE MULTIPLIED
* LD LOADCN
* IDX,Y (0. ONLY NEG. IND ROW OF XY) ARE ACTIVE
* >0. ALL XY ARE ACTIVE
* >0. ONLY POSITIVE IND OF XY ARE ACTIVE
* NY FIRST DIMENSION OF X Y
* IOP (0. SUBTRACTION =0, MULT ONLY >0, ADDITION
* OUTPUT
* Y OUTPUT VECTOR (SHOULD BE INITIALIZED)
* DIMENSION A(I),IND(I),X(NY,1),Y(NY,1)
* COMMON /SOLVA/ NBAND, NBNDF
* DUMY=0.0
DO 8000 I=1,NY
IF(IOP.NE.0) GO TO 1400
DO 1100 K=I,LD
1100 Y(I,K)=0.0
1400 IF(IDY*IND(I).LT.0) GO TO 8000
BEGIN=I-NBAND+I
IF(BEGIN.GT.ND) BEGIN=ND
IF(IDX*IND(J).LT.0) GO TO 3000
L=(J-I)*NBAND+I-J+1
IF(IOP.LT.0) GO TO 2000
DO 1800 K=1,LD
1800 Y(I,K)*Y(K,I)=A(K)*Y(I,K)
GO TO 3000
2000 DO 2200 K=1,LD
2200 Y(I,K)=Y(I,K)-A(K)*Y(J,K)
3000 CONTINUE
IF(IDY.EQ.NY) GO TO 8000
JEND=I+NBAND+1
IF(END.GT.NY) JEND=NY
JB=I+1
IF(JB.GT.1) GO TO 8000
DO 6000 J=JB,1END
IF(IDX'IND(J).L T.O) GO TO 6000
L=(I-I)'NBAND+I-I+J
IF(IOP.LT.O) GO TO 5000
DO 4800 K=I,LD
4800 Y(I,K)=Y(J,IC)+A(L)'X(J,K)
GO TO 6000
5000 DO 5200 K=I,LD
5200 Y(I,K)=Y(I,K-A(L)'X(J,K)
6000 CONTINUE
8000 CONTINUE
RETURN

*ROUTINE LOUT(TIME,NSTA,X,Y,Z, VX, VY, VZ, ASIG, EPSEL)
FUNCTION(SP. NO. =36)
WRITES THE RESULTS.
CALLED BY:
OUTPUT
CALLS:
PAGE
INPUT & OUTPUT
* NITE = INNER ITERATION NUMBER.
* NSTA = STAGE NUMBER.
* X = X AXIAL COORDINATES OF NODAL POINTS.
* Y = Y AXIAL COORDINATES OF NODAL POINTS.
* Z = Z AXIAL COORDINATES OF NODAL POINTS.
* VX = NODAL POINT VELOCITY IN X-DIRECTION
* VY = NODAL POINT VELOCITY IN Y-DIRECTION
* VZ = NODAL POINT VELOCITY IN Z-DIRECTION
* ASIG = AVERAGE STRESS OF THE ELEMENT.
* EPSL = EQUIVALENT STRAIN OF ELEMENTS.

*** COMMON /ASVA/ NUMEL, NUMNP, NUMDF, NUMLD
*** COMMON /FILU N/, LP, IDF, ISF
*** COMMON /ABEV/ HEAD(20), IFLAG(20), CLIM, DFACT,ISTIF
*** , IENITY, MODEL
*** COMMON /RESULTI
*** COMMON /FILE/ IN , LP , IFK , ISF
*** COMMON /LINESI
*** COMMON /INPVAf HEAD(20) , IFLAG(20) , CUM, DFACT,ISTIF
*** , INITV , MODEL
*** COMMON X(I), Y(I),Z(I),VX(I), VY(I), VZ(I),ASIG(I),EPSEL(N,1)
*** INITIAL DATA SET.
*** TITLE AND STAGE NUMBER.
WRITE(LP,6000) HEAD
WRITE(LP,6010) NSTA
CONVERGENCE INFORMATION.
WRITE(LP,6020) NITE , RATIO, CUM
COORDINATES AND VELOCITY.
WRITE(LP,6030) DO 100 N=I,NUMNP
100 CONTINUE
AVERAGED ELEMENT STRESS AND STRAIN.
WRITE(LP,6050) DO 200 N=I,NUMEL
200 CONTINUE
RETURN
*** WRITE FORMAT.
6000 FORMAT(IHI,I,IX,20A4)
6010 FORMAT(IHI,11X,20A4)
6020 FORMAT(IHI,10X,20A4)
6030 FORMAT(IHI,20X,20A4)
6040 FORMAT(IHI,10X,20A4)
6050 FORMAT(IHI,10X,20A4)
6060 FORMAT(IHI,10X,20A4)
END

SUBROUTINE ROUT(NUMSTA,NSTA,X,Y,Z, VX, VY, VZ, ASIG,EPSEL,NUMEL)
FUNCTION(SP. NO. =36)
WRITES THE RESULTS.
Improvements on the 3-D RIPLE Program

* CALLED BY:  
* OUTPUT  
* CALLS:  
* none  

* INPUT & OUTPUT:  
* NITE = INNER ITERATION NUMBER.  
* NSTA = STAGE NUMBER.  
* X = X AXIAL COORDINATES OF NODAL POINTS.  
* Y = Y AXIAL COORDINATES OF NODAL POINTS.  
* Z = Z AXIAL COORDINATES OF NODAL POINTS.  
* VX = NODAL POINT VELOCITY IN X-DIRECTION.  
* VY = NODAL POINT VELOCITY IN Y-DIRECTION.  
* VZ = NODAL POINT VELOCITY IN Z-DIRECTION.  
* ASIG = AVERAGE STRESS OF THE ELEMENT.  
* EPS7, 8* = EQUIVALENT STRAIN RATE OF ELEMENTS.  
* EPS8, 9* = EQUIVALENT STRAIN OF ELEMENTS.  

* COMMON BASVA, NUMEL, NUMNP, NUMDF, NUMLD  
* COMMON (SPVA, HEAD20, IFLAG, CLM, PFACT, STIF  
* INITV, MODEL  
* COMMON (XYCOR, YCOR, ZCOR)  
* COMMON (I, J, K, T, TIME)  
* COMMON (XCOR(I, 000), YCOR(I, 000), ZCOR(I, 000), ASIG(I), EPSEL(I))  

* IF(NSTA.LT.9) THEN  
  TIME=TIME+TTIME  
  ELSE IF(MOD(NSTA, 8).EQ.0) THEN  
    TIME=TIME+TTIME  
  ELSE  
    TIME=TIME+TTIME  
  END IF  

6001 FORMAT(F10.5)  
XDIS=0.0  
YDIS=0.0  
ZDIS=0.0  
IFLAGS=IFLAG(S)  
IFLAG9=IFLAG(9)  
IF(IFLAG9.EQ.0) THEN  
  NCASE=IFLAGS  
ELSE  
  NCASE=IFLAGS/IFLAG9  
END IF  
DO 100 N=1,NUMNP  
  XDIS=X(N)-XCOR(N)  
  YDIS=Y(N)-YCOR(N)  
  ZDIS=Z(N)-ZCOR(N)  
  WRITE(32,1000) N, XDIS, YDIS, ZDIS  
XDIS=0.0  
YDIS=0.0  
ZDIS=0.0  
100 CONTINUE  

** AVERAGED ELEMENT STRESS AND STRAIN.  
** NTYPE=8  
** NOIT=4  
** NOPT=1  
** IF(NSTA.EQ.1) WRITE(36,1200) NUMEL, NCASE  
** DO 200 N=1, NUMEL  
** WRITE(36,2000) N , NTYPE, NOIT, NOPT  
** WRITE(36,3000) ASIG(N), EPS7(N), EPS8(N), EPS9(N)  
200 CONTINUE  
1100 FORMAT(2I10)  
1000 FORMAT(2X,3(E10.3,1X))  
1200 FORMAT(2X)  
2000 FORMAT(4I5)  
3000 FORMAT(4E10.3)  
RETURN  
END

OUTP -- SUBROUTINE OUTPUT --  
* SUBROUTINE OUTPUT(A,NITE,NSTA)  
* FUNCTION(KP, NO. 35)  
* WRITE THE RESULTS  
* CALLED BY  
* MAIN  
* CALLS.
Improvements on the 3-D RIPLE Program

* LOUT

* INPUT.
* A = BLANK COMMON AREA.
* NITE = INNER ITERATION NUMBER.
* NSTA = STAGE NUMBER.
* IFLAG(9) = FLAG FOR PRINT INTERVAL.
* OUTPUT.
* NONE.

* COMMON ARES1M, M1MAX, M2MNS, M3MNN, M4MNE, M5MNS,
* M6MNE, M7MNS, M8MNE, M9MNS, M10MNE, M11MNS,
* M12MNE, M13MNS, M14MNE, M15MNS, M16MNE, M17MNS,
* M18MNE, M19MNS, M20MNE, M21MNS, M22MNE, M23MNS,
* M24MNE, M25MNS, M26MNE, M27MNS, M28MNE, M29MNS,
* M30MNE.
COMMON ARES2M, M31MNS, M32MNE, M33MNS, M34MNE, M35MNS,
* M36MNE, M37MNS, M38MNE, M39MNS, M40MNE, M41MNS,
* M42MNE, M43MNS, M44MNE, M45MNS, M46MNE, M47MNS,
* M48MNE, M49MNS, M50MNE, M51MNS, M52MNE, M53MNS,
* M54MNE, M55MNS, M56MNE, M57MNS, M58MNE, M59MNS,
* M60MNE.

COMMON ARESV1, ARESV2, ARESV3, ARESV4, ARESV5, ARESV6,
* ARESV7, ARESV8, ARESV9, ARESV10, ARESV11, ARESV12,
* ARESV13, ARESV14, ARESV15, ARESV16, ARESV17, ARESV18,
* ARESV19, ARESV20, ARESV21, ARESV22, ARESV23, ARESV24,
* ARESV25, ARESV26, ARESV27, ARESV28, ARESV29, ARESV30,
* ARESV31, ARESV32, ARESV33, ARESV34, ARESV35, ARESV36,
* ARESV37, ARESV38, ARESV39, ARESV40, ARESV41, ARESV42,
* ARESV43, ARESV44, ARESV45, ARESV46, ARESV47, ARESV48,
* ARESV49, ARESV50, ARESV51, ARESV52, ARESV53, ARESV54,
* ARESV55, ARESV56, ARESV57, ARESV58, ARESV59, ARESV60,
* ARESV61, ARESV62, ARESV63, ARESV64, ARESV65, ARESV66,
* ARESV67, ARESV68, ARESV69, ARESV70, ARESV71, ARESV72,
* ARESV73, ARESV74, ARESV75, ARESV76, ARESV77, ARESV78,
* ARESV79, ARESV80, ARESV81, ARESV82, ARESV83, ARESV84,
* ARESV85, ARESV86, ARESV87, ARESV88, ARESV89, ARESV90,
* ARESV91, ARESV92, ARESV93, ARESV94, ARESV95, ARESV96,
* ARESV97, ARESV98, ARESV99, ARESV100, ARESV101, ARESV102,
* ARESV103, ARESV104, ARESV105, ARESV106, ARESV107, ARESV108,
* ARESV109, ARESV110, ARESV111, ARESV112, ARESV113, ARESV114,
* ARESV115, ARESV116, ARESV117, ARESV118, ARESV119, ARESV120,
* ARESV121, ARESV122, ARESV123, ARESV124, ARESV125, ARESV126,
* ARESV127, ARESV128, ARESV129, ARESV130, ARESV131, ARESV132,
* ARESV133, ARESV134, ARESV135, ARESV136, ARESV137, ARESV138,
* ARESV139, ARESV140, ARESV141, ARESV142, ARESV143, ARESV144,
* ARESV145, ARESV146, ARESV147, ARESV148, ARESV149, ARESV150,
* ARESV151, ARESV152, ARESV153, ARESV154, ARESV155, ARESV156,
* ARESV157, ARESV158, ARESV159, ARESV160, ARESV161, ARESV162,
* ARESV163, ARESV164, ARESV165, ARESV166, ARESV167, ARESV168,
* ARESV169, ARESV170, ARESV171, ARESV172, ARESV173, ARESV174,
* ARESV175, ARESV176, ARESV177, ARESV178, ARESV179, ARESV180,
* ARESV181, ARESV182, ARESV183, ARESV184, ARESV185, ARESV186,
* ARESV187, ARESV188, ARESV189, ARESV190, ARESV191, ARESV192,
* ARESV193, ARESV194, ARESV195, ARESV196, ARESV197, ARESV198,
* ARESV199, ARESV200, ARESV201, ARESV202, ARESV203, ARESV204,
* ARESV205, ARESV206, ARESV207, ARESV208, ARESV209, ARESV210,
* ARESV211, ARESV212, ARESV213, ARESV214, ARESV215, ARESV216,