Constitutive Equations

In a Finite Element code, constitutive equations are used at each integration point to simulate material behavior. Z-mat adds capability by providing advanced constitutive equations based on the rigorous thermodynamic state variable formulation. The primary numerical task for material simulation code for FEA calculations is to provide the value of the stress tensor and the state variables at the end of a loading increment, knowing the values at the beginning of the increment and a trial strain increment. More precisely, we can define an "object" *BEHAVIOR* as the collection of :

- primal (prescribed) variable (e.g. strain) and dual (associated) variable (e.g. stress)
- set of state dependent variables **V**
- set of auxiliary variables, **A**, not necessary for the computation, but used in the post-processing
- external parameters, **P**, prescribed by the user and acting like an external load (e.g. temperature in a mechanical uncoupled calculation);
- material parameters, **M**, to be identified for each material.

The constitutive equations are normally expressed as a first order ordinary differential system (ODE), the derivative of the state variables being defined as functions of the prescribed primal variable, of **A**, **P** and **M**. The integration of this system can be made by explicit and implicit methods. Both techniques are included in Z-mat as follows :

- Runge-Kutta method with automatic time stepping
- Modified midpoint (θ-method), solved with a Newton algorithm

The explicit integration is easier to implement, and therefore is used for fast prototyping. Implicit integration however demands the definition of the local Jacobian matrix defining the behavior which is a more complex affair, but the method is more robust for large time steps, as it provides (for free!) the information needed to compute the global consistent tangent matrix, and therefore permit quadratic global convergence.

Z-mat treats all model formulations in the most general sense, employing object-oriented design to achieve a high level of flexibility. With this approach, the models are described in terms of material building "bricks," such as yield criterion, isotropic or kinematic hardening evolution, or viscoplastic flow. A model is much more powerful with this approach because all the different "brick" types can be combined by the user to effectively make new models at run-time. In fact, even the coefficients are generalized (abstracted), allowing constant values, tabular values, or c-type syntax functions.

How to use Z-mat

There are two ways of using Z-mat

User-mode

Developer-mode

As many constitutive equations are already implemented in Z-mat (see next page), most users can simply make use of those in their calculations. A small change in the ABAQUS input file is necessary to access the Z-mat library, and an external material definition file must also be created.

The input file modifications must give the material name, which will become the name of an external material definition file to control Z-mat. The number of state variables is also required - Z-mat does not use any information regarding corfficients or material parameters from the ABAQUS input. Any number of Z-mat materials is accepted in the same run.

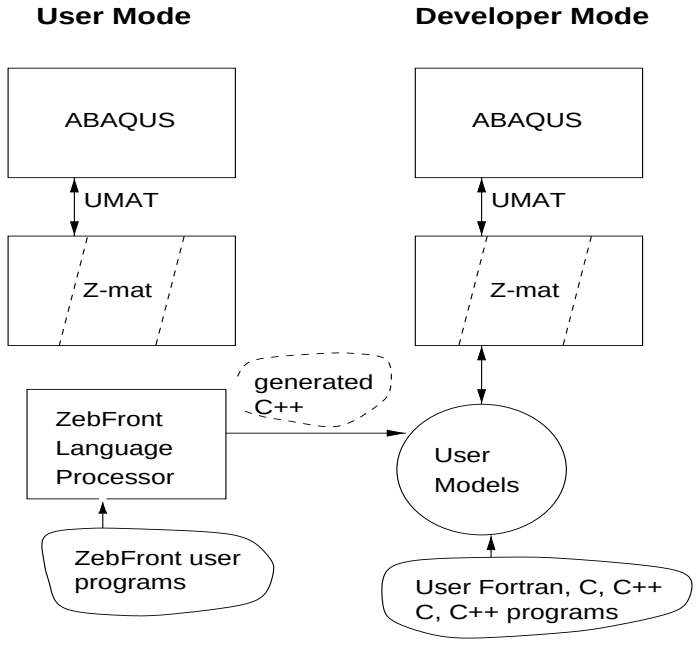


Figure 1. The two modes of using Z-mat

User Mode

Using the material laws contained in the Z-mat library involves changing the material definition in the ABAQUS .inp file, and creating a Z-mat material definition file. These Z-mat input files may be stored in a common location, allowing a site to easily manage their different material inputs. The following figure shows examples of these two files:

ABAQUS .inp file

```
*NODE, NSET=all
1,0.,0.
...
**----------------------------------
*SOLID
SECTION,ELSET=ALL,MATERIAL=steel
*MATERIAL,NAME=steel
*DEPVAR
  13
*USER MATERIAL,CONSTANTS=1
0.0
*USER SUBROUTINES,INPUT=umat.f
```

Z-mat material file

```
***material
  *integration theta_method_a 1.0 1.e-12 1500
***behavior  gen_evp
  **elasticity isotropic
     young   210000.0
     poisson  0.33
  **potential gen_evp ep
    *criterion  mises
    *flow  plasticity
    *isotropic constant
     R0 150.0
    *kinematic nonlinear
     D  69.31
     C  8317.77
***return
```

The modifications of the ABAQUS .inp file are essentially constant between different analyses, except for the number of integrated state variables (using the DEPVAR keyword). For a given Z-mat material file, the utility program Zpreload will output all the lines which are needed. Note that Z-mat does not use the abaqus input material coefficients, but this line is still needed by ABAQUS to use the UMAT routine.

The Z-mat material file customizes the way materials are entered into ABAQUS. Local convergence and automatic time step parameters can be adjusted, material rotations specified, and variables initialized. The material definitions itself is marked by the keyword ***behavior, followed by a keyword for the type of behavior, and its options. All material “building bricks” are constructed in this way, and each such “object” creation may be added to with user programs.

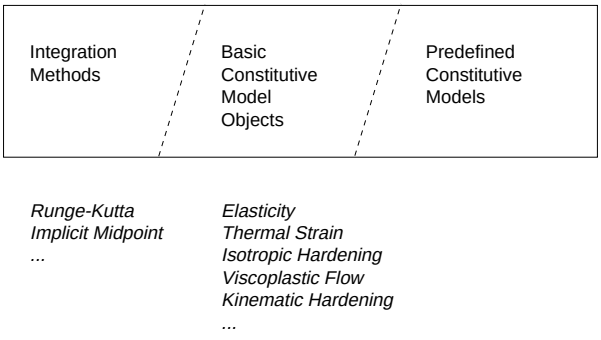
Some Bricks Available

- All coefficients are variable, allowing constant, tabular, step change, or interpreted high-level function definitions.
- Model “bricks”: viscoplastic rate laws, yield criteria, isotropic and kinematic hardenings
- Linear thermo-elasticity; isotropic, orthotropic, anisotropic, etc.
- Generalized Maxwell Viscoelasticity.
- Porous plasticity for Gurson damage or powder compaction. This model provides additional capabilities to that in ABAQUS, including other potential formulations such as Rousillier, etc.
- Complete Chaboche model with coupled damage in explicit/implicit integration. Modifications for multipotential flow (e.g. creep, viscoplastic, and time independently plasticity combined), static recovery on all hardening variables, coupling with damage.
- .No limit to the number of kinematic hardenings, interaction with criteria, state coupling of variables, isotropic evolution of saturation rate, linear-nonlinear evolution with a criterion, etc.
- MATMOD model of Miller
- Bodner-Partom viscoplastic model
- SUVIC, SUVIC-D model of rock with damage
- Micro-macro models for single and poly crystals. Polycrystal yield deforms and expands using slip-plane isotropic and kinematic hardening. Complex flow rules, Taylor and self hardening. Crystals for FCC, BCC, and HCP.
- Composite materials, including anisotropic damage with fiber closures.
- Implicit/Explicit material integration with consistent tangent matrices.

Developer

Extended use of Z-mat can be made by adding other levels in the library. In fact, the large number of predefined constitutive models represent only a part of the code — the developer also has access to:

- basic objects, which can be used in his own constitutive equations, offering the power of already tested bricks of behavior: for instance using the "elasticity" object allows the user, without any effort, to have a direct access to any kind of elasticity in the code *without any flag* in the local code : the type of elasticity is automatically determined by the code from the data file, and the corresponding behavior is then dynamically applied to the material.
- integration methods; the developer has then the choice of using the integrators only, or use basic objects in the integrators.



The developing user can program any number of additions, including multiple additions of the same “type.” For example, multiple new material behaviors can be added, along with additional isotropic hardening “bricks.” These later models can be used by the new user behavior models, and by all the standard Z-mat library models. The input interface for user “bricks” is also *exactly the same* as for the standard models supplied with Z-mat, so user extensions are seamlessly integrated into the library (in contrast to the user definition of umat in the abaqus file).

Additions may be programmed in C++, or use the “material programming” language of ZebFront. ZebFront adds special commands for declaring material model data and functions (such as scalar, vector, or tensorial state variables, and coefficients), and automatically generates material file read functions and function prototypes. The end result is a material model which can be close to a 1:1 listing of the model equations. These equations can be written in full tensor or matrix form as well, eliminating many program loops.

A typical ZebFront model includes a “class” declaration, where the model data structures are defined, a post step calculation, and either or both explicit or implicit integration implementations (the model equations). The class can be “derived” from more than one base type, which determines the functionality of the class. Often material models are derived from both BASIC_NL_BEHAVIOR and BASIC_SIMULATOR to provide FEA behavior and simulation with the same code.

To assist with the developer mode, the Z-mat library comes with a set of development tools to generate makefiles from multiple source files, link shared libraries on many platforms, and manage multiple architectures. Debugging tools are provided as well, to assist in debugging code running in ABAQUS. An example of the Z-mat developer steps is shown below:

Schematic for ZebFront Program

```
@Class ZUSER : BASIC_NL_BEHAVIOR {
  @Name  Zuser;
  @SubClass ELASTICITY E;
  @Coefs   C1, C2;
  @VarInt  evi, eel, ...;
  @VarAux X,Y;
};

@StrainPart { /* post increment calcs */
  evi = eto - eel;
  sig = *E*eel;
}

@Derivative {
  devi = ... ;
  deel = ... ;
}
```

Compilation

```
   % Zsetup

   % Zmake
```

Z-mat material file

```
%
%  In file steel  ...
%
***material
   *integration theta_method_a    1.0 1.e-9  25
***behavior   Zuser
**elasticity isotropic
  young     210.e3
  poisson     0.3
**model_coef
  C1        temperature
  100.0    23.0
   50.0   500.0
  C2  500.0
***return
```

ABAQUS Input file (from running Zpreload steel)

```
**********************************************
*SOLID SECTION, ELSET=ALL, MATERIAL=steel
*MATERIAL,NAME=steel
*USER SUBROUTINES, INPUT=umat.f
...
```

Example 1: Generalized Chaboche Model

This is an example using the generalized Chaboche model available in the Z-mat library (i.e. User Mode). A particular form of the model will be used, including thermal strain, non-linear isotropic hardening, linear and non-linear kinematic hardening, and some coefficients which vary with temperature.

The behavior model is only a framework for building up particular hardening/criterion/flow/etc combinations. The basic form of the model is stated simply below:

$$\boldsymbol{\sigma} = \mathbf{E} : \boldsymbol{\varepsilon}_{el}$$
$$\dot{\boldsymbol{\varepsilon}}_{el} + \sum \dot{\boldsymbol{\varepsilon}}_i = \dot{\boldsymbol{\varepsilon}}_{tot} + \dot{\boldsymbol{\varepsilon}}_{th}$$

where the $\dot{\boldsymbol{\varepsilon}}_i$ terms are individual components of inelastic deformations. Evolution for these deformations is determined by a particular inelastic *dissipation potential*. Several different forms of dissipation potential are available in Z-mat, to provide models including the classical Chaboche viscoplasticity, single crystal deformation, fully associated plasticity, special plasticity for ratchetting, modified static recovery, among others. For Chaboche viscoplasticity, the potential the following general equations can be written:

$$f = f(\boldsymbol{\sigma}, \mathbf{X}) - R(v) \qquad \mathbf{X} = \sum \mathbf{X}_i \qquad \mathbf{X}_i = \tfrac{2}{3} C_i \boldsymbol{\alpha}_i$$
$$\dot{\boldsymbol{\varepsilon}} = \dot{v} \frac{\partial f}{\partial \boldsymbol{\sigma}} \qquad \textit{with} \qquad \dot{v} = \dot{v}(f, v)$$

with f being the criterion, R an isotropic measure of the yield size (usually the yield point in uniaxial tension), v the cumulated viscoplastic strain equivalent, and **X** a series of kinematic translations of the yield (back stresses). The model uses material "objects" for the criterion, the rate equation for v, the equation for R, and each of the kinematic variables.

To start the material file, the ***behavior keyword must be used to indicate a generic behavior object is to be created. All such objects will have a "class" keyword (here, behavior), followed by a particular type of that class. For this model, the type is gen_evp for generalized elasto-viscoplastic. It should be noted that the asterisks in the Z-mat file indicate not only keywords, but a hierarchy of sub-commands. Thus all keywords of **-level are sub-commands of the behavior, and the next ***-level command (here ***return) terminates the behavior read. For this model, the elasticity matrix, thermal strain, and potential keywords are used.

The potential object is the most interesting one here, as it defiles all the plasticity characteristics. The potential model chosen is again named gen_evp (but could be one of many others), and the characters following are an optional *name* for the potential. Output variables and coupling use this user determined name.

criterion mises $\quad f = \left[\tfrac{3}{2}(\mathbf{s}-\mathbf{X}) : (\mathbf{s}-\mathbf{X})\right]^{\frac{1}{2}}$

flow norton $\quad \dot{v} = \left\langle f / K \right\rangle^n$

isotropic nonlinear $\quad R = R_0 + Q\left(1 - e^{-bv}\right)$

kinematic linear $\quad \dot{\boldsymbol{\alpha}}_2 = \dot{\boldsymbol{\varepsilon}}_{vp}$

kinematic nonlinear $\quad \dot{\boldsymbol{\alpha}}_2 = \dot{\boldsymbol{\varepsilon}}_{vp} - \dot{v} \frac{3}{2} \frac{D}{C} \mathbf{X}_2$

Note that the model supports any number of kinematic hardening variables, and all coefficients in the model may be variable.

```
%
% Example Chaboche material file
% with some different objects and coefficient
% input formats
%
***behavior gen_evp
 **elasticity isotropic
    young  function
       200000.-100.*temperature^2.;
    poisson   0.3
 **thermal_strain isotropic
    alpha  temperature
    0.12500E-04     0.0
    0.14500E-04   500.00
    0.14500E-04  1000.00
 **potential gen_evp   ev
   *criterion mises
   *flow  norton
     n   4.
     K  temperature
      200.    20.
      300.   650.
      250.   950.
  *isotropic nonlinear
     R0  temperature
       400.    0.
       300. 1000.
     Q   -200.
     b      4.
  *kinematic nonlinear
     D  function 800.+2.5*(temperature-475.)^2;
     C  75000.
  *kinematic linear
     C  1050.
***return
```

Example 2: User Model with Explicit Integration

For this example a user model is created using the ZebFront preprocessor. The model adds an effect in isotropic hardening to account for aging by adding a term R^* to the yield function f :

$$f = J(\boldsymbol{\sigma} - \mathbf{X}) - R - R^*$$

where R^* depends upon a new state variable a used to characterize material ageing:

$$R^* = R_0^*(1-a) \qquad \dot{a} = \frac{1}{\tau}(1-a)$$

Declaration of material parameters, state dependent, and auxiliary variables is done by means of the `@Class` keyword (lines 1 to 9 of the ZebFront model below). Compared to a standard Chaboche model, this ageing model introduces two new material parameters R_0^* and τ declared at line 3, while the additional ageing a variable is declared at line 6.

In the second block `@StrainPart` keyword at lines (11 to 15) the user has only to define how to derive stress σ from the different variables manipulated by the model. Note that high level *material objects*, such as the ELASTICITY class defined in the Z-mat library can be called directly in a user model leading to fast and easy implementation (lines 13,18).

Then, for explicit Runge-Kutta integration, differential equations governing the evolution of the state variables are defined in the last program block `@Derivative` keyword from line 17 to 36. C++ variables names such as d*varname* are automatically generated by ZebFront for each state variable *varname* declared at lines 5,6, and all data is initialized properly for the calculation.

The final implementation is very concise (only 36 lines of which one third is simply declaration) for quite a complex model. Use of high level mathematical objects such as tensors and their associated operators allows the user to write lines of code that closely look like the mathematical equations defining the model. More involved examples using implicit integration and the returning of consistent tangent matrices are dealt with in following examples. This model runs with any element formulation using the BFGS solution procedure.

An example material input file for this model is shown (below right). Note that the input syntax follows exactly the that for the "native" gen_evp behavior of the last example. All input handling is automatically generated for this example, but additional user code could be given for special situations. Any of the coefficients can be variable here of course. Thermal strain is also automatically accounted for.

```
 1  @Class AGEING : BASIC_NL_BEHAVIOR {
 2     @SubClass ELASTICITY E;
 3     @Coefs    R0, Q, b, R0_star, tau;
 4     @Coefs    K, n, C, D;
 5     @tVarInt  eel, alpha;
 6     @sVarInt  evcum, age;
 7     @tVarAux  evi, X;
 8     @sVarAux  R, R_star;
 9  };
10
11  @StrainPart {
12    evi  = eto - eel;
13    sig  = *E*eel;
14    if (m_flags&CALC_TG_MATRIX) m_tg_matrix=*E;
15  }
16
17  @Derivative {
18    sig    = *E*eel;
19    X      = (2.0*C/3.0)*alpha;
20    R      = R0 + Q*(1.-exp(-b*evcum));
21    R_star = R0_star*(1.0-age);
22
23    TENSOR2 sigeff = deviator(sig - X);
24    double  J      = sqrt(1.5*(sigeff|sigeff));
25    double  f      = J - R - R_star;
26
27    deel   = deto;
28    dage   = (1.0-age)/tau;
29
30    if (f>0.0) {
31                   devcum   = pow(f/K,n);
32       TENSOR2     norm     = sigeff*(1.5/J);
33                   deel    -= devcum*norm;
34       if (C>0.0) dalpha = devcum*(norm - D*alpha);
35    }
36  }
```

$$\boldsymbol{\sigma} = \mathbf{E} : \boldsymbol{\varepsilon}_{el}$$
$$\mathbf{X} = \tfrac{2}{3} C\boldsymbol{\alpha}$$
$$R = R_0 + Q(1 - e^{-bv})$$
$$R^* = R_0^*(1-a)$$
$$\boldsymbol{\sigma}_{eff} = \mathbf{s} - \mathbf{X}$$
$$J = \sqrt{\tfrac{3}{2}\boldsymbol{\sigma}_{eff} : \boldsymbol{\sigma}_{eff}}$$
$$f = J - R - R^*$$

$$\dot{a} = (1-a)/\tau$$
$$\dot{v} = \left\langle f \middle/ K \right\rangle^n$$
$$\mathbf{n} = \frac{\partial f}{\partial \boldsymbol{\sigma}} = \frac{3}{2}\frac{\boldsymbol{\sigma}_{eff}}{J}$$
$$\dot{\boldsymbol{\varepsilon}}_{vp} = \dot{v}\mathbf{n}$$
$$\dot{\boldsymbol{\alpha}}_{vp} = \dot{v}\,[\mathbf{n} - D\boldsymbol{\alpha}]$$

```
***behavior ageing
 **E isotropic
    young  80000.
    poisson 0.3
**model_coef
    K  1200.0
    n   3.0
    C 20995.0
    D  1105.0
    R0  400.0
    Q   0.
    b   1.
    tau 2000.
    R0_star -200.
***return
```

Example 3: Implicit Integration with Tangent

The last example used the explicit Runge-Kutta integration scheme. Explicit integration is easier to write, as only the rate equations are needed, but is less accurate for large time steps and does not provide a good tangent matrix for the global solution proceedure. The real benefit of explicit integration is that it allows "rapid prototyping" of material models. This section in contrast illustrates the implementation with a modified midpoint implicit method (θ-method) which yields an efficient final implementation. In place of the `@Derivative` subroutine, one has to specify the residual and the Jacobian matrix by means of the `@CalcGradF` subroutine, as shown below for a simple Norton creep model.

Over each time increment Δt the constitutive equation for this model is written in terms of assumed increments of the state variables. For a given set of such increments, the following *residual* is written:

$$\mathbf{F}_{el} = \Delta\boldsymbol{\varepsilon}_{el} + \Delta v\,\mathbf{n} - (\Delta\boldsymbol{\varepsilon}_{tot} + \Delta\boldsymbol{\varepsilon}_{th}) \qquad F_v = \Delta v - \Delta t\left\langle \frac{f}{K}\right\rangle^n$$

where the tensorial elastic deformation ε_{el} and the viscoplastic cumulated deformation *v* are state dependent variables declared respectively as `eel` and `evcum` in the definition of the ZebFront code below (at lines 5 and 6). Upon convergence, this residual will be close (by an adjustable limit) to zero. The corresponding residual is defined at lines 32-35 of the ZebFront code (the F vectors are initialized to the variable increment, and all diagonal terms of the Jacobian are already initialized to unity).

As this set of equations will be solved by a Newton-Raphson algorithm to find the increment of state dependent variables $\Delta\varepsilon_{el}$ and Δv one must also specify the partial derivatives of the residual in terms of the each variable. C++ variables are automatically generated with names f_vec_*varname* for the residual and d*varnamei*_d*varnamej* for the partial derivatives based on the declaration of a *varname* state dependent variable (`@VarInt` declaration). The Jacobian matrix is then automatically assembled to form the system of equations solved at each Newton-Raphson iteration:

$$\begin{Bmatrix}\mathbf{R}_{el}\\ R_v\end{Bmatrix} = \begin{bmatrix}\partial\mathbf{R}_{el}/\partial\Delta\boldsymbol{\varepsilon}_{el} & \partial\mathbf{R}_{el}/\partial\Delta v\\ \partial R_v/\partial\Delta\boldsymbol{\varepsilon}_{el} & 1\end{bmatrix}\begin{Bmatrix}\Delta\boldsymbol{\varepsilon}_{el}\\ \Delta v\end{Bmatrix}$$

Note that the 1,1 component of the inverted Jacobian can be used to find the tangent matrix:

$$\frac{\partial\Delta\boldsymbol{\varepsilon}_{el}}{\partial\Delta\boldsymbol{\varepsilon}_{tot}} = \left[\nabla F^{-1}\right]_{el}$$

$$\frac{\partial\Delta\boldsymbol{\sigma}}{\partial\Delta\boldsymbol{\varepsilon}_{tot}} = \frac{\partial\boldsymbol{\sigma}}{\partial\boldsymbol{\varepsilon}_{el}}\left[\nabla F^{-1}\right]_{el} = \mathbf{E}\left[\nabla F^{-1}\right]_{el}$$

$$\boldsymbol{\sigma} = \mathbf{E} : \boldsymbol{\varepsilon}_{el}$$

$$\boldsymbol{\sigma}_{eff} = \mathbf{s} - \mathbf{X}$$

$$J = \sqrt{\tfrac{3}{2}\boldsymbol{\sigma}_{eff} : \boldsymbol{\sigma}_{eff}}$$

$$\mathbf{n} = \frac{\partial f}{\partial\boldsymbol{\sigma}} = \frac{3}{2}\frac{\boldsymbol{\sigma}_{eff}}{J}$$

$$F_{el} = \Delta\boldsymbol{\varepsilon}_{el} + \Delta v\,\mathbf{n} - (\Delta\boldsymbol{\varepsilon}_{tot} + \Delta\boldsymbol{\varepsilon}_{th})$$

$$F_v = \Delta v - \Delta t\left\langle J/K\right\rangle^n$$

$$\theta\Delta v\frac{\partial\mathbf{n}}{\partial\boldsymbol{\sigma}} = \theta\Delta v\frac{3}{2J}\left[\mathbf{1} - \mathbf{n}\otimes\mathbf{n}\right]$$

$$\frac{\partial F_{el}}{\partial\Delta\boldsymbol{\varepsilon}_{el}} = \mathbf{1} + \theta\Delta v\frac{\partial\mathbf{n}}{\partial\boldsymbol{\sigma}}\frac{\partial\boldsymbol{\sigma}}{\partial\Delta\boldsymbol{\varepsilon}_{el}} = \mathbf{1} + \theta\Delta v\frac{\partial\mathbf{n}}{\partial\boldsymbol{\sigma}}\mathbf{E}$$

$$\frac{\partial F_{el}}{\partial\Delta v} = \mathbf{n}$$

$$\frac{\partial F_v}{\partial\Delta\boldsymbol{\varepsilon}_{el}} = \frac{\theta\Delta tn}{K}\left\langle\frac{f}{K}\right\rangle^{n-1}\mathbf{nE}$$

```
1  @Class NORTON_BEHAVIOR : BASIC_NL_BEHAVIOR {
2     @Name norton;
3     @SubClass ELASTICITY E;
4     @Coefs    K, n;
5     @tVarInt  eel;
6     @sVarInt  evcum;
7     @Implicit
8  };
9
10 @StrainPart {
11   sig = *E*eel;
12   SMATRIX tmp(psz,f_grad,0,0);
13   if (Dtime>0.0) m_tg_matrix=*E*tmp;
14   else           m_tg_matrix=*E;
15 }
16
17 @CalcGradF {
18   sig = *E*eel;
19   f_vec_eel -= deto;
20
21   TENSOR2 sigeff = deviator(sig);
22   double  J      = sqrt(1.5*(sigeff|sigeff));
23
24   if (J>0.0) {
25      TENSOR2        norm            = sigeff*(1.5/J);
26                     f_vec_eel      += norm*devcum;
27                     f_vec_evcum    -= dt*pow(J/K,n);
28
29      SMATRIX dn_ds = theta*devcum*(unit32 - (norm^norm)/J;
30      TENSOR2 df_fs   = tdt*n*pow(J/K,n-1)/K *norm;
31
32      deel_deel         += dn_ds*(*E);
33      deel_devcum       += norm;
34
35      devcum_deel       -= df_fs*(*E);
36   }
37 }
```

A Small Application

This simple example illustrate the influence of constitutive equations on structural results, and provide an idea of the CPU time spent for such nonlinear material computations. This section shows finite element simulation for a simplified 3D structure under cyclic thermomechanical loading, which mimics the critical part of a combustion chamber for a car engine. The interest of having a good description of the material is demonstrated, and comparison between the CPU times obtained for the various models is made.

The finite element model is a three-dimensional solid (continuum) involving C3D20 and C3D15 type elements with full integration. The mesh consists of 531 elements and 2985 nodes. The lateral faces are fixed in x and y direction, so that thermomechanical stresses are produced when a non homogeneous temperature field is applied (fig.2a). Figure 2b shows the temperature cycles applied during analysis.

The component is made of an aluminium alloy. It will be modeled successively by three types of constitutive equations :

- time independent plasticity;
- viscoplasticity;
- viscoplasticity with ageing.

The material dependent parameters will be given in the material files used by Z-mat.

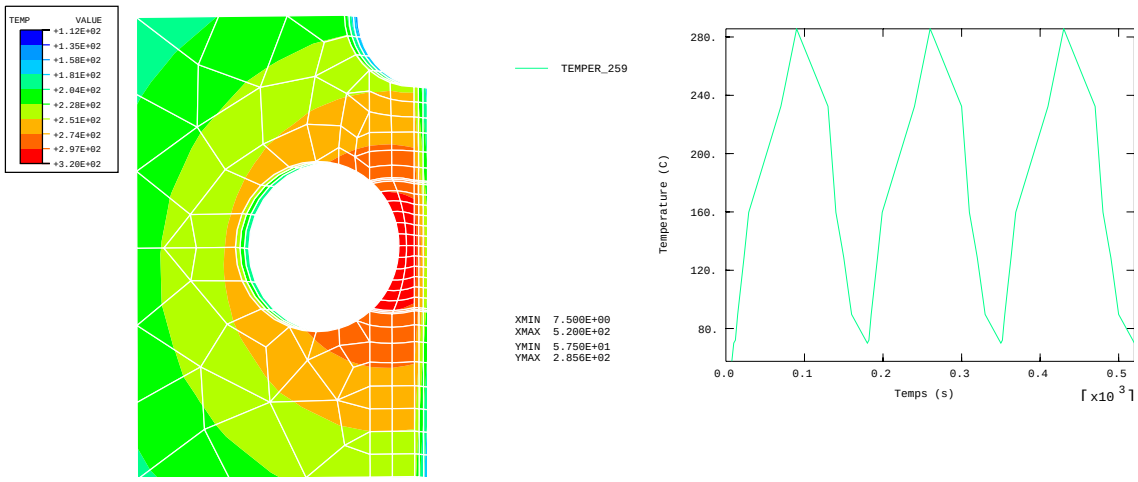


Figure 2. a) Temperature contours. b) Thermal cycle.

Results

The results of calculations are shown for the time corresponding to the maximum of temperature. The results of elastic material are discussed first. The contours of the von Mises equivalent stress are given in figure 3a.

When a nonlinear behavior is used, stress redistribution occurs in the component. The first model used is a Chaboche's model with non linear kinematic hardening, as described by Abaqus option *PLASTIC, HARDENING=COMBINED.

The results are illustrated in the curve on the next page, and figure 3b which shows the equivalent plastic strain. The curve on the next page (bottom) is the evolution of the diagonal components of the stress and strain tensor of the material in the critical zone. The maximum tension stress is equal to 232 MPa, while the compression stress is equal to -219 MPa.

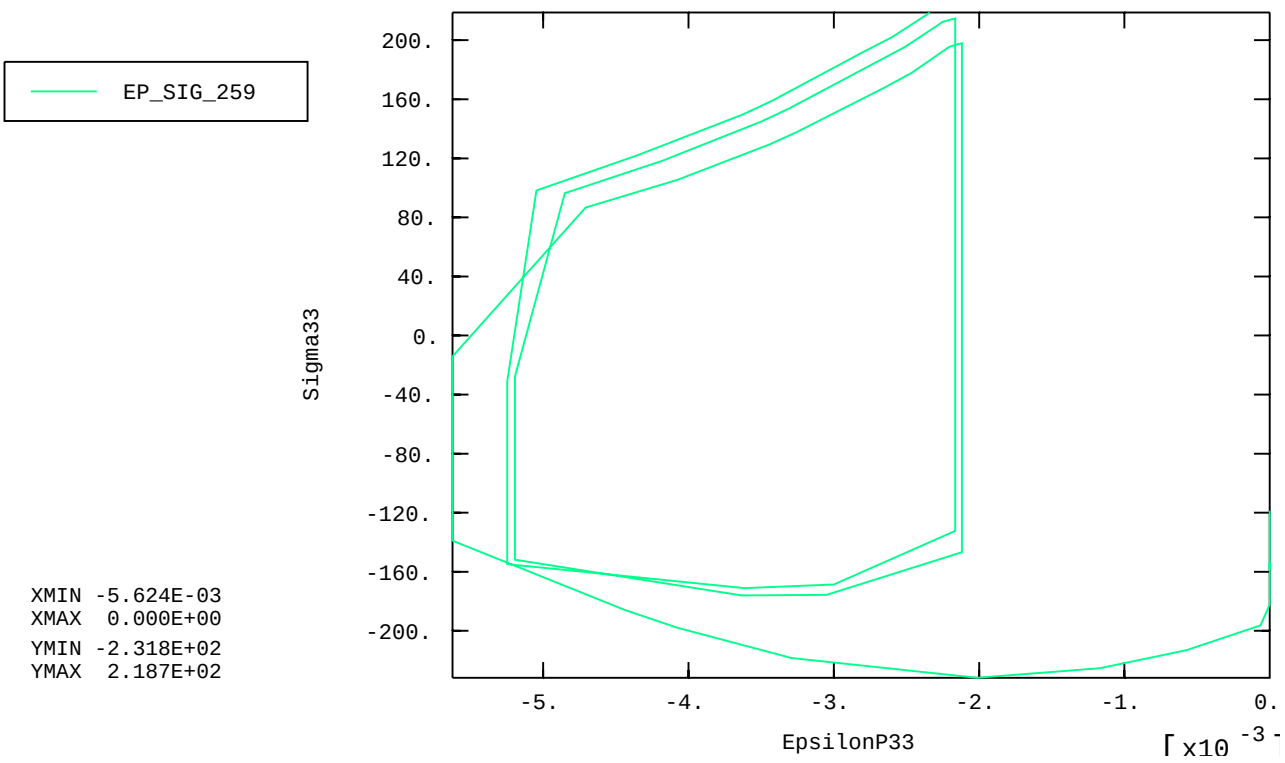
Time independent plasticity is generally not sufficient to have a good evaluation of the stress state in the component, due to i) the influence of creep during the hold time, ii) ageing of the material at high temperature. In order to take into account the time dependence of material behavior, the previous model version is transformed into a viscoplastic framework. This can be made with only a slight change in the material file, in Z-mat's syntax, as shown at top,right of the next page.

** ELASTOPLASTIC MODEL

```
***behavior  gen_evp
**elasticity isotropic
     young    , temperature
     76000.0     20.0
     72000.0     100.0
     60000.0     300.0
     poisson  0.33
  **potential gen_evp ep
   *criterion  mises
   *flow  plasticity
   *isotropic constant
     R0 , temperature
      160.0   20.0
      143.0  100.0
      122.0  200.0
       74.   300.0
   *kinematic nonlinear
       D  ,temperature
       155. 20.0
       155. 100.
       155. 300.
       C   , temperature
    21700.  20.
    22000.  100.
     8460.  200.
     3780.  300.
***return
...
```

** VISCOPLASTIC MODEL

```
***behavior  gen_evp
**elasticity isotropic
     young    , temperature
     76000.0     20.0
     72000.0     100.0
     60000.0     300.0
     poisson  0.33
**potential gen_evp ep
 *criterion  mises
 *flow  norton
      n 8.5
      K  temperature
       250. 20.
       256.5 200.
       226.5 300.
  *isotropic constant
    R0   temperature
     80.0   20.0
     63.0  100.0
     52.0  200.0
      4.   300.0
  *kinematic nonlinear
      D   temperature
      155. 20.0
      200.  100.0
      420.  300.
      C temperature
   21700.  20.
   22000.  100.
   8460.   200.
   3780.   300.
 ***return
```



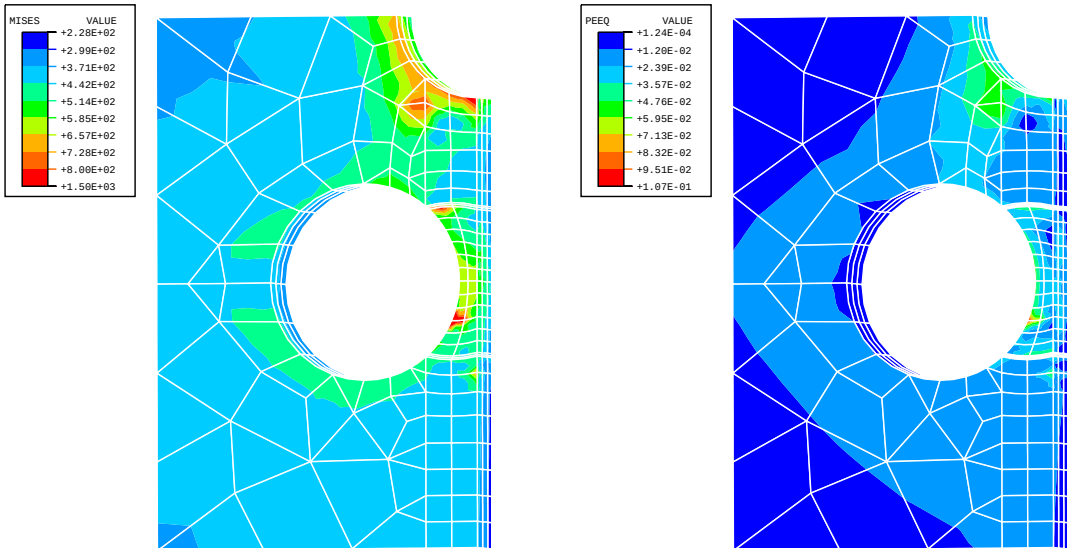


Figure 3. a) Contours of von Mises stress. b) Contour of equivalent plastic strain magnitude.

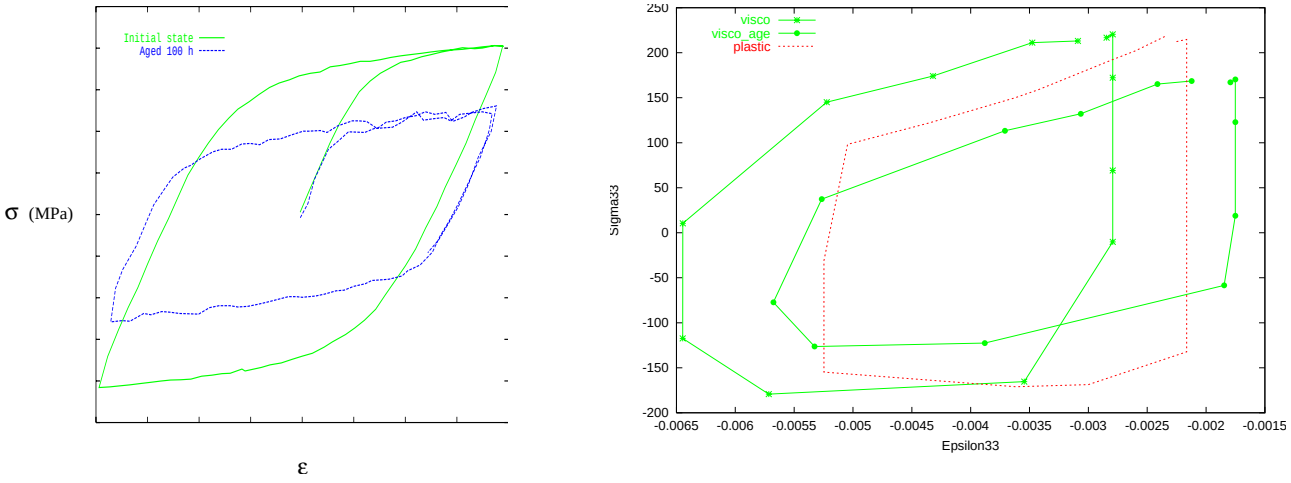


Figure 4a (above left) shows the effects of ageing on the material hardening. It is observed that is influences the isotropic hardening. After one hundred hours ageing period, a progressive softening can be seen after ageing. The full numerical implementation of the related model was shown as example 2 on page 5.

The number of increments needed for performing one cycle with each model type is given in in the table at right, together with the total CPU time. The Chaboche plastic model in Z-mat takes 15% more time than for Abaqus native computations. That has to be related with the additional exchanges generated by `umat`'s use. Viscoplastic models require more CPU time, since the yield limit is lower in that case, so that the number of nonlinear increments is bigger. The time needed remains nevertheless quite reasonable (2.2 factor).

Type of Calculation	Steps	Total iters	total CPU	Ratio
Elastic	1 Inc	1	20	
Plastic-Abaqus	12 Inc	43	787	1.0
Plastic Z-mat	12 Inc	26	904	1.15
Viscoplastic Z-mat	12 Inc	36	1751	2.22
Visco-aging	12 Inc	36	1737.7	2.20