Investigation on pen extrusion with LBA and UBET

Eindhoven University of Technology
Faculty of Mechanical Engineering
Department of Production Technology & Automation
Laboratory for Forming Technology

Investigation on Pen Extrusion

with LBA and UBET

Wang Shunlong

Apr. 1992 WPA 1304

Report on the courses:

"Analysis of plasticity processes"

"Research on UBET"

I. Introduction

Forging technology is rather old, but it is still playing an important role in the modern manufacturing industry. In order to effectively use the valuable materials and energy and to reduce the dependence on experience, the mechanical engineers have been trying to apply the theory of plasticity into the analysis of forging processes. Forging processes are usually complicated, it is very difficult to get a closed form solution. Until now, just a few theoretical solutions(such as by slip line field theory) are obtained which describe tailored processes with a lot of simplifications.

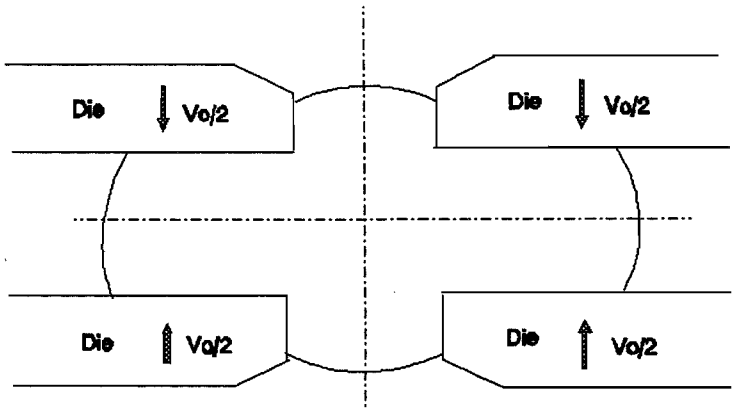
The closed form solution to a metal forming process must be statically stress admissible and kinematic velocity admissible. Generally speaking, 18 simultaneous partial derivative equations have to be solved. The difficulty in obtaining the exact solution by solving so many partial derivative equations lead to three important ways for analysing metal forming problems.

The first is the development of limit analysis method. Disregarding the statically stress admissibility condition, an upper bound solution can be obtained. Disregarding the kinematical velocity admissibility condition, a lower bound solution can be obtained.

The second are the numerical methods (the Finite Difference Method and the Finite Element Method).

The third are the experimental methods, such as the Moiré method, the photoplasticity method, and the visioplasticity method.

In this report, the author investigated a typical forging process--the



billet height : 14 mm

billet radius : 25 mm

die inner radius : 5 mm

displacement size : 4 mm

friction factor : 1.0 --

Fig. 1. The combined pen extrusion

combined pen-extrusion (Fig. 1). First , the lower bound of forming load is derived, which can be used to evaluate the upper bound solution; then to continue the work of Sniekers[1], improved the algorithm for describing a curved surface with cubic spline interpolation method and used the new UBET package to get the forming load.

II. Lower bound analysis

The limit analysis is based on the calculus of variations to get the limit value of plastic forming load. This avoids solving the complicated partial differential equations, it is very simple to be used in the engineering problems.

For the lower bound analysis, a statically admissible stress field σ_{ij}^* is assumed, where σ_{ij}^* has to meet the following conditions:

1. the equilibrium equation has to be satisfied

$$\sigma_{ij}^*{}_{,j}=0$$

2. satisfy the force boundary condition

$$T_i = \sigma_{ij}^* * n_j$$

T_i: the surface force

n_j: the orientation cosine of the surface normal line

3. the yield criterion in plastic deformation region cannot be exceeded

$$s_{ij}^* * s_{ij}^* <= 2K^2$$

where s_{ij}^* : the stress deviator

K : the shear strength

Then the lower bound theorem is expressed as: Among all statically admissible stress fields, the actual one maximizes the expression:

$$I=\int_{S_v} T_i V_i ds$$

Where I is the computed power supplied by the tool over surfaces over which velocity is prescribed.

Supposing the actual material behaves as von Mises's material and the friction is known and acts according to the constant shear factor expression (the von Mises friction model).

$$\tau_{fric} = m * K$$

For a lower bound solution, first a statically admissible stress field has to be constructed. For pen extrusion, the shear stress τ_{rz} is assumed to be distributed as a linear function in z direction (Fig. 2), in region $R_n \leq R \leq R_o$

$$\tau_{rz} = -\frac{2mK}{h} Z \qquad (2.1)$$

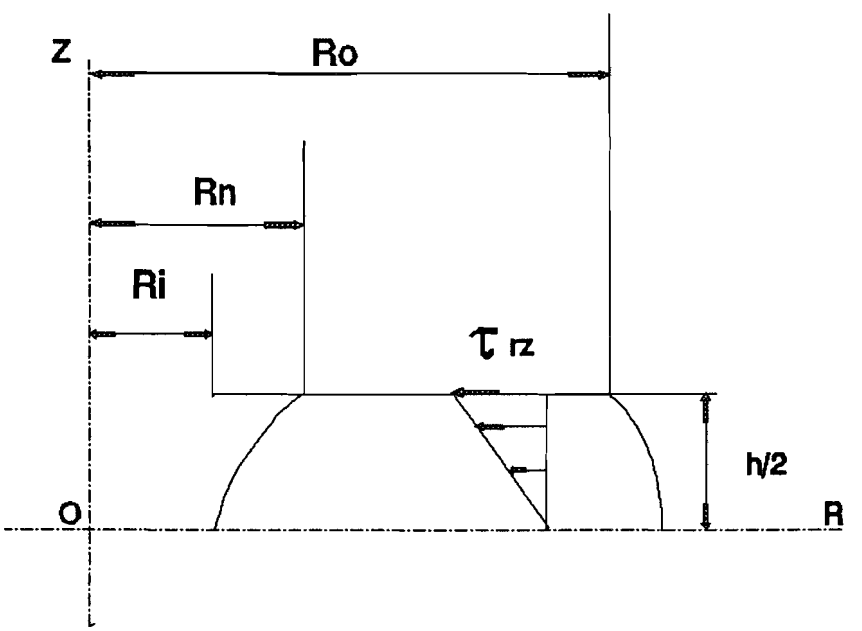


Fig. 2. The shear stress distribution

The equilibrium equation for axisymmetric problems in cylindrical coordinate system are:

$$\frac{\partial \sigma_r}{\partial r}+\frac{\partial \tau_{rz}}{\partial z}+\frac{\sigma_r-\sigma_\theta}{r}=0 \tag{2.2}$$

$$\frac{\partial \tau_{rz}}{\partial r}+\frac{\partial \sigma_z}{\partial z}+\frac{\tau_{rz}}{r}=0 \tag{2.3}$$

Using the Shield assumption for axisymmetric forming problems $\sigma_r=\sigma_\theta$, substitute equation (2.1) into (2.2), (2.3), the following expressions are obtained:

$$\sigma_r=\frac{2mK}{h}r+f_1(z) \tag{2.4}$$

$$\sigma_z=\frac{mK}{hr}z^2+f_2(r) \tag{2.5}$$

The boundary condition is at point $r=R_o$, $z=0$, so $\sigma_r=0$

So in the $z=0$ plane, $f_1(z)=-(2mKR_o)/h$

$$\sigma_r=\frac{2mK}{h}(r-R_o) \tag{2.6}$$

Using the yield criterion for axisymmetric problems:

$$(\sigma_r-\sigma_z)^2+3\tau_{rz}^2=3K^2 \tag{2.7}$$

$$\left[\frac{2mK}{h}r+f_1(z)-\frac{mK}{hr}z^2-f_2(r)\right]^2+3\left(\frac{-2mK}{h}z\right)^2=3K^2 \tag{2.8}$$

In the $z=0$ plane, the $f_2(r)$ can be obtained

$$f_2(r)=-\sqrt{3}K-\frac{2mK}{h}(R_o-r)$$

$$\sigma_z=\frac{mK}{hr}z^2-\sqrt{3}K-\frac{2mK}{h}(R_o-r) \tag{2.9}$$

Substitute equation (2.9) into (2.7), the σ_r can be obtained

$$\sigma_r=\sqrt{3}K\sqrt{1-\frac{4m^2z^2}{h^2}}+\frac{mK}{hr}z^2-\sqrt{3}K-\frac{2mK}{h}(R_o-r) \tag{2.10}$$

The same way, in region $R_i \le r < R_n$, the shear stress is assumed as:

$$\tau_{rz}=\frac{2mK}{h}Z \tag{2.11}$$

Then the following expressions can be found:

$$\sigma_z=-\frac{mK}{hr}z^2-\sqrt{3}K+\frac{2mK}{h}(2R_n-R_o-r) \tag{2.12}$$

$$\sigma_r=\sqrt{3}K\sqrt{1-\frac{4m^2z^2}{h^2}}-\frac{mK}{hr}z^2-\sqrt{3}K+\frac{2mK}{h}(2R_n-R_o-r) \tag{2.13}$$

By integration the σ_z, in the region $R_i \le r \le R_o$, the lower bound of working force can be found:

$$\begin{aligned}
P&=-\int_{R_i}^{R_o}\sigma_z*(2\pi r)dr\\
&=2\pi K\{\int_{R_i}^{R_n}[\frac{mz^2}{hr}+\sqrt{3}-\frac{2m}{h}(2R_n-R_o-r)]rdr\\
&\quad+\int_{R_n}^{R_o}[\sqrt{3}+\frac{2m}{h}(R_o-r)-\frac{mz^2}{hr}]rdr\}\\
&=2\pi K[\frac{\sqrt{3}}{2}(R_o^2-R_i^2)+\frac{2m}{3h}(2R_n^3-R_o^3-R_i^3)\\
&\quad+\frac{m}{h}(R_o^3-2R_n^3+2R_nR_i^2-RR_i^2)+\frac{mz^2}{h}(2R_n-R_i-R_o)]
\end{aligned} \tag{2.14}$$

The average pressure can be presented as:

$$p=\frac{P}{\pi\sigma_s(R_0^2-R_i^2)}$$
$$=\frac{2}{\sqrt{3}(R_0^2-R_i^2)}[\frac{\sqrt{3}}{2}(R_o^2-R_i^2)+\frac{2m}{3h}(2R_n^3-R_o^3-R_i^3) \quad (2.15)$$
$$+\frac{m}{h}(R_o^3-2R_n^3+2R_nR_i^2-R_oR_i^2)+\frac{mz^2}{h}(2R_n-R_i-R_o)]$$

Here R_o: the outer radius at z

R_0: the initial billet outer radius

R_i: the die inner radius

h : the instantaneous height of the workpiece.

Due to the principle of maximum work,the R_n should be optimized to get the highest lower bound of load.

$$\frac{\partial p}{\partial R_n}=0$$

$$R_n=\sqrt{R_i^2+mz^2} \quad (2.16)$$

For the comparison with Sniekers's result, the constant friction factor m = 1 is also used.

By substituting equation (2.16) into equation (2.15), the highest lower bound solution is obtained. Using the boundary condition m = 1 and z = h/2, the lower bound of average pressure on the tool can be presented as:

$$p=\frac{P}{\pi\sigma_s(R_0^2-R_i^2)}$$
$$=\frac{2}{\sqrt{3}(R_0^2-R_i^2)}[\frac{\sqrt{3}}{2}(R_o^2-R_i^2)+\frac{2}{3h}(2R_n^3-R_o^3-R_i^3) \quad (2.17)$$
$$+\frac{1}{h}(R_o^3-2R_n^3+2R_nR_i^2-R_oR_i^2)+\frac{h}{4}(2R_n-R_i-R_o)]$$

The lower bound analysis is based on the assumption of a statically stress admissible field, a suitable velocity field which satisfy the velocity boundary condition and the volume constancy condition cannot be found. So the geometry of the workpiece for the successive steps cannot be obtained by lower bound analysis. This is a big weakness of lower bound analysis. For this special statically stress admissible field, from equation (2.17), the inner radius R_i, the outer radius R_o (at $z=h/2$), the instantaneous height h are needed to obtain the lower bound solution. For the purpose of comparison of lower bound and upper bound analysis, the upper bound solutions of the workpiece geometry (the outer radius at $z=h/2$) are used to get the lower bound solution.

Table 1 gives the highest lower bound solutions of forming load (for this statically admissible stress field), these results will be compared with the upper bound results in the next section.

Table 1 The results of LBA

	Ro(z=h/2)	h	p
step0	25.0	14	1.533
step1	27.326	10	2.300
step2	32.836	6	5.226

III. Upper Bound Analysis

It is very difficult to get a closed form solution including the local parameters(strain, stress, strain rate, temperature etc) analytically for metal forming processes as is pointed out in the introduction.Sometimes just an engineering solution of the forming energy (load) is needed, and only a rational and practical solution by a scientific and simple method is necessary. In the former section, the lower bound of forming load for combined pen extrusion is obtained. But in real production processes, an upper bound solution is necessary for choosing forming machines scientifically. The most important task to be done is to construct the velocity field as closely to the real deformation field as possible, so that not only a fairly good upper bound solution for the load can be obtained, but also a fairly accurate prediction of the workpiece geometry.

In the last two decades, the UBET (abbreviation for Upper Bound Elemental Technique) has been widely used in the analysis of metal forming processes. Many scholars contributed to the research of UBET, among them are Kudo[6], Bramley[7], Kiuchi[8] etc. In the M.Sc thesis of Sniekers, he constructed elements with admissible velocity fields including internal degrees of freedom and applied this to combined pen-extrusion process and obtained fairly good results compared with the original UBET.

In this section, the author improved the algorithm for describing a curved surface with cubic spline interpolation method and used the new UBET package to get the forming load.

III.1 The spline interpolation

In UBET analysis, the curved surface connecting two elements should be described accurately. There are many ways to describe a curved surface, such as multinominal interpolation, the linear interpolation in every subinterval, the spline interpolation. The high-degree multinominal interpolation sometimes exhibits the large oscillations characteristic. The linear interpolation in every subinterval is not accurate enough. The spline functions yield smooth interpolating curves in the whole interval, give very accurate description of the curved surface.

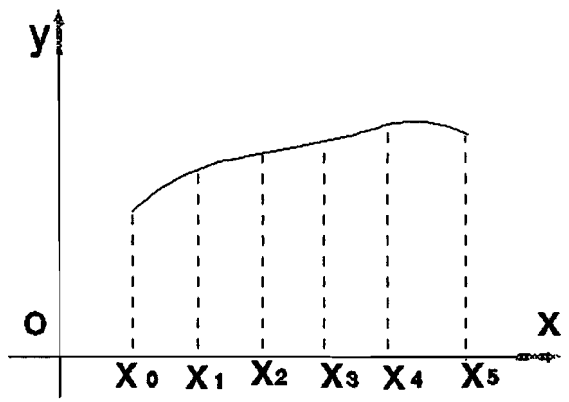


Fig. 3 The spline interpolation

Given (n+1) ordinal numerical points on XOY plane (x_0,y_0), (x_1,y_1), ... (x_n,y_n), and $x_0<x_1<.....<x_n$, a function s(x) can be constructed(Fig.2), s(x) meets the conditions as follows:

1. $s(x_i)=y_i$, $(i=0,1,2,.....n)$;
2. in every subinterval $[x_i,x_{i+1}]$, s(x) is the multinominal with 3 degree free of x;
3. s(x) is twice continuously differential on $[x_0,x_n]$.

s(x) is called the cubic spline function or the cubic interpolation multinominal.

The spline function can be constructed by the first derivatives at the nodal points m_i, the following expressions is obtained

$$\lambda_i m_i - 1 + 2m_i + \mu_i m_{i+1} = c_i \quad (i=1,2,.....n-1) \tag{3.1}$$

here

$$h_i = x_i - x_{i-1}$$

$$\mu_i = \frac{h_i}{h_i + h_{i+1}}$$

$$\lambda_i = 1 - \mu_i = \frac{h_{i+1}}{h_i + h_{i+1}}$$

$$c_i = 3[\mu_i \frac{y_{i+1} - y_i}{h_{i+1}} + \lambda_i \frac{y_i - y_{i+1}}{h_i}]$$

The equation (3.1) is the cubic spline function expression of the first derivative m_i, it is a (n-1) system of linear equations with (n + 1) unknowns (m_0, m_1, ..., m_n).

To solve equation (3.1), two additional boundary conditions is needed. Usually the boundary values of m_0, m_n, is not known, so the author suggests a fairly good way should be to use the "three points Lagrange interpolation" to get the first derivatives at the boundary.

$$m_0=\frac{2x_0-(x_1+x_2)}{(x_0-x_1)(x_0-x_2)}y_0+\frac{x_0-x_2}{(x_1-x_0)(x_1-x_2)}y_1+\frac{x_0-x_1}{(x_2-x_1)(x_2-x_0)}y_0$$

$$m_n=\frac{2x_n-(x_{n-1}+x_{n-2})}{(x_n-x_{n-1})(x_n-x_{n-2})}y_n+\frac{x_n-x_{n-2}}{(x_{n-1}-x_n)(x_{n-1}-x_{n-2})}y_{n-1}+\frac{x_n-x_{n-1}}{(x_{n-2}-x_{n-1})(x_{n-2}-x_i}$$

$$\begin{aligned}s_i(x)=&[\frac{3}{h_i^2}(x_{i+1}-x)^2-\frac{2}{h_i^3}(x_{i+1}-x)^3]y_i\\&+[\frac{3}{h_i^2}(x-x_i)^2-\frac{2}{h_i^3}(x-x_i)^3]y_{i+1}\\&+h_i[\frac{3}{h_i^2}(x_{i+1}-x)^2-\frac{1}{h_i^3}(x_{i+1}-x)^3]m_i\\&-h_i[\frac{3}{h_i^3}(x-x_i)^2-\frac{1}{h_i^3}(x_{i+1}-x)^3]m_{i+1}\end{aligned} \qquad (3.2)$$

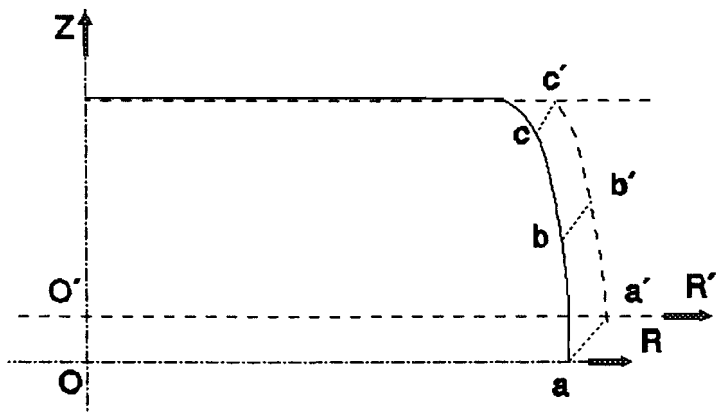
With m_0, m_n, the system of linear equation (3.1) can be solved by Gaussian elimination algorithm to get m_i. Then the spline functions $s_i(x)$ can be expressed by equation (3.2).

The cubic spline interpolation algorithm is written into UNIT INTERPOL.

III.2 On the outer-radius calculation

In calculating the new geometry for the new step--the outer-radius, when the friction factor is high enough and the deformation is also large enough, there will be the foldover phenomenon, that is the side surface material of the workpiece folding over to contact with the upper die(Fig. 4). The solid line repersents the Nth step and the dashed line represents the (N+1)th step. The free surface of cd in Nth step foldover to the upper die in the (N+1)th step. For the example--pen extrusion, the friction factor is chosen as m=1 and the deformation is fairly large,

the foldover phenomenon must be happen. This phenomenon must be handled properly in order to get the correct new geometry.



Solid line: Nth step

Dashed line: (N+1) step

Fig. 4 The foldover phenomenon

The outer-radius is solved in Unit Solv_out.pas, a condition statement is added in the PROCEDURE integrate_velocity_time for calculating the outer radius. That is

```
int1: = billet_geometry.height0/2-0.5*proces_data.velocity
  *(proces_counter-1+(k+1)*time_level);
IF loop_point.z+axi_velo*proces_data.delta_time > = int1
THEN new_point.z= int
ELSE new_point.z=loop_point+axi_velo*proces_data.delta_time
```

By this way, the outer radius is calculated from the suggested velocity field, which can be used to compare with the experimental results and also for the calculation of the lower bound solution.

III.3 The upper bound results

Using the velocity field for pen extrusion suggested by Sniekers and UNIT Interpol, the following upper bound results(Fig.2) is obtained. The "difference" represent 2*(UBET-LBA)/(UBET+LBA)*%, they show the quality of the statically admissible stress field and the kinematically admissible velocity field for pen extrusion.

Table 2. The power consumption

	LBA	UBET	DIFFERENCE(%)
step 0	1.533	1.564	2.00
step 1	2.300	2.430	5.50
step 2	5.226	5.460	4.38

From table 2, it can be seen that the difference between the LBA results and the UBET results is fairly small. This shows that both the LBA results and the UBET results are fairly good approximation to the real solution, so both the statically admissible stress field and the kinematically admissible velocity field are fairly good to model the pen extrusion process. Combining the lower bound analysis and the upper bound analysis makes it very easy to check the quality of the limit analysis, this avoids to do some experiments to verify the results. If we just try to verify the quality of the limit analysis by doing experiments, we must be very carefully to choose the parameters to coincide with the parameters in the limit analysis, this is very difficult and is not necessary. So the combination of LBA and UBA is a very

good way to verify the quality of limit analysis. Of course, it is necessary to combine the LBA and UBA closely with the experimental observation to construct the best kinematically admissible velocity field.

For the total power consumption, the friction power on element I counts a large part. For example, in the first step, total power is 5.8923E+03, the friction power(mainly the friction power on element I) is 1.2347E+03. So inspired by the lower bound analysis, the author thinks that the following kinematically admissible velocity field (element I) may give more accurate results. This velocity field based on the fact that when the constant friction factor m is zero, the material will deform uniformly; when m = 1 (sticking friction), the radical velocity of the material at the die/workpiece interface will be zero.

For a given friction factor m, the boundary condition can be expressed as:

$$u_r(z=h/2)=\frac{v_0 r}{2h}(1-\frac{R_n^2}{r^2})(1-m)$$

$$u_z(z=h/2)=-\frac{v_0}{2}$$

$$u_z(z=0)=0$$

So u_r, u_z can be chosen easily to satisfy the above boundary conditions. Further work should be done to see the quality of the new velocity field.

VI. Discussion

The limit analysis is a useful way to analysis metal forming processes. But there are still some problems in the limit analysis, especially in the analysis of non-steady state processes.

The combined pen extrusion is a typical non-steady state process. This kind of metal forming process must be discrete into several steps. For both lower bound and upper bound analysis, the information of the last step is needed. For lower bound analysis, just a statically admissible stress field is assumed, the velocity field cannot be obtained, so it is impossible to continue the lower bound analysis for next step. This is a serious weakness of the lower bound analysis. For upper bound analysis, a kinematically admissible velocity field is assumed, it is possible to integrate the velocity field to calculate the geometry for the next step. But for the non-steady state large deformation processes, especially when there is a unconstrained surface, the UBA results are sometimes unreliable. This is because, when carrying out the calculation procedure incrementally, the accumulation of possible errors in the predicted deformation increments may lead to considerable deviations of the predicted current workpiece shape and tool/workpiece contact area from reality. So it is necessary to construct the velocity field as closely to the real velocity field as possible (especially for non-steady state processes), and simultaneously to carry out lower bound analysis. Only when the LBA results and the UBA results have little differences, the geometry predicted by upper bound analysis can be considered to be reliable.

So the main task is to construct the high quality velocity field which can model the real metal forming processes very well.

V. Conclusion

The pen extrusion process are investigated with the lower bound analysis and UBET. The results show that both the lower bound solution and the UBET solution are of high quality. It is a easy and very useful way to combine the lower bound analysis and the upper bound analysis to verify the accuracy of the limit analysis results. A new velocity field is suggested.

There are some weaknesses in lower bound analysis and upper bound analysis. The lower bound analysis cannot predict the geometry for the analysis of the next step, it can just analyze a special step, so it is difficult to use it for non-steady state process. The upper bound analysis has some unreliability for analysing large deformation with unconstrained surfaces by incremental calculation. It is the most important task to construct a kinematically admissible velocity field which can model the real forming processes.

The UBET is an effective way for analysing the forming processes. The UBET should be developed to contain the characteristics of treating material work hardening, the curved tool boundary, the temperature change, pressure distribution over tool surfaces and the material failure. It is also possible to combine the UBET analysis with the metallurgy to predict the microstructure(the grain size). So a lot of work should be done to perfect the UBET analysis.

Acknowledgement

The author wishes to thank Ir. S. Hoogenboom and Ir. R. Sniekers for their help during the project.

Reference

1. R. Sniekers, UBET Research on friction influenced elements(M.Sc thesis, Appendix B: Analysis, Appendix C: Listings), Eindhoven University of Technology & University of Bath, July, 1990
2. B. Avitzur, Metal Forming: Processes and Analysis, 1968
3. E. P. Unksov, An engineering theory of plasticity, 1961
4. W. F. Hosford, R. M. Caddell, Metal Forming: Mechanics and Metallurgy, 1983
5. D. Y. Yang, J. H. Kim, C. K. Lim, An arbitrarily inclined triangular element and it's application to combined forging, Transactions of the ASME, may 1985
6. S. Kobayashi, Upper bound solutions of axisymmetric forming problems-I,II, Transactions of ASME, 1964
7. H. Kudo, An upper bound approach to plane strain forging and extrusion--I,II, International Journal of Mechanical Sciences,1960
8. R. P. McDermott, A. N. Bramley, An elemental upper-bound technique for general use in forging analysis, 1974
9. M. Kiuchi, Y. Murata, Study on application of UBET, Proc. 4th Int. Conf. Prod. Eng., 1980
10. O. M. Ettouney, K. A. Stelson, An approximate model to calculate foldover and strains during cold upsetting of cylinders Part I: Formulation and evaluation of the foldover model, Journal of engineering for industry, Transactions of the ASME, Aug. 1990
11. O. M. Ettouney, K. A. Stelson, An approximate model to calculate foldover and strains during cold upsetting of cylinders Part II: Use of the foldover model to estimate friction, Journal of engineering for industry, Transactions of the ASME, Aug. 1990

12. P. Dadras, J. F. Thomas. Jr., Analysis of axisymmetric upsetting based on flow pattern observations, Int. J. Mech. Sci., Vol. 25, No. 6, 1983

13. P. Dadras, P. R. Burte, Nonisothermal axisymmetric forging, Journal of engineering for industry, Transactions of the ASME, Nov. 1986

14. D. Y. Yang, Y. Choi, J. H. Kim, Analysis of upset forging of cylindrical billets considering the dissimilar frictional conditions at the two flat die surfaces, Int. J. Mach. Tools Manufact., Vol. 31, No. 3, 1991

15. W. T. Carter, Jr., D. Lee, Further analysis of axisymmetric upsetting, Journal of engineering for industry, Transactions of the ASME, Aug. 1986

16. J. A. H. Ramaekers, S. Hoogenboom, The prediction of tool pressures in coining, Advances in Manufacturing Technology, 1985

17. Wang Zutang, Mechanics of metal forming processes, Tsinghua University Press, 1987

18. Li Qingyang, Wang Nengchao, Yi Dayi, Numerical analysis, Middle China University of Science and Technology Press, 1982

19. J. Stoer, R. Bulirsch, Introduction to numerical analysis, Springer-Verlag, 1980

20. Handbook of mathematics(in Chinese), Higher Education Press, 1979

```
{*************************************************************}
{*                                                           *}
{*                                                           *}
{*     UNIT          :INTERPOL                               *}
{*     Purpose       :find corresponding values in an array by means of  *}
{*                    cubic spline interpolation.            *}
{*     Version       :1.00                                   *}
{*     Date          :27 Mar. 1991                           *}
{*     Language      :Turbopascal 5.5  (Borland)             *}
{*     System        :Cirp-AT with EGA graphics and math coprocessor  *}
{*     Programmer    :Wang Shunlong                          *}
{*     Information   :Eindhoven University of Technology     *}
{*                    Department of Mechanical Engineering   *}
{*                    Group Production technologie and -Automation  *}
{*                    Laboratory of Forging Technology       *}
{*     Telephone     :040-474158  Wang Shunlong              *}
{*                    :040-816969  Wang Shunlong             *}
{*                                                           *}
{*                                                           *}
{*************************************************************}


UNIT interpol;

INTERFACE

 USES
  BULGVAR;

 TYPE
  rreal_array_type = array [0..discrete] of real;

 VAR
  cc: rreal_array_type;


   FUNCTION  spline(xx:  point_array_type;r:  real;  k:  integer;
```

```
cc:rreal_array_type):real;

  FUNCTION get_z_at_r(xx: point_array_type; r :REAL): REAL;

  FUNCTION get_r_at_z(xx: point_array_type; z: REAL): REAL;

  PROCEDURE find_cross( VAR point: point_type);

IMPLEMENTATION

{************************************************************}

FUNCTION spline(xx: point_array_type;r:real;k: integer; cc:rreal_array_type):real;

{*** This function constructs the cubic spline interpolation function    *}
{*** with the first order derivative for a group of points               *}

VAR

 i,n,j: INTEGER;
 t1,k1,k2,s1,s2: REAL;
 hh,ss,bb,dd: ARRAY[0..discrete] OF REAL;
 int1,int2,int3: REAL;
 int4,int5,int6: REAL;
 int7,mm,mm0  : REAL;

BEGIN

 n:= discrete;

 mm0:=0;
 FOR j:=1 TO discrete DO
 BEGIN
 mm:=abs(xx[j].r-xx[j-1].r);
 IF mm>mm0 THEN mm0:=mm;
 END;
 IF mm0<=0.01  THEN spline := xx[0].z
```

```
ELSE BEGIN
FOR i:=0 TO n DO
cc[i]:=0;

FOR i:=1 TO n DO
 hh[i]:= xx[i].r-xx[i-1].r;

FOR i:=1 TO n-1 DO
 BEGIN
  ss[i]:= hh[i]/(hh[i]+hh[i+1]);
  bb[i]:= 1-ss[i];
  cc[i]:= 3*ss[i]*(xx[i+1].z-xx[i].z)/hh[i+1];
  cc[i]:= cc[i]+3*bb[i]*(xx[i].z-xx[i-1].z)/hh[i];
 END;

k1:= xx[2].r-xx[0].r;
s1:= -((hh[1]+k1)*xx[0].z)/(hh[1]*k1)+(k1*xx[1].z)/(hh[1]*hh[2]);
s1:= s1-(hh[1]*xx[2].z)/(k1*hh[2]);
k2:= xx[n].r-xx[n-2].r;
s2:= hh[n]*xx[n-2].z/(k2*hh[n-1])-k2*xx[n-1].z/(hh[n-1]*hh[n]);
s2:= s2+(hh[n]+k2)*xx[n].z/(k2*hh[n]);
ss[0]:= 0;
ss[n]:= 0;
cc[0]:= 2*s1;
cc[n]:= 2*s2;
bb[n]:= 0;
bb[0]:= 0;

FOR i:=0 TO n DO
 dd[i]:=2;

 ss[0]:= ss[0]/dd[0];
 cc[0]:= cc[0]/dd[0];

FOR i:=1 TO n DO
 BEGIN
  t1:= dd[i]-ss[i-1]*bb[i];
```

```
  ss[i]:= ss[i]/t1;
  cc[i]:= (cc[i]-cc[i-1])*bb[i]/t1;
 END;

FOR i:=n-1 DOWNTO 0 DO
 cc[i]:=cc[i]-ss[i]*cc[i+1];



BEGIN

 int1:=xx[k].r-r;
 int2:=r-xx[k-1].r;
 hh[k]:= xx[k].r-xx[k-1].r;
 int3:= sqr(int1)/sqr(hh[k]);
 int4:= int1*sqr(int1)/(hh[k]*sqr(hh[k]));
 int5:= sqr(int2)/sqr(hh[k]);
 int6:= int2*sqr(int2)/(hh[k]*sqr(hh[k]));

 int7:= (3*int3-2*int4)*xx[k-1].z;
 int7:= int7+(3*int5-2*int6)*xx[k].z;
 int7:= int7 + (int3-int4)*hh[k]*cc[k-1];
 int7:= int7 - (int5-int6)*hh[k]*cc[k];
 spline:= int7;
 writeln('spline= ',int7);
 END;
 END;
 END;

{*************************************************************}

FUNCTION get_z_at_r (xx: point_array_type; r: REAL): real;

VAR

 i: INTEGER;
```

```
BEGIN

 IF xx[discrete].r > = xx[0].r THEN BEGIN
 i: = 0;
 IF r > = xx[discrete].r THEN get_z_at_r : = xx[discrete].z
 ELSE IF r < xx[0].r THEN get_z_at_r : = xx[0].z
 ELSE BEGIN
  REPEAT
   INC(i);
  UNTIL ((xx[i].r > =r) and (xx[i-1].r < =r));
  writeln('i = ',i);
  get_z_at_r : = spline(xx, r,i,cc);
  END;
 END;

 IF xx[discrete].r < xx[0].r THEN BEGIN
  i: = 0;
  IF r < = xx[discrete].r then get_z_at_r : = xx[discrete].z
  ELSE IF r > xx[0].r THEN get_z_at_r : = xx[0].z
  ELSE BEGIN
   REPEAT
     INC(i);
   UNTIL ((xx[i].r < = r) and (xx[i-1].r > = r));
   writeln('i = ',i);
   get_z_at_r : = spline(xx, r,i,cc);
   END;
  END;
 END;


{*************************************************************}

FUNCTION get_r_at_z(xx: point_array_type; z: REAL): REAL;


VAR
```

```
j: INTEGER;
contem: REAL;


BEGIN

  FOR j:=0 TO discrete DO
  BEGIN
   contem:= xx[j].z;
   xx[j].z:= xx[j].r;
   xx[j].r:= contem;
  END;
  get_r_at_z:=get_z_at_r(xx,z);
  FOR j:=0 TO discrete DO
  BEGIN
   contem:= xx[j].z;
   xx[j].z:= xx[j].r;
   xx[j].r:= contem;
  END;
 END;

{*************************************************************}

PROCEDURE find_cross( VAR point : point_type );

VAR
 int1, int2     : REAL;
 a1, a2, b1, b2 : REAL;
 i              : INTEGER;

BEGIN
 i:= -1;
 {* determinination of the closest available points of the two arrarys to *}
 {* the cross points of the curves                                        *}

 REPEAT
   INC(i);
```

```
      UNTIL (saved_discontinuity_23[i].z >= discontinuity_23[i].z);
 IF i = 0 THEN BEGIN
            point.r := die_geometry.inner_radius;
  point.z := mom_height;
 END
 ELSE BEGIN
 {* calculation cross point of two intersecting lines                *}
 {* straight line : y = a*x + b                                      *}

            int1 := discontinuity_23[i].z - discontinuity_23[i-1].z;
            int2 := discontinuity_23[i].r - discontinuity_23[i-1].r;
  a1   := int1/int2;
            b1   := discontinuity_23[i].z - a1*discontinuity_23[i].r;

            int1 := saved_discontinuity_23[i].z - saved_discontinuity_23[i-1].z;
            int2 := saved_discontinuity_23[i].r - saved_discontinuity_23[i-1].r;
  a2   := int1/int2;
            b2   := saved_discontinuity_23[i].z - a2*saved_discontinuity_23[i].r;

  point.r := (b2-b1)/(a1-a2);
            point.z := get_z_at_r(discontinuity_23,point.r);
 END;
END;

{*************************************************************}
END. {of unit}
```

```
{*************************************************************}

PROCEDURE integrate_velocity_time2 ( radial_velocity : velocity_function;
                                     axial_velocity  : velocity_function;
                                     time            : REAL;
                                     old_point       : point_type;
                                     VAR new_point : point_type;
                                     VAR proces_counter: integer );

{*** This procedure integrates the velocities, with respect to time, so as    *}
{*** a result, the displacements are calculated. The foldover phenomenon   *}
{*** is treated.                                                             *}

VAR
  intermediate1    : REAL;
  intermediate2    : REAL;
  old_time         : REAL;
  axi_velo         : REAL;
  rad_velo         : REAL;

      loop_point   : point_type;
      point_est    : ARRAY [ 0..discrete ] OF point_type;

  level            : INTEGER;
  time_level       : REAL;
  iint,iint1,etha  : REAL;
  j,k              : INTEGER;



BEGIN
  etha        := 0.01;
  j           := 0;
      old_time      := time - proces_data.delta_time;
      axi_velo      := axial_velocity  (old_point,old_time);
      rad_velo      := radial_velocity ( old_point,old_time);
      iint: =billet_geometry.height0/2-0.5*proces_data.velocity*proces_counter;
```

```
      IF old_point.z + axi_velo * proces_data.delta_time>iint THEN
      new_point.z:=iint  ELSE
      new_point.z:= old_point.z+axi_velo*proces_data.delta_time;
      new_point.r   := old_point.r + rad_velo * proces_data.delta_time;
      point_est[j]  := new_point;
 level        := 1;
 time_level    := proces_data.delta_time;
 REPEAT
  INC(j);
  level       := level * 2;
  time_level  := time_level/2;
            loop_point  := old_point;
  FOR k := 0 TO (level-1) DO BEGIN
                  axi_velo     := axial_velocity   ( loop_point, old_time +
                               k*time_level);
                  rad_velo     := radial_velocity ( loop_point, old_time +
                               k*time_level);
                  iint1                    :=  0.5*billet_geometry.height0 -
                               0.5*proces_data.velocity
                               *(proces_counter-1+(k+1)*time_level);
            IF loop_point.z + axi_velo * time_level>iint1 THEN
                 loop_point.z:=iint1  ELSE
                  loop_point.z := loop_point.z + axi_velo * time_level;
                  loop_point.r := loop_point.r + rad_velo * time_level;
  END;
            point_est[j] := loop_point;
            IF point_est[j].r = 0 THEN intermediate1 := 0
            ELSE intermediate1  := point_est[j-1].r/point_est[j].r;
            IF point_est[j].z = 0 THEN intermediate2 := 1
            ELSE  intermediate2 := point_est[j-1].z/point_est[j].z;
      UNTIL ((ABS(1-intermediate1)) <= etha) and ((ABS(1-intermediate2)) <=
etha);
      new_point := point_est[j];
END;

{*************************************************************}
```

```
PROCEDURE calculate_radius(proces_counter:integer);

VAR
      r1        : TEXT;
      i         : INTEGER;
      new_point : point_type;
      old_point : point_type;


BEGIN
      assign(r1,path + 'rad' + file_count_string + '.dat');
      rewrite(r1);
      FOR i := 0 TO discrete DO BEGIN
            old_point := radius_history[i];
            integrate_velocity_time2 ( radial_velocity_l,
                                       axial_velocity_l,
                                       mom_time,
                                       old_point,
                                       new_point,proces_counter );
            radius_history[i] := new_point;
            writeln(r1, radius_history[i].r);
            writeln(r1, radius_history[i].z);
      END;
      close(r1);
END;

{*************************************************************}
```

hjhjhjhjhjhjhjh (3.3)