Preparing input for INCREMENTALDRIVER

A.Niemunis

22.10.2019

1 Introduction

The INCREMENTALDRIVER is a small Fortran 90 program for numerical simulations of *element tests*. A constitutive routine `umat` (= user's material) is driven by the INCREMENTALDRIVER in a series of time increments following a desired loading path. `Umat` is an independent piece of software and not a part[1] of the INCREMENTALDRIVER. `Umat` should be available as a source code[2]. Different `umats` can be tested upon a prescribed path and the results can be compared with the laboratory data. The INCREMENTALDRIVER can also be helpful in calibration of material constants. `Umat` handles a single strain increment only and hence one needs the INCREMENTALDRIVER to calculate the whole path. Given the current state of a material $\mathbf{T}^t$ and $\texttt{statev}^{\,t}$ and a strain increment $\Delta\boldsymbol{\epsilon}$ `umat` calculates $\mathbf{T}^{t+\Delta t}$ and $\texttt{statev}^{\,t+\Delta t}$ at the end of the increment. This is called an update after an increment.

The INCREMENTALDRIVER itself is freely available as a Fortran 90 source code and the current version can be downloaded from http://www.pg.gda.pl/ aniem/dyd.html . Several source-form `umat`s are available at https://web.natur.cuni.cz/uhigug/masin/umat/. An unrestricted usage of *debuggers* with the INCREMENTALDRIVER and with `umat` is of great advantage, especially for testing of new `umat` routines. They are seldom free of errors. `Umat` for the INCREMENTALDRIVER should be written strictly according to the ABAQUS rules described in ABAQUS *User Subroutines Reference Manual* 1.1.40.

The update corresponds to the implicit time integration. It is performed by `umat` and the data is stored by ABAQUS[3]. Apart from the update, `umat` returns the *Jacobian matrix* $(\partial\mathbf{T}/\partial\boldsymbol{\epsilon})$ further called the stiffness. This stiffness is required in the equilibrium iteration of the FEM. The INCREMENTALDRIVER also needs the stiffness, whenever a stress path or a mixed path is prescribed, see Section 4. In the simplest case of a prescribed strain path (all $\Delta\epsilon_{ij}$ components are known) the INCREMENTALDRIVER invokes `umat` with $\Delta\epsilon_{ij}$ and stores the stress path as a material response. However, when following a prescribed *stress path* $\mathbf{T}(t)$ or a *mixed path*, the INCREMENTALDRIVER must invoke `umat` iteratively (several times within a single increment) and the stiffness is necessary for the corrections of strain increment. This procedure is similar to the equilibrium iteration in the FEM: strain increments are adjusted to satisfy conditions formulated in terms of stress. The prescribed components of strain and stress increment are termed the *load* here. The calculated (remaining) components are called the material *response*. The increments of `statev` are always passive, i.e. they are treated as the material *response*.

1.1 A typical simulation of an element test

The INCREMENTALDRIVER is a FORTRAN 90 project which consists of the following files to be compiled and linked

```
incrementalDriver.f
aba_param.inc
umat.f & sub-files if any
```

$\xrightarrow{\textit{compile}}$

```
incrementalDriver.exe
```

After the compilation the resulting program `incrementalDriver.exe` can be executed, usually in DOS window. The program needs three input files and it produces a single output file. The name of the output file should be specified in the input file `test.inp`

[1]A simple linear elastic constitutive model is delivered with the INCREMENTALDRIVER as an example in a separate file `umat.f`. This file should be replaced by a 'serious' constitutive routine, of course.

[2]`Umat` should be written as a Fortran 90 source code to be easily compiled and linked with the INCREMENTALDRIVER and with ABAQUS. Other languages and semi-compiled forms are also possible, see Appendix.

[3]ABAQUS Standard requires implicit integration, contrarily to ABAQUS Explicit.

$$\begin{tikzcd}\boxed{\begin{array}{l}\texttt{initialConditions.inp}\\ \texttt{parameters.inp}\\ \texttt{test.inp}\end{array}} \arrow[r, "incrementalDriver.exe"] & \boxed{\texttt{outputfile.out}}\end{tikzcd}$$

The above default file names can be overridden via the command-line parameters. The following example keeps the default names of input but redirects the output to the file `x.out`:

```
incrementalDriver test=test.inp param=parameters.inp ini=initialconditions.inp out=x.out verbose=true
```

`verbose=false` restricts the screen output during calculation. All command-line parameters of `incrementalDriver` are optional. They may appear in any order. Spaces are treated as separators and should not appear at =.

1.2 General remarks on preparing the input files

For all input files: `test.inp`, `parameters.inp` and `initialconditions.inp` the following rules apply:

- No empty lines and no comment lines;
- The end-of-line comments should start with `#`. Insert some spaces (say 10) between input strings and `#`.
- Input integers must not have a decimal point;
- Strings in input must not have apostrophes. They should start from the first column (no preceding spaces).
- The asterisk is the first character of each keyword. The capital letters are important in the keywords.
- Only spaces can be used as separators of data in a single line (no commas or semicolons)

2 Input file with material constants (`parameters.inp`)

```
cmname
nprops
props(1)
props(2)
...
props(nprops)
```

The initial conditions are usually read from the file `parameters.inp`. This name can be overridden in the command line as described in Section 1.1. In `parameters.inp` you specify the material name `cmname` (only one material can be used) and the number `nprops` of material constants. These two lines should be followed by a list of constants `props(1), props(2)` .... The meaning of `props` depends on their internal interpretation in `umat`. Input just one material constant per line. The end-of-line comments after the data are allowed but note that `cmname` is `character(len=80)`.

3 Input file that describes initial conditions (`initialconditions.inp`)

```
ntens
stress(1)
stress(2)
stress(3)
...
stress(ntens)
nstatv
statev(1)
statev(2)
...
statev(nstatv)
```

The initial conditions are usually read from the file `initialconditions.inp`. This name can be overridden in the command line, see Sec. 1.1.

a. specify first `ntens` components of stress, usually `ntens=6` and `stress(1...6)` are : $T_{11}, T_{22}, T_{33}, T_{12}, T_{13}, T_{23}$

b. The initial stress is always defined with the Cartesian components. As yet, you cannot input initial Roscoe invariants instead.

c. If the end of file (EOF) is encountered while reading `statev( )` then the remaining components of `statev( )` will be padded with zeros.

4 Input file with prescribed path (`test.inp`)

The first line of `test.inp` contains the name `character(len=260)` of the output file.

```
outputFileName   #  optional heading  which will be copied to the outputFile
```

If the character # is encoutered then the portion before # is interpreted as the name of the output file and the portion after # is copied into the output file as heading[4]. The output file name is obligatory and there is no default. The output file name read from `test.inp` can be overridden by `anyName.out` in the command line if the argument `out=anyName.out` appears in the command line invoking `incrementalDriver`. This may be useful for writing scripts. The heading (i.e. the text after #) cannot be overridden.

The subsequent commands in `test.inp` describe sequences if increments that follow the desired path, i.e. each command followed by parameters describes a steps (understood as in ABAQUS) usually consisting of many increments.

4.1 Description of a step

Each step begins with a keyword like `*LinearLoad`, `*Deformationgradient`, `*ImportFile` etc. The second line is common for all steps. It contains the number of increments `ninc`, the desired number of equilibrium iterations `maxiter` and the total time interval of the step (the time increment is `deltaTime / ninc`). If you calculate thousands of increments you can optionally suppress the output using the integer parameter `every`, after the colon. For example

```
*LinearLoad
10000   10    500.0  :  100
...
```

applies 10000 increments with 10 iterations with the time increment of 0.05 and every 100th state will be written to the output file. If the colon is absent then `every = 1` is assumed, i.e. time, stress, strain and all state variables will be printed after each increment.

The third line describes the type of components and can be `*Cartesian`, `*Roscoe`, `RoscoeIsomorph` etc. The next six lines prescribe the components of stress or strain incremens. In very popular steps like `TriaxialE1` or `OedometricS1` the type of components is not used and the description of loading is abbreviated.

4.2 Proportional stress/strain paths

During a proportional loading all `ninc` increments within the step are identical. In `test.inp` such linear loading is prescribed by command `*LinearLoad` followed by some data and some options

```
*LinearLoad
ninc  maxiter  deltaTime  :  every  !   number of increments, max. number of equil. iterations  and  step time interval
*Cartesian                        !   other possibilities here are:  *Roscoe, *RoscoeIsomorph, *Rendulic
ifstress(1)   deltaLoad(1)  !   0/1 Flag (0=strain 1=stress) and  the change of the 1st component in the step
ifstress(2)   deltaLoad(2)
...
ifstress(6)   deltaLoad(6)   !   here Delta T23 or Delta gamma23
```

The prescribed components `deltaLoad` of "generalized load" and the time interval `deltaTime` pertain to the whole *step* and will be apportioned to *individual increments*, usually dividing `deltaLoad` by `ninc`. Depending on the flag `ifstress(i)` the respective (i-th) component of prescribed strain or stress increment is prescribed as `ddstress(i) = deltaLoad/ninc` or strain increment `dstran(i) = deltaLoad/ninc`.

The required number of iterations `maxiter` is defined a priori and kept constant for all increments in a given step. The time interval for a given step may be of importance for example in rate dependent constitutive models. The load increments ΔT_{ij} or $\Delta\epsilon_{ij}$ may be defined using different components of stress and strain (work - conjugated)

- `*Cartesian` $T_{11}, T_{22}, T_{33}, T_{12}, T_{13}, T_{23}$ and $\epsilon_{11}, \epsilon_{22}, \epsilon_{33}, \gamma_{12}, \gamma_{13}, \gamma_{23}$
- `*Roscoe` $p, q, z, T_{12}, T_{13}, T_{23}$ and $\epsilon_v, \epsilon_q, \epsilon_z, \gamma_{12}, \gamma_{13}, \gamma_{23}$
- `*RoscoeIsomorph` $P, Q, Z, T_{12}, T_{13}, T_{23}$ and $\epsilon_P, \epsilon_Q, \epsilon_Z, \gamma_{12}, \gamma_{13}, \gamma_{23}$
- `*Rendulic` $T_{11}, \sqrt{2}T_{22}, Z, T_{12}, T_{13}, T_{23}$ and $\epsilon_{11}, \sqrt{2}\epsilon_{22}, \epsilon_Z, \gamma_{12}, \gamma_{13}, \gamma_{23}$

with:

$$p = -(T_{11}+T_{22}+T_{33})/3, \quad q = -(T_{11} - \tfrac{1}{2}T_{22} - \tfrac{1}{2}T_{33}), \quad z = -(T_{22}-T_{33})$$
$$\epsilon_v = -(\epsilon_{11}+\epsilon_{22}+\epsilon_{33}), \quad \epsilon_q = -\tfrac{2}{3}(\epsilon_{11} - \tfrac{1}{2}\epsilon_{22} - \tfrac{1}{2}\epsilon_{33}), \quad \epsilon_z = -(\epsilon_{22}-\epsilon_{33})/2,$$
$$P = -(T_{11}+T_{22}+T_{33})/\sqrt{3}\ , \quad Q = -\sqrt{\tfrac{2}{3}}(T_{11} - \tfrac{1}{2}T_{22} - \tfrac{1}{2}T_{33}), \quad Z = -(T_{22}-T_{33})/\sqrt{2},$$
$$\epsilon_P = -(\epsilon_{11}+\epsilon_{22}+\epsilon_{33})/\sqrt{3}, \quad \epsilon_Q = -\sqrt{\tfrac{2}{3}}(\epsilon_{11} - \tfrac{1}{2}\epsilon_{22} - \tfrac{1}{2}\epsilon_{33}), \quad \epsilon_Z = -(\epsilon_{22}-\epsilon_{33})/\sqrt{2}$$

[4] For EASYPLOT one can use `output.ep #  /td "nnnnnnnnyxnnnnn"` in order to indicate which column should be plotted

The change within the step of the Hencky strain $\boldsymbol{\epsilon} = \ln \mathbf{U}$ is applied in `ninc` equal increments $\Delta\boldsymbol{\epsilon}$. In a 1D we would have $\epsilon = \ln \frac{H_n}{H_0} = \sum_{i=1}^{n} \ln \frac{H_i}{H_{i-1}}$ so equal strain increments $\ln \frac{H_i}{H_{i-1}} = \text{const}$ do not imply equal increments of displacement.

In the current version of the program there is *no convergence criterion*. The fixed number `maxiter` of equilibrium iterations is carried out irrespectively of the convergence. After the last equilibrium iteration the next load increment is applied, no matter whether the previous iteration was successful (acceptably small out-of-balance stress) or not. Hence, we should check whether the desired stress path is equal to the one actually admitted by the material.

The loading prescribed in `test.inp` may demand a stress increment which cannot be achieved by the material (e.g. going outside the yield surface despite softening). In this case the equilibrium iteration in the IncrementalDriver cannot achieve the desired stress but the calculation is not interrupted. Note that the insufficient accuracy problem within the stress controlled regime may also be caused by too small number `maxiter` of iterations. The number of equilibrium iterations should be increased in the case of an inconsistent Jacobian matrix in `umat` or when large increments are applied.

Equal increments of displacement can be also imposed by `*DeformationGradient`, for example:

```
*DeformationGradient
ninc maxiter deltaTime :every ! number of inc, max. number of EI, step time interval and suppressed output
deltaLoad(1)               ! ifstress flag is automatically 0 (=strain-like) and the axes are  *Cartesian
deltaLoad(2)
...
deltaLoad(9)
```

This example defines nine components of the deformation gradient $\{F_{11}, F_{22}, F_{33}F_{12}, F_{21}, F_{13}, F_{31}, F_{23}, F_{32}\}$ between the the beginning of the step and the end of the step. The `ninc` equal increments are calculated with $\Delta\mathbf{F} = \frac{1}{n}(\mathbf{F} - \mathbf{1})$ They all take the configuration at the beginning of the step as the reference.

4.3 Import of a loading path from an external file (new in 2016)

Many laboratory element test results are available in the form of text files in a tabular form, say a file `followMe.inp`. Let us assume that such file consists of chronologically measured states written in individual records (lines), e.g. www.torsten-wichtmann.de and that the columns correspond to the individual components of stress or strain or to different state variables.

In a `test.inp` we could define a loading step of say 100 equal increments taking the end-values minus the start-values of a step from `followMe.inp` and dividing these differences by 100. Such "manual definition" of a step assumes proportionality. If the actual increments (differences between the subsequent records) from `followMe.inp` are not proportional we may obtain discrepancies. Hence, the best method to deal with a complicated loading programme (available in a form of a data file) is to tell the IncrementalDriver to read lines of `followMe.inp` directly. The lines of `followMe.inp` will be read one by one and the increments will be calculated as differences between the subsequent lines.

Of course, we must specify how to interpret the columns and which ones should be used to define the prescribed path. For all this we use a special `step` syntax called `*ImportFile`. First, the IncrementalDriver needs the name of the file, here `followMe.inp`. The description of the step begins with the line `ninc maxiter deltaTime : every` common for all steps. In the next line we define the coordinate system, here `*Cartesian`. In the next six records we must specify six columns from the file `followMe.inp`, where the prescribed loading path can be read. Zero column number has a special meaning. In such case zero *as a value* will be assigned to the corresponding increment.

```
*ImportFile   followMe.inp  |  ncols    ! name of file and number of reals to be input per line
ninc  maxiter  deltaTime : every  ! number of incr., max. number of  EI and step time and suppressed output
*Cartesian                  ! obligatory
ifstress(1)  column(1)      !  (0=strain 1=stress) & column in the file.  column==0 means  no increment
ifstress(2)  column(2)    *  ImportFactor(2)    ! multiplier for input increments
...
ifstress(6)  column(6)
columnWithTime              ! only if deltaTime < 0 in the second line
```

Non-numeric lines[5] (for example a description of test) are allowed in the heading of `followMe.inp` only. They will be ignored by the IncrementalDriver there. Elsewhere, non-numeric characters may cause errors.

The values read from the specified column of `followMe.inp` can be multiplied by a numerical `ImportFactor`. It can be optionally defined after the number of column and after asterisk `*`. The `ImportFactor` can be useful, if the strains imported from `followMe.inp` are in [%].

[5]Lines starting (after spaces or tabs) with a character different than one from this list: `1234567890+-`.

In order to import strain increments $\Delta\epsilon_{22}$ stored as ϵ_{22} [%] in the 5-th column of `followMe.inp` we write in the 2-nd record (after `*Cartesian`) as `0 5 * 0.01`, where `0` indicates the strain-type of loading, here ϵ_{22}, `5` denotes the 5th column of `followMe.inp` and `* 0.01` is the multiplier to convert [%] to [-] before writing the increments $\epsilon_{22}(t_{n+1}) - \epsilon_{22}(t_n)$ to `deltaLoad(2)`. The line `1 4 * -1` one converts the geotechnical stress from the 4th column into the mechanical stress[6]. Even the `columnWithTime` can be scaled[7] with an `ImportFactor`.

You must always give the name of the external file followed by the number `|` `ncols`, of columns be be read. It must be known prior to importing a file. The IncrementalDriver assumes that each line of the imported line consists of exactly `ncols` real numbers `real(8)`. The number `ncols` satisfies following inequality: max(`column(1...6)` , `columnWithTime`) $\leq$ `ncols` $\leq$ number of data in each line of the file. Be sure that each data line in `followMe.inp` has at least `ncols` numerical items to be input with `read(1 ,*) real(8), real(8) ....`

If the number of increments `ninc` is smaller than the number of lines in the `followMe.inp` file then exactly `ninc` increments from the file will be executed. However, if `ninc` is larger than the length of `followMe.inp` then the end-of-file will be encountered by the IncrementalDriver while reading the file. It does not cause an error, so counting lines in `followMe.inp` is not necessary if `ninc` is sufficiently large. The IncrementalDriver simply closes the external file `followMe.inp`, ends the step `*ImportFile` and continues with the next step from the main file `test.inp`.

If `deltaTime` is positive in the second line of `*ImportFile` then the time increment will be `deltaTime` in all increments. If `deltaTime` is negative you must specify the `columnWithTime`, i.e. the position of the time column in `followMe.inp`. The time will be read as a `real(8)` number in seconds. Timestamps like 2:30:59 will not be recognized. You must not write `columnWithTime` if the earlier specified `deltaTime` is positive (no blank line , no zero value). There is no need to start the column `columnWithTime` with a zero value.

4.3.1 Comparison of simulation with the laboratory results

Developing or calibrating a constitutive model we are often given a laboratory data file, say the file `followMe.inp`. We compare the stress path $\boldsymbol{\sigma}^L(t)$ from the laboratory with the stress path $\boldsymbol{\sigma}^S(t)$ from the numerical simulation obtained using the strain path read from `followMe.inp`. An automatic comparison and evaluation of discrepancies $\boldsymbol{\sigma}^S(t) - \boldsymbol{\sigma}^L(t)$ is performed independently for each step. In order to eliminate the discrepancies inherited from the previous history we may want to start a new step from the perfect stress i.e. we bring the values of all stress components to the values from the laboratory before the the first increment of a new step available from `followMe.inp`. This somewhat artificial update of the calculated stresses consists in setting them to the laboratory values from `followMe.inp`. It is performed after a step is completed. The respective end-of-step records in `followMe.inp`, the so-called reversals, are detected by a separate programme. We must indicate the numbers of records at which such *alignment* operation should be performed. This is done via `followMe.rev`. If the file `followMe.rev` does not exist then simply no *alignment* is performed. The file `followMe.rev` contains

```
kblank, nrec, kReversal, ncol
irecRev1,  irecRev2, ....
sig11col, sig22col, sig33col, sig12col,sig13col,sig23col
sig11Ifac , sig22Ifac, sig33Ifac, sig12Ifac,sig13Ifac,sig23Ifac
eps11col, eps22col, eps33col, eps12col, eps13col,eps23col
eps11Ifac , eps22Ifac, eps33Ifac, eps12Ifac,eps13Ifac,eps23Ifac
```

These variables help reading the laboratory file `followMe.inp`. They denote

1 `kblank` number of empty (or non-numerical) lines in the heading of the file `followMe.inp`
`nrec` number of records (numerical lines) in the heading of the file `followMe.inp`
`kReversal` number of reversals
`ncol` number of columns in `followMe.inp`

2 `irecRev1, irecRev2, ....` list of reversals (up to 100, so this line can be very long)

3 `sig11col, sig22col,...` a list of six columns in `followMe.inp` with stress components (zero = no value)

4 `sig11Ifac , sig22Ifac,...` six multipliers rendering the column values to be stress components for Abaqus.

5 `eps11col, eps22col,...` a list of six columns in `followMe.inp` with strain components (zero = no value)

6 `eps11Ifac , eps22Ifac,...` six multipliers rendering the column values to be strain components for Abaqus.

The file `followMe.rev` can be written by hand or produced semi-automatically from `followMe.inp` using the program REVERSALS.EXE, e.g.

[6]The expression `1 2 * (-1)` will not work , although it is mathematically correct, because the parser in the IncrementalDriver cannot recognize brackets.
[7]Useful to change hours into seconds.

```
reversals.exe dataFile=followMe.dat ncols=9 e1=9*-0.01 e2=9*0.005 e3=9*0.005  s1=5*-1 s2=6*-1.0 s3=6*-1
              Lchako=0.00001 cosThmax=0.7
```

The REVERSALS.EXE can recognize the reversals and kinks in the data and write out the respective records. In some cases this separation of the test data into steps is not trivial due to the noise in the measurement and due to the fact that an excessive smoothing of data could erase some physically important oscillations. Two parameters `Lchako` (applied after scaling of data) and `cosThmax` define the strain spans used in the detection of kinks. The parameter `ncols` indicates the number of columns in the `followMe.dat`, parameters `e1, .. e6` indicate numbers of strain-data columns and `s1, .. s6` indicate numbers of strain-data columns. The multipliers may be needed for scaling the values read from these columns, e.g. `e1=9*-0.01 e2=9*0.005 e3=9*0.005` composes the undrained strain path from the axial component in the 9th column (in %, and compression positive)

The stress paths from individual steps (from Laboratory $\boldsymbol{\sigma}^L(t)$ or from Simulation $\boldsymbol{\sigma}^S(t)$) can be quantified using $\boldsymbol{\sigma}(z) \approx \boldsymbol{\sigma}^B + \boldsymbol{\sigma}' z + \frac{1}{2}\boldsymbol{\sigma}'' z^2$, wherein $z = \int \|\dot{\boldsymbol{\epsilon}}\| \mathrm{d}t$. The program FITSTEP.EXE reads files `followMe.dat` and `followMe.rev` and writes the output file `followMe.fit` containing $z_B, \Delta z, \boldsymbol{\sigma}^B, \boldsymbol{\sigma}', \boldsymbol{\sigma}''$ with six components per stress in a separate line for each step.

4.4 Harmonic load

A single oscillation loop can be input with *CirculatingLoad

```
*CirculatingLoad
ninc   maxiter   deltaTime  : every   !  number of incr., max. number of  EI and step time and suppressed output
*Cartesian                                           ! alternatives here:  *Roscoe, *RoscoeIsomorph, *Rendulic
ifstress(1) deltaLoadCirc(1) phase0(1) deltaLoad(1) ! 1/0Flag, amplitude A_1, phase w0_1 and superposed linload B_1
ifstress(2) deltaLoadCirc(2) phase0(2) deltaLoad(2)
...
ifstress(6) deltaLoadCirc(6) phase0(6) deltaLoad(6)
```

The applied load increments are calculated from

$$\Delta L_i = \dot{\omega} \Delta t A_i \cos(\dot{\omega} t + \omega_i^{(0)}) + B_i \tag{1}$$

wherein ΔL_i denotes either the increment of stress `ddstress(i)` or the increment of strain `dstran(i)`, depending on the value of `ifstress(i)` for the i-th component. Other variables are

`deltaTime` = the period T so that $\dot{\omega}$ = `2*Pi/deltaTime`

Δt = `dtime = deltaTime/ninc`

A_i = `deltaLoadCirc(i)` = amplitude

B_i =`deltaLoad(i)/ninc` = linear shift

t = `time(1) + dtime/2 !  step time in the middle of the increment`

$\omega_i^{(0)}$ `= phase0(i)`

Closed harmonic loop can be superposed by a linear load B_i = `deltaLoad(i)/ninc`, with $i = 1, \dots 6$.

Each component of `*CirculatingLoad` may be individually shifted in phase using $\omega_i^{(0)}$ = `phase0(i)` so that oval paths can also be defined.

4.5 Definition of loading path by applying restrictions

The most flexible method for mixed control is offered by the command `*ObeyRestrictions`. This command reads 6 linear equations describing stress and/or strain change (per step not per increment), e.g.

```
*ObeyRestrictions
100 20   1.0                ! ninc, maxiter, deltaTime, full output (equiv. to  100 20 1.0 : 1)
sd1 + 1.0*sd2 + sd3 = 0     ! restriction 1 = isobaric condition
ed1 = -0.01                 ! restriction 2
ed3 = 0.0                   ! ....
ed4=0
ed5    =    0.0d0
ed6  = -0                   ! restriction 6
```

The change of i-th stress component within the step is denoted as `sd`i. The change of j-th strain component within the step is denoted as `ed`j. All restrictions have the formal structure

$$\mathsf{M}^t \cdot \Delta \mathbf{T} + \mathsf{M}^e \cdot \Delta \boldsymbol{\epsilon} = \mathbf{m}, \tag{2}$$

wherein all components of $\mathsf{M}^t, \mathsf{M}^e$ and $\mathbf{m}$ are known. The k-th restriction with $k = 1, \dots 6$ should be input as a separate line : $\sum_i M^t_{ki}$ `sd`i $+ \sum_{j \neq i} M^e_{kj}$ `ed`$j = m_k$ with sum over complementary indices i, j from 1 to 6. The parsing abilities of the IncrementalDriver are very limited. Here are the formal rules:

a. The input line cannot be longer than 120 characters.
b. No brackets `()`, no exponents `**` or divisions `/` may appear.
c. An index number at components `sd1,sd2,sd3,sd4,sd5,sd6,ed1,ed2,ed3,ed4,ed5,ed6` may appear at most once in a restriction (at a strain or at a stress increment), i.e. one cannot write `sd1 + sd1 = 0.0d0`. Lower case is obligatory for `sd` and `ed`.
d. Each component constitutes a summand and can be preceded by a single factor, e.g. `9.0d-4*sd1` but not `3*3.0d-4*sd1` and not `sd1*3.14`. Summands without stress or strain components must be delegated to the RHS, i.e. `sd1 + 1.0 + sd2 = 5.0` should be rewritten as `sd1 + sd2 = 4.0` .
e. Only numerical coefficients are allowed. You should write `4.0 *sd1 = 1.0` instead of two lines `t = 4` and `t*sd1 = 1.0`.
f. In all multiplications the asterisk `*` is obligatory. For example you should write `3.0*sd3 = 2.0` and not `3.0 sd3 = 2.0.`.
g. The summands are separated by `+` or `-`.
h. All spaces are ignored (spaces inside numbers cause errors, of course).
i. There must be just a single constant value in each restriction and it must appear on the right-hand side of the restriction. The constant can be preceded by `-` but not by `+`. No multiplications like `3.0*4.0` are allowed for.

The end-of-line comments must begin with `#`. Exactly 6 lines with restrictions must be input.

Example (from Javeriana): the projection of the stress path on the deviatoric plane should be a straight line, say $\dot{T}^*_1 = a\dot{T}^*_2$ We derive the restriction with Mathematica as follows:

```
eqs = {sddev1==sd1-pd, sddev2==sd2-pd, sddev3==sd3-pd, sddev1== a*sddev2, sddev3== -sddev1-sddev2, pd ==(sd1 + sd2 + sd3)/3} ;
Eliminate[eqs, {sddev1, sddev2, sddev3, pd}]  (* obtaining    (1 + 2 a)sd2 +   (1- a) sd3 -  (2 + a) sd1 == 0 *)
```

4.6 Response envelopes in stress or in strain

A perturbation of stress `*PerturbationsS` or strain `*PerturbationsE` can be applied to an arbitrary state as a separate step in `test.inp` with the following syntax

```
*PerturbationsE                     ! here alternatively  *PerturbationsS  for stress probes
 ninc, maxiter, deltaTime
*RoscoeIsomorph                      ! here alternatively  *Rendulic
deltaLoad(1)
```

Perturbations of the first two components of strain (or stress) are performed in such way that $\sqrt{(\breve{\Delta}\epsilon_P)^2 + (\breve{\Delta}\epsilon_Q)^2} =$ `deltaLoad(1)` holds. The strain probes are applied radially in `ninc` different directions equally distributed. Use either `*RoscoeIsomorph` or `*Rendulic` (only these transformations isometric).

4.7 Random walk (new 2019)

A random strain (or stress) path can be useful in testing new constitutive laws. A separate step in `test.inp` may have the following syntax

```
*RandomWalk
  1000    10   1.0  :  1           !   ninc  maxiter  deltaTime  :  every
*Cartesian                         ! here alternatively  *Rendulic or  ...
0     1.0e-4                       ! ifstress(1)    deltaLoad(1)
0     0.5e-4
0     0.5e-4
0     1.0e-4
0     1.0e-4
0     1.0e-5                       ! ifstress(6)    deltaLoad(6)
```

Here strain increments will be generated. The `deltaLoad( )` values describe the range in which the component of random strain will be generated, for example $\Delta\epsilon_{11} \in$ `( - deltaLoad(1), deltaLoad(1) )` .

4.8 Repetition of a group of steps

The keyword `*Repetition` has just two parameters `nSteps` and `nRepetitions`. It must be followed by description of `nSteps` steps which will be subsequently applied within a loop and repeated `nRepetitions` times.

Each step should be defined according to its own syntax. Exactly `nSteps` steps must be defined. Often `nSteps` is set to 2 in order to describe a simple stress (or strain) cycle. There is no end-of-repetition-loop statement so the next steps (beyond the position `nRepetitions`) will be executed just once unless wrapped in another `*Repetition`.

```
*Repetition
nSteps   nRepetitions
....
....! here follow nSteps preceded by their own keywords
...
```

For 1000 undrained triaxial stress cycles with the Amplitude $q^{\text{ampl}} = 10$ kPa and with 1Hz we can write:

a saw-like version with `*LinearLoad`

```
*LinearLoad
10   10   0.25
*Roscoe
0  0
1  10
0  0
0  0
0  0
0  0
*Repetition
2   1000
*LinearLoad
20   10   0.5
*Roscoe
0  0
1  -20
0  0
0  0
0  0
0  0
*LinearLoad
20   10   0.5
*Roscoe
0  0
1  20
0  0
0  0
0  0
0  0
*LinearLoad
10   10   0.25
*Roscoe
0  0
1  -10
...
```

or a harmonic version

```
*Repetition
1    1000
*CirculatingLoad
40    10    1.0
*Roscoe
0    0    0    0
1    10    0    0
0    0    0    0
0    0    0    0
0    0    0    0
0    0    0    0
```

or a saw-like with predefined shearing

```
*TriaxialUq
10  10  1.0
10.0
*Repetition
2    1000
*TriaxialUq
20    10    0.5
-20.0
*TriaxialUq
20    10    0.5
20.0
*TriaxialUq
10    10    0.2
-10
```

4.9 Predefined popular paths

Several short step descriptions have been predefined in the IncrementalDriver for convenience of geotechnical users. They are:

`*OedometricE1`, `*OedometricS1`, `*TriaxialE1`, `*TriaxialS1`, `*TriaxialUEq`, `*TriaxialUq`, `*PureRelaxation`, `*PureCreep`, `*UndrainedCreep`

These paths define the principal stresses / strains assuming x_1-axial symmetry of the applied components. The material response depends on `umat` and it need not be axisymmetric, of course. The shear components of strain are prescribed as constant

```
*OedometricE1
ninc maxiter dtime   : every
ddstran(1)    # lateral strain is assumed constant

*OedometricS1
ninc maxiter dtime : every
ddstress(1)  # lateral strain is assumed constant

*TriaxialE1
ninc maxiter   dtime : every
ddstran(1)     # lateral stress is assumed constant

*TriaxialS1
```

```
ninc maxiter dtime : every
ddstress(1)    # lateral stress is assumed constant

*TriaxialUEq
ninc  maxiter dtime : every
ddstran(2)    #  volume = constant, Roscoe's Delta epsilon_q is applied

*TriaxialUq
ninc  maxiter  dtime : every
ddstress(2)    # volume = const, Roscoe's Delta q is applied

*PureRelaxation
ninc maxiter  dtime : every

*PureCreep
ninc  maxiter dtime : every

*UndrainedCreep      # Roscoe's  Delta eps_v = 0 and Delta q  = 0   other stress inc also =0
ninc  maxiter dtime : every

*End   # can be used to terminate the calculation
```

5 Enforced exit from a step on a predefined condition (new in 2016)

Each step-command may be extended by a short inequality condition which will be evaluated at the end of each increment. If this condition is satisfied the remaining increments of the current step will be skipped. The next step will be commenced from the currently reached state. For example, we may interrupt an oedometric compression if the horizontal stress is large enough, say if $T_2 < -200$, by writing

```
*OedometricE1 ? s2 < -200.0
100  10 1 : 2
-0.0002   #  lateral strain is assumed constant
```

The exit condition is optional, so the old input files should work fine with the new version. The exit condition must be written in the same line as the description of step after the separator `?` and the whole line must not exceed 40 columns. You can use addition and multiplication, but both on the left-hand side of the inequality only. The right-hand side must be a single number. Parsing rules from Section 4.5 apply. No end-of-line comments and no `=` character may appear (sharp inequalities only).

Only the following variables can be used in the exit condition (extended in 2020):
`s1,s2,s3,s12,s13,s23` for stress components (mech. sign, tension positive) $T_{11}, T_{22}, T_{33}, T_{12}, T_{13}, T_{23}$
`e1,e2,e3,g12,g13,g23` for strain components (mech. sign, tension positive) $\epsilon_{11}, \epsilon_{22}, \epsilon_{33}, \gamma_{12}, \gamma_{13}, \gamma_{23}$
`p,q,P,Q` for geotechnical stress invariants (q with sign, i.e. $q = -(T_1 - T_3)$).
`ev,eq,eP,eQ` for geotechnical strain invariants (ε_q with sign, i.e. $\varepsilon_q = -\frac{2}{3}(\epsilon_1 - \epsilon_3)$).
`v1,v2,...v9` for state variables (just the first nine state variables have been implemented)
`q,Q,eq,eQ` deviatoric invariants are signed, e.g. $q = -(T_1 - T_3)$ assuming triax. symmetry $T_2 = T_3$

6 The command *End

In order to terminate calculations the command `*End` can be used. Commands behind `*End` will not be executed. Otherwise the file `test.inp` is read until the end of file is encountered. No empty lines should follow the description of the last step because the IncrementalDriver will try to read and interpret them. It may lead to strange errors. Hence, writing `*End` after the last step is recommended as a good practice.