AFRL-MN-EG-TR-2000-7026

2D Axisymmetric Lagrangian Solver for Taylor Impact with Johnson-Cook Constitutive Model , Vol.1

William H. Cook

Air Force Research Laboratory
Munitions Directorate (AFRL/MNAC)
101 W. Eglin Blvd, Suite 334
Eglin AFB, FL 32542-6810

APRIL 2000

INTERIM REPORT FOR PERIOD
DECEMBER 1999 - MARCH 2000

Approved for public release; distribution unlimited.

AIR FORCE RESEARCH LABORATORY, MUNITIONS DIRECTORATE

Air Force Materiel Command ■ United States Air Force ■ Eglin Air Force Base

Revised

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	April 2000	Interim Dec 1999-March 2000

4. TITLE AND SUBTITLE
2D Axisymmetric Lagrangian Solver for Taylor Impact with Johnson-Cook Constitutive Model Vol 1

5. FUNDING NUMBERS
In-house, Project 25020729, PE 62602F

6. AUTHOR(S)
William H. Cook

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Air Force Research Laboratory
Munitions Directorate (AFRL/MNAC)
101 W. Eglin Blvd, Suite 348
Eglin AFB, FL 32542-6810

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Air Force Research Laboratory
Munitions Directorate (AFRL/MNAC)
101 W. Eglin Blvd, Suite 348
Eglin AFB, FL 32542-6810

10. SPONSORING/MONITORING AGENCY REPORT NUMBER
AFRL-MN-EG-TR-2000-7026

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

A Taylor impact solver has been developed, documented, and provided for installation on personal computers. This code operates under Microsoft Windows environments on personal computers to allow the interpretation of cylinder impact data in terms of high strain rate material constants. The code allows the determination of the Johnson-Cook constitutive model constants from cylinder impact data. The code also demonstrates the importance of the dynamic behavior of materials at high rates of loading.

See Vol 2 (Software to accompany Vol 1)

[redacted]

14. SUBJECT TERM
Material Modeling Constitutive Model High Strain Rate Taylor Impact Hydrocode

15. NUMBER OF PAGES
29

16. PRICE CODE

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

NOTICE

When Government drawings, specifications, or other data are used for any purpose other than in connection with a definitely Government-related procurement, the United States Government incurs no responsibility or any obligation whatsoever. The fact that the Government may have formulated or in any way supplied the said drawings, specifications, or other data, is not to be regarded by implication, or otherwise in any manner construed, as licensing the holder, or any other person or corporation; or as conveying any rights or permission to manufacture, use, or sell any patented invention that may in any way be related thereto.

This technical report is releasable to the National Technical Information Services (NTIS). At NTIS it will be available to the general public, including foreign nations.

This technical report has been reviewed and is approved for publication.

Frederick A. Davis
Technical Director
Assessment and Demonstrations Division

William H. Cook, Ph.D.
Senior Engineer

If your address has changed, if you wish to be removed from our mailing list, or if your organization no longer employs the addressee, please notify AFRL/MNAC, Eglin AFB FL 32542-6810, to help us maintain a current mailing list.

Do not return copies of this report unless contractual obligations or notice on a specific document requires that it be returned.

Table of Contents

Table of Figures

Table of Tables

Introduction

The accurate prediction of the performance of impacting and explosively formed metals requires high strain rate descriptions of material behavior. One such description is the Johnson-Cook model (Reference 1), which was originally developed for the accurate prediction of explosively formed metal penetrators. The Johnson-Cook model was specifically developed from a set of well-defined laboratory data, including low and high strain rate tests as well as elevated temperature tests. The original description of the model explains the method for determining the constants in the model from laboratory experiments.

An alternative way of relatively easily obtaining constants for the model is by conducting a cylinder impact experiment (Taylor test) and performing iterative hydrocode predictions of the experiment while changing material constants until a reasonable match of the deformed profile occurs compared to the computation. This approach requires access to an axisymmetric transient wave propagation code (hydrocode) which is capable of accurately modeling the behavior of high velocity impact.

This report documents a Lagrangian hydrocode developed specifically of this purpose. The code is designed to be fully interactive, allowing the user to vary material model inputs very easily. The code is a Lagrangian code with the reference frame fixed in the material, which is particularly well suited to careful study of the material behavior. This code will be referred to here as simply the Taylor impact solver.

Objectives and Approach

The Taylor impact solver was written in Microsoft Visual C++ to maximize ease of use through a user interface that is consistent with Microsoft Windows standards. The code operates as a simple form with all inputs, outputs, and graphical displays visible at all times.

The primary use of the code was intended to be rough approximations of material model constants from cylinder impact test data, but the Taylor impact solver is also a powerful tool to demonstrate the importance of material properties to high strain rate events.

Governing Equations and the Solver

This code solves the equations of conservation of mass, momentum, and energy subject to material models in a finite element, explicit Lagrangian framework. Details of the equations and the finite element implementation for axisymmetric triangular finite elements are provided in Reference 2. Neither the governing equations nor the numerical approximations of these equations will be presented here because to do so would needlessly duplicate the more complete presentation provided in Reference 2. Instead, the C++ listing of the solver implementing the numerical solution of the conservation equations is included as Appendix III. The listing is intended to be self documenting for anyone with a basic understanding of C++ programming, and some sense of the physics being computed. Unfortunately, a complete description of all of the variables was also considered to be beyond the scope of this basic documentation, so the reader is expected to interpret the variables in the coding as best possible. Hopefully the listing will be more helpful than confusing. The listing also has been annotated with extra comments to assist the reader in understanding the numerical solution steps and the equations as implemented in the solver. The relative simplicity of the Johnson-Cook model is also made clear by this presentation. It is highlighted in particular, and since equation of state constants are made available for the user to change, that section is also marked.

Johnson-Cook Constitutive Model

The Johnson-Cook constitutive was developed for metals undergoing rapid rates of deformation during processes such as high speed impact or explosive loading. The model describes stress as a function of strain, strain rate, and temperature as described below:

$$\sigma = A + B\varepsilon^{n}[1 + C\ln\dot{\varepsilon}^{*}][1 - T^{*m}]$$

Where

σ = von Mises flow stress (psi)
ε = equivalent plastic strain
ε^{*} = plastic strain rate
T^{*M} = homologous temperature (degrees F) defined as:

$$T^{*m} = (T - T_{ROOM})/(T_{MELT} - T_{ROOM})$$

with 5 Johnson-Cook model constants for each material:

A = Yield Strength (psi)
B = Work Hardening Extent (psi)
C = Strain Rate Effect
m = Thermal Softening Shape
n = Work Hardening Shape

Several symbols are used for the same constants in the display, the code, and this report. The interchangeable symbols used are:

Display	Code	Report
c1	m_c1	A
c2	m_c2	B
c3	m_c3	C
am	m_am	m
an	m_an	n

The effect of each of these constants on material behavior is demonstrated by three of the four plots in the center of the display (refer to Figure 1 in Appendix I.) The upper left plot shows stress vs. strain and changes with changes in A, B, and n. The upper right plot of a dimensionless multiplier on stress vs. log strain rate varies as C varies. The lower left plot of a dimensionless stress multiplier vs. homologous temperature varies with m. The plot of pressure vs. volumetric strain on the lower right illustrates the behavior of the material represented with the equation of state, affected by grun, c, s, and d. Details of the Mie-Gruneisen equation of state are given in reference 2 and elsewhere.

Installing the Code

The code may be installed and used on IBM compatible personal computer (PC) systems running Windows 95/98 or Windows NT 4.0 or newer operating systems. Install the code by copying the files from the enclosed 3½ inch floppy disk to the directory you choose to run from. This may simply be the desktop if you prefer. Two files are required for operation—the executable and one data file. These files are called HydroForm.exe and Taylor.dat. The data file (.dat) will be used to read from and write to, storing particular user runs. This feature allows a user to build a library of inputs of interest by simply renaming files. The data file is an ASCII file that can be edited by many common editors. For example, Wordpad, Word, or Write may be used to look at and change input data. Of course, data sets may also be created and modified by changing the default values that are set on initiation of the code, and saving the configuration to the data file.

The default data set supplied on the floppy disc contains the same data set that would be generated by taking the default values on program startup and writing the data file. Four additional files are included on the installation floppy for setups with Oxygen Free High Conductivity (OFHC) copper, Armco iron, 2024-T4 aluminum, and 4340 steel. These files are shown in Appendix II. The names of the files indicate the materials they describe. This data set is consistent with the data published in Reference 1.

Inputs and Operation

The inputs to the code are collected on the left of the dialog box, consisting of two grouping of inputs—general inputs for run control, and inputs for the material models.

If the button on the lower right side is depressed to read a data file, the file named material.dat in the default directory that the code is run from will be read and used to reset the values of the variables displayed as inputs on the left side of the dialog box, as well as the profile data on the lower right. The format of the data file is given in Tables I and II in Appendix II.

Changes to the material constants in the inputs are reflected immediately as changes to the four material plots to the right of these constants. For example, changing c1, which is effectively the yield strength of the material, will immediately result in a change in the top left plot of stress vs. strain.

Changes to the inputs for general run control, such as number of nodes along the length or across the radius will immediately effect the meshing of the cylinder and be visible in the contour plot on the upper right of the dialog display. Variables plotted may be changed prior to, during, or after a run.

Saving the displayed values to a data file will overwrite the material.dat file in the default directory where the Taylor impact solver was executed. It is the responsibility of the user to rename critical data files and keep track of them. Other than the example material data files supplied on the source floppy disk, no material data or problem librarian is supplied with this code. Limiting the supplied data was intentional to make wider distribution of the software tool possible than might have been permitted with the rather extensive material data set that might be available for the Johnson-Cook model.

Example Applications

The examples used throughout this section presume a 0.3000 diameter by 1.00 inch long cylinder impacting at 6000 inches per second. These conditions are similar to the conditions for Taylor cylinder impacts used to validate the original Johnson-Cook constants described in Reference 1. The examples are generated using the data sets supplied on the distribution diskette, with modifications as noted.

Example 1: OFHC Copper.

Figures 1 and 2 show the initial setup and final results for the default setting of the code, utilizing constants for Oxygen Free High Conductivity (OFHC) copper.

Example 2: Armco Iron.

Figures 3 and 4 show the initial setup and the final results for the Armco iron data file.

Example 3: 4340 Steel.

Figures 5 and 6 show the initial setup and the final results for the 4340 steel data file.

Example 4: Aluminum.

Figures 7 and 8 show the initial setup and the final results for the 2024 aluminum data file.

Example 5: Deformed OFHC Profile

Figures 9 and 10 repeat Figures 1 and 2, but with a profile of the deformed cylinder provided based upon the computation. This profile is used in the remaining examples to demonstrate the effects of changes in the parameters of the Johnson-Cook model.

Example 6: Effect of OFHC Johnson-Cook Parameter c3

Example 6 in Figure 11 demonstrates the effect of changing the term c3 in the Johnson-Cook model to 0.0, eliminating the strain rate effect. The result is clear in the reduced cylinder length.

Example 7: Effect of OFHC Johnson-Cook Parameter c1

Example 7 in Figure 12 demonstrates the effect of doubling the Johnson-Cook parameter c1, which is effectively the yield strength. The result is clear in both the reduced primary diameter at the rigid boundary interface and the increased final length.

Example 8: Effect of OFHC Johnson-Cook Parameter c2

Example 8 in Figure 13 demonstrates the effect of reducing the amount of work hardening by changing the Johnson-Cook parameter c2 from 42,300 psi to 13,000 psi. The result is a larger primary diameter at the rigid boundary interface, a larger secondary diameter at the "second cone", and a reduced final length. This is the first reference to the secondary cone, but as an aside, it is interesting to observe the dynamic deformation/hardening process that caused the first cone to form at the rigid boundary interface, and then the secondary cone forms as an impact against the first cone.

Example 9: Effect of OFHC Johnson-Cook Parameter am

Example 9 in Figure 14 demonstrates the effect of changing the thermal softening parameter am from 1.09 to 0.50, increasing the softening effect for temperatures between melt and room temperature. The result is demonstrated in the reduced overall final length.

Example 10: Effect of OFHC Johnson-Cook Parameter an

Example 10 in Figure 15 demonstrates the effect of increasing the Johnson-Cook parameter am from 0.31 to 0.50. The effect is to decrease work hardening at lower strains, by changing the shape of the work hardening. The resulting larger secondary diameter and shorter final length is observed in Figure 15.

Summary and Conclusions

A Taylor impact solver has been developed, documented, and provided for installation on personal computers. This code operates under Microsoft Windows environments on personal computers to allow the interpretation of cylinder impact data in terms of high strain rate material constants. The code allows the determination of the Johnson-Cook constitutive model constants from cylinder impact data. The code also demonstrates the importance of the dynamic behavior of materials at high rates of loading.

References

1. Johnson, G. R and Cook, W. H., "A Constitutive Model and Data for Metals Subjected to Large Strains, High Strain Rates, and High Temperatures," Proceedings of Seventh International Symposium on Ballistics, the Hague, The Netherlands, April 1983.
2. Johnson, G. R. Stryk, R. A., Holmquist, T. J., and Beissel, S. R., "Numerical Algorithms in a Lagrangian Hydrocode," Wright Laboratory, Armament Directorate, WL-TR-1997-7039, June 1997.

Appendix I. Figures

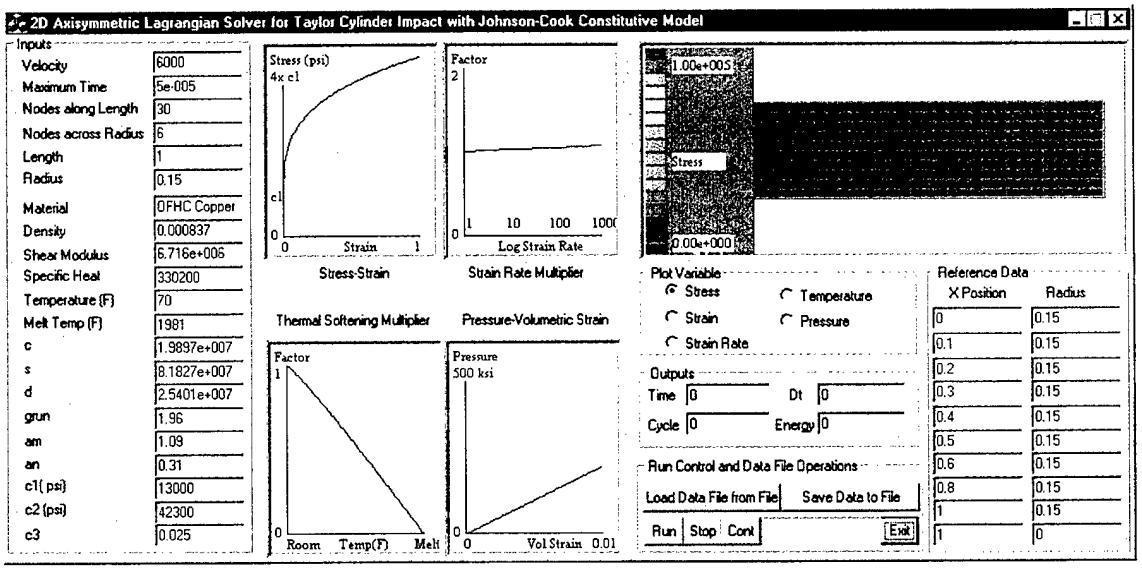

Figure 1. View of screen on startup.

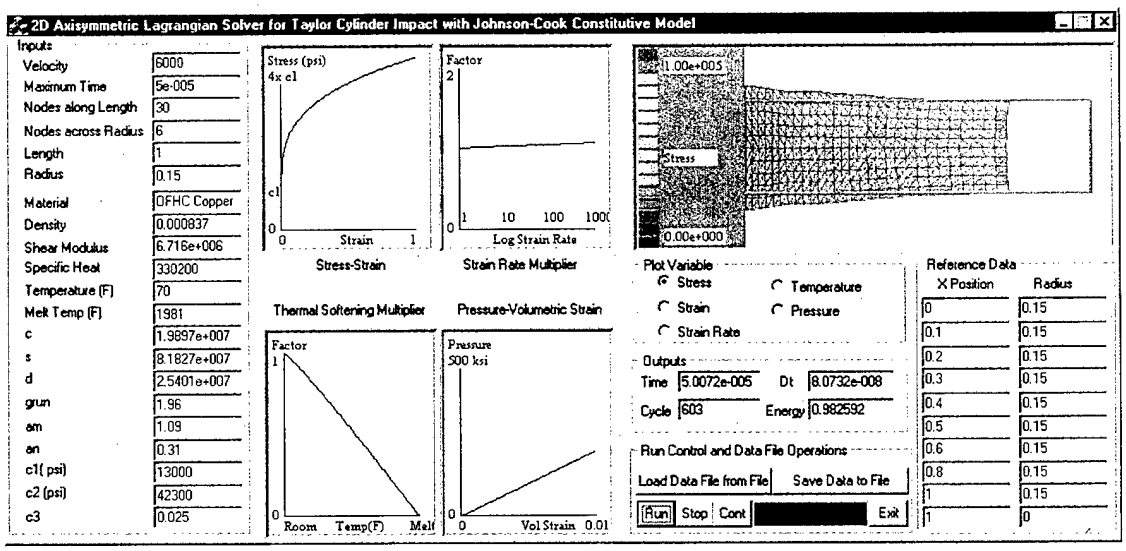

Figure 2. View of screen after execution of defaults.

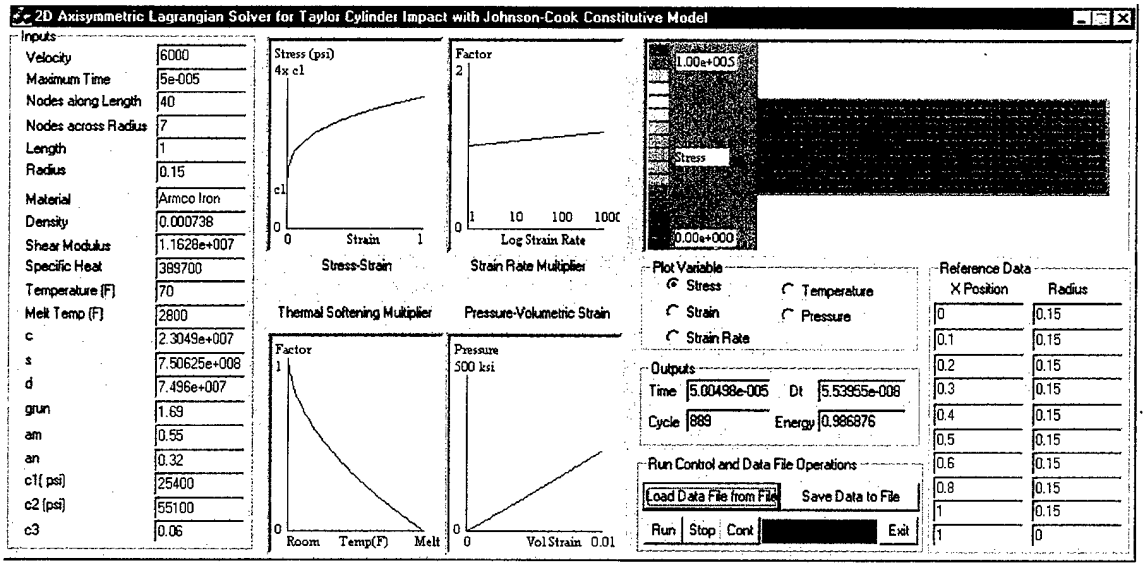

Figure 3. Armco iron before run.

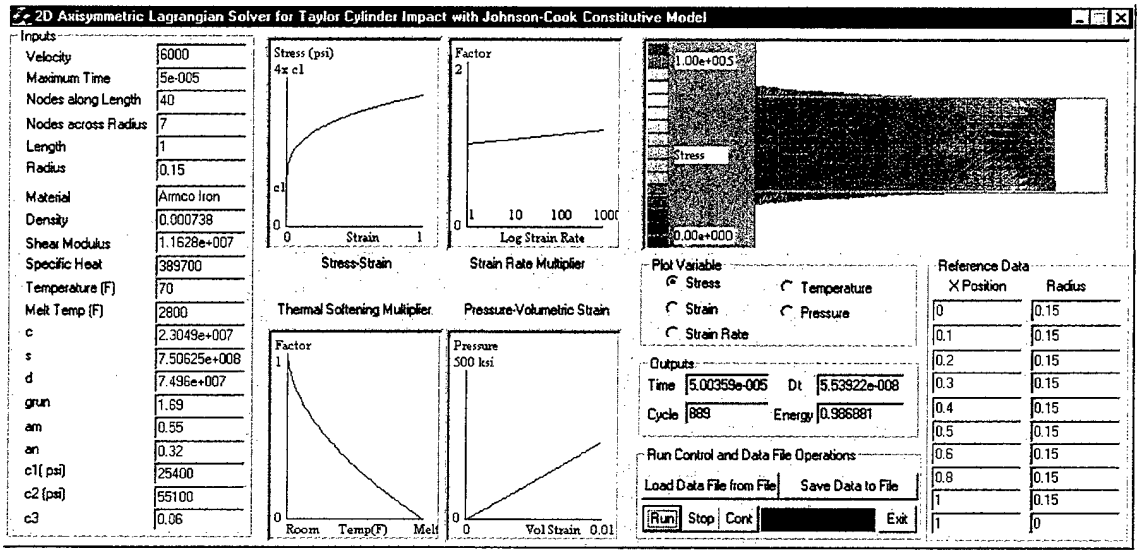

Figure 4. Armco iron after execution.

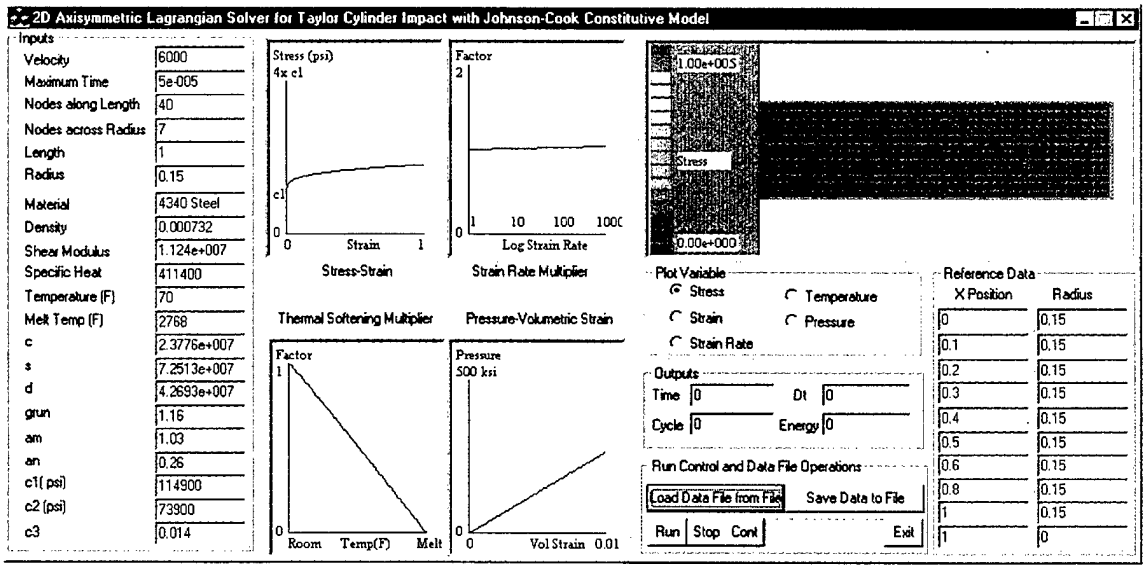

Figure 5. 4340 steel before run.

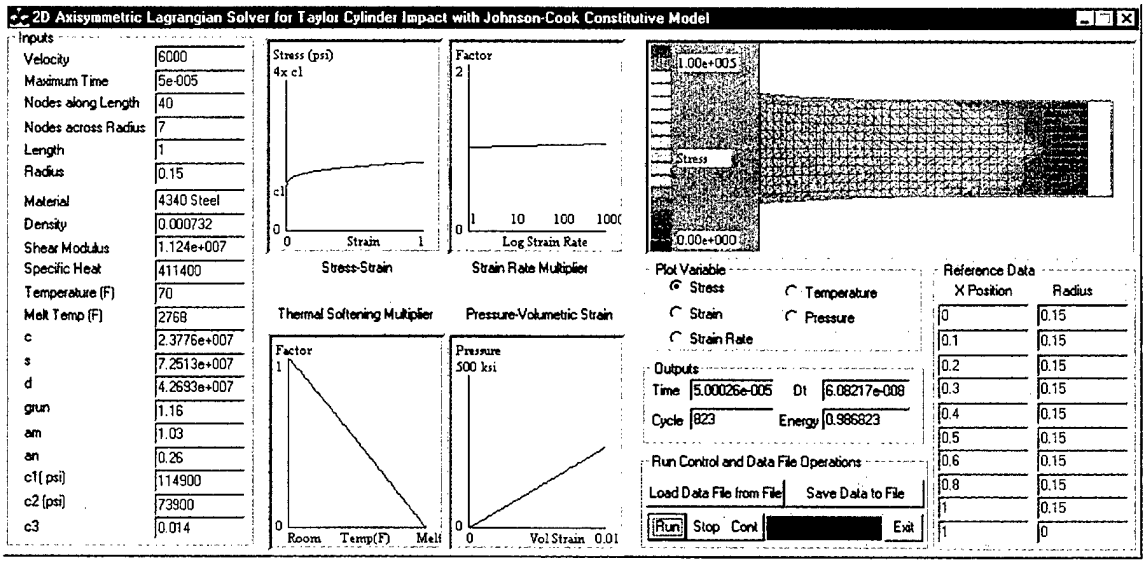

Figure 6. 4340 steel after execution.

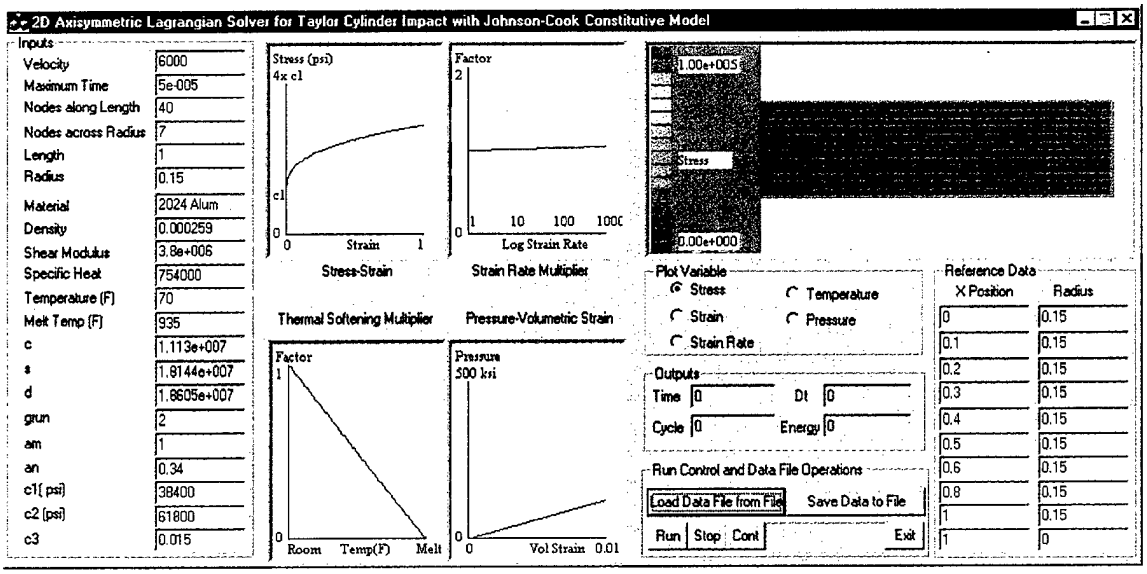

Figure 7. 2024-T4 aluminum before run.

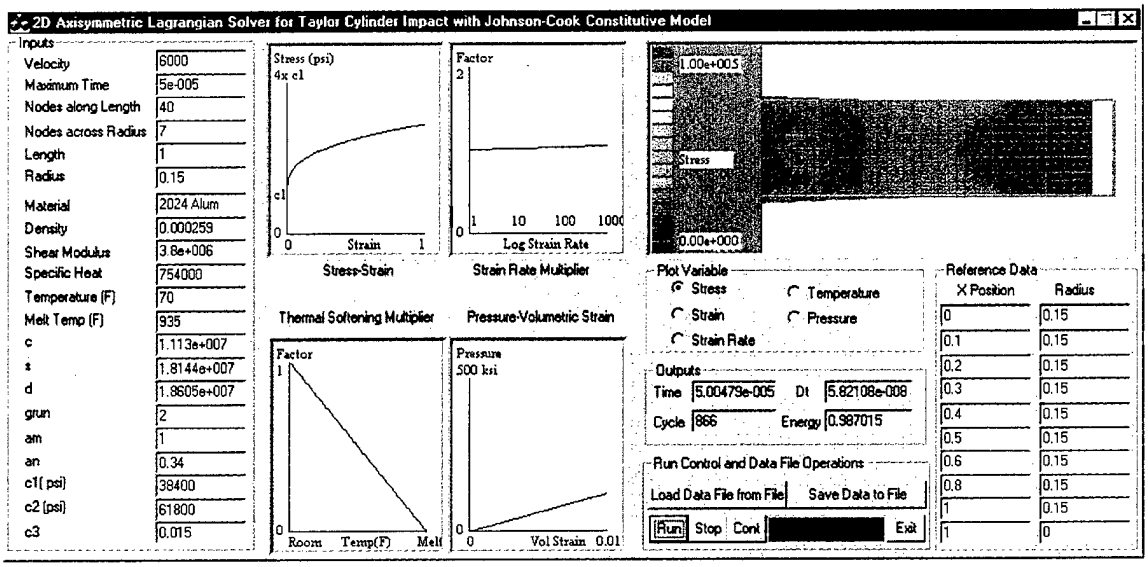

Figure 8. 2024-T4 aluminum after execution.

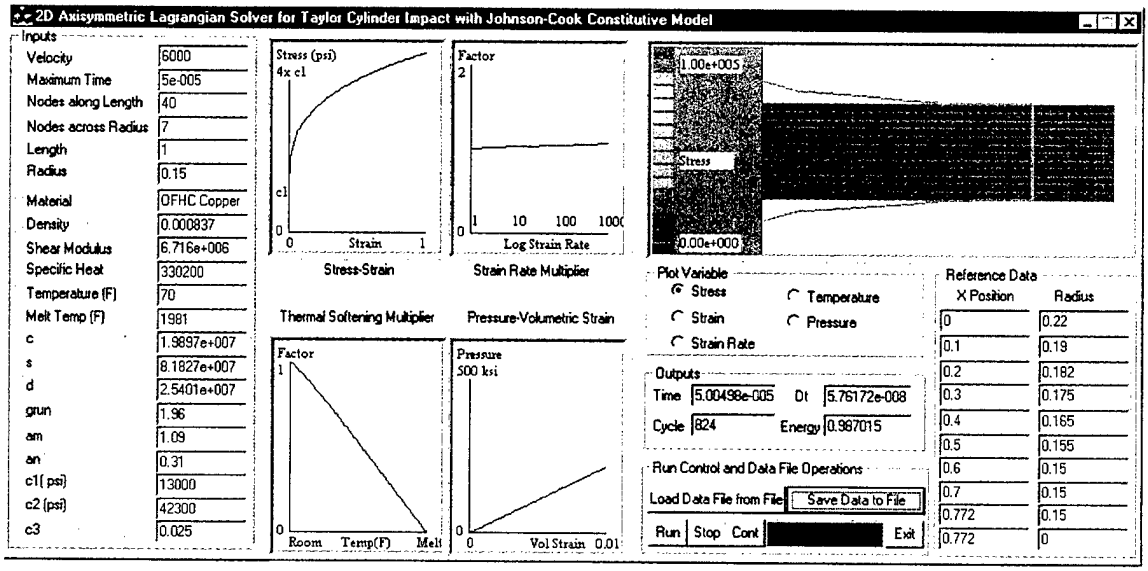

Figure 9. Outline of deformed copper cylinder shown—otherwise same as defaults

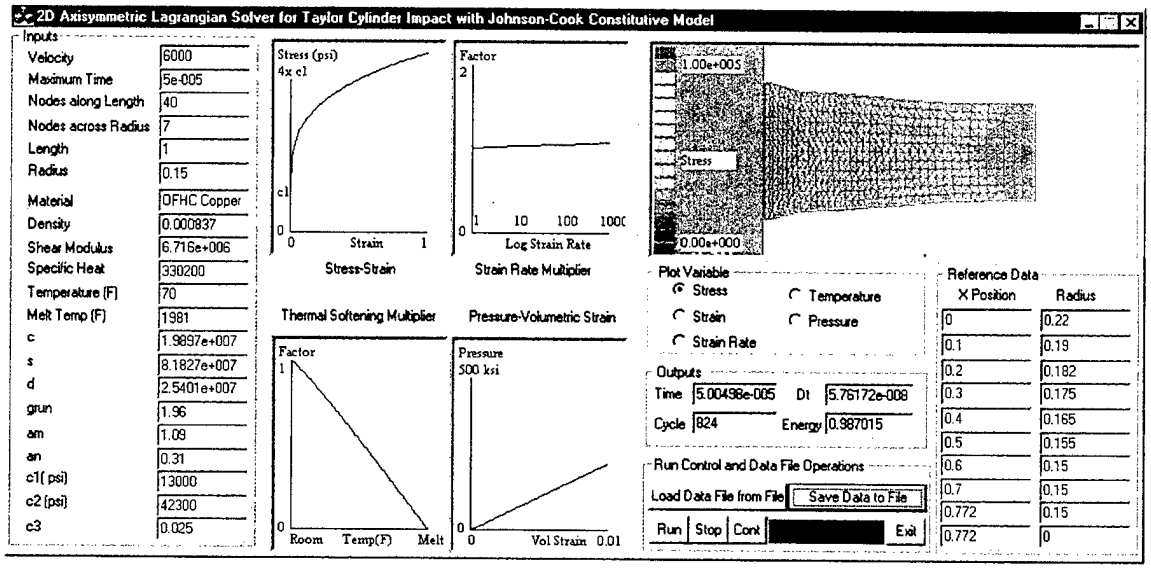

Figure 10. Default values to establish cylinder outline for subsequent comparisons

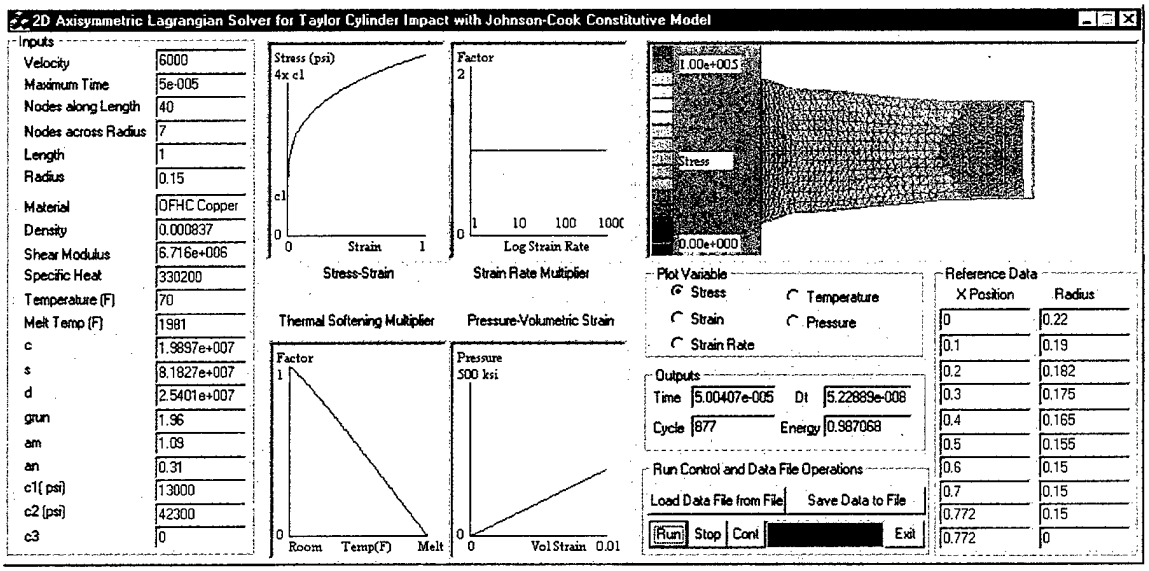

Figure 11. Effect of elimination of strain rate effect (c3 = 0)

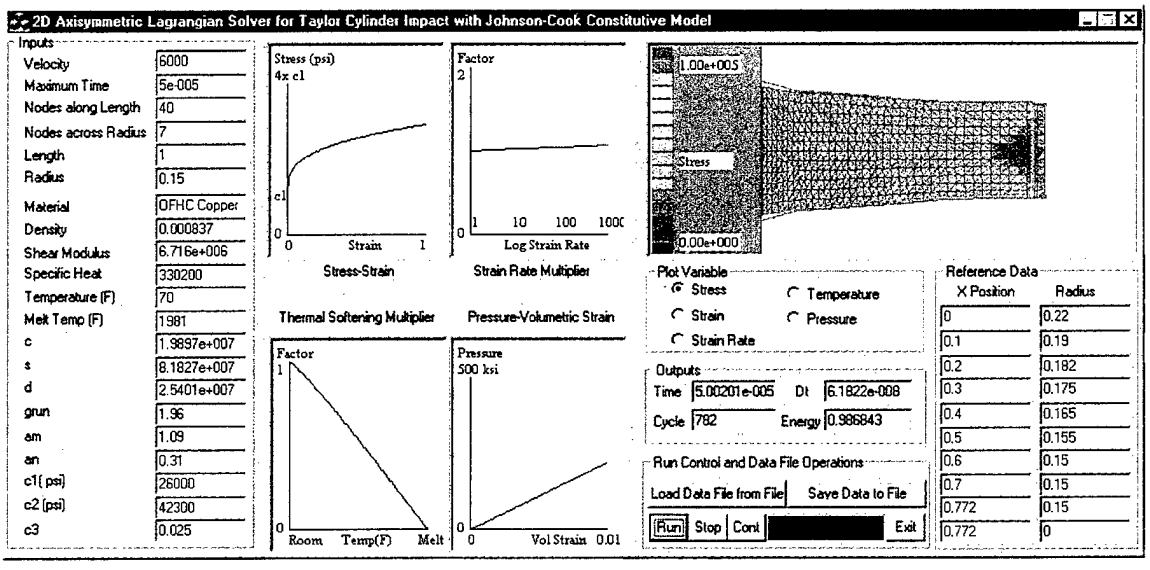

Figure 12. Effect of change in value of c1 (yield strength)

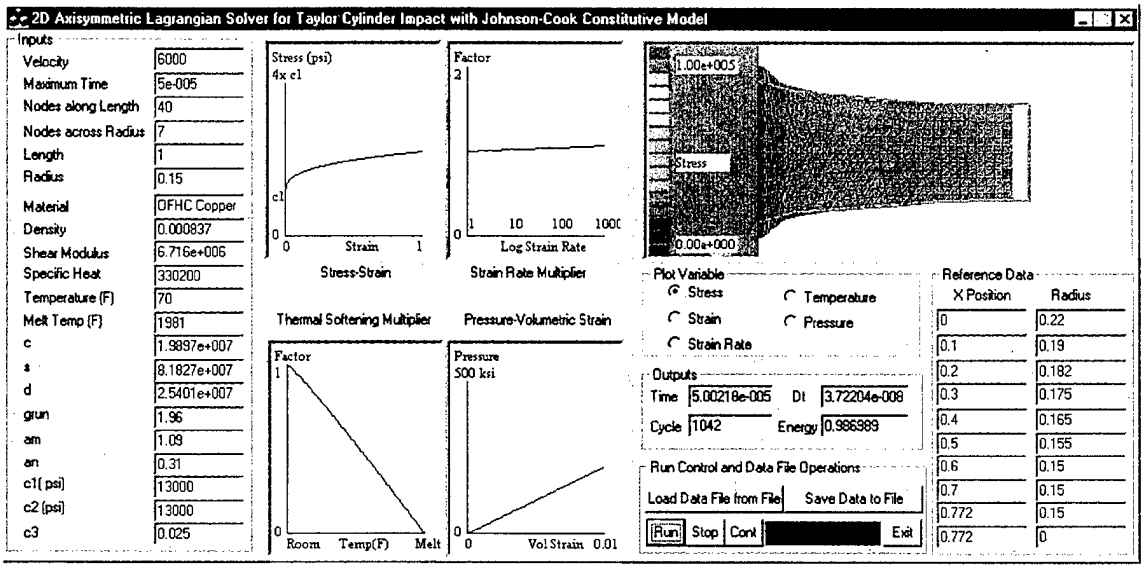

Figure 13. Effect of change in value of c2 (extent of work hardening)

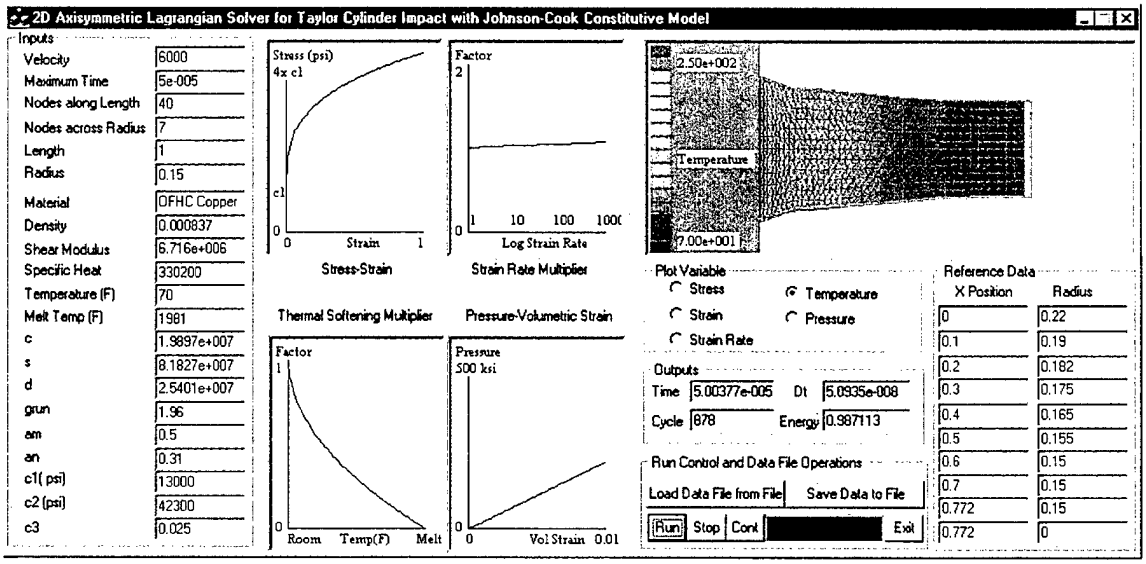

Figure 14. Effect of Thermal Softening Constant Change- note plot here is temperature contours

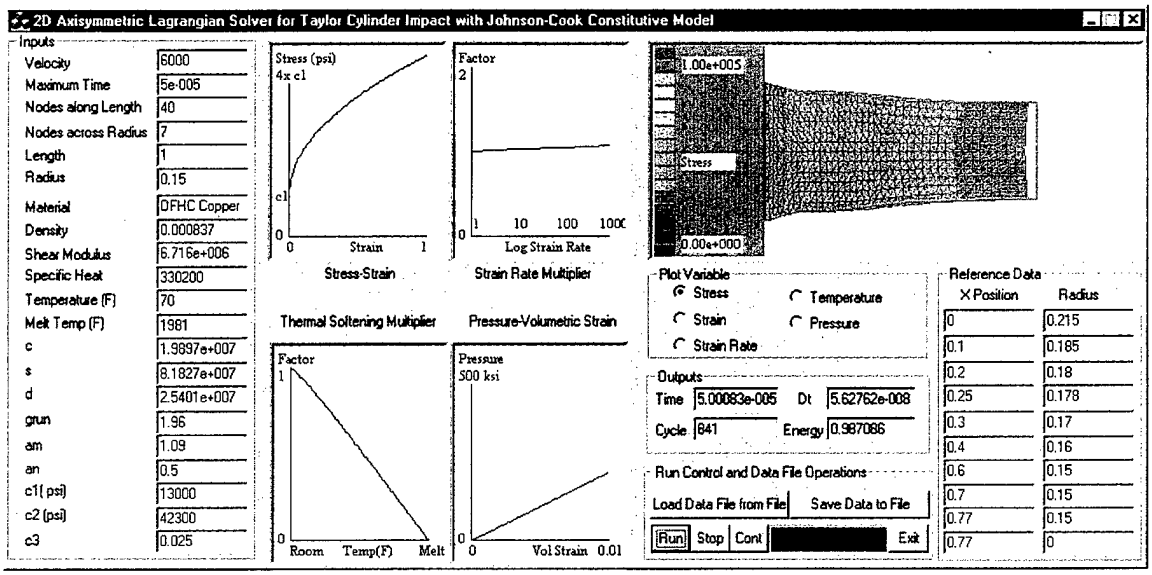

Figure 15. Effect of Strain Rate Hardening Exponent Change

Appendix II. Tables

6000.000000
0.000050
40
7
1.000000
0.150000
OFHC Copper
0.000837
6716000.000000
330200.000000
70.000000
1981.000000
19897000.000000
25401000.000000
81827000.000000
1.960000
1.090000
0.310000
13000.000000
42300.000000
0.025000
0.000000
0.100000
0.200000
0.250000
0.300000
0.400000
0.600000
0.700000
0.770000
0.770000
0.215000
0.185000
0.180000
0.178000
0.170000
0.160000
0.150000
0.150000
0.150000
0.000000

Table 1. Data file as read by code

6000.000000	Impact velocity (inches/sec)
0.000050	Problem maximum time (seconds)
40	Number of nodes along X
7	Number of nodes across radius
1.000000	Cylinder length (inches)
0.150000	Cylinder radius (inches)
OFHC Copper	Material Description
0.000837	Density
6716000.000000	Shear Modulus (psi)
330200.000000	Specific Heat
70.000000	Room Temperature (degrees F)
1981.000000	Melt temperature (degrees F)
19897000.000000	Mie-Gruneisen EOS constant k1
25401000.000000	Mie-Gruneisen EOS constant k2
81827000.000000	Mie-Gruneisen EOS constant k3
1.960000	Gruneisen Constant (usually called Gamma)
1.090000	Johnson Cook parameter m (Thermal Softening Shape)
0.310000	Johnson Cook parameter n (Work Hardening Shape)
13000.000000	Johnson Cook parameter A (psi) (Yield Strength)
42300.000000	Johnson Cook parameter B (psi) (Work Hardening Extent)
0.025000	Johnson Cook parameter C (Strain Rate)
0.000000	X Station 1 for reference profile
0.100000	X Station 2 for reference profile
0.200000	X Station 3 for reference profile
0.250000	X Station 4 for reference profile
0.300000	X Station 5 for reference profile
0.400000	X Station 6 for reference profile
0.600000	X Station 7 for reference profile
0.700000	X Station 8 for reference profile
0.770000	X Station 9 for reference profile
0.770000	X Station 10 for reference profile
0.215000	Radius 1 for reference profile
0.185000	Radius 2 for reference profile
0.180000	Radius 3 for reference profile
0.178000	Radius 4 for reference profile
0.170000	Radius 5 for reference profile
0.160000	Radius 6 for reference profile
0.150000	Radius 7 for reference profile
0.150000	Radius 8 for reference profile
0.150000	Radius 9 for reference profile
0.000000	Radius 10 for reference profile

Table 2. Annotated data file

6000.000000
0.000050
40
7
1.000000
0.150000
OFHC Copper
0.000837
6716000.000000
330200.000000
70.000000
1981.000000
19897000.000000
25401000.000000
81827000.000000
1.960000
1.090000
0.310000
13000.000000
42300.000000
0.025000
0.000000
0.100000
0.200000
0.300000
0.400000
0.500000
0.600000
0.800000
1.000000
1.000000
0.150000
0.150000
0.150000
0.150000
0.150000
0.150000
0.150000
0.150000
0.150000
0.000000

Table 3. Supplemental data for OFHC Copper

6000.000000
0.000050
40
7
1.000000
0.150000
Armco Iron
0.000738
11628000.000000
389700.000000
70.000000
2800.000000
23049000.000000
74960000.000000
750625000.000000
1.690000
0.550000
0.320000
25400.000000
55100.000000
0.060000
0.000000
0.100000
0.200000
0.300000
0.400000
0.500000
0.600000
0.800000
1.000000
1.000000
0.150000
0.150000
0.150000
0.150000
0.150000
0.150000
0.150000
0.150000
0.150000
0.000000

Table 4. Supplemental data for Armco Iron

6000.000000
0.000050
40
7
1.000000
0.150000
2024 Alum
0.000259
3800000.000000
754000.000000
70.000000
935.000000
11130000.000000
18605000.000000
18144000.000000
2.000000
1.000000
0.340000
38400.000000
61800.000000
0.015000
0.000000
0.100000
0.200000
0.300000
0.400000
0.500000
0.600000
0.800000
1.000000
1.000000
0.150000
0.150000
0.150000
0.150000
0.150000
0.150000
0.150000
0.150000
0.150000
0.000000

Table 5. Supplemental data for 2024-T4 Aluminum

6000.000000
0.000050
40
7
1.000000
0.150000
4340 Steel
0.000732
11240000.000000
411400.000000
70.000000
2768.000000
23776000.000000
42693000.000000
72513000.000000
1.160000
1.030000
0.260000
114900.000000
73900.000000
0.014000
0.000000
0.100000
0.200000
0.300000
0.400000
0.500000
0.600000
0.800000
1.000000
1.000000
0.150000
0.150000
0.150000
0.150000
0.150000
0.150000
0.150000
0.150000
0.150000
0.000000

Table 6. Supplemental data for 4340 Steel

Appendix III. Annotated Listing of Solver

```
//      solver, do until time exceeds simulation limits or dt is too small
        while ( (time_prob < m_tmax) && (dt > min_dt) )
        {
                dtbar = (dt + dtlast) * one_half;
                dtnext = dt * 1.1f;
//      begin main node loop on i
                for (i = 0; i < number_of_nodes; ++i)
                {
//      update new velocities
                        nd[i].xdot += (nd[i].fx) * dtbar / nd[i].mass;
                        nd[i].ydot += (nd[i].fy) * dtbar / nd[i].mass;
//      check fixed boundary conditions
                        if (nd[i].x <= 0.0f)
                        {
                                nd[i].x    = 0.0f;
                                nd[i].xdot = 0.0f;
                        }
//      compute new displacements after saving old displacements
                        nd[i].x += nd[i].xdot * dt;
                        nd[i].y += nd[i].ydot * dt;
                }
//      end main node loop on i
//      constrain centerline nodes to centerline
                for (i = 0; i < number_of_nodes; ++i)
                {
                        if (nd[i].centerline || (nd[i].y < 0.0) )
                        {
                                nd[i].y    = 0.;
                                nd[i].ydot = 0.;
                        }
                }
//      zero force arrays for internal forces
//      and mass array to reset masses
                for (i = 0; i < number_of_nodes; ++i)
                {
                        nd[i].fx   = 0.0;
                        nd[i].fy   = 0.0;
                        nd[i].mass = 0.0;
                }
//      begin preliminary element loop on i
//      and to adjust nodal masses due to axisymmetric radial motion
                for (i = 0; i < number_of_elements; ++i)
                {
//      establish nodes 1,2, & 3 for element i and material type
                        n1 = el[i].node1;
                        n2 = el[i].node2;
                        n3 = el[i].node3;
//      compute current area x 2 and ybar
```

```
                areax2 = (nd[n2].y - nd[n3].y) * (nd[n3].x - nd[n1].x)
                        - (nd[n3].y - nd[n1].y) * (nd[n2].x - nd[n3].x);
                ybar = (nd[n1].y + nd[n2].y + nd[n3].y) * one_third;
//      adjust nodal masses due to axisymmetric radial motion
                nd[n1].mass += m_Density * el[i].vol *
                        (nd[n1].y / (ybar * 12.0f) + one_fourth);
                nd[n2].mass += m_Density * el[i].vol *
                        (nd[n2].y / (ybar * 12.0f) + one_fourth);
                nd[n3].mass += m_Density * el[i].vol *
                        (nd[n3].y / (ybar * 12.0f) + one_fourth);
//      save new volumetric strain for element
                el[i].dvol_old = el[i].dvol;
                el[i].dvol = (ybar * pi * areax2) / el[i].vol - 1.0f;
        }
//      compute kinetic energy
        float kinetic_energy = 0.0f;
        for (i = 0; i < number_of_nodes; ++i)
                kinetic_energy += nd[i].mass * one_half * (nd[i].xdot*nd[i].xdot
                                                        +nd[i].ydot*nd[i].ydot);
//      begin main element loop on i
        for (i = 0; i < number_of_elements; ++i)
        {
                el[i].u = -el[i].dvol / (1.0f + el[i].dvol);
                u = el[i].u;
                dvdot = (el[i].dvol - el[i].dvol_old) / dt;
//      compute average volumetric strain (dvbar)
                dvbar = el[i].dvol - dvdot * one_half * dt;
//      establish nodes 1,2, & 3 for element i
                n1 = el[i].node1;
                n2 = el[i].node2;
                n3 = el[i].node3;
//      compute current area x 2
                areax2 = (nd[n2].y - nd[n3].y) * (nd[n3].x - nd[n1].x)
                        - (nd[n3].y - nd[n1].y) * (nd[n2].x - nd[n3].x);
                ybar = (nd[n1].y + nd[n2].y + nd[n3].y) * one_third;
//      compute minimum altitude of element
                y1 = (nd[n1].y - nd[n2].y) * (nd[n1].y - nd[n2].y)
                        +(nd[n1].x - nd[n2].x) * (nd[n1].x - nd[n2].x);
                y2 = (nd[n2].y - nd[n3].y) * (nd[n2].y - nd[n3].y)
                        +(nd[n2].x - nd[n3].x) * (nd[n2].x - nd[n3].x);
                y3 = (nd[n3].y - nd[n1].y) * (nd[n3].y - nd[n1].y)
                        +(nd[n3].x - nd[n1].x) * (nd[n3].x - nd[n1].x);
                ymax = max(y1,y2);
                ymax = float(sqrt((max(ymax,y3))));
                hmin = areax2 / ymax;
                x1 = nd[n2].x - nd[n3].x;
                x2 = nd[n3].x - nd[n1].x;
                x3 = nd[n1].x - nd[n2].x;
                y1 = nd[n3].y - nd[n2].y;
                y2 = nd[n1].y - nd[n3].y;
                y3 = nd[n2].y - nd[n1].y;
//      compute strain rates (exdot...edot)
                exdot = (y1 * nd[n1].xdot + y2 * nd[n2].xdot + y3 * nd[n3].xdot)
                        / areax2;
```

```
                eydot = (x1 * nd[n1].ydot + x2 * nd[n2].ydot + x3 * nd[n3].ydot)
                        / areax2;
                ezdot = (nd[n1].ydot + nd[n2].ydot + nd[n3].ydot)
                        / (ybar * 3.0f);
                exydot = (y1 * nd[n1].ydot + y2 * nd[n2].ydot +
                        y3 * nd[n3].ydot + x1 * nd[n1].xdot +
                        x2 * nd[n2].xdot + x3 * nd[n3].xdot) / areax2;
//      total strains
                el[i].ex += exdot * dt;
                el[i].ey += eydot * dt;
                el[i].ez += ezdot * dt;
                el[i].exy += exydot * dt;
                de = (exdot + eydot + ezdot) / 3.0f;
                exdot -= de;
                eydot -= de;
                ezdot -= de;
//      compute the von mises equivalent strain rate
                edot = float(sqrt(((exdot-eydot)*(exdot-eydot)
                        +(eydot-ezdot)*(eydot-ezdot)
                        +(exdot-ezdot)*(exdot-ezdot)
                        +exydot*exydot * 1.5f) * two_ninths));
//      calculate measure of element rotation
                rotation = one_half * (
                         x1 * nd[n1].xdot
                        + x2 * nd[n2].xdot
                        + x3 * nd[n3].xdot
                        - y1 * nd[n1].ydot
                        - y2 * nd[n2].ydot
                        - y3 * nd[n3].ydot) / areax2;
//      determine average normal stress (sbar)
                el[i].sbar = (el[i].sy + el[i].sx + el[i].sz) * one_third;
//      determine previous deviator stresses (sy1,sx1,sz1)
                sx1 = el[i].sx - el[i].sbar;
                sy1 = el[i].sy - el[i].sbar;
                sz1 = el[i].sz - el[i].sbar;
                sxy1 = el[i].sxy;
//      compute trial deviator and shear stresses
                el[i].sy = sy1 + 2.0f * (m_G * eydot - (sxy1 * rotation)) * dt;
                el[i].sx = sx1 + 2.0f * (m_G * exdot + (sxy1 * rotation)) * dt;
                el[i].sz = sz1 + 2.0f * (m_G * ezdot) * dt;
                el[i].sxy = sxy1 + (m_G * exydot + ((sy1 - sx1) * rotation)) * dt;
//      compute von mises flow stress
                el[i].vmises = float(sqrt((el[i].sy*el[i].sy
                        +el[i].sx * el[i].sx
                        +el[i].sz * el[i].sz ) * 1.5f
                        +el[i].sxy * el[i].sxy * 3.0f));
//      Begin Johnson_Cook Constitutive Model
                float tstar = (el[i].tmp-m_Temp)/(m_MTemp-m_Temp);
                tstar = float(max(tstar,0.));
                tstar = float(min(tstar,1.));
                edot = float(max(edot,1e-4));
                float smax = (m_c1 + m_c2 * float(pow(el[i].ebar,
                                m_an))) * (m_c3 * float(log(edot)) + 1.0f);
                if (m_am > 0.) smax *= 1.0f - float(pow(tstar,m_am));
```

```
//      End Johnson_Cook Constitutive Model
                    if (smax < 0.) smax = 0.;
                    if (el[i].vmises > smax)
                    {
                            factor = smax / el[i].vmises;
                            el[i].sx *= factor;
                            el[i].sy *= factor;
                            el[i].sz *= factor;
                            el[i].sxy *= factor;
                            el[i].vmises = float(sqrt((el[i].sy*el[i].sy
                                    +el[i].sx*el[i].sx
                                    +el[i].sz*el[i].sz)* 1.5f
                                    +el[i].sxy*el[i].sxy * 3.0f));
                    }
//      compute internal energy (edev) from stresses of previous cycle
                    edev = ((el[i].sy + sy1) * eydot + (el[i].sx + sx1)
                            * exdot + (el[i].sz + sz1) * ezdot +
                            (el[i].sxy + sxy1) * exydot) * one_half * (dvbar + 1.0f) * dt;
                    gdt = m_G * dt;
//      adjust strain rates into plastic strain rates
                    exdot -= (el[i].sx - sx1) / (gdt * 2.0f);
                    eydot -= (el[i].sy - sy1) / (gdt * 2.0f);
                    ezdot -= (el[i].sz - sz1) / (gdt * 2.0f);
                    exydot -= (el[i].sxy - sxy1) / gdt;
//      compute equivalent plastic strain rate (epdot)
                    el[i].epdot = float(sqrt(((eydot-exdot)*(eydot-exdot)
                            +(eydot-ezdot)*(eydot-ezdot)
                            +(exdot-ezdot)*(exdot-ezdot)
                            +exydot*exydot* 1.5f) * two_ninths));
//      compute updated plastic strain (ebar)
                    el[i].ebar += el[i].epdot * dt;
//      Begin Equation of State
//      eos provides pressure (p), sound speed squared (ss2), and artificial viscosity (q)
                    if (u > 0.)
                    {
                            ss2 = (m_c * (1.0f - m_grun * u) +
                                    m_d * (u * 2.0f - m_grun * 1.5f * u * u) +
                                    m_s * (u * 3.0f * u - m_grun * 2.0f * u * u * u) +
                                    el[i].engyi * m_grun +
                                    float(fabs(el[i].sbar)) * m_grun / (u + 1.0f)) / m_Density;
                    }
                    if (u <= 0.)
                            ss2 = (m_c + (el[i].engyi +
                                    float(fabs(el[i].sbar))) * m_grun) / m_Density;
                    ss2 += m_G * four_thirds / m_Density;
                    ss2 = float(max(ss2,1e-10));
//      compute artificial viscosity, q  (q=0.0 for dvdot(i).ge.0.)
                    float q = 0.;
                    if (dvdot < 0.)
                            q = m_Density * 4.0f * (u + 1.0f) *
                                    (( hmin * dvdot * (u + 1.0f))*
                                    ( hmin * dvdot * (u + 1.0f)))
                                    - m_Density * 0.2f * (u + 1.0f)
                                    * float(sqrt(ss2)) * hmin * dvdot * (u + 1.0f);
```

```
//      compute total internal energy (engyi) and hydrostatic press (p)
                pmin1 = spall;
//      in the case of u ge 0.0
                if (u >= 0.)
                {
                        p1 = (m_c * u + m_d * u * u + m_s * u *
                                u * u) * (1.0f - m_grun * one_half * u);
                        el[i].engyi = (el[i].engyi + (el[i].sbar- p1 - q)
                                * one_half * dvdot * dt + edev)
                                / (m_grun * one_half * (u + 1.0f) * dvdot * dt + 1.0f);
                        p = p1 + m_grun * (u + 1.0f) * el[i].engyi;
                }
//      in the case of u lt 0.0
                else if (u < 0.)
                {
                        p1 = m_c * u * (1.0f - m_grun * one_half * u);
                        etrial = (el[i].engyi + (el[i].sbar - p1 -
                                q) * one_half * dvdot * dt +
                                edev) / (m_grun * one_half * (
                                u + 1.0f) * dvdot * dt + 1.0f);
                        p = p1 + m_grun * (u + 1.0f) * etrial;
                        pmin1 = -spall;
                        if (p < pmin1)
                        {
                                p = pmin1;
                                el[i].engyi = el[i].engyi + (
                                        el[i].sbar - p - q) * one_half * dvdot * dt + edev;
                        }
                        else
                        {
                                el[i].engyi = etrial;
                        }
                }
//      End Equation of State
//      compute total normal stresses (deviator - p - q)
                el[i].sbar = -p - q;
                el[i].sy += el[i].sbar;
                el[i].sx += el[i].sbar;
                el[i].sz += el[i].sbar;
                float b2 = q * 4.0f / (m_Density * (u + 1.0f));
                float dtt = ssf * hmin / (float(sqrt(b2)) + float(sqrt(b2 + ss2)));
//      determine allowable integration time increment: dtnext
                if (dtt < dtnext) dtnext = dtt;
        }
//      ends main element loop
//      compute forces on the nodes
        float current_energy = 0.0f;
        for (i = 0; i < number_of_elements; ++i)
        {
//      establish nodes 1,2, & 3 for element i
                n1 = el[i].node1;
                n2 = el[i].node2;
                n3 = el[i].node3;
//      compute current area x 2
```

```
                areax2 = (nd[n2].y - nd[n3].y) * (nd[n3].x - nd[n1].x)
                        - (nd[n3].y - nd[n1].y) * (nd[n2].x - nd[n3].x);
                ybar = (nd[n1].y + nd[n2].y + nd[n3].y) * one_third;
                x1 = nd[n2].x - nd[n3].x;
                x2 = nd[n3].x - nd[n1].x;
                x3 = nd[n1].x - nd[n2].x;
                y1 = nd[n3].y - nd[n2].y;
                y2 = nd[n1].y - nd[n3].y;
                y3 = nd[n2].y - nd[n1].y;
                float fhoop= one_third * areax2 * pi * el[i].sz;
                nd[n1].fx -= (ybar * pi * (y1 * el[i].sx + x1 * el[i].sxy));
                nd[n2].fx -= (ybar * pi * (y2 * el[i].sx + x2 * el[i].sxy));
                nd[n3].fx -= (ybar * pi * (y3 * el[i].sx + x3 * el[i].sxy));
                nd[n1].fy -= (ybar * pi * (x1 * el[i].sy + y1 * el[i].sxy) + fhoop);
                nd[n2].fy -= (ybar * pi * (x2 * el[i].sy + y2 * el[i].sxy) + fhoop);
                nd[n3].fy -= (ybar * pi * (x3 * el[i].sy + y3 * el[i].sxy) + fhoop);
//      compute temperature
                el[i].tmp = temp1 + el[i].engyi / (m_sph * m_Density);
                current_energy += el[i].engyi * el[i].vol;
        }
//      ends second main element loop
        current_energy += kinetic_energy;
        normalized_energy = current_energy / original_energy;
        DrawContourGL();
        dtlast = dt;
        ++ncycle;
        time_prob += dt;
        dt = dtnext;
        m_Time = time_prob;
        m_cycle = ncycle;
        m_energy = normalized_energy;
        m_dt = dt;
}
```

DISTRIBUTION LIST
AFRL-MN-EG-TR-2000-7026

Defense Technical Information Center 8725 John J. Kingman Road, Ste 0944 Ft. Belvoir, VA 22060-6218	1
Air University Library 600 Chennault Circle Bldg 1405 Maxwell AFB, AL 36112-6424	1

EGLIN AFB OFFICES:

AFRL/CA-N	1
AFRL/MNOC-1 (STINFO Office)	1
AFRL/MNA	1
AFRL/MNAC(Dr Cook)	10
AFRL/MNMW	5