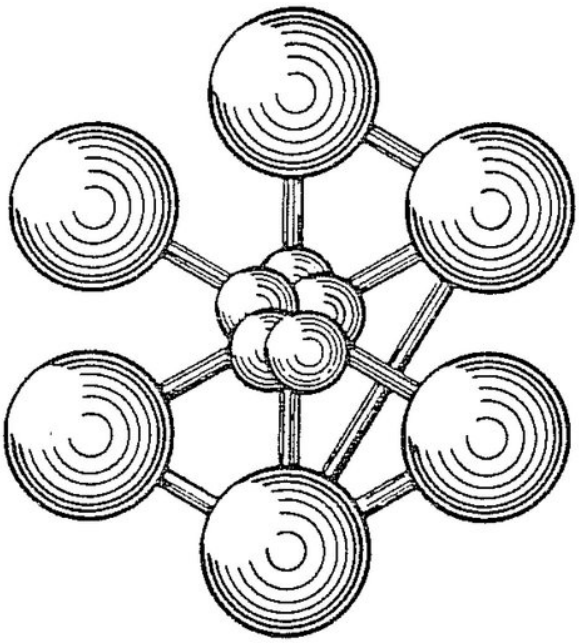
USING CAST3M

S. PASCAL

# A NOT SO SHORT INTRODUCTION TO CAST3M



cea

2026

mailto: serge.pascal@cea.fr

Cast3M is a finite element toolbox for structural analysis and fluid mechanics modelling. Cast3M is developed by the "Département de Modélisation des Systèmes et Structures" (DM2S) of the "Direction des Énergies" of the french Alternative Energies and Atomic Energy Commission (CEA).

The development of Cast3M is part of a research activity in mechanics. The goal is to develop a high-level tool that can serve as a reliable basis for the design, sizing, and analysis of structures and components.

Accordingly, Cast3M provides a complete system that includes not only calculation functions, but also capabilities for model development (pre-processor) and result analysis (post-processor). Cast3M is a "toolbox" program that allows users to develop functions tailored to their own needs.

Cast3M is more particularly used in the domain of nuclear energy as a simulation tool, as well as a generic plateform for the development of specialized program applications. In particular, Cast3M is used by the French Authority for Nuclear Safety and Radiation Protection (ASNR) in safety analysis of the french nuclear facilities.



# Contents


**1 Introduction** **7**
- 1.1 About Cast3M . . . 7
- 1.2 About this document . . . 7

**2 First mechanical analysis with Cast3M** **9**
- 2.1 Modelling in interactive mode . . . 9
  - 2.1.1 Cast3M Command Line Interface (CLI) . . . 9
  - 2.1.2 Global program options . . . 9
  - 2.1.3 User parameters . . . 11
  - 2.1.4 Check a result with **LIST** . . . 12
  - 2.1.5 Point definition . . . 12
  - 2.1.6 First mesh: a straight line . . . 13
  - 2.1.7 Get a manual page with **INFO** . . . 14
  - 2.1.8 Meshing the beam geometry . . . 15
  - 2.1.9 Display window . . . 16
  - 2.1.10 **MODE**l definition and **MATE**rial characteristics . . . 18
  - 2.1.11 Boundary conditions . . . 18
  - 2.1.12 Mechanical load . . . 19
  - 2.1.13 Linear resolution . . . 20
  - 2.1.14 Result analysis . . . 21
  - 2.1.15 Save and restore . . . 24
  - 2.1.16 Export and import results . . . 26
  - 2.1.17 End of analysis . . . 26
- 2.2 Modelling in a script file ".dgibi" . . . 27
  - 2.2.1 Default log file and script file `beam1.dgibi` . . . 27
  - 2.2.2 Highlighting Gibiane . . . 28
  - 2.2.3 Executing a script file . . . 30
  - 2.2.4 Printing graphical displays to a PostScript file . . . 30

**3 Nonlinear static analysis with PASAPAS** **33**
- 3.1 More about Gibiane . . . 33
  - 3.1.1 Basic object types . . . 33
  - 3.1.2 Conditional expressions . . . 35
  - 3.1.3 Iterative expressions . . . 35
  - 3.1.4 Gibiane procedures . . . 36
  - 3.1.5 Gibiane notices . . . 37
- 3.2 Example of nonlinear static analysis . . . 40
  - 3.2.1 Presentation of the problem . . . 40
  - 3.2.2 Definition of some parameters . . . 40
  - 3.2.3 Meshing . . . 41

3.2.4 Nonlinear mechanical behavior . . . . . 43
3.2.5 Boundary conditions . . . . . 44
3.2.6 Mechanical loading . . . . . 47
3.2.7 Resolution with PASAPAS . . . . . 48
3.2.8 Result analysis . . . . . 50
3.3 How to step in during a calculation with PASAPAS . . . . . 54
3.3.1 Operating principle using the procedure PERSO1 . . . . . 54
3.3.2 Element mesh removal . . . . . 55
3.3.3 Visualization during processing . . . . . 56
3.3.4 Script file beam3.dgibi . . . . . 58

**A Screenshots** **63**

**Index** **71**

# Chapter 1

# Introduction

## 1.1 About Cast3M

Cast3M is a finite element toolbox for structural analysis and fluid mechanics modelling. Cast3M is based on a set of elementary processes, a library of functions that the user combines to define and solve his problem, and to analyse the solution. These functions are available *via* Gibiane, the command language of Cast3M. Therefore using Cast3M requires learning Gibiane.

Gibiane has all the features of a computer language. Users can then program macro functions in Gibiane to adapt Cast3M to their needs. This makes Cast3M especially interesting for research and development activities, in which one has to develop new processes by combining existing ones.

The versatility of Gibiane makes also Cast3M a useful platform for developing calculation applications based on the finite element method. Moreover, as Cast3M is available on all common computer systems, namely Windows, Linux and Mac OS, these applications prove to be completely portable.

## 1.2 About this document

This document aims to provide non-French-speaking users with access to Cast3M. It deals only with structural mechanics, not fluid mechanics. Following this brief introduction, the document consists of two chapters. The first one presents the Cast3M environment and its command language, Gibiane, through an example of linear elastic analysis. The second chapter provides additionnal information on Gibiane and explains how to perform a nonlinear static analysis with **PASAPAS**, the main nonlinear solver in Cast3M for structural mechanics.

Currently, the Gibiane instructions presented in this document run on the last three versions of Cast3M (2024, 2025, 2026). But over time, some examples will no longer be completely up to date with the new ones. But that should not matter too much since most of what is presented in this document concerns the basics of the code, which has been under stable development for years.

This document was written with the help of the free versions of Google Translate and DeepL.

# Chapter 2

# First mechanical analysis with Cast3M

In this chapter, you are going to discover Cast3M and the Gibiane language through the example of computing the static equilibrium of a bending beam in three dimensions. Beyond this example, you will learn some fundamentals about Gibiane, as well as some useful functions for obtaining code documentation or for checking results on the fly. The chapter is composed of two parts: a first in which you use Cast3M in interactive mode, and a second in which you use a script file.

## 2.1 Modelling in interactive mode

### 2.1.1 Cast3M Command Line Interface (CLI)

Once Cast3M is installed on your computer, open a "cmd command-line" on Windows system, or a terminal on Linux or Mac-OS, and type "`castem24`" to launch the version 2024 of Cast3M[1]. You can also double-click on the Cast3M icon on your desktop to launch it in a new terminal. However, in this latter case, the terminal window will automatically close when you exit Cast3M. Anyway, the result should look like the screenshot presented in Figure 2.1, that presents the Command Line Interface (CLI) of Cast3M.

Using Cast3M in interactive mode involves typing code statements in the (CLI) to build a model, solve it, and analyze the results. Cast3M can also read code statements from a script file, as you will see in the next part of this chapter. But to get started with Cast3M, let's first use it interactively. The code statements are given in the command language of Cast3M: Gibiane.

### 2.1.2 Global program options

Let us first define some global options, which are the space dimension of the modelling and the type of finite element to use for meshing. For this, we use the **directive**[2] `OPTI`on. Please type or copy-paste the following Gibiane statement into the CLI:

```
OPTI 'DIME' 3 'ELEM' 'CUB8' ;
```

The keyword `'DIME'` stands for the space dimension (here, 3) and the keyword `'ELEM'`, for the type of element to use by default (a "`'CUB8'`" is a linear hexaedron).

You could have defined the element type (`'ELEM'`) before the space dimension (`'DIME'`), the order of the keywords do not matter. You could also have defined these two options in two succesive statements. The options `'ELEM'` and `'DIME'` are program state variables that can be modified at any time,

[1] For version 20XX of Cast3M, type "castemXX".
[2] In Cast3M, a "directive" is a function that do not return any result.




Figure 2.1: Cast3M header and Command Line Interface (CLI).



independently.

Finally, **you could have also given longer keywords** to make the code statement more readable, such as:

```
OPTION 'ELEMENT' 'CUB8' 'DIMENSION' 3 ;
```

**Note** Gibiane statements are decoded from left to right, searching for the first 4 characters of each keywords in the set of function names. As a result, if you add extra characters to function names, they are not taken into account. Note that this is only true for function names, and not for their arguments, whose names can be longer than 4 characters.

The directive **OPTI**on has many other functions than defining the space dimension or the default finite element type. They are all documented in its manual page, available on the Cast3M website. It is also available *via* the directive **INFO**, as you will see later in this chapter.

**Note** Every Cast3M function is documented in a manual page. When you use a new function or if you encounter any difficulty with one, refer to its manual page. There is one principle to keep in mind: **it is the manual page that is right, not the way the code works.** Therefore, if the code do not work as mentioned in the manual page, it is an anomaly. Please, do not hesitate to report any anomaly to the Cast3M support (see the website).

### 2.1.3 User parameters

It is convenient to define the dimensions of the beam as program parameters. Please type or copy-paste the following Gibiane statements into the CLI:

```
w1    = 0.06 ;
H1    = 0.1  ;
l1    = 2    ;
```

**Parameter names** (**w1**, **H1** and **l1**) **are freely defined by the user**. This is true for any type of result. However, **the length of result names is currently limited to 24 characters**.

**Gibiane is not case sensitive**, which is useful when using Cast3M in an interactive mode, as it is not necessary to pay attention to the letter case when typing statements. This implies that "**H1**" and "**h1**" designate the same parameter[3].

**In Gibiane, results are dynamically typed**. Hence, **w1** and **H1** are floating point numbers, while **l1** is an integer since there is no dot after the number 2. Gibiane has implicit conversion rules between integers and floating point numbers but you can use type conversion functions to master them, namely:

- **FLOT**: to convert an integer into a "FLOTTANT" (float),
- **ENTI**: to convert a float into an "ENTIER" (integer).

**Note** As you can freely name your results, you can overwrite the name of a function. In this case, the function is no longer available since it then designates the named result. To avoid this to happen, *it is recommended to put a figure in the result names*, like here-above **w1**, **H1** or **l1**, since function names contain only letters[4]. Anyway, if you happen to overwrite a function name in a user variable, *you can always call it by writing its name in uppercase letters enclosed in single quotes*.

[3] The code statements presented in this document intentionally mix upper and lower case letters to illustrate this feature.
[4] There are some rare exceptions, see the manual pages.

### 2.1.4 Check a result with LIST

If in doubt, **you can check a result with the directive LIST**. Please enter into the CLI:

```
list w1 ;
```

The CLI then returns the message presented in Figure 2.2. This message is in french. It means: "Real value: 6.00000E-02".

```
$ list w1 ;
* list w1 ;
Reel valant: 6.00000E-02
$
```

Figure 2.2: Listing of the parameter **w1**.

To get this message in english, change the **LANG**uage option to "anglais" (english):

```
OPTION 'LANGUAGE' 'ANGLAIS' ;
```

Actually, if you type the list command again, you get the message presented in Figure 2.3.

```
$ OPTION LANGUAGE ANGLAIS ;
* OPTION LANGUAGE ANGLAIS ;
$ list w1 ;
* list w1 ;
Real number equalling: 6.00000E-02
$
```

Figure 2.3: Listing of the parameter **w1** with the language option set to "anglais".

**Note** You can recall previous commands by pressing the up and down keys on the keyboard. Go through the previous ones, then modify one to list **l1**. You can then verify that **l1** is an integer.

### 2.1.5 Point definition

Before meshing, it is convenient to define some reference points for the geometry. Please enter the following Gibiane statements into the CLI:

```
P0 = 0. 0 0 ;
P1 = L1 0 0 ;
P2 = 0 W1 0 ;
P3 = 0 0 H1 ;
```

Remark that you can mix integer and real numbers in a point definition without any trouble (point **P0**). Figure 2.4 presents the listing of this point.

In this Figure, you can see that a point has a "number" (here, 1), which is its node number. This feature is useful when examinating meshes but you should not use this numbering in Cast3M, especially because it can be changed by the program. For this reason, *it is highly recommended to name points of interest and to never refer to their node number*.

You can also see that a point has a "Density". This is relative to what size the elements of a mesh should be at this point. The default value can be defined with the directive **DENS**ity or **OPTI**on. Note

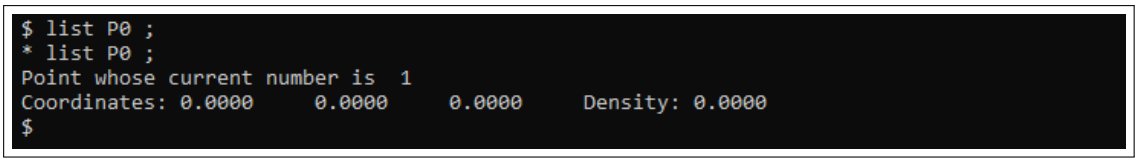



Figure 2.4: Listing of point **P0**.

that the density must be set before creating the point.

In Cast3M, you can define as many points as you wish, even at the same geometrical location:

```
OPTI 'DENS' 0.2 ;
PX = 0 0 0 ;
```

The listing of point **PX** is given in Figure 2.5. Indeed the point **PX** has the same coordinates as the point **P0** but different node number and density.

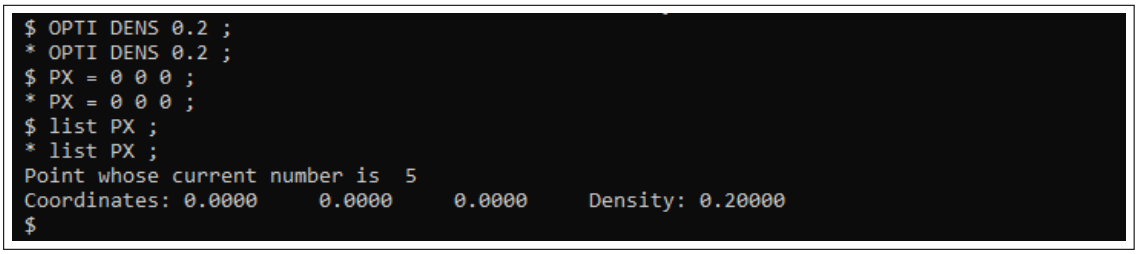



Figure 2.5: Listing of point **PX**.

### 2.1.6 First mesh: a straight line

To mesh a line from point **P0** to point **P2** containing 3 elements, use the **operator**[5] "**DROI**te" (straight line). The syntax is rather straightforward:

```
Line1 = P0 droi 3 P2 ;
```

In this statement, the order in which you give the arguments: **P0**, **3** and **P2**, is not mandatory. For example, these three other statements are equivalent:

```
Line1 = droi 3 P0 P2 ;
Line1 = 3 P0 P2 droi ;
Line1 = P0 P2 DroI 3 ;
```

For this feature, Gibiane is said to be a **non-positional language**, which means that the result of a Gibiane statement does not depend on the order of its arguments, as far as the order has no matter on the result.

This is a really useful feature: if you remember the name of an operator and if its arguments are of evidence, then you just have to put them in any order to get the result.

Of course, *the order of the arguments sometimes change the result* and you must be careful about this. For example, the following statement:

```
Line1 = P2 droi P0 3 ;
```

[5] In Cast3M, an "operator" is a function that returns one or more results.

Line1 = P2 droi P0 3 ;
is not strictly equivalent to the previous ones because the orientation of **Line1** is no longer from **P0** to **P2**, but reversed.

You can see the difference between this last syntax and any of the previous ones by listing **Line1** in each case (use the directive **LIST**). Do this by recalling previous commands using the up and down keys on your keyboard.

The importance of ordering arguments is obvious in comparison expressions. For instance, if $X1$ is a numerical value, we have this mathematical relation:

$$(X1 < 0) \neq (0 < X1).$$

But even in this case, there can be several equivalent syntaxes in Gibiane. For example, these three Gibiane statements are equivalent (A1=B1=C1):

```
A1 = X1 < 0. ;
B1 = X1 0. < ;
C1 = < X1 0. ;
```

This characteristic of Gibiane to be non-positional is not always well satisfied. But in all cases, **the syntax given in the manual page must work.**

### 2.1.7 Get a manual page with INFO

**You can always access to the manual page of any Cast3M functions with the directive INFO.** Please enter into the CLI:

```
info droi ;
```

to get the manual page of the operator **DROI**. Figure 2.6 shows the result of this command.

The manual page of a Cast3M function (operator or directive) generally exhibits three parts:

- a functional specification,
- a description of its purpose,
- the content of its inputs and outputs.

The functional specification is the general syntax of the function. In Figure 2.6, it is the line:

```
 GEO1 = DROIT (N1) POIN1 POIN2  ('DINI' DENS1)  ('DFIN' DENS2) ;
```

The names of the arguments are generally related to their type: **N1** is an integer, **POIN1** a point, **GEO1** a geometry (mesh)... Arguments into parentheses are optional.

When you are in the manual page, pressing the "Enter" key takes you to the next page, until returning to the command line. Pressing the "q" key returns immediatly to the command-line. Some manual pages are structured: **INFO** then displays a single-line menu at the bottom of the terminal window to navigate through the different parts.

**The set of manual pages integrated into a Cast3M version forms its "reference manual"**. If you find a way to use a function that is not documented in its manual page, do not use it since the result is not guaranteed. This is usually a hidden debugging option or a old syntax that is no longer maintained.

```
$ info droi ;
* info droi ;
                                          DATE     11/09/12

    Operateur DROIT (alias D)             Voir aussi : COMP
    -------------------------

    GEO1 = DROIT (N1) POIN1 POIN2  ('DINI' DENS1)  ('DFIN' DENS2) ;



    Description :
    _____________

    The DROIT operator constructs the straight line segment which
 connects the following two points : POIN1 and POIN2.

    Contents :
    __________

    POIN1, POIN2 : points fixing the limits of the segment of straight
                   line (POINT type)

    DENS1, DENS2 : densities associated with the points beginning and
                   ending the segment of straight line (FLOTTANT type)

    N1           : number of generated elements (ENTIER type)

    GEO1         : segment of straight line (MAILLAGE type)


    Note 1 :
    _________

    If N1 is not specified, the number of generated elements as well
as their size will be computed according to the extreme point densities.
    If N1 is specified and is positive, N1 elements of equal length
 will be generated.
    If N1 is negative, N1 elements will be generated and their
 size will be computed according to the extreme point densities.

    Note 2 :
    _________
```

Figure 2.6: The directive **INFO** displays the manual page of any Cast3M function (here, operator **DROI**).

The manual pages available on the website are those of the "version of the day". This version of Cast3M is only avalaible at the CEA. These manual pages may differ from those of any annual version, even the latest release.

To avoid any misunderstanding, please always refer to the manual pages integrated into your version by using the directive **INFO**. They are also provided in HTML format in the installation directory of Cast3M's annual versions. *But remenber that the absolute reference is always the manual page given by **INFO**.*

### 2.1.8 Meshing the beam geometry

You can mesh the beam cross section by translating **Line1** according to the vector defined by point **P3**. For this, use the operator **TRAN**slation to create the surface **Surf1**, putting 5 elements in the direction of translation:

```
Surf1 = Line1 TRAN 5 P3 ;
```

By the same way, you can mesh the volume of the beam by translating **Surf1** according to the vector defined by point **P1**. Use the operator **VOLU**me, option **'TRAN'**slation, putting 50 elements in the direction of translation:

```
Beam1 = Surf1 VoLu 'TRAN' 50 P1 ;
```

You can list the results **Surf1** and **Beam1**. However, the listing of **Beam1** turns out to be long since the mesh contains 750 elements. To only get the header of the listing and some of its first lines, use the option "**'RESU'**mé" (summary) of **LIST**:

```
List 'RESU' beam1 ;
```

The result of this command is shown in Figure 2.7. The listing shows that **Beam1** is a "MAILLAGE" (mesh) of pointer number 2122072. It also gives the number and type of its elements. The subreferences are pointer numbers of named parts of the mesh. Finally it gives in column the color of each element (DEFA is the "default" color) and their constitutive nodes.

```
$ List Resu beam1 ;
* List Resu beam1 ;
MAILLAGE 2122072 : 750 element(S) of CUB8 type
3 subreference(s)
List of (the) subreference(s) :
  2121995   2122142   2122212
1st line  number element : 2nd color : 3rd... node(s)
          1       2       3       4       5       6       7       8       9      10
       DEFA    DEFA    DEFA    DEFA    DEFA    DEFA    DEFA    DEFA    DEFA    DEFA

          3      14      15      20      28      29      21      30      31      22
         14      15       1      28      29      24      30      31      25      32
         28      29      24      30      31      25      32      33      26      34
         20      28      29      21      30      31      22      32      33      23
         60      61      64      63      62      65      69      68      70      73
         61      64      66      62      65      67      68      70      71      72
         62      65      67      68      70      71      72      74      75      76
         63      62      65      69      68      70      73      72      74      77
$
```

Figure 2.7: Listing of the mesh **Beam1** with the option **'RESU'**.

**Note** You may have noticed that each line of statement typed in the CLI is repeated by the program. This feature can be switch off by setting the **OPTI**on **'ECHO'** to zero:

```
OPTI 'ECHO' 0 ;
```

**'ECHO'** is set to 1 by default.

### 2.1.9 Display window

You can display the mesh with the directive **TRAC**e. For example, copy-paste into the CLI this statement:

```
TRAC 'CACH' Beam1 'TITR' 'First mesh with Cast3M !' ;
```

The result of this command is shown in Figure 2.8. The option **'CACH'** hiddes the part of the mesh in the backside of the view (try without). The option **'TITRE'** adds a title at the bottom of the display window. *Note the use of single quotes to define a character string.* The directive **TRAC** has many other options, see its manual page.

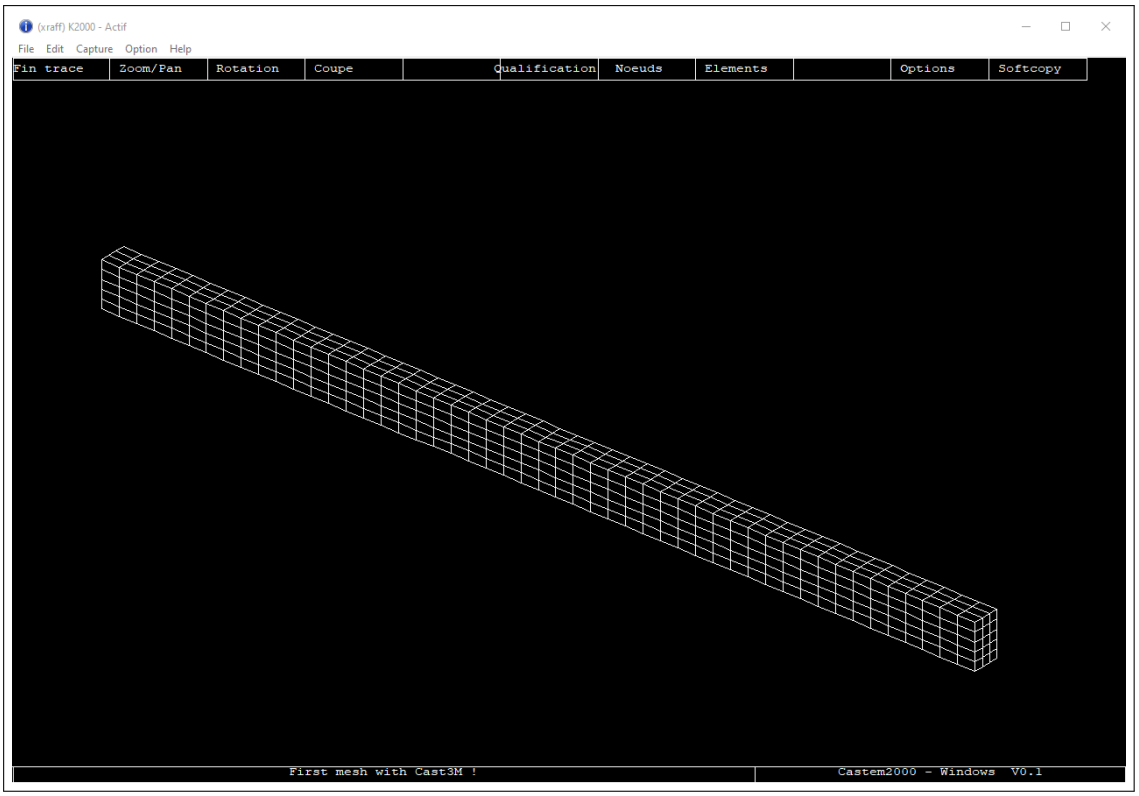



Figure 2.8: Display of the mesh Beam1.

There are several buttons at the top of the display window. Try them except "Fin trace", which is the "Quit" button:

- The button "Rotation" displays a square with different sectors. This square is a plane representation of a unit sphere, in which the horizontal lines represent the equator and latitudes at ±45°, and the vertical lines represent longitudes at ±90° apart.
- The button "Zoom/Pan" opens a menu to zoom on a part of the mesh by clicking two times in the display window to define the diagonal of the local frame. When using the submenu "Pan", click once to define a point to be moved in the view, and again to define where it must go.
- The button "Coupe" (cut) is used to define a cutting view of the mesh, parallel to the view plane. Clicking on the button brings up three submenus : "Retour" (return), "Annulation", to cancel the current cut, and "Position", to define a new cut. Clicking on this last button displays an elongated rectangle, a kind of ruler: the left edge of this ruler represents the closest point of the mesh to your viewpoint, and its right edge, the furthest point. Clicking in this ruler defines the relative position of the cutting plane in the mesh.
- The menus "Qualification", "Noeuds" (nodes) and "Elements" are fairly straightforward.
- The button "Softcopy" opens a menu to print the display in a PostScript file.
- The button "Options": give it a try!

Now, click on "Fin trace" in the display window to quit and give hand to the CLI.

### 2.1.10 MODEl definition and MATErial characteristics

The mesh **beam1** represents the geometry. Now, you have to assign a physics to this geometry. In Cast3M, this is achieved by defining on the mesh a "model" with the operator **MODE**l:

```
MOD1 = MODE Beam1 'MECANIQUE' 'ELASTIQUE' ;
```

**MOD1** is actually a "model" in a Cast3M sense. The listing of **MOD1** is presented in Figure A.1 in appendix. As you can read in the header of this listing, MOD1 is an object of type "MMODEL" that contains 1 elementary zone.

The key word **'MECANIQUE'** defines the physics assigned to the beam: in Cast3M, it is called the "formulation" of the model. If you refer to the manual page of **MODE**, you will see that there are several formulations available in Cast3M: **'THERMIQUE'**, **'DIFFUSION'**...

The key word **'ELASTIQUE'** defines the material behavior assigned to the beam. The manual page of **MODE** also shows that there are many mechanical behavior laws available in Cast3M.

To define the material characteristics of **MOD1**, use the operator **MATE**erial:

```
MAT1 = MATE MOD1 'YOUN' 30.e9 'NU' 0.2 ;
```

**MAT1** is an "element field", *i.e.* a field which values are defined at integration points (or "Gauss points") of the finite elements. The key words **'YOUN'** and **'NU'** stand for the Young's modulus and the Poisson's ratio, respectively. The values given are typical elastic properties of concrete. The units must be consistent with those of the mesh dimensions. Here, the Young's modulus is given in pascal, which is consistent with dimensions in meters.

The listing of **MAT1** is presented in Figure A.2. The header indicates that **MAT1** is an object of type MCHAML, which is not a meaningful name but results from the code development history. The listing also refers to the pointer number of the mesh onto which this field is defined (the same as in **MOD1** listing) and gives other information that we let you investigate.

One could also define on **beam1** a thermal model **MOD2**, with characteristics **MAT2**, as well as a diffusion model **MOD3**, with characteristics **MAT3**, and so on, to describe all the physics to take into account in the modelling of this beam.

In the same way, if there were several meshes, you could define on them models with different mechanical behaviors (plasticity, viscoelasticity...) or, for a given mechanical behavior, different material characteristics.

In Cast3M, you can define as many models and characteristics as you need to build your modelling. Model and element field objects can be summed with the operator "**ET**" (and). The syntax is basically:

```
MOD0 = MOD1 ET MOD2 ET MOD3 ... ;
MAT0 = MAT1 ET MAT2 ET MAT3 ... ;
```

### 2.1.11 Boundary conditions

The beam is embedded at its first end, along the surface mesh **Surf1**. To prescribe a condition to the displacements, use the operator "**BLOQ**uer" (block):

```
BC1 = BLOQ 'DEPL' surf1 ;
```

The key word **'DEPL'** stands for all the components of displacement, *i.e.* **'UX'**, **'UY'**, **'UZ'**. If you want to block only the **'UX'** component, for instance, replace **'DEPL'** by **'UX'**. See the manual page of **BLOQ** for more information.

### 2.1.12 Mechanical load

A force is apllied downward at the other end of the beam. To prescribe this force, it is necessary to define the mesh on which the force is applied.

#### Load application surface

There are several possibilities to get this surface mesh. One of them is to define a mesh of points (or point set) using the operators **POIN**t and **COOR**dinates:

```
PTX1 = (beam1 coor 1) poin 'MAXI' ;
```

It is the first time that parenthesis are used in a code statement example. In Gibiane, as in many other computer languages, the expression into parentheses is first evaluated before executing the rest of the statement.

In the statement in parentheses, the operator **COOR** build the "point field" of the coordinates of **beam1** in the first (1) direction of space coordinates, which is the direction of the beam length. Generally speaking, a "point field" is a field of values defined at some points. In this case, these are the points of **beam1**.

In Cast3M, a "point field" is an object of type `CHPOINT`. However, in this case there is no way to get a listing of this field since it is a temporary result passed directly to the operator **POIN**.

The option **'MAXI'** of operator **POIN** takes all the points at which the "point field" is maximal. In this case, these are all the points at the maximal coordinate along the beam length. The result **PTX1** is then a mesh made up of elements of type `POI1`, which is the element type for a mesh of points in Cast3M, as it is mentionned in the listing of **PTX1** (see Figure A.3 in appendix).

You can visualize the location of **PTX1** by displaying it together with the edges of the beam:

```
titr 'Beam1 edges and PTX1 node set (in red)' ;
Trac ((aret beam1) et (PTX1 coul 'ROUGE')) ;
```

The result of these statements is presented in Figure 2.9.

The first statement defines the display title. Here, **TITR** is not used as an option of the directive **TRAC** but as a directive in its own. In fact, the title in the display window is a global variable of the program that you can change by using the directive **TITR** or the option of the operator **TRAC**. If you do not specify a title, Cast3M will reuse the previous one. The default title is "`CASTEM FECIT`" ("Cast3M did it" in latin).

The 2nd statement displays the edges of **Beam1**, together with the mesh **PTX1** colored in red. It includes several statements into prentheses, that construct temporary results that will no longer be available once you leave the display.

In this 2nd statement, the operator **ARET** (edge) first constructs the mesh of the edges of **Beam1**. Second, the operator "**COUL**eur" (color) creates a copy of PTX1 colored in "rouge" (red). Third, the operator **ET** concanates these two meshes into a global one. Lastly, the directive **TRAC** displays this global mesh. Actually, **PTX1** contains the points at the opposite side of **Surf1**. To have no doubt, recall and modify the previous command to add **Surf1** to the display.

#### Force definition

To apply a force on the mesh **PTX1**, use the operator **FORC**e:

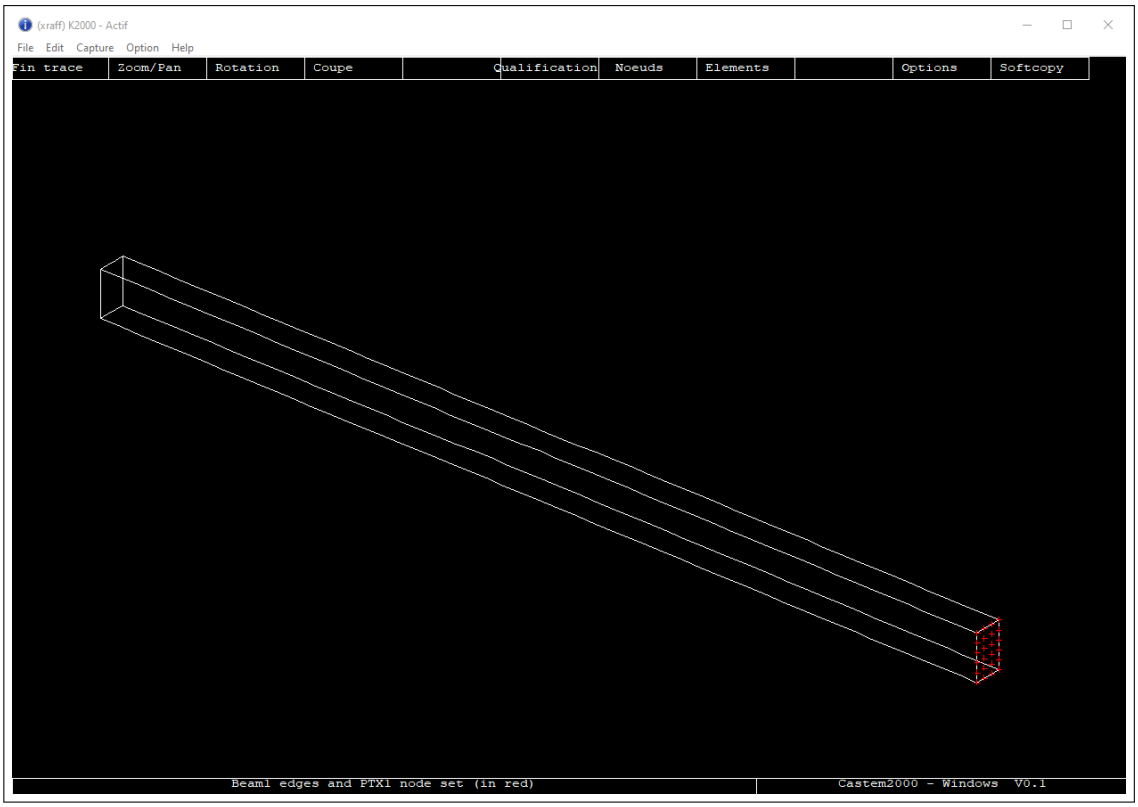



Figure 2.9: Display of the mesh PTX1 (red cross), together with the edges of the mesh **beam1**.

```
F1 = forc PTX1 (0 0 -1000) ;
```

This statement is relativaly obvious to understand. The unit of the force magnitude must be consistent with the units of the other values involved in the problem. Here, the force is expressed in newton.

The Figure A.4 in appendix presents the listing of **F1**. The listed values of the component UZ are not equal to 1000. This is because the force magnitude is divided by the number of node of **PTX1** in order to get a resultant force equal to the prescribed value. Once again, remember that if you do not know how an operator exactly works, have a look to its manual page and display its result with **LIST**.

### 2.1.13 Linear resolution

To determine the displacement field in **Beam1**, one has to build the usual linear system KU=F. To do this, we first build the elastic stiffness **KEL1** with the operator **RIGI**dity:

```
KEL1 = RIGI MOD1 MAT1 ;
```

The object type of **KEL1** is "RIGIDITE" (rigidity). Note that the listing of this object type can be long, even with the **'RESU'** option.

To take into account the boundary conditions, we add **BC1** to **KEL1**:

```
K1 = KEL1 ET BC1 ;
```

Indeed, **BC1** is also an object of type RIGIDITE, which projects the space of unknown displacements onto the space of constraints.

To calculate the displacement field **DEP1** at the nodes of **Beam1**, we solve the linear system defined by **K1** and **F1** with the operator "**RESO**udre" (solve):

```
DEP1 = RESO K1 F1 ;
```

Figure A.5 in appendix presents the listing of **DEP1**. In this figure, you can see that **DEP1** is an object of type CHPOINT, containing two sub-fields with component names LX for the first, and UX, UY, UZ for the second. The sub-field of component LX is the "node field" of Lagrange multiplier.

### 2.1.14 Result analysis

#### Displacement field

To analyse the result, we can first display the displacement field **DEP1**:

```
TRAC DEP1 BEAM1 'TITRE' 'Displacement field' ;
```

Figure 2.10 presents the contour plot of the component UZ of **DEP1**. The statement here above first shows the contour plot of the component UX. To change it, click on the button "Composantes" (components) and choose UZ.

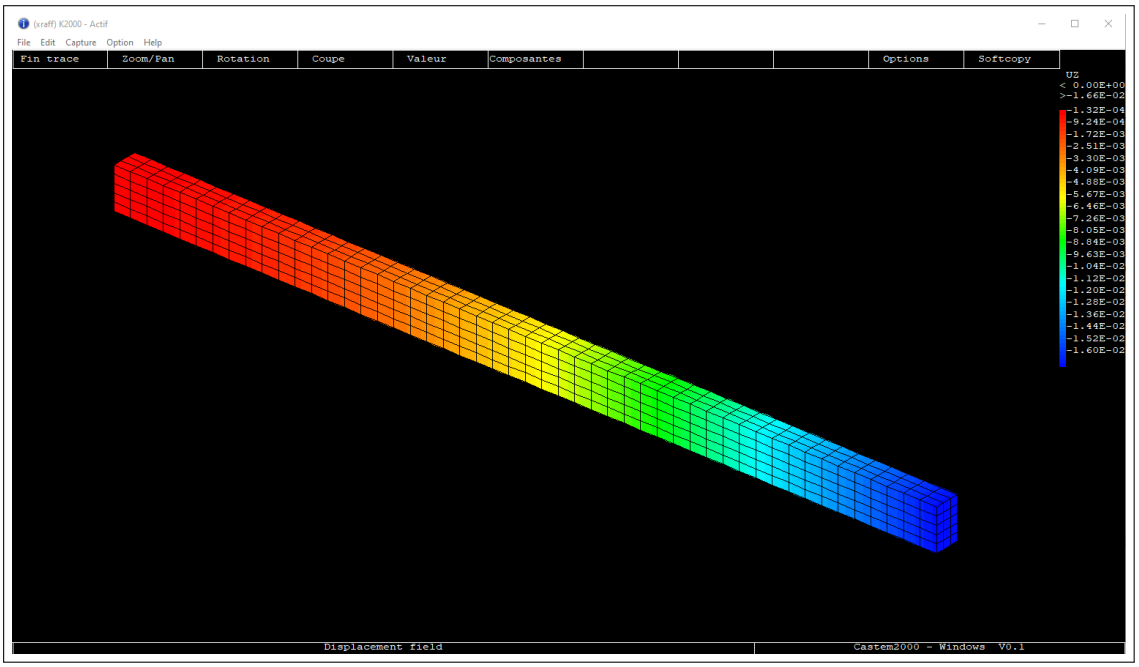



Figure 2.10: Contour plot of the component UZ of the displacement field **DEP1**.

The extremal values of UZ are indicated at the top of the legend. You can see in Figure 2.10 that the maximal deflection is equal to -1.66E-2. You can also click on the button "Valeur" (value), then click anywhere in the contour plot to get the value of the field at that point.

**Note** When displaying a CHPOINT type object (here **DEP1**), you must also provide a mesh (here **BEAM1**) to interpolate the values in the finite elements, as this type of object only contains values at the point field and no information about how they vary in between. With the version 2022 of Cast3M and later, try without giving the mesh.

#### Deformed shape

To display the deformed shape, we first build an object of type "DEFORMEE" (deformed shape) with the operator DEFO, then display the result with TRAC:

```
DEF0 = DEFO DEP1 BEAM1 0. 'BLANC' ;
DEF1 = DEFO DEP1 BEAM1    'ROUGE' ;
titre 'Deformed shape (red) Vs initial shape' ;
trac (0 -100 0) (def0 et def1) ;
```

The first two statements here above build the deformed shapes **DEF0** and **DEF1** of the mesh **Beam1** according to the displacement field **DEP1**. The magnification of **DEF0** is set to zero and its color to `'BLANC'` (white). The magnification of **DEF1** is not specified and its color is set to `'ROUGE'` (red). The title of the display is defined before calling the directive **TRAC**. The result of these four statements is given in Figure 2.11.

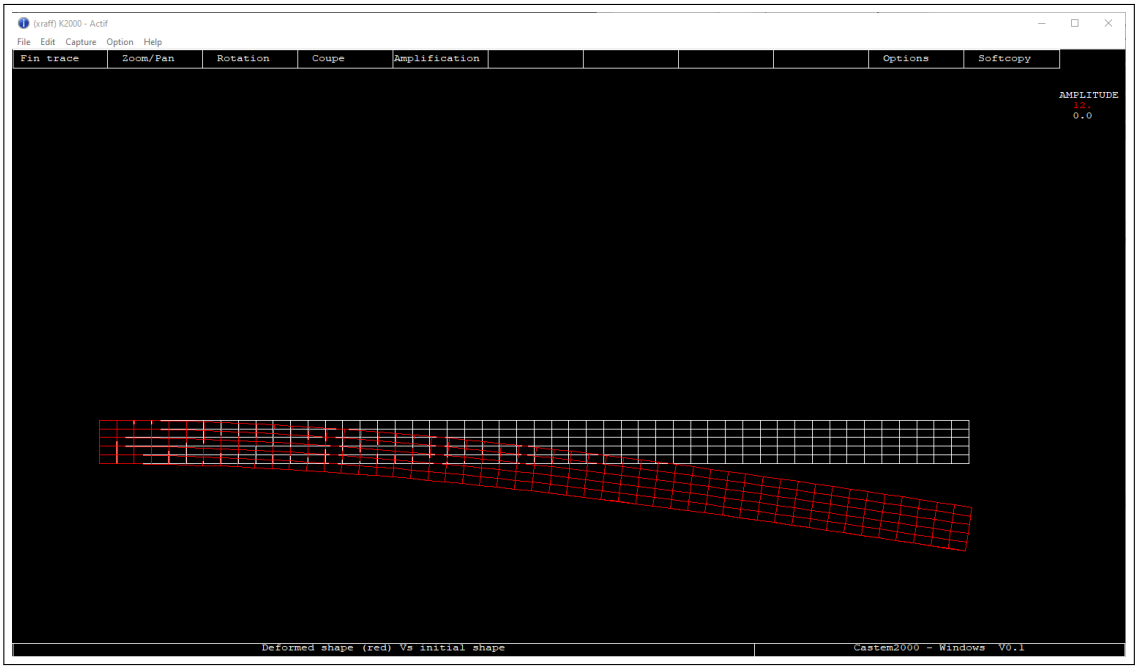



Figure 2.11: Deformed shape of **Beam1** (in red), together with the initial shape (in white). The viewpoint of the display is set at point `(0 -100 0)` in order to see the deformed shapes in the (O,y) direction.

In this Figure, you can see that the magnification of **DEF1** is automatically computed by the program. Its value is indicated on the right of the display window, under the legend title "`Amplitude`". It is written in the same color as the deformed shape (here 12). Of course, the magnification of **DEF0** is also indicated. Lastly, the viewpoint of the display is defined at the point `(0 -100 0)` in order to see the deformed shapes according to the (0,y) direction.

### Stress field

To compute the stress field **SIG1**, we use the operator **SIGM**a:

```
SIG1 = SIGM MOD1 MAT1 DEP1 ;
```

In this statement, **SIGM** first computes the linear strains derived from the displacement field **DEP1** according to the finite element model **MOD1**, then the stress field **SIG1** by applying the Hook's law with the elastic properties given in **MAT1**.

We can now display the stress field with **TRAC**:

```
Trac (100 -100 80) (1.e-6*SIG1) MOD1 'TITRE' 'Stress field (MPa)' ;
```

In this statement, **SIG1** is multiplied by `1.e-6` to convert its values in megapascal (MPa). The viewpoint is set at `(100 -100 80)`. The result is shown in Figure 2.12.

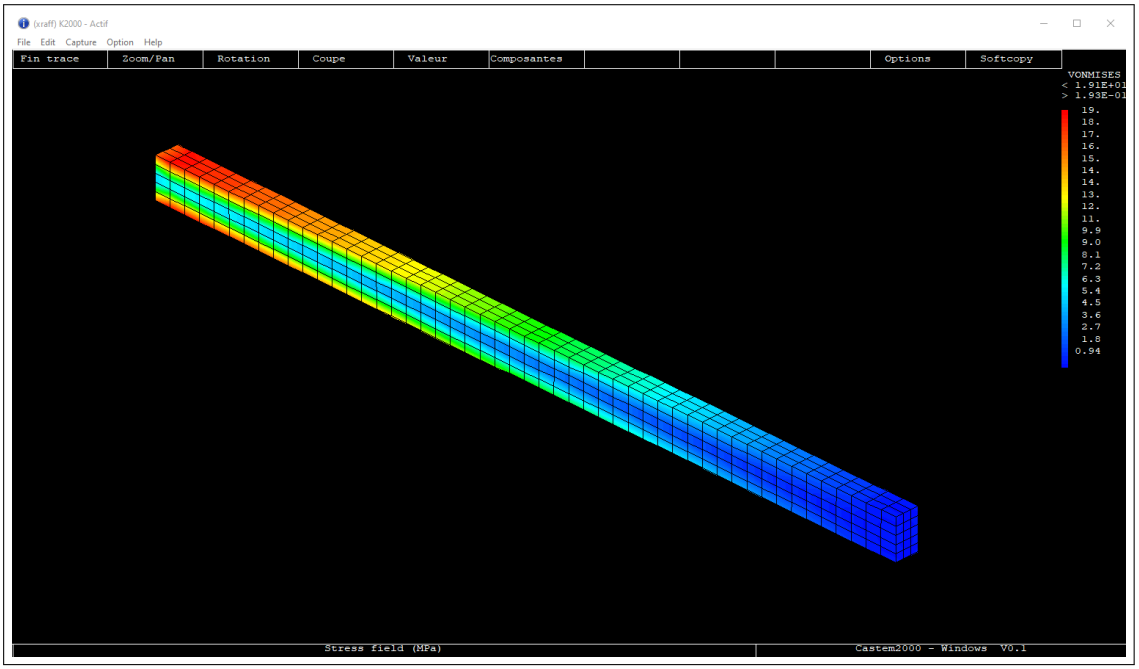



Figure 2.12: Stress field **SIG1** (MPa). The viewpoint is at `(100 -100 80)`.

When giving a stress field to **TRAC**, it automatically computes the von Mises stress and displays it. You can display the other components of **SIG1** by clicking on the button "`Composantes`" and choosing the one you want. The name of the displayed component and its extremal values are indicated at the top of the legend.

The listing of **SIG1** is presented in Figure A.6 in appendix. You can see that **SIG1** is a `MCHAML` type object ("element field"), which means that its values are defined at the integration points of the finite elements, here the integration points for the stresses.

**Note** The extremal values of **SIG1** given in the display window are values *extrapolated to the nodes* of the mesh to be consistent with the contour plot. If you want to know the maximal or minimal values of **SIG1** at the integration points, you can use the operators **MAXI** or **MINI** to obtain and list them.

### Evolution of the displacement along a line

Consider the edge located along the length of the beam and bounded by the points **P0** and **P1**, and let's name this edge **Line2**. You can construct **Line2** from **Beam1** with the following statements:

```
PtsL2 = Beam1 poin 'DROI' P0 P1 1.e-5 ;
Line2 = (Beam1 aret) elem 'APPU' 'STRI' PtsL2 ;
```

The first statement get all the points of **Beam1** located at a distance lower or equal to `1.e-5` along the "**droi**te" (straight line) defined by **P0** and **P1**. The point set **PtsL2** is a mesh of points.

The second statement first builds the edges of **Beam1** with the operator "**ARET**" (edge). Then, the operator **ELEM** get all the elements of the edges "**appu**yés **'STRI'**ctement" (stritly supported) by the

point set **PtsL2**[6]. Try to display **Line2** colored in red, together with the edges of **Beam1**.

To build the EVOLUTIOn of the component UZ of **DEP1** along the **Line2**, call the operator **EVOL**ution with the option **CHPO**:

```
EVUZ1 = EVOL 'VERT' 'CHPO' DEP1 'UZ' Line2 ;
```

**EVUZ1** is an object of type EVOLUTIOn. This type of object describes the evolution of a quantity as a function of another. It is made up of two lists of real numbers of the same length, giving for each abscissa the value of the ordinate. An EVOLUTIOn object has different attributes, like a color. Here, it is colored in "**'VERT'**" (mygreen). Besides, several EVOLUTIOns can be summed into one by using the operator **ET**. The listing of EVUZ1 is presented in Figure A.7 in appendix.

To plot an EVOLUTIOn, use the directive "**DESS**iner" (draw). You can previously define the title of the display with the directive **titr** and attach a label to the curve in a caption table:

- first, initialize a TABLE type object (**tleg1**) with the operator **tabl**,
- then, modify the default line style, which is solid line, by adding to the curve crosses of small size : **'MARQ S CROI'**,
- last, attach the label **'Uz (m)'** to the curve in the subtable **tleg1.titre**.

All this leads to the following statements:

```
tleg1 = tabl ;
tleg1 . 1 = mot 'MARQ S CROI' ;
tleg1 . titre = tabl ;
tleg1 . titre . 1 = 'Uz (m)' ;
titr 'Evolution of the displacements along Line2' ;
dess EVUZ1 'LEGE' tleg1 'XBOR' 0. 2.1 'XGRA' 0.25 'GRIL' 'POIN' 'GRIS' ;
```

In the statement beginning by **DESS** here-above, the key-word **'LEGE'** is for the caption table. The key-word **'XBOR'** defines the bounds of the abscissa and **'XGRA'** the graduation. Finally, the optional expression: **'GRIL' 'POIN' 'GRIS'**, displays a dotted gray grid. The directive **DESS** has many other options, see its manual page.

The result of these statements is presented in Figure 2.13. Starting from negative values, the curve increases until reaching zero at abscissa 2, where a triangle mark indicates that this is point **P0**. This mark shows that **Line2** is oriented from **P1** to **P0**. You could reverse the orientation of **Line2** with the operator "**INVE**erser" (reverse).

You may have noticed that the display shows no mark for **P1**. The reason is that point **P1** is not part of **Beam1**, and therefore not of **Line2** either, by construction of the mesh by the operator **VOLU**.

### 2.1.15 Save and restore

To save results in a file, you must first define the name of the file with the directive **OPTI**, option **SAUV**. You are free to name this file whatever you want. However, it is convenient to use the same file name as the script but with a different extension, for example ".sauve" (or ".save"). For the script file "beam1.dgibi", this leads to the following statement:

```
OPTI 'SAUV' 'beam1.sauve' ;
```

This statement can be placed anywhere in the script. Once the file name is defined, you can record the results with the directive **SAUV**:

[6] An element is strictly supported by a point set if all its nodes are in the point set.

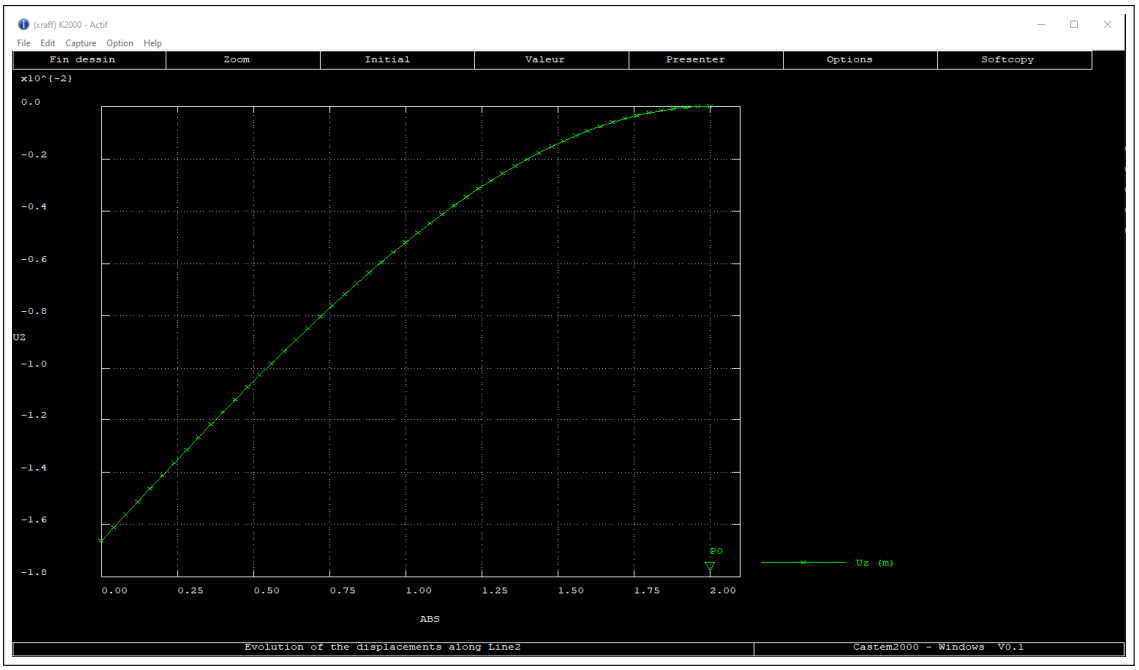



Figure 2.13: Evolution of the displacement $U_z$ along Line2.

```
SAUV ;
```

Without any arguments, **SAUV** saves all the results. But you can save only some results in order to reduce the size of the record. You simply have to precise the result names. For example the statement:

```
SAUV Beam1 ;
```

only saves the mesh **Beam1**.

The directive **SAUV** can be called at any time, and as many times as required. In incremental recordings (several calls to **SAUV**), each recording is identified by a label (see the manual page of **SAUV**). But it is usually called once at the end of the script to save all the results.

The statements to restore results from a file are identical to the **SAUV** function. First, you define the name of the file to restore with the directive **OPTI**, option **REST**. Then use the directive **REST** to open the file and load the results:

```
OPTI 'REST' 'beam1.sauve' ;
REST ;
```

Figure 2.14 shows a screenshot of the terminal after restoring the file "beam1.sauve". The directive **REST** prints several informations about the file and its contents, mainly:

- the version number of the recording format: `NIVEAU DU FICHIER 24`,
- the error level, space dimension and default density: `NIVEAU D'ERREUR 0...`,
- the number and type of recorded results: `12 MAILLAGE`, `8 CHPOINT`, `9 RIGIDITE...`
- the name and number of backup label: `LABEL_AUTOMATIQUE_1` and `LABEL_AUTOMATIQUE_2`,

and indicates that the operation proceeded normally. The 2nd label is due to the 2nd call to **SAUV**.

```
$  * OPTI REST 'beam1.sauve' ;
The type of restoration file is XDR
$  * REST ;


 NIVEAU DU FICHIER 24
 TAILLE DES COMPOSANTES   8


 NIVEAU D'ERREUR  0 DIMENSION  3 DENSITE  2.00000E-01
 LECTURE DE       34 OBJETS MAILLAGE
 LECTURE DE        8 OBJETS CHPOINT
 LECTURE DE        9 OBJETS RIGIDITE
 LECTURE DE        2 OBJETS TABLE
 LECTURE DE       12 OBJETS IMATRI
 LECTURE DE        2 OBJETS DEFORME
 LECTURE DE        3 OBJETS LISTREEL
 LECTURE DE        1 OBJETS EVOLUTIO
 LECTURE DE        3 OBJETS FLOTTANT
 LECTURE DE        1 OBJETS ENTIER
 LECTURE DE       65 OBJETS MOT
 LECTURE DE        1 OBJETS LISTMOTS
 LECTURE DE     1299 OBJETS POINT
 LECTURE DE        1 OBJETS CONFIGUR
 LECTURE DE        1 OBJETS MMODEL
 LECTURE DE        2 OBJETS MCHAML
 LECTURE DE        5 OBJETS MINTE
 LECTURE DE        8 OBJETS IELVAL
FIN DE LECTURE DU LABEL :
LABEL_AUTOMATIQUE_1

 LECTURE DE        0 OBJETS POINT
 LECTURE DE        1 OBJETS CONFIGUR
FIN DE LECTURE DU LABEL :
LABEL_AUTOMATIQUE_2

 Retrieval normal termination
 $  * FIN ;
```

Figure 2.14: Screenshot of the terminal after restoring the file "beam1.sauve".

**Note** In Cast3M, the principle of save and restore operations is to be able to resume the program as if it had never stopped. Therefore, these operations are not limited to saving/returning results, but also cover all the program state variables: space dimension, element type by default, etc. *The save and restore operations must preserve the referential integrity of the data.*

### 2.1.16 Export and import results

You can export EVOLUTIOn result types in a .csv file by calling the directive "**SORT**ir" (exit) with the option **'EXCEL'**, and more general result types with the option **'VTK'**, file format supported by ParaView. The dual directive of **SORT** is "**LIRE**" (read). See their manual pages for more details.

### 2.1.17 End of analysis

To finish with Cast3M and exit the program, simply type:

```
FIN ;
```

into the CLI, and the program ends!



## 2.2 Modelling in a script file ".dgibi"

### 2.2.1 Default log file and script file `beam1.dgibi`

The current directory should now contains a file named "fort.98". This is the default name for the log file of Cast3M, when used without an input file.

Copy or rename this file "beam1.dgibi". The extension "dgibi" is the common file extension for input files (or script files) of Cast3M. "dgibi" means "**d**onnées **gibi**ane" (gibiane data). However, this extension is not mandatory and you can name your input file as you wish.

When opening "beam1.dgibi" with a text editor, you can read all the statement lines that you have typed into the CLI.[7] It should look like the script file presented in Listing 2.1.

Listing 2.1: Script file "beam1.dgibi".

```
1  OPTI 'DIME' 3 'ELEM' 'CUB8' ;
2  OPTION 'ELEMENT' 'CUB8' 'DIMENSION' 3 ;
3  w1    = 0.06 ;
4  H1    = 0.1  ;
5  l1    = 2    ;
6  list w1 ;
7  OPTION 'LANGUAGE' 'ANGLAIS' ;
8  list w1 ;
9  list L1 ;
10 P0 = 0. 0 0 ;
11 P1 = L1 0 0 ;
12 P2 = 0 W1 0 ;
13 P3 = 0 0 H1 ;
14 list P0 ;
15 OPTI 'DENS' 0.2 ;
16 PX = 0 0 0 ;
17 list PX ;
18 Line1 = P0 droi 3 P2 ;
19 Line1 = droi 3 P0 P2 ;
20 Line1 = 3 P0 P2 droi ;
21 Line1 = P0 P2 DroI 3 ;
22 Line1 = P2 droi P0 3 ;
23 List Line1 ;
24 info droi ;
25 Surf1 = Line1 TRAN 5 P3 ;
26 Beam1 = Surf1 VoLu 'TRAN' 50 P1 ;
27 List 'RESU' beam1 ;
28 OPTI 'ECHO' 0 ;
29 OPTI 'ECHO' 1 ;
30 TRAC 'CACH' Beam1 'TITR' 'First mesh with Cast3M !' ;
31 MOD1 = MODE Beam1 'MECANIQUE' 'ELASTIQUE' ;
32 list MOD1 ;
33 MAT1 = MATE MOD1 'YOUN' 30.e9 'NU' 0.2 ;
34 list MAT1 ;
35 BC1 = BLOQ 'DEPL' surf1 ;
36 PTX1 = (beam1 coor 1) poin 'MAXI' ;
37 list PTX1 ;
38 titr 'Beam1 edges and PTX1 node set (in red)' ;
39 Trac ((aret beam1) et (PTX1 coul 'ROUGE')) ;
```

[7] Depending on the Cast3M version you are using, you may have a space character at the beginning of each line of the log file. It is up to you to keep it or delete it.

```
Trac ((aret beam1) et (PTX1 coul 'ROUGE') et Surf1) ;
F1 = forc PTX1 (0 0 -1000) ;
List 'RESU' F1 ;
KEL1 = RIGI MOD1 MAT1 ;
K1 = KEL1 ET BC1 ;
DEP1 = RESO K1 F1 ;
List 'RESU' DEP1 ;
TRAC DEP1 BEAM1 'TITRE' 'Displacement field' ;
DEF0 = DEFO DEP1 BEAM1 0. 'BLANC' ;
DEF1 = DEFO DEP1 BEAM1    'ROUGE' ;
titre 'Deformed shape (red) Vs initial shape' ;
trac (0 -100 0) (def0 et def1) ;
SIG1 = SIGM MOD1 MAT1 DEP1 ;
Trac (100 -100 80) (1.e-6*SIG1) MOD1 'TITRE' 'Stress field (MPa)' ;
List 'RESU' SIG1 ;
PtsL2 = Beam1 poin 'DROI' P0 P1 1.e-5 ;
Line2 = (Beam1 aret) elem 'APPU' 'STRI' PtsL2 ;
trac 'QUAL' ((Line2 coul 'ROUG') et (aret beam1))
     'TITR' 'Beam edges and Line2 (red)' ;
EVUZ1 = EVOL 'VERT' 'CHPO' DEP1 'UZ' Line2 ;
List 'RESU' EVUZ1 ;
tleg1 = tabl ;
tleg1 . 1 = mot 'MARQ S CROI' ;
tleg1 . titre = tabl ;
tleg1 . titre . 1 = 'Uz (m)' ;
titr 'Evolution of the displacements along Line2' ;
dess EVUZ1 'LEGE' tleg1 'XBOR' 0. 2.1 'XGRA' 0.25 'GRIL' 'POIN' 'GRIS' ;
OPTI 'SAUV' 'beam1.sauve' ;
SAUV ;
SAUV Beam1 ;
OPTI 'REST' 'beam1.sauve' ;
REST ;
FIN ;
```

### 2.2.2 Highlighting Gibiane

The script presented in Listing 2.1 is quite dense. To make it easier to read, you can add blank lines to separate different parts. You can also add comment lines by placing the character "*" at the beginning of the line. Another way of placing comments is to use the operator **COMM**ent. In this case, do not forget to end your comment line with a semicolon ";" to make the operator **COMM**ent working.

Figure 2.15 shows a more readable version of "beam1.dgibi", also cleared of most calls to the directive **LIST**. This picture is a screenshot of the script file opened with Notepad++ text editor, enhanced with a syntax highlighting format available on Cast3M website.

```
1  *-------------------------------------------------------------*
2  * Meshing a rectangular beam in 3 dimensions.
3  * Beam size : L1 x H1 x W1
4  *-------------------------------------------------------------*
5
6  * Space dimensions and FE type :
7  OPTI DIME 3 ELEM CUB8 ;
8
9  * Beam dimensions :
10 w1    = 0.06 ;
11 H1    = 0.1  ;
12 l1    = 2    ;
13
14 * Switch to english language
15 OPTION LANGUAGE ANGLAIS ;
16
17 * Points definition :
18 P0 = 0. 0 0 ;
19 P1 = L1 0 0 ;
20 P2 = 0 W1 0 ;
21 P3 = 0 0 H1 ;
22
23 * Mesh definition :
24 Line1 = P0    droi 3 P2 ;
25 Surf1 = Line1 TRAN 5 P3 ;
26 Beam1 = Surf1 VoLu TRAN 50 P1 ;
27
28 * Short listing of mesh beam1 :
29 List Resu beam1 ;
30
31 * Display :
32 TRAC CACH Beam1 titre 'First mesh with Cast3M !' ;
33
34 FIN ;
35
```

Figure 2.15: A more readable version of "beam1.dgibi". Screenshot of the file opened in Notepad++ text editor with syntax highlighting.

### 2.2.3 Executing a script file

To execute a script file with Cast3M, for example "mymodel.dgibi", type in a command window:

```
castem24 mymodel.dgibi
```

It should also work without the file extension:

```
castem24 mymodel
```

When executing an input file, the log file extension is ".trace", *i.e.* "mymodel.trace". A Cast3M log file can be executed and the results must be the same as with the initial input file.

You can stop the execution of a file by inserting the following statement in the script:

```
OPTI 'DONN' 5 ;
```

When reading this line, Cast3M stops decoding the script and hands over to the user in the CLI. One can then check the results of previous statements, for example using the operator **LIST** or **TRAC**, and correct the script if necessary, after trying the correction in the CLI.

To resume execution, type:

```
OPTI 'DONN' 3 ;
```

Cast3M then starts reading statements in the script file again, just after the line `OPTI DONN 5`. You can insert as many `OPTI DONN 5` statements as you need.

**Note** The meaning of `OPTI DONN 5` comes from the Fortran 77 language, where "UNIT 5" is the keyboard. `OPTI DONN 3` comes from the fact that the input file is initially copied and read on the "UNIT 3" (fort.3 file).

### 2.2.4 Printing graphical displays to a PostScript file

You can redirect the graphical displays to a PostScript file using the following Gibiane instruction:

```
OPTI 'TRAC' 'PSC' ;
```

The keyword `'PSC'` stands for "PostScript Color".

By default, the name of the PostScript file is the same as the ".dgibi" file but with the extension ".ps". It is also the file name of the outputs generated by clicking the SoftCopy button on the display window (see Figure 2.8 page 17).

You can specify the PostScript file name with the option `'FTRA'`. For example, the following statement exports the graphical displays into the file "`pictures.ps`":

```
OPTI 'FTRA' 'pictures.ps' ;
```

You must set the `'PSC'` option before using the `'FTRA'` option, however this instruction goes in error. You can define as many PostScript files as you like by changing the file name from one display to another, but it is good practice to group all your post- processing images into a single PostScript file.

The default value of the `'TRAC'` option is "`'X'`" in reference to the X Window System of the X11 library used in Cast3M for graphical displays. You can alternate between displaying in the X Window or in a PostScript file by switching the `'TRAC'` option between `'X'` and `'PSC'`.

Lastly, the OpenGL library is also available in Cast3M for graphical displays. You can use it by setting the `'TRAC'` option to `'OPEN'`. Give it a try!

Figures 2.16 and 2.17 illustrate the rendering of PostScript outputs with two examples taken from the script file `beam1.dgibi`. The first one is the contour plot of the stress field presented in Figure 2.12 page 23. It should be noted that the PostScript format is better suited for presentation in a report than screenshots taken from similar displays. In particular, the isovalue legend is easier to read.

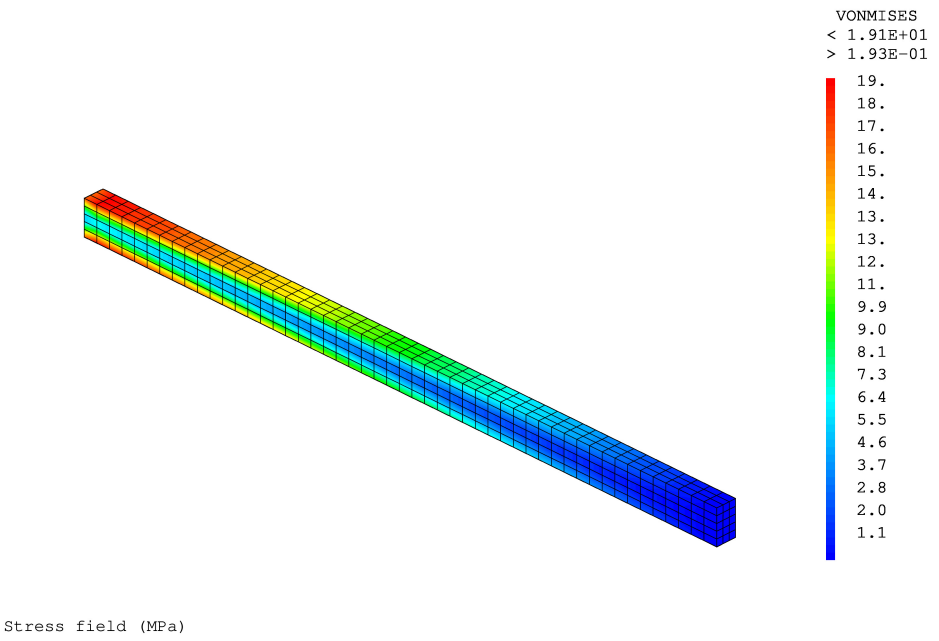



Figure 2.16: Contour plots of the stress field (MPa) (`beam1.dgibi`).

Figure 2.17 illustrates the plot of an `EVOLUTIONn` object with the directvive **`DESS`** in PostScript format. This example is taken form the display presented in Figure 2.13 page 25.

There are many tools to convert PostScript files in other image formats. On Linux system, you can use `psselect` to extract a page of a PostScript file and `convert`, a function of ImageMagik, to convert it to another format. You can also convert a PostScript file to PDF using `ps2pdf`. Cross-platform ports of these tools are also available for Windows.

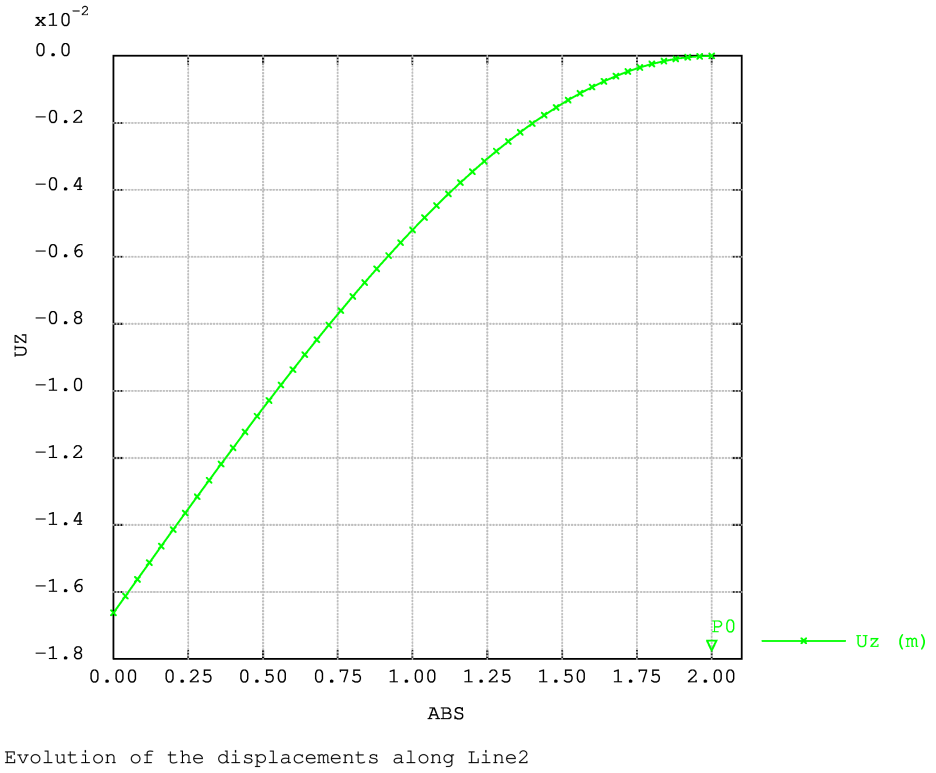



Figure 2.17: Evolution of the $U_z$ component of displacement along **`Line2`** (`beam1.dgibi`).



# Chapter 3

# Nonlinear static analysis with PASAPAS

This chapter is divided into three parts. In the first one, you will learn more about Gibiane in order to acquire the knowledge needed in the following: basic object types, conditonnal and iterative expressions (loops), and Gibiane procedures. The two following parts focuse on the use of Cast3M. The second one presents an example of nonlinear static analysis through the computation of a reinforced concrete beam subjected to three-point bending. The third and last part explains how to execute Gibiane instructions during a calculation with the procedure PASAPAS.

## 3.1 More about Gibiane

### 3.1.1 Basic object types

**"LOGIQUE" type object (logical)**

As in other computer languages, a LOGIQUE can only take two values: "VRAI" (true) or "FAUX" (false). To define **LOG1** as a variable of type LOGIQUE, simply set it to VRAI or FAUX:

```
LOG1 = VRAI ;
LOG1 = FAUX ;
```

**"MOT" type object (word)**

A "MOT" is a character string. Its length is currently limited to 500 characters. To define **MOT1** as a variable of type MOT, you can give its value into single quotes or use the operator **MOT**:

```
MOT1 = 'Hello' ;
MOT1 = MOT hello ;
MOT1 = MOT 'Hello' ;
```

The operator **MOT** avoids using quotes but, without quotes, the character string is converted to uppercase, while the case of letters is conserved within the quotes (**LIST** the result in each case). You can combine both syntaxes.

The operator **MOT** is mandatory when you want to define a MOT which value is the same as a function name of Cast3M, for example:

```
MOT1 = MOT MOT ;
```

to assign to **MOT1** the value MOT.

When the character string contains spaces, use the syntax with quotes:

```
MOT1 = 'Hello world !' ;
```

You can print messages with the directive **MESS**age:

```
MESS MOT1 ;
```

**MESS** can combine MOT variables with ENTIER and FLOTTANT. It is useful for printing values when your script is running or simply to inform you that your test case succeeded before exiting Cast3M.

Lastly, you can concatenate several MOT with the operator **ET** (place the operator fisrt) or with the operator **CHAI**n. This last operator offers format options when combining MOT with ENTIER and FLOTTANT (see its manual page).

### LISTENTI type object

A LISTENTI is a list of integers. To define **LENT1** as a variable of type LISTENTI, use the operator **LECT**:

```
LENT1 = LECT ;
```

In this case, **LENT1** contains no value. To initialize **LENT1** with a given set of values, simply add them to **LECT**, for example:

```
LENT1 = LECT 1 2 3 ;
```

You can define a LISTENTI containing *n* times the same value with the option "*". You can also define a LISTENTI containing values increasing or decreasing according to a given step with the option "PAS" (step):

```
LENT1 = LECT 5 * 1 ;       COMM defines a list of 5 times the number 1 ;
LENT2 = LECT 1 'PAS' 2 5 ; COMM defines the list of numbers {1 3 5} ;
```

You can mix the different options, see the manual page of **LECT**.

To concatenate two LISTENTI or add an ENTIER to a LISTENTI, use the operator **ET**. Lastly, to get the dimension of a LISTENTI (number of values in the list), use the operator **DIME**nsion.

### LISTREEL type object

A LISTREEL is a list of real numbers. To define **LREE1** as a variable of type LISTREEL, use the operator **PROG**, for example:

```
LREE1 = PROG ;                  COMM initialize LREE1 as a LISTREEL ;
LREE1 = LREE1 ET (PROG 0. 1.) ; COMM add 0. and 1. to LREE1 ;
LREE1 = LREE1 ET 2. ;           COMM add 2. to LREE1 ;
LREE2 = PROG 0. 1. 2. ;         COMM the same list as LREE1 ;
LREE3 = PROG 0. 'PAS' 1. 2. ;   COMM idem ;
LREE4 = PROG 3 * 1. ;           COMM a list of 3 times the number 1. ;
LREE5 = LREE1 ET LRRE4 ;
```

**PROG** works such as **LECT** but it as more options, see its manual page. The operators **ET** and **DIME** work on a LISTREEL as for a LISTENTI.

### LISTMOTS type object

A LISTMOTS is a list of words. To define **LMOT1** as a variable of type LISTMOTS, use the operator "**MOTS**" (words), for example:

```
LMOT1 = MOTS 'Hi' 'Salut' 'Ola' 'Ciao' 'Hallo' ;
LMOT2 = MOTS 3 * 'YES' 2 * 'NO' 'PLUS' ;
LMOT3 = LMOT1 ET LMOT2 ;
```

The words of the list do not need to be the same length. The operators **ET** and **DIME** work on a LISTMOTS as for a LISTENTI.



#### TABLE type object

As already mentioned in chapter 2, use the operator **TABL** to define a variable of type TABLE. You can put any type of variable in a TABLE and it can also be indexed by any type of variable. As a consequence, you can put a TABLE in a former one to define a subtable. Here is an example:

```
TAB1  = TABL ;
TAB1 . 'TITLE'   = 'My PhD' ;
TAB1 . 'CHAPTER' = TABL ;
TAB1 . 'CHAPTER' . 1 = 'Introduction' ;
TAB1 . 'CHAPTER' . 2 = 'Bibliography' ;
TCHAP3  = TABL ;
TCHAP3  = 'Exeprimental study' ;
TAB1 . 'CHAPTER' . 3 = TCHAP3 ;
```

This example uses only MOT and ENTIER as indices, and **TAB1** and its subtables contain only data of type MOT.

The operators **ET** and **DIME** work on a TABLE as for a LISTENTI. You can also get all the indices of a TABLE using the operator **INDE**x (see its manual page).

#### Operator TYPE

If you want to know the type of a variable **OBJ1**, use the operator **TYPE**:

```
MOT1 = TYPE OBJ1 ;
```

### 3.1.2 Conditional expressions

You can program conditional expessions with the directives "**SI**" (if), "**SINO**n" (else) and "**FINS**i" (endif):

```
SI VRAI ;
  MESS 'You will read this message' ;
SINON ;
  MESS '& not this one' ;
FINSI ;
```

Remember that you can add extra characters in function names to make the code more readable (here, **SINON** and **FINSI**).

You can define conditions with comparison and logical operators. The Table 3.1 presents the common ones. When comparing objects of type MOT, put the operator first in the statement. You must compare object of the same type, otherwise the result is not guaranteed.

Table 3.1: Common comparison and logical operators.

| | < | ≤ | = | ≠ | ≥ | > | AND | OR | NOT |
|---|---|---|---|---|---|---|---|---|---|
| Cast3M | < | <EG | EGA | NEG | >EG | > | ET | OU | NON |

### 3.1.3 Iterative expressions

You can program iteration loops with the directives **REPE**at and **FIN**. Loops are identified by a name, freely chosen by the user, and you can specify the number of iterations. Call in a loop, the directive **ITER**ate jumps to the next iteration and the directive **QUIT** leaves the loop. Here is an example:

```
N1 = 10 ;
REPE Loop1 N1 ;
  SI (&Loop1 MULT 2) ; ITER Loop1 ; fins ;
  MESS 'Iteration No:' &Loop1 ;
  SI (&Loop1 ega  7) ; QUIT Loop1 ; fins ;
FIN Loop1 ;
```

In these statements, the loop name is **Loop1** and the number of iterations is **N1**, initialized to 10 before the loop start statement. If the number of iteration is 0, Cast3M ignore the loop block. If you do not specify the number of iteration, the loop is infinite. Attached to the loop name, the character "&" forms the loop counter (here, **&Loop1**).

In the first conditional expression of the loop, Cast3M jumps to the next iteration if the loop counter is a multiple of 2 (**MULT**). If not, it prints a message giving the current iteration number. In the second conditional expression, Cast3M leaves the loop is the loop counter is equal to 7.

Remember to specify the loop name to the directive **FIN**, otherwise you leave the program. The value of naming loops becomes evident in nested loops. Note the conditional expressions defined on a single line. Of course, this is not mandatory.

### 3.1.4 Gibiane procedures

You can define Gibiane subtoutines, called PROCEDUR. PROCEDURs are freely named by the user. A PROCEDUR is delimited by the directives **DEBP** and **FINP**. The PROCEDUR name is given to the directive **DEBP**, together with its input data. Input data are Gibiane variables which type can be specified. Output data are specified to the directive **FINP**. Here is an exemple with the function factorial:

```
DEBP FACTORIAL N1*'ENTIER' LOG1/'LOGIQUE' ;
* Initialization of LOG1 if it do not exist:
  SI (NON (EXIS LOG1)) ; LOG1 = FAUX ; FINS ;
* If LOG1 is true : print message
  SI LOG1 ;
    MESS 'Call to FACTORIAL: N =' N1 ;
  FINS ;
* Recursive call to FACTORIAL to compute the result:
  SI (N1 > 0) ;
    RES1 = N1 * (FACTORIAL (N1-1) LOG1) ;
  SINO ;
    RES1 = 1 ;
  FINS ;
FINP RES1 ;
```

In the first statement, you can see that the input type is attached to the variable: **N1*'ENTIER'** and **LOG1/'LOGIQUE'**. When the type is specified with the symbol "*", the argument is mandatory. When it is specified with the symbol "/", it is optional. Here, this last possibility is used to print a message when entring the PROCEDUR in order to follow the recursive calls.

Arguments can also be read by the directive **ARGU** and the result written by the directive **RESP**. They can be called anywhere in the PROCEDUR. They enable conditional management of input/output data. You can leave a PROCEDUR by calling the directive **QUIT**, as for the loop statements, and error management can be achieved with the directive "**ERRE**ur" (error).

You can program a PROCEDUR in a separate file, with extension name ".procedur". In our example, it would be "factorial.procedur". To make a PROCEDUR available to Cast3M, place its file in the directory where Cast3M is executed, or in a subdirectory named "./procedur", or in the installation directory of

your Cast3M version.

Figure 3.1 shows the use of the PROCEDUR **factorial**, defined in the file "factorial.procedur", when running Cast3M in interactive mode in the same directory. The PROCEDUR **factorial** is called twice: once by activating the option to print a message, once without. You can see that one use a PROCEDUR as any other Cast3M function. It is easy to check that the result is correct with **LIST** (`fac4 = 24`). The last statement applies the directive **LIST** to **factorial**. You can see that **LIST** displays the content of the PROCEDUR, as for any object in Cast3M.

```
 $ fac4 = factorial 4 VRAI ;
 * fac4 = factorial 4 VRAI ;
Call to FACTORIAL: N =4
Call to FACTORIAL: N =3
Call to FACTORIAL: N =2
Call to FACTORIAL: N =1
Call to FACTORIAL: N =0
 $ list fac4 ;
 * list fac4 ;
 Entier valant: 24
 $ fac4 = factorial 4 ;
 * fac4 = factorial 4 ;
 $ list fac4 ;
 * list fac4 ;
 Entier valant: 24
 $ list factorial ;
 * list factorial ;
  Procedure courante de Cast3M.
      Liste de la PROCEDURE
      *********************
    1  DEBP FACTORIAL N1*'ENTIER' LOG1/'LOGIQUE' ;
    2  * Initialization of LOG1 if it do not exist:
    3  SI (NON (EXIS LOG1)) ; LOG1 = FAUX ; FINS ;
    4  * If LOG1 is true : print message
    5  SI LOG1 ;
    6    MESS 'Call to FACTORIAL: N =' N1 ;
    7  FINS ;
    8  * Recursive call to FACTORIAL to compute the result:
    9  SI (N1 > 0) ;
   10    RES1 = N1 * (FACTORIAL (N1-1) LOG1) ;
   11  SINO ;
   12    RES1 = 1 ;
   13  FINS ;
   14  FINP RES1 ;
$
```

Figure 3.1: Execution of the PROCEDUR **factorial**.

### 3.1.5 Gibiane notices

You can document a PROCEDUR in a manual page, called NOTICE[1]. A NOTICE is a text file that usually bears the name of the PROCEDUR it documents, but truncated to 8 characters[2]. The NOTICE file extension must be ".notice". For the PROCEDUR `factorial`, it should therefore be "factoria.notice".

A NOTICE has a header starting with "$$$$", followed by the name of the function, and the file must end with "$$$$" (last line). It should contain:

- the name of the function as title;

[1] Indeed, "manual page" is our translation of "notice".

[2] This 8-character limitation has been lifted for variable names but not yet for NOTICE. It is drawback since the NOTICE name may not be the same as the name of the PROCEDUR. One solution is to limit the PROCEDUR name to 8 characters.

- a functionnal specification;
- a description of the function;
- a description of the input and output data, specifying their type.

To start writing a NOTICE, you can take example on an existing one. They are all supplied in the install directory of Cast3M.

You can manage sections written in different languages using tags. The tag for a section in french is "FRAN====", and "ANGL====" for a section in english. In Cast3M, every function is documented in both french and english. Informative and error messages are also available in these two languages.

The first four characters of the tag is the key word given to the directive OPTI to select the language. As a consequence, you can define sections in any other language by defining a key word for it. For example, if you want to write manual pages in spanish, you could choose the key word ESPA (for español), and the statement:

```
OPTI 'LANG' 'ESPA' ;
```

will make the directive INFO displays the sections tagged with "ESPA====".

The tag "========" defines sections which are displayed whatever the language option is. Functional specifications has to be written in such sections. Here is an example of what the NOTICE of the PROCEDUR factorial could be:

```
$$$$ FACTORIA
FRAN====
  Procedure FACTORIAL
  -------------------

  Syntaxe :
  ---------

ANGL====
  FACTORIAL procedure
  -------------------

  Syntax :
  --------

========
  ENT2  =  FACTORIAL  ENT1  (LOG1) ;

FRAN====
  Objet :
  -------
  La procedure FACTORIAL...

  Commentaire :
  -------------
  ENT1   =   ENTIER dont on veut calculer la factorielle
  LOG1   =   ...

ANGL====
  Description:
  ------------
  The FACTORIAL procedure...
```

```
  Contents:
  ---------
  ENT1    =    ENTIER to compute the factorial
  LOG1    =    ...

$$$$
```

```
$ info factoria ;
* info factoria ;
       Procedure FACTORIAL
       -------------------

       Syntaxe :
       ---------

       ENT2  =  FACTORIAL  ENT1  (LOG1) ;

       Objet :
       -------
       La procedure FACTORIAL...

       Commentaire :
       -------------
       ENT1   =   ENTIER dont on veut calculer la factorielle
       LOG1   =   ...

$ opti lang angl ;
* opti lang angl ;
$ info factoria ;
* info factoria ;
       FACTORIAL procedure
       -------------------

       Syntax :
       --------

       ENT2  =  FACTORIAL  ENT1  (LOG1) ;

       Description:
       ------------
       The FACTORIAL procedure...

       Contents:
       ---------
       ENT1   =   ENTIER to compute the factorial
       LOG1   =   ...

$ opti lang chineese ;
* opti lang chineese ;
$ info factoria ;
* info factoria ;
       ENT2  =  FACTORIAL  ENT1  (LOG1) ;

$
```

Figure 3.2: Display of the NOTICE factoria by the directive INFO in french, english, and chineese.

Figure 3.2 shows the display of the NOTICE factoria with the directive INFO in french, english, and chineese. For french and english, the directive INFO prints the sections identified by the tags for these languages. For chineese, it only prints the functional specification, identified by the tag "========" since there is no tag defined for this language.

Finally, you can structure a NOTICE with chapters and parts using the CHAP and PART tags. To see how to use them, take the NOTICE of the operators MANU, CHAN, MODE, or MATE as examples.

## 3.2 Example of nonlinear static analysis

In this part, we are going to see how to perform a nonlinear static analysis with Cast3M. As an example, we resume the elastic beam presented in the previous chapter to model a reinforced concrete beam submitted to three-point bending.

### 3.2.1 Presentation of the problem

Figure 3.3 presents a diagram of a reinforced concrete beam subjected to three-point bending. The diagram on the left represents a front view of the beam, into which the red dashed lines represent the reinforcing bars, or "rebars", embedded in the concrete. On the right, the diagram represents the beam section, in which the rebars appear as red circles.

The beam dimensions are: $l_b = 2$ m, $h_b = 0.1$ m and $w_b = 0.06$ m. Rebars are placed symmetrically in relation to the height and width of the beam (see the beam section diagram). The distance between the top (or bottom) surface of the beam and the reinforcing bars is noted $h_r = 0.01$ m.

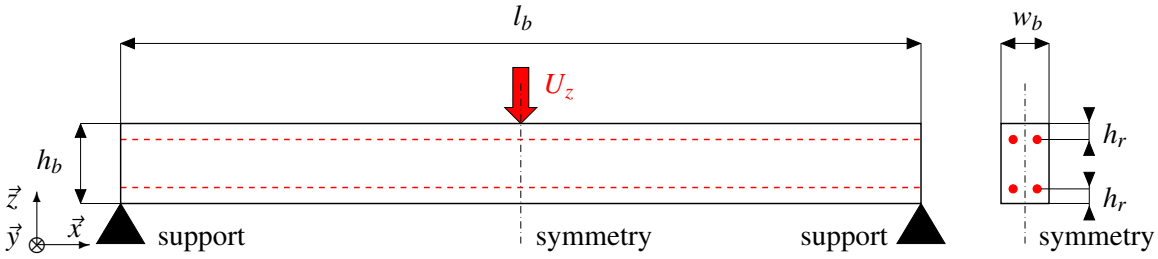



Figure 3.3: Diagram of a reinforced concrete beam subjected to three-point bending. On the left : front view of the beam, on the right: beam section.

A displacement $U_z$ is imposed in the middle of the beam's upper surface, while lower supports are placed at its two ends. Thanks to the symmetry of the problem, we only need to model one quarter of the beam geometry: a half-length and a half-width.

The damage behavior of concrete is assumed to follow Mazars' constitutive law, while the mechanical behavior of rebars is assumed to be elastoplastic, with a linear isotropic hardening.

The analysis will be carried out in the frame of "large displacement" as an example, although the small displacement assumption is usually sufficient in static analysis of concrete beams.

### 3.2.2 Definition of some parameters

Let's start modeling our problem. After a short heading of comments, we define the space dimension and the finite element type to use by default using the directive **OPTI**on:

```
* fichier : beam3.dgibi
*-----------------------------------------------------------------------*
* Reinforced concrete beam subjected to 3 points bending
*-----------------------------------------------------------------------*
* Space dimension and finite element type:
OPTI 'DIME' 3 'ELEM' 'CUB8' ;
```

Next, we define some model parameters, mainly relative to geometry:

```
* --- 1. Model parameters
* Beam geometry parameters (in meters):
```



```
lb1 =  1.00 ;
hb1 =  0.10 ;
wb1 =  0.03 ;
hr1 =  0.01 ;
sr1 =  0.5e-4;
uz1 = -0.0015 ;

* Mesh density:
df1  = 0.01 ;
dc1  = 0.10 ;
```

Among these parameters, we recognize the dimensions presented in Figure 3.3. However **lb1** et **wb1** are only half of $l_b$ and $w_b$, respectively, since we only model one quarter of the problem. The value of the displacement $U_z$ is specified: -1.5 mm. Parameter **sr1** is the cross-sectional area of a rebar. We also define a "fine" and "coarse" mesh density, **df1** and **dc1** respectively.

### 3.2.3 Meshing

To mesh the beam and the rebars using the above parameters, it is useful to define a frame of coordinates:

```
* --- 2. Meshing
* Frame:
o1  = 0 0 0 ;
x1  = 1 0 0 ;
y1  = 0 1 0 ;
z1  = 0 0 1 ;
```

Then, let's mesh the beam first:

```
* Beam mesh:
l1  = o1 DROI (wb1*y1) 'DINI' df1 'DFIN' df1 ;
s1  = l1 TRAN (hb1*z1) 'DINI' df1 'DFIN' df1 ;
v1  = s1 VOLU 'TRAN' (0.5*lb1*x1) 'DINI' dc1 'DFIN' dc1 ;
s2  = v1 FACE 2 ;
v2  = s2 VOLU 'TRAN' (0.5*lb1*x1) 'DINI' df1 'DFIN' df1 ;
v0  = v1 ET v2 ;
```

In these instructions, we start to mesh the line **l1** from the origin **o1**, to a point located at the distance **wb1** in the (O,y) direction. Remember that **o1**, **x1**, **y1** and **z1** are points, so you can see that the operator * also works for the product between a scalar and a vector.

The number of elements in **l1** is determined by the mesh density, actually the element size, which can vary from the initial point of the line, whose density is defined by **'DINI'**, to the final point, whose density is defined by **'DFIN'**. Here, the value of the density is the same at both ends of the line and equal to **df1**. Therefore, the element size is homogenoeous in **l1** and their number approximately equal to **wb1/df1**, since the length of **l1** is equal to **wb1**.

Next, the line **l1** is translated in the (O,z) direction over the distance **hb1** to create the surface **s1**, using the mesh density **df1**. Finally, the surface **s1** is translated in the (O,x) direction over half of the length **lb1** to define a first volume **v1**, using the coarse mesh density **dc1**.

To create the second half of the beam mesh length, we first retrieve the face at the extremity of **v1** with the operator **FACE**, and name this surface **s2**. The volume mesh **V2** is obained by translating **s2** in the same way as **V1**, but using the fine mesh density parameter **df1**. Volume **v1** and **v2** are finally added together with the operator **ET** to form the concrete beam mesh **v0**.

Now, let's mesh the rebars:

```
* Rebars mesh:
pr1 = (0.5*wb1*y1) PLUS (hr1*z1) ;
pr2 = pr1 PLUS (lb1*x1) ;
lr1 = pr1 DROI pr2 'DINI' (0.97*df1) 'DFIN' (0.97*df1);
lr2 = lr1 PLUS ((-2. * hr1 + hb1)*z1) ;
lr0 = (lr1 ET lr2) COUL 'ROUG' ;
TRAC (100 -50 20) ((ENVE V0) ET lr0)
  'TITR' 'Meshes of the concrete beam and the rebars (red lines)' ;
```

In the first instruction here-above, we create the point **(0.5*wb1*y1)** at mid-width of the beam in the (O,y) direction, then add the vector **(hr1*z1)** to define the point **pr1**. Operator **PLUS** defines the addition in the affine space of coordinates. It also acts on other types of objects, such as "MAILLAGE" (mesh) (see its manual page).

Point **pr2** is defined in the same way to be located at the other end of the beam in the (O,x) direction. Then the line **lr1** is created between these two points to define the mesh of the rebar located at the bottom of the beam. Note that the density is slightly modified **(0.97*df1)**, so that the rebar nodes are not conforming the beam elements.

The mesh of the second rebar **lr2** is obtained by adding the vector **((-2.*hr1+hb1)*z1)**[3] to the mesh **lr1**, $(h_b - 2h_r)$ being the distance between the two rebars in the (O,z) direction. Lines **lr1** and **lr2** are added together to define the entire mesh of the rebars **lr0**, which is colored in "ROUGe" (red) with the operator "**COUL**eur" (color).

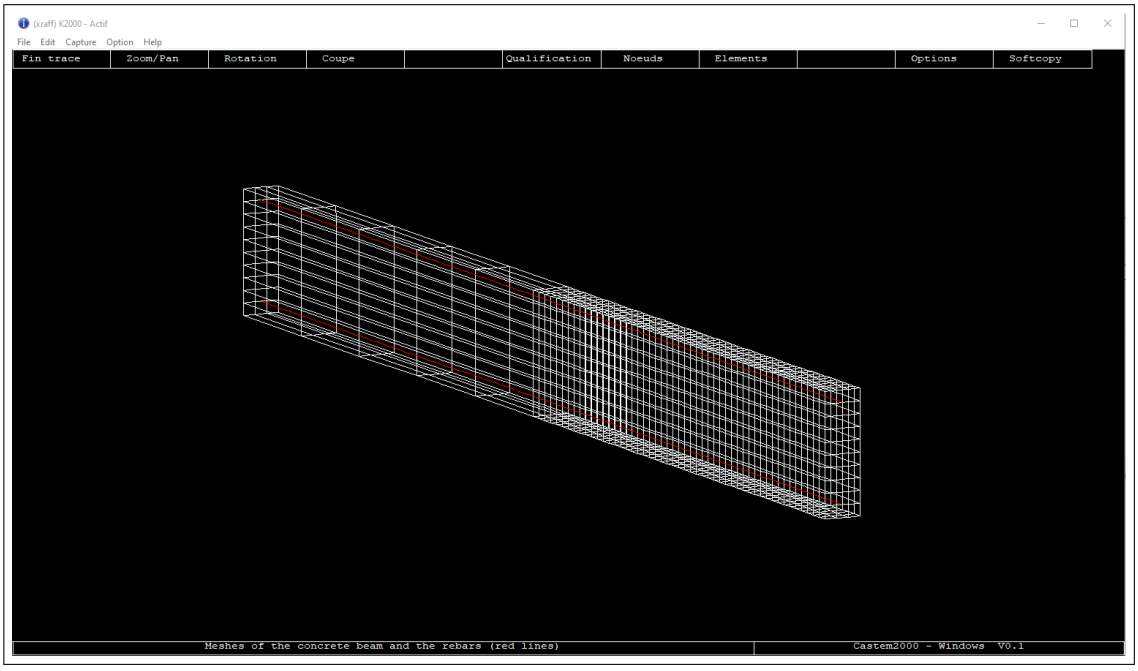



Figure 3.4: Meshes of the beam and the rebars.

Finally, the beam and rebars meshes are displayed using the directive **TRAC**. We only display the enveloppe of the beam mesh, using the operator **ENVE**, in order to better see the rebars inside. The point `(100 -50 20)` given to **TRAC** is the viewpoint of the drawing. Figure 3.4 presents the result of

[3]Note that **hr1** is first multiplied by **-2** before adding to **hb1**, as Gibiane does not respect the priority of arithmetic operations.



this display.

### 3.2.4 Nonlinear mechanical behavior

#### Model definition

Once the structure has been meshed, we need to specify the physic that applies to it by defining a model with the operator **MODE**. The input data of **MODE** are mainly[4]:

- a mesh (MAILLAGE),
- one or more keywords (MOT), that define the physical model,
- for structural elements (beam, shells...), the name of the structural element (MOT),
- eventually, a constituent name (MOT).

For more details, see the manual page of **MODE**.

The model definitions for the beam and rebars are as follows:

```
* --- 3. Models & Material characteristics
* Concrete and rebars mechanical models:
mod1 = MODE V0  'MECANIQUE' 'ELASTIQUE' 'ENDOMMAGEMENT' 'MAZARS'
                'CONS' 'CONCRETE' ;
mod2 = MODE lr0 'MECANIQUE' 'ELASTIQUE' 'PLASTIQUE' 'ISOTROPE' 'BARR'
                'CONS' 'REBARS' ;
* Model addition:
mod0 = mod1 ET mod2 ;
```

The concrete beam has an elastic-damage mechanical behavior according to the Mazar's constitutive law, as defined in the model **mod1**. The constituent name "concrete" is given to **mod1** using the **'CONS'** option of **MODE**. This is useful when you want to **EXTR**act a particular model from a global one.

The rebars have an elastoplastic mechanical behavior with a linear isotropic hardening, as defined in the model **mod2**. Its contituent name is "rebars". For the rebar model, we also need to specify the structural behavior of the one-dimensionnal finite elements. This is the meaning of the keyword "**'BARR'**" (truss): BARR elements only work in tension or compression ; they have no flexural stiffness. Lastly, the two models are added together with the operator **ET** to define a "global" model **mod0**.

#### Material definition

The material characteristics of models are defined with the operator **MATE**. The input data of **MATE** are mainly:

- a model (MMODEL),
- one or more keywords (MOT), that are the model parameters, generally related to a physical propery (Young's modulus, thermal conductivity...),
- the value of each model parameter, that can be of various types: a real number (FLOTTANT), a tensile curve or a temperature dependent propery (EVOLUTIOn), a tabulated propery according to different parameters ("NUAGE" (cloud)), a vector defining a direction of anisotropy (POINT), etc.

[4] Cast3M object type is indicated in parentheses.

For more details, see the manual page of **MATE**.

For the Mazars behavior law, they are the followings:

```
* Concrete mechanical characteristics: Mazars behavior law parameters:
ym1   = 31.e9 ;
nu1   = 0.2 ;
epsd1 = 1.e-04 ;
acom1 = 1.2 ;
bcom1 = 1500. ;
atra1 = 1.0 ;
btra1 = 10000. ;
beta1 = 1.06 ;
mat1 = MATE mod1 'YOUN' ym1 'NU' nu1 'KTR0' epsd1 'ACOM' acom1
                'BCOM' bcom1 'ATRA' atra1 'BTRA' btra1 'BETA' beta1 ;
```

In these instructions, we first define the numerical values of the Mazars model parameters, then we call the operator **MATE** to build the element field of material characteristics **mat1** associated with **mod1**.

The material characteristics of the rebars are those of a common steel. For the elastoplastic law of **mod2**, the plastic behavior is fully determined by the hardening curve of the material, which is the curve $\sigma(\varepsilon^p)$. Here, we simply represent the hardening curve as a straight line with a slope of about 1 GPa, and we consider a yield stress of 210 GPa:

```
* Rebars mechanical characteristics: elastoplastic with hardening curve:
ecro1 = EVOL 'VERT' 'MANU'  (PROG 0.     1.)
                    'ECRO1' (PROG 210.e6 1.e9) ;
DESS ecro1 'TITR' 'Hardening curve of rebars' ;
mat2  = MATE mod2 'YOUN' 210.e9 'NU' 0.3 'ECRO' ecro1 'SECT' sr1 ;

* Material characteristics addition:
mat0  = mat1 ET mat2 ;
```

In these instructions, we first define the hardening curve by creating an object of type "EVOLUTIOn" **ecro1**, using the operator **EVOL**. Next, we draw this curve with the directive **DESS**. Then, we call the operator **MATE** to define the material characteristics **mat2** associated with **mod2**. Note that the definition of **mat2** also requires the cross-section of the BARR elements **sr1**. The last instruction adds together **mat1** and **mat2** to build the global field of material charcteristics **mat0**.

### 3.2.5 Boundary conditions

In Cast3M, there are two main operators to prescribe boundary conditions or, more generally, kinematic constraints: **BLOQ**uer and **RELA**tion. Their input data are mainly (but not only):

- one or more meshes (MAILLAGE), where the constraints apply,
- one or more keywords (MOT), to specify the name of the unknowns concerned (UX, T...),
- other keywords (MOT) for options.

For more details, see their manual pages.

The boundary conditions are the followings:

- **bc1**: $U_z = 0$ at the nodes of the beam mesh in contact with the lower support,
- **bc2**: $U_y = 0$ at all nodes of the beam mesh with coordinate $y = 0$, to represent the plane of symmetry in the (O,y) direction,



- **bc3**: $U_x = 0$ at the end of the beam mesh opposite to the lower support, to represent the plane of symmetry in the (O,x) direction,
- **bc4**: $U_z =$ Uz1 at the nodes where displacement is imposed,
- **bc5**: $\vec{U}$(rebars) = $\vec{U}$(beam) at all points of the rebars, in order to represent the embedding of the rebars into the beam.

#### Boundary condition bc1

```
* --- 4. Boundary conditions and loading
* Support boundary conditions at one end of the beam:
pz0  = (v0  COOR 3) POIN 'MINI' ;
pxz0 = (pz0 COOR 1) POIN 'MINI' ;
bc1  = BLOQ pxz0 'UZ' ;
TRAC ((pxz0 COUL 'ORAN') ET (ARET V0))
  'TITR' 'Lower support BC: Uz=0 on orange points' ;
```

The first instruction creates the node field of third coordinate on the beam mesh **v0**, then retrieves all the points for which the value is equal to the minimum, *i.e.* the points on the underside of the beam. The second instruction acts in the same way on the points **pz0**, to obtain the subset of points **pxz0** whose first coordinate is minimal.

Then, **bc1** is created by blocking the UZ component of displacement of the points **pxz0**. The last instruction displays the points **pxz0** colored in orange with the edges of the beam mesh **V0**. Figure **??** in appendix shows this display.

#### Boundary condition bc2

```
* Symmetry in the (O,y) direction:
py0  = (v0 COOR 2) POIN 'MINI' ;
bc2  = BLOQ py0 'UY' ;
trac ((py0 COUL 'ORAN') ET (ARET v0))
  'TITR' 'Mid-width beam symmetry: Uy=0 on orange points' ;
```

These instructions are quite similar as those for **bc1**. The first one retrieves all the nodes of the beam mesh at the minimal coordinate in the (O,y) direction, and the second block the UY component of the displacements of all these points. The last instruction displays the nodes involved in **bc2** in orange with the edges of the beam. Figure 3.5 shows this display.

#### Boundary condition bc3

```
* Symmetry in the (O,x) direction:
s3   = v2 FACE 2 ;
bc3  = BLOQ s3 'UX' ;
TRAC ((s3 COUL 'ORAN') ET (ARET v0))
  'TITR' 'Mid-length beam symmetry: Ux=0 on orange elements' ;
```

For **bc3**, the way to retrieve the mesh to apply the boudary conditions is somewhat different: we apply the operator **FACE** on the volume mesh **v2** to obtain the surface mesh **s3** at the end of the beam. **FACE** is useful when a volume is created by translating a surface: the initial surface defines the first face, the ending surface, the second face, the third face being the complement of these last two to form the envelope of the mesh.

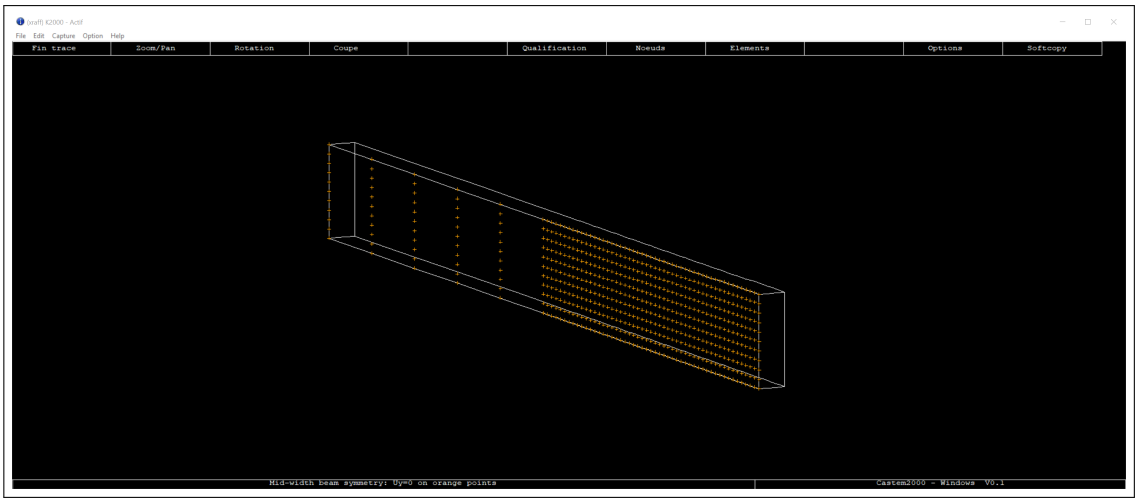



Figure 3.5: Display of the points where **bc2** is applied.

Then the component UX of displacements is blocked on the surface **s3**, to account for the symmetry of the problem in the middle of the "real" beam. The last instruction displays the surface mesh **s3** colored in orange with the edges of the beam. Figure **??** in appendix shows this display.

### Boundary condition bc4

```
* BC for applied displacements:
pxz1 = (S3 COOR 3) POIN 'MAXI' ;
bc4  = BLOQ 'UZ' pxz1 ;
TRAC ((pxz1 COUL 'ORAN') et (ARET v0))
   'TITR' 'Points (in orange) of applied displacements' ;
```

In these instructions, we first retrieve the nodes at the top of the surface **s3**, then block the component UZ of their displacements, and finally displays the points where **bc4** is applied. Figure **??** in appendix shows this display. The magnitude of the displacement is specified in the following.

### Boundary conditions bc5 and bc0

```
* Anchoring rebars in concrete:
bc5  = RELA 'ACCRO' lr0 v0 ;

* BC stiffnesses addition:
bc0  = bc1 ET bc2 ET bc3 ET bc4 ET bc5 ;
```

In the first instruction, the option "'ACCRO'cher" (hang) of the operator **RELA** defines a kinematic relation **bc5** between the unknowns of displacement of the rebar mesh **lr0** and the beam mesh **v0**. This relation is not symmetric: the nodes of the rebars are constrained to follow the displacements of the concrete beam, but it is not reciprocal. The displacement of the concrete beam at the point of the rebar nodes are interpolated using the shape functions of the mesh **v0**. Finally, we add all the boudary conditions into a global one **bc0** with the operator **ET**.

### 3.2.6 Mechanical loading

The stiffness **bc4** implies that the displacement UZ is zero at the points **pxz1**. To specify the non-null value **uz1**, we apply to **bc4** the operator **DEPI**, meaning "DEPlacement Imposé" (imposed displacement). **dbc4** is an object of type CHPOINT (node field), which is the mechanical loading of our problem:

```
* Mechanical loading: time evolution of the applied displacement
dbc4 = DEPI bc4 uz1 ;
mot1 = CHAI 'FORMAT' '(F6.2)'
       'Applied displacements: Uz =' (1000. * uz1) ' (mm)' ;
TRAC ((dbc4 EXCO 'FLX' 'UZ') VECT 'DEPL' 20. 'ROUG') (ARET V0) 'TITR'
   mot1 ;

ev1  = EVOL 'MANU' (PROG 0. 1.) (PROG 0. 1.) ;
cg1  = CHAR 'DIMP' dbc4 ev1 ;
```

The two following instructions concern the display. We first define the display title **mot1** by using the operator "**CHAI**ne" (character string). This operator is useful to specify the printing format of the digits: "(F6.2)" specifies the printing of a 6-digit floating-point number, with 2 digits after the decimal point (see Fortran printing format options). Note that **uz1** is multiplied by 1000 to convert it to millimeters.

Then, we display the vector of applied displacements with the directive **TRAC**. In this instruction, the operator **EXCO** extracts the component FLX of **dbc4** and rename it UZ. Then, the operator **VECT** creates an object of type "VECTEUR" (vector), of red color (**'ROUG'**) and with a magnification of 20. The keyword **'DEPL'** indicates to **VECT** that this is a displacement. This vector is displayed with the edges of the beam mesh **v0** and the title **mot1**. Figure 3.6 shows the display.

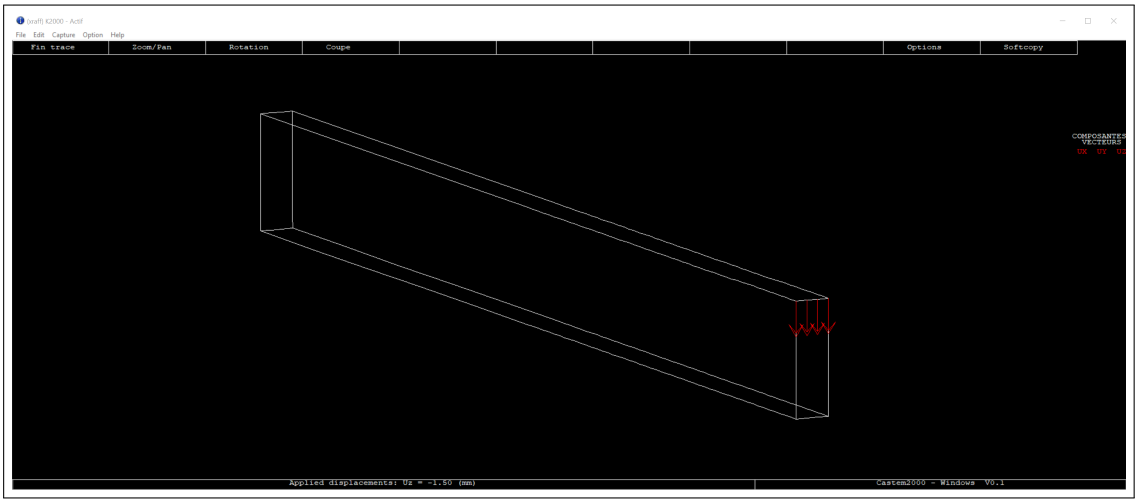



Figure 3.6: Display of the vector of applied displacements.

The two last instructions concern the time-evolution of the mechanical loading. For nonlinear problems that are not explicitly time-dependent, the mechanical loading is applied progressively according to a load parameter and a pseudo-time. We consider that the load parameter of applied displacement varies from 0 to 1 over a pseudo-time-period of 1. This time-evolution is defined by **ev1**, object of type EVOLUTIOn.

To modulate the mechanical loading **dbc4** over time, we define **cg1**, object of type CHARGEMEnt, using the operator **CHAR**. The keyword **'DIMP'** is the name of the CHARGEMENT and stands for "Déplacement IMPposé" (imposed displacement). This name is mandatory when using the **PASAPAS** solver.

### 3.2.7 Resolution with PASAPAS

#### About PASAPAS

"**PASAPAS**" (step-by-step) is a nonlinear solver of general purpose, mainly used for solving thermo-mechanical problems. It solves "step by step" a time-evolution problem according to a given time discretization. This may be "real time", as in transient thermal analysis, or a load parameter, as in some non linear mechanical analysis.

**PASAPAS** is a Gibiane procedure, made up of different sub-procedures. This means that it can be easily modified and overloaded by users, which is useful for improving the solver or developing new numerical methods in Cast3M.

**PASAPAS** accepts only one argument, a TABLE, containing the specification of the numerical problem to be solved and the choice of different options. Results are written in the input table. Thus, its functional syntax is quite simple:

```
PASAPAS TAB1 ;
```

You can distinguish input data from output data using the following syntax:

```
TAB2 = PASAPAS TAB1 ;
```

In this case **TAB1** and **TAB2** point on the same TABLE object.[5]

The entries of **TAB1** are defined by a name assigned to each input data. For example, the entry named "MODELE" contains the user defined model, for example **MOD1**:

```
TAB1 . 'MODELE' = MOD1 ;
```

All entries of the **PASAPAS** input TABLE are documented in the procedure manual page.

Few inputs are mandatory: a model (MMODEL), its characteristics (MCHAML), a loading (CHARGEMEnt) and the list of time-steps (LISTREEL). Boundary conditions are not mandatory since there may be no constraints applied to the primary unknowns, as it happens in dynamic or in transient thermal analysis. For more information, see the manual page of **PASAPAS**.

#### PASAPAS input table

Let us define the entries of the input table **tpas1** corresponding to our mechanical modelling:

```
* --- 6. Resolution with PASAPAS
* PASAPAS input table:
tpas1 = TABLE ;
tpas1 . 'MODELE'              = mod0 ;
tpas1 . 'CARACTERISTIQUES'    = mat0 ;
tpas1 . 'CHARGEMENT'          = cg1 ;
tpas1 . 'BLOCAGES_MECANIQUES' = bc0 ; COMM mech. boundary conditions ;
tpas1 . 'TEMPS_CALCULES'      = PROG 0. 0.5 'PAS' 0.05 1.0 ;
```

The names of the entries of **tpas1** are rather straightforward.

The entry **'TEMPS_CALCULES'** defines the list of time-steps (LISTREEL). In this list, the size of the time steps is not uniform. It has been defined after several trials an errors so as not spend too much

[5] This means that the same TABLE has two different names: either **TAB1** or **TAB2**.



time in the elastic part of the structure's response. In PASAPAS, there is no automatic procedure for reducing or increasing the time step during resolution.

We want to solve the problem in the frame of "large displacement". Given that PASAPAS assumes the "small displacement hypothesis" by default, we must set the 'GRANDS_DEPLACEMENTS (large displacement) option to VRAI:

```
* Large displacements analysis options:
tpas1 . 'GRANDS_DEPLACEMENTS' = vrai ;
tpas1 . 'LAGRANGIEN'          = MOT 'REACTUALISE' ;
tpas1 . 'PREDICTEUR'          = MOT 'HPP' ;
```

Large displacement analysis can be achieved using two formulations: "total lagrangian" or "updated lagrangian". Here, we chose the option 'REACTUALISE' (updated). Note the use of the operator MOT to protect the character string 'REACTUALISE' in order to avoid calling the operator REACtion[6], which calculates the reaction forces associated with some boundary conditions.

The last option 'PREDICTEUR' (predictor) set the use of the 'HPP'[7] method to improve the convergence in large displacement analysis. In this case, the predictor is simply an estimation of the solution in the frame of small displacements.

### Resolution

To solve our modelling with PASAPAS, we just have to type:

```
* PASAPAS procedure call:
PASAPAS tpas1 ;
```

In interactive mode, the procedure starts immediately. A screenshot of the PASAPAS procedure startup is shown at figure 3.7. First, a few lines summarizes the main calculation assumptions : "`Calcul MECANIQUE`" (mechanical analysis), "`*** Hypothese calcul des deformations...`" (deformation calculation assumption), etc.

Then the resolution of the first time-step starts. A header mentions the time-step number ("`Numero du pas`") and the time value at the end of the time-step ("`temps :  5.00000E-01`"). It is followed by different information about the resolution process printed in columns:

- "`Iter`": the iteration number.
- "`Nplas`": the number of Gauss points that have met the plastic criterion.
- "`Critere`": criterion, a measure of the equilibrium, basically a norm of the force residual divided by the force intensity. By default, this figure must be less than 1.e-4 to achieve convergence.
- "`Deps.max`": the maximal strain increment over the time-step.
- "`Eps.max`": the maximal strain.
- "`Crit.flex`": a criterion based on bending forces, which differs from the above criterion only when the problem involves finite elements with rotations and moments (beam, shell, etc.).

A message from the linear solver is inserted in the second line of the time-step messages: "`Taille de la matrice:...`". It gives information about the linear system that has been solved:

[6] Remember that Cast3M searches for an operator name among the first four character of each word in the instruction line.

[7] HPP: "Hypothèse des Petites Perturbations" (small perturbation hypothesis).

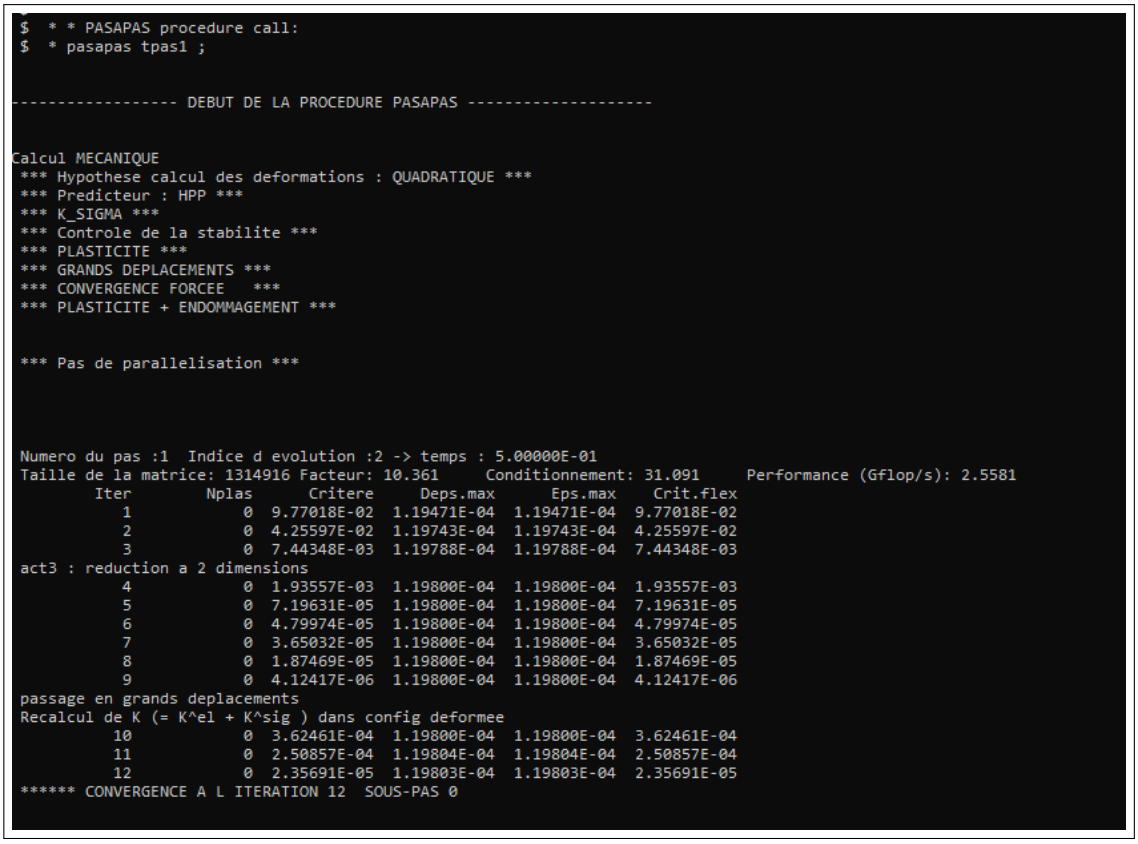



Figure 3.7: Screenshot of the PASAPAS procedure startup.

- "`Taille de la matrice`": matrix size in memory (in words).
- "`Conditionnement`": matrix conditioning, the ratio between the largest and smallest values on the diagonal of the matrix.
- "`Performance`": relative to the hardware.

The solver performs numerous iterations at each time step, displaying various information related to the resolution process. We will not go into further detail, as this is not the purpose of this document, but one must note that the solver iterates quite a lot due to the damaging behavior of the beam, which lead to almost zero stiffness in fully damaged elements. Damage variable is however limited to remain greater than zero in order to avoid a total loss of stiffness in those elements. The first step ends with the message: "`'****** CONVERGENCE A L ITERATION 12 SOUS-PAS 0`".

### 3.2.8 Result analysis

The results are stored in the input table under new indices, the name of which depend on the type of analysis performed (thermal, mechanical...). Let us examine the indices now present in the table `tpas1`.

#### Listing of the table `tpas1`

Figure 3.8 shows the contents of table **`tpas1`** after resolution with PASAPAS. Compared to the initial input table, the new indices are the following:

- **'SOUTYPE'**: subtype of the table, mention that **tpas1** contains **'PASAPAS** results.
- **'WTABLE'**: Working TABLE, a table used by **PASAPAS**.
- **'CONTINUATION'**: a table which contains the information necessary to continue the calculation.
- **'TEMPS'**: a table which contains the time value of each time step.
- **'DEPLACEMENTS'**: a table which contains the node field of displacements of each time step.
- **'CONTRAINTES'**: a table which contains the element field of stresses of each time step.
- **'REACTIONS'**: a table which contains the node field of reaction forces of each time step.
- **'VARIABLES_INTERNES'**: a table which contains the element field of internal variables of each time step.
- **'ESTIMATION'**: a table containing the same information as **'CONTINUATION'**.

For an analysis containing N time steps, the time steps are numbered from 0 to N, time step 0 corresponding to the initial state. As a consequence, there are N+1 recordings in the **'TEMPS'**, **'DEPLACEMENTS'**, **'CONTRAINTES'**... tables of **tpas1**.

```
$  * list tpas1 ;
TABLE de pointeur 2428105
       Indice                          Objet
     Type    Valeur                 Type   Valeur
MOT        MODELE                  MMODEL     2426233
MOT        CARACTERISTIQUES        MCHAML     2426793
MOT        CHARGEMENT              CHARGEME   2428081
MOT        BLOCAGES_MECANIQUES     RIGIDITE   2427937
MOT        TEMPS_CALCULES          LISTREEL   2428089
MOT        GRANDS_DEPLACEMENTS     LOGIQUE    VRAI
MOT        LAGRANGIEN              MOT        REACTUALISE
MOT        PREDICTEUR              MOT        HPP
MOT        SOUSTYPE                MOT        PASAPAS
MOT        WTABLE                  TABLE       2428353
MOT        CONTINUATION            TABLE      2428161
MOT        TEMPS                   TABLE      2429289
MOT        DEPLACEMENTS            TABLE      2429417
MOT        CONTRAINTES             TABLE      2429649
MOT        REACTIONS               TABLE      2429665
MOT        VARIABLES_INTERNES      TABLE      2429721
MOT        ESTIMATION              TABLE      2430009
$  *
```

Figure 3.8: Listing of the table `tpas1` after the resolution with `PASAPAS`.

### Deformed shape

As we saw in section 2.1.14, the operator **DEFO** computes an object "DEFORMEE" (deformed shape) according to a mesh and a node field of displacements. To display the deformed shape of the beam at the last time step, we then have to retrieve the corresponding displacement field from the results stored in **tpas1**. The following statements illustrate one way to do this.

```
if1   = (DIME tpas1 . 'TEMPS') - 1 ;
dep1  = tpas1 . 'DEPLACEMENTS' . if1 ;
def0  = DEFO (ARET V0) dep1   0. 'BLAN' ;
```

```
def1  = DEFO V0        dep1 100. 'ROUG' ;
TITR 'Deformed shape of the damaged beam &'
     ' initial shape (white edges)' ;
TRAC (2 -5 2) 'CACH' (def0 ET def1) ;
```

In these statements, the first instruction determines the index value **if1** of the last time step. The operator **DIME** returns the dimension of the table **tpas1.temps** (number of indices), then 1 is substracted since the table numbering starts at 0. You can **list** **tpas1.temps** to display its indices, then **if1** to check that its value is equal to the last of them.

In the second instruction, the index value of the time step **if1** is used to retrieve the corresponding displacement field. There, you can also **LIST** the table **tpas1.deplacements** to display its indices and note that they are the same as for the table **tpas1.temps**.

In the two following instructions, the operator **DEFO** first builds the initial shape of the beam (**def0**) on the edges (**ARET**) of the mesh **V0**, applying a magnification of 0 on the displacement field and coloring the result in white (**'BLAN'**"). Second, operator **DEFO** computes the deformed shape of the whole mesh **V0**, applying a magnification of 100 on the displacement field and coloring the result in red (**'ROUG'**").

Lastly, the two deformed shapes are fused in one object (**ET**) and displayed together with the directive **TRAC**, setting the viewpoint at **(2 -5 2)** and hiding the hidden parts of the deformed shapes with the **'CACH'** option. The title is defined at the previous line with the directive **TITR**. A screenshot of this displayed is presented in Figure 3.9.

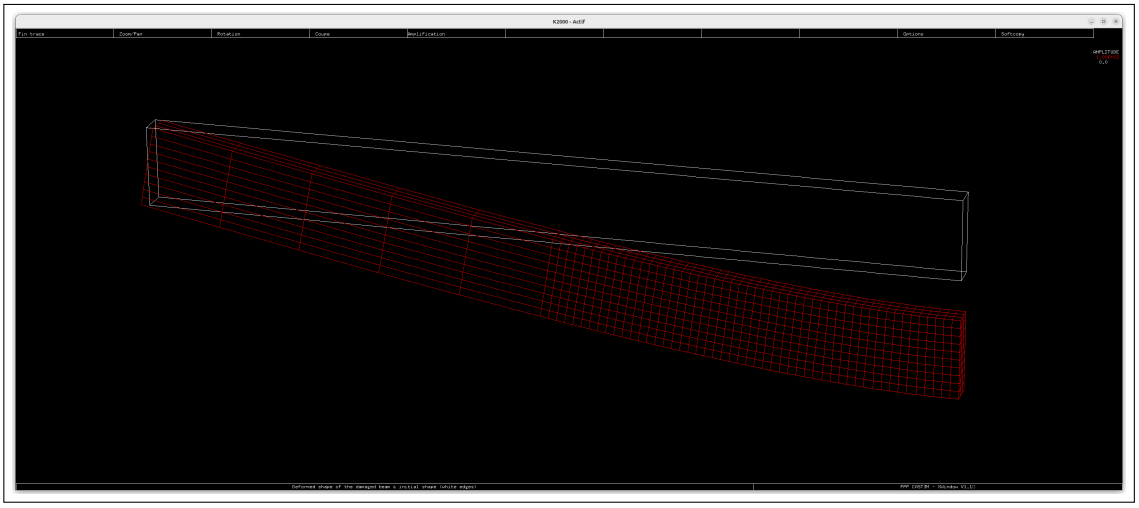


Figure 3.9: Display of the deformed shape of the damaged beam (red), together with the edges of its initial shape (see page 51).

### Stresses in the concrete beam

To postprocess the stresses, we retrieve the stress field corresponding to the last time step from the table **tpas1**, just as we did before for the displacements, as you can see at the first line of the following code statements.

```
sig1  = tpas1 . 'CONTRAINTES' . if1 ;
smxx1 = EXCO sig1 'SMXX' ;
```



```
dfx1  = DEFO V0 dep1 100. smxx1 mod1 ;
TITR 'Sxx stress component on the deformed shape of the damaged beam';
TRAC (2 -3 2) dfx1 ;
```

At the second instruction line, the operator **EXCO** extracts the component named **'SMXX'** from **sig1**, which is the $\sigma_{xx}$ component of the stress field.

Then, **defo** builds the deformed shape of this field. Note that **mod1** must be provided to specify the model associated to **smxx1**. Here, we give the model of the concrete beam (**mod1**) to get the stresses in the concrete.

Finally, directive **TRAC** displays the deformed shape **dfx1**, with the title defined at the previous instruction line by **TITR**. Figure A.11 in appendix shows this display.

#### Stresses in the steel rebars

The following statements lead to the display of the longitudinal stresses in the rebars.

```
effx1 = (REDU sig1 mod2) EXCO 'EFFX' 'SMXX' ;
effx1 = effx1 / sr1 * 1.e-6 ;
dfx1  = DEFO lr0 dep1 100. effx1 mod2 ;
TRAC (2 -5 2) dfx1 'TITR' 'Sxx stress component in the rebars' ;
```

The first instruction line postprocesses the generalized stress **effx1** in the rebars, which are modelled with truss elements. For this, operator **REDU** first reduces the stress field **sig1** to the model of the rebars (**mod2**), then operator **EXCO** extracts the component named **'EFFX'** from this reduced field and renames it **'SMXX'**.

In truss elements, the generalized stress is homogeneous to a force. So, at the second instruction line, **effx1** is divided by the rebar section **sr1** to be homogenous to a stress, then multiplied by $10^{-6}$ to be expressed in megapascals.

Next, operator **DEFO** builds the display of this field in the deformed configuration, as it was done previously for the stress field in the concrete beam. Note that the input mesh for operator **DEFO** is that of the rebars (**lr0**).

Lastly, directive **TRAC** displays **dfx1** with a title defined on the same instruction line. Figure A.12 in appendix shows this display.

#### Damage variable in the concrete beam

The following statements lead to the display of the damage variable in the concrete beam.

```
vard1 = (tpas1 . 'VARIABLES_INTERNES' . if1) EXCO 'D' ;
conf0 = FORM ;
liso1 = PROG 0.1 'PAS' 0.1 0.9 0.99 ;
FORM (100. * dep1) ;
TRAC liso1 vard1 mod1 'TITR' 'Damage in the concrete beam' ;
```

The first instruction line retrieves the element field of internal variable at the last time step from the table **tpas1**, then extracts its component `'D'`, which corresponds to the damage variable of the Mazars behavior law.

In these statements, the element field **vard1** is displayed in the deformed configuration of the beam without creating a DEFORMEE object with the operator **DEFO**. Instead, we use the operator **FORM** to record the current (initial) configuration **conf0**, then to move to the deformed configuration defined

by the displacement field **dep1**, magnified 100 times (third instruction line).

Once the display is achieved with the directive TRAC, directive FORM is applied to **conf0** to go back to the initial configuration. Note that FORM can be used either as an operator or as a directive. Note also that the model **mod1** is required to display an element field with TRAC.

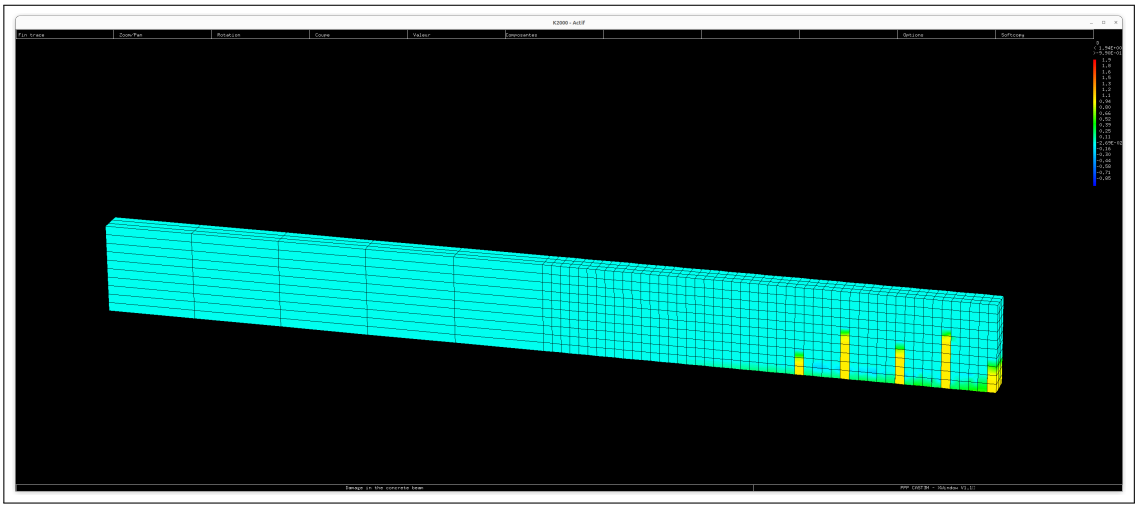


Figure 3.10: Display of the damage variable on the deformed configuration of the damaged beam (see page 53).

Figure 3.10 presents a screenshot of this display. On that one, it may be surprising to find that the damage values can be negative or superior to 1. This is due to the extrapolation of values from Gauss points to the nodes. By listing the maximal and minimal values of **vard1** (type "LIST (MAXI vard1) ;" for example), one can check that they effectivelly range form 0 to 1.

## 3.3 How to step in during a calculation with PASAPAS

### 3.3.1 Operating principle using the procedure PERSO1

The procedure PASAPAS provides some entries that allow to modify the process during the calculation. One of them is the procedure PERSO1. The use of PERSO1 is quite simple:

- First, program the Gibiane instructions that modify the process as needed within a procedure named PERSO1.
- Second, ask PASAPAS to call this procedur during the calculation by setting to "VRAI" (true) the following input option:

```
TAB1 . 'PROCEDURE_PERSO1' = VRAI ;
```

  where **TAB1** is the table of PASAPAS.

The procedure PERSO1 is called by PASAPAS at the end of each time step, after the equilibrium solution (the procedure that solves the equilibrium is called UNPAS (ONESTEP)). The structure of the procedure PERSO1 must comply the following model:



```
DEBP PERSO1 TT1*'TABLE' ;
...
FINP TT1 ;
```

The input table **TT1** is the table of PASAPAS. It therefore contains the entire modelling as well as the processed results. The next two sections present two examples of the use of the procedure PERSO1.

The procedure PASAPAS provides several other entries such as PERSO1. The procedure PERSO2 is called at the end of each step of a thermal analysis, the procedures CHARTHER and PARATHER during the iterations of a thermal analysis, the procedure CHARMECA, during the iterations of a mechanical analysis... Consult the Cast3M website to find examples using these procedures (not all of them), as well as the training materials about PASAPAS.

### 3.3.2 Element mesh removal

Let us go back to the example of a damaging reinforced concrete beam (see 3.2.1). We intend to remove from the model those finite elements for which the damage variable is close to 1 (greater than 0.99) during the calculation. The main stages to achieve this are as follows:

- retrieve the current element field of damage variable from the PASAPAS table,
- EXTRact the finite elements for which the damage variable is less than 0.99,
- REDUce the model to that mesh,
- update the PASAPAS table with this new model.

The following procedure PERSO1 is an example of implementation:

```
1  DEBP PERSO1 tt1*table ;
2  * Modify the model mesh:
3    et1  = tt1 . 'ESTIMATION' ;
4    wt1  = tt1 . 'WTABLE' ;
5    mox0 = wt1 . 'MODELE' ;
6    mox1 = mox0 EXTR 'CONS' 'CONCRETE' ;
7    mox2 = mox0 EXTR 'CONS' 'REBARS' ;
8    vax1 = et1 . 'VARIABLES_INTERNES' ;
9    vad1 = (EXCO vax1 'D' 'D') REDU mox1 ;
10   vad1 = vad1 CHAN 'GRAVITE' mox1 ;
11   elx1 = vad1 ELEM 'INFE' 0.99 ;
12   mox1 = REDU mox1 elx1 ;
13   wt1 . 'MODELE' = mox1 ET mox2 ;
14 FINP tt1 ;
```

The first and last lines of instructions correspond to procedure specification presented in the previous section. At lines 3 and 4, we define **et1** and **wt1** as local variables for the tables ESTIMATION and WTABLE (see 3.2.8 page 50).

At line 5, the current calculation model **mox0** is retrieved from the table WTABLE. The model initially provided by the user is stored by PASAPAS at the index MODEL of the table WTABLE, which allows modifying this model while preserving the initial one. Note that modifying the initial input model at the index **TT1.'MODELE'** would have no effect on the calculation since PASAPAS uses the one stored in the table WTABLE. At lines 6 and 7, we extract from the global model **mox0** the model correponding to the concrete beam (**mox1**) and the one correponding to the reinforcement bars (**mox2**).

Then, we retrieve from the table ESTIMATION the element field of internal variables **vax1** (line 8), from which we extract the field of damage variable (component 'D'), that we reduce to the model of

the concrete beam (**vad1** at line 9). At line 10, this field is interpolated at the center of gravity of the finite elements (also referred to as the centroid in other software).

Next, we extract from **vad1** the finite elements for which the damage variable is strictly lower than 0.99 (**elx1**, line 11). The model of the concrete beam (**mox1**) is then reduced to this mesh (line 12), which means that this model no longer contains elements for which the damage variable is greater than 0.99. Finally, the models of the concrete beam **mox1** and that of the reinforcement bars **mox2** are combined together and stored at the calculation model index of the PASAPAS table (line 13).

All these instructions appear consistent, but there is no prof that the "damaged finite elements" are actually removed from the calculation. In this case, it may be useful to visualize what happens during the calcultion process. That is the objective of the following section.

### 3.3.3 Visualization during processing

In order to visualize the modifications made to the calculation model, we add the following Gibiane instructions to the procedure PERSO1 developed in the previous section:

```
1   wt1 . 'MODELE' = mox1 ET mox2 ;
2 * Display damage variable on the deformed configuration:
3   vad1 = REDU vad1 mox1 ;
4   dex1 = et1 . 'DEPLACEMENTS' ;
5   dfx1 = DEFO (EXTR mox1 'MAIL') dex1 100. vad1 mox1 ;
6   tps1 = et1 . 'TEMPS' ;
7   mot1 = CHAI 'Damage variable on deformed shape at time t=' tps1 ;
8   TRAC 'NCLK' (1.2 -0.5 0.2) (PROG 0.1 'PAS' 0.1 0.9 0.99) dfx1
9        'TITR' mot1 ;
10 FINP tt1 ;
```

The added instructions begin with the comment at line 2. The element field of damage variable **vad1** is first reduced to the updated model **mox1** of the concrete beam (line 3). Then, the current node field of displacements **dex1** is retrieved from the table ESTIMATION (line 4) to create a DEFORMEE **dfx1** on which the damage variable **vad1** is displayed (line 5). Note that the operator DEFO uses a mesh extrated from the model **mox1**, from which the "damaged finite elements" have been removed.

Next, the current value of the time step **tps1** is retrieved from the table ESTIMATION (line 6) to create **mot1**, the display title, with the operator CHAI (line 7). At line 8, the directive TRAC finally displays the values of the damage variable in the deformed configuration of the "damaged mesh". On that line, note the use of the nclk option (meaning "No CLicK"), which allows to quit the display window without clicking, and automatically move on to the next plot. Furthermore, the LISTREEL defined using the operator PROG specifies the scale of the contour plot. The last line ends the procedure PERSO1.

Figure 3.11 presents two contour plots of the damage variable generated by the procedure PERSO1 during the calculation of PASAPAS. These contour plots were automatically saved in a postscript file using the option 'TRAC' 'PSC' of the directive OPTI.

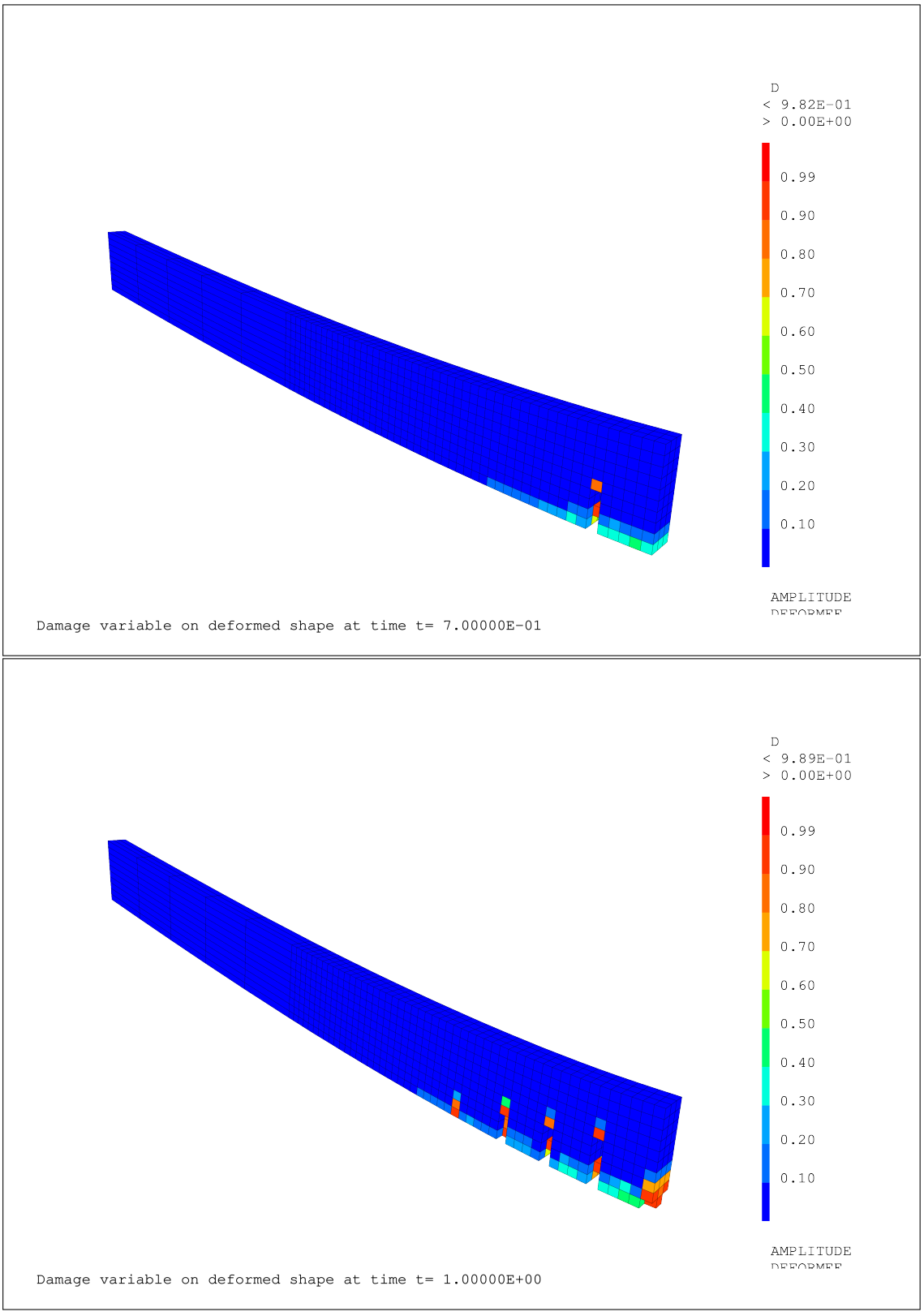



Figure 3.11: Contour plots of the damage variable on the deformed shaped of the concrete beam achieved by the procedure **PERSO1** during the calculation of **PASAPAS**.

### 3.3.4 Script file `beam3.dgibi`

All the statements presented in this chapter regarding the nonlinear static analysis of a reinforced concrete beam submitted to three-point bending are resumed in the script file `beam3.dgibi` below. Besides, you will find some additional statements in the analysis of the results, in particular the use of the procedure "PECHE" (fish) to retrieve results from the PASAPAS table. We leave them to the reader.

Listing 3.1: Script file "beam3.dgibi".

```
* fichier : beam3.dgibi
*------------------------------------------------------------------------*
* Reinforced concrete beam subjected to 3 points bending
*------------------------------------------------------------------------*
* Space dimension and finite element type:
OPTI 'DIME' 3 'ELEM' 'CUB8' ;

* --- 1. Model parameters
* Beam geometry parameters (in meters):
lb1 =  1.00 ;
hb1 =  0.10 ;
wb1 =  0.03 ;
hr1 =  0.01 ;
sr1 =  0.5e-4;
uz1 = -0.0015 ;

* Mesh density:
df1  = 0.01 ;
dc1  = 0.10 ;

* --- 2. Meshing
* Frame:
o1  = 0 0 0 ;
x1  = 1 0 0 ;
y1  = 0 1 0 ;
z1  = 0 0 1 ;

* Beam mesh:
l1  = o1 DROI (wb1*y1) 'DINI' df1 'DFIN' df1 ;
s1  = l1 TRAN (hb1*z1) 'DINI' df1 'DFIN' df1 ;
v1  = s1 VOLU 'TRAN' (0.5*lb1*x1) 'DINI' dc1 'DFIN' dc1 ;
s2  = v1 FACE 2 ;
v2  = s2 VOLU 'TRAN' (0.5*lb1*x1) 'DINI' df1 'DFIN' df1 ;
v0  = v1 ET v2 ;

* Rebars mesh:
pr1 = (0.5*wb1*y1) PLUS (hr1*z1) ;
pr2 = pr1 PLUS (lb1*x1) ;
lr1 = pr1 DROI pr2 'DINI' (0.97*df1) 'DFIN' (0.97*df1);
lr2 = lr1 PLUS ((-2. * hr1 + hb1)*z1) ;
lr0 = (lr1 ET lr2) COUL 'ROUG' ;
TRAC (100 -50 20) ((ENVE V0) ET lr0)
  'TITR' 'Meshes of the concrete beam and the rebars (red lines)' ;

* --- 3. Models & Material characteristics
* Concrete and rebars mechanical models:
mod1 = MODE V0  'MECANIQUE' 'ELASTIQUE' 'ENDOMMAGEMENT' 'MAZARS'
                'CONS' 'CONCRETE' ;
mod2 = MODE lr0 'MECANIQUE' 'ELASTIQUE' 'PLASTIQUE' 'ISOTROPE' 'BARR'
```



```
                'CONS' 'REBARS' ;
* Model addition:
mod0 = mod1 ET mod2 ;

* Concrete mechanical characteristics: Mazars behavior law parameters:
ym1   = 31.e9 ;
nu1   = 0.2 ;
epsd1 = 1.e-04 ;
acom1 = 1.2 ;
bcom1 = 1500. ;
atra1 = 1.0 ;
btra1 = 10000. ;
beta1 = 1.06 ;
mat1 = MATE mod1 'YOUN' ym1 'NU' nu1 'KTR0' epsd1 'ACOM' acom1
                'BCOM' bcom1 'ATRA' atra1 'BTRA' btra1 'BETA' beta1 ;

* Rebars mechanical characteristics: elastoplastic with hardening curve:
ecro1 = EVOL 'VERT' 'MANU'  (PROG 0.     1.)
                    'ECRO1' (PROG 210.e6 1.e9) ;
DESS ecro1 'TITR' 'Hardening curve of rebars' ;
mat2  = MATE mod2 'YOUN' 210.e9 'NU' 0.3 'ECRO' ecro1 'SECT' sr1 ;

* Material characteristics addition:
mat0  = mat1 ET mat2 ;

* --- 4. Boundary conditions and loading
* Support boundary conditions at one end of the beam:
pz0  = (v0  COOR 3) POIN 'MINI' ;
pxz0 = (pz0 COOR 1) POIN 'MINI' ;
bc1  = BLOQ pxz0 'UZ' ;
TRAC ((pxz0 COUL 'ORAN') ET (ARET V0))
  'TITR' 'Lower support BC: Uz=0 on orange points' ;

* Symmetry in the (O,y) direction:
py0  = (v0 COOR 2) POIN 'MINI' ;
bc2  = BLOQ py0 'UY' ;
trac ((py0 COUL 'ORAN') ET (ARET v0))
  'TITR' 'Mid-width beam symmetry: Uy=0 on orange points' ;

* Symmetry in the (O,x) direction:
s3   = v2 FACE 2 ;
bc3  = BLOQ s3 'UX' ;
TRAC ((s3 COUL 'ORAN') ET (ARET v0))
  'TITR' 'Mid-length beam symmetry: Ux=0 on orange elements' ;

* BC for applied displacements:
pxz1 = (S3 COOR 3) POIN 'MAXI' ;
bc4  = BLOQ 'UZ' pxz1 ;
TRAC ((pxz1 COUL 'ORAN') et (ARET v0))
   'TITR' 'Points (in orange) of applied displacements' ;

* Anchoring rebars in concrete:
bc5  = RELA 'ACCRO' lr0 v0 ;

* BC stiffnesses addition:
bc0  = bc1 ET bc2 ET bc3 ET bc4 ET bc5 ;

```

```
* Mechanical loading: time evolution of the applied displacement
dbc4 = DEPI bc4 uz1 ;
mot1 = CHAI 'FORMAT' '(F6.2)'
       'Applied displacements: Uz =' (1000. * uz1) ' (mm)' ;
TRAC ((dbc4 EXCO 'FLX' 'UZ') VECT 'DEPL' 20. 'ROUG') (ARET V0) 'TITR'
   mot1 ;

ev1  = EVOL 'MANU' (PROG 0. 1.) (PROG 0. 1.) ;
cg1  = CHAR 'DIMP' dbc4 ev1 ;

* --- 5. Remove damaged elements using PERSO1 procedure
* PERSO1 procedure definition:
DEBP PERSO1 tt1*table ;
* Modify the model mesh:
  et1  = tt1 . 'ESTIMATION' ;
  wt1  = tt1 . 'WTABLE' ;
  mox0 = wt1 . 'MODELE' ;
  mox1 = mox0 EXTR 'CONS' 'CONCRETE' ;
  mox2 = mox0 EXTR 'CONS' 'REBARS' ;
  vax1 = et1 . 'VARIABLES_INTERNES' ;
  vad1 = (EXCO vax1 'D' 'D') REDU mox1 ;
  vad1 = vad1 CHAN 'GRAVITE' mox1 ;
  elx1 = vad1 ELEM 'INFE' 0.99 ;
  mox1 = REDU mox1 elx1 ;
  wt1 . 'MODELE' = mox1 ET mox2 ;
* Display damage variable on the deformed configuration:
  vad1 = REDU vad1 mox1 ;
  dex1 = et1 . 'DEPLACEMENTS' ;
  dfx1 = DEFO (EXTR mox1 'MAIL') dex1 100. vad1 mox1 ;
  tps1 = et1 . 'TEMPS' ;
  mot1 = CHAI 'Damage variable on deformed shape at time t=' tps1 ;
  TRAC 'NCLK' (1.2 -0.5 0.2) (PROG 0.1 'PAS' 0.1 0.9 0.99) dfx1
       'TITR' mot1 ;
FINP tt1 ;

* --- 6. Resolution with PASAPAS
* PASAPAS input table:
tpas1 = TABLE ;
tpas1 . 'MODELE'               = mod0 ;
tpas1 . 'CARACTERISTIQUES'     = mat0 ;
tpas1 . 'CHARGEMENT'           = cg1 ;
tpas1 . 'BLOCAGES_MECANIQUES' = bc0 ; COMM mech. boundary conditions ;
tpas1 . 'TEMPS_CALCULES'       = PROG 0. 0.5 'PAS' 0.05 1.0 ;

* Large displacements analysis options:
tpas1 . 'GRANDS_DEPLACEMENTS' = vrai ;
tpas1 . 'LAGRANGIEN'           = MOT 'REACTUALISE' ;
tpas1 . 'PREDICTEUR'           = MOT 'HPP' ;

* Call to procedure PERSO1 option:
tpas1 . 'PROCEDURE_PERSO1'     = vrai ;

* PASAPAS procedure call:
PASAPAS tpas1 ;

LIST tpas1 ;

```



```
* --- 7. Post-processing
* Deformed shape:
SI (tpas1 . 'PROCEDURE_PERSO1') ;
  if1   = (DIME tpas1 . 'TEMPS') - 1 ;
  tps1  = tpas1 . 'TEMPS' . if1 ;
  dep1  = PECHE tpas1 'DEPLACEMENTS' tps1 ;
  mof1  = (tpas1 . 'WTABLE' . 'MODELE') EXTR 'CONS' 'CONCRETE' ;
  Vf1   = mof1 EXTR 'MAIL' ;
  def0  = DEFO (ARET V0) dep1   0. 'BLAN' ;
  def1  = DEFO Vf1       dep1 100. 'ROUG' ;
  TITR 'Deformed shape of the damaged beam &'
       ' initial shape (white edges)' ;
  TRAC (2 -5 2) 'CACH' (def0 ET def1) ;
SINO ;
  if1   = (DIME tpas1 . 'TEMPS') - 1 ;
  dep1  = tpas1 . 'DEPLACEMENTS' . if1 ;
  def0  = DEFO (ARET V0) dep1   0. 'BLAN' ;
  def1  = DEFO V0        dep1 100. 'ROUG' ;
  TITR 'Deformed shape of the damaged beam &'
       ' initial shape (white edges)' ;
  TRAC (2 -5 2) 'CACH' (def0 ET def1) ;
FINS ;

LIST tpas1 . 'TEMPS' ;
LIST if1 ;

* Stress in the concrete beam:
SI (tpas1 . 'PROCEDURE_PERSO1') ;
  sig1  = PECHE tpas1 'CONTRAINTES' tps1 ;
  smxx1 = (EXCO sig1 'SMXX') REDU mof1 ;
  dfx1  = DEFO Vf1 dep1 100. smxx1 mof1 ;
  TITR 'Sxx stress component on the deformed shape of the damaged beam';
  TRAC (2 -5 2) dfx1 ;
SINO ;
  sig1  = tpas1 . 'CONTRAINTES' . if1 ;
  smxx1 = EXCO sig1 'SMXX' ;
  dfx1  = DEFO V0 dep1 100. smxx1 mod1 ;
  TITR 'Sxx stress component on the deformed shape of the damaged beam';
  TRAC (2 -3 2) dfx1 ;
FINS ;

* Stress in the rebars:
effx1 = (REDU sig1 mod2) EXCO 'EFFX' 'SMXX' ;
effx1 = effx1 / sr1 * 1.e-6 ;
dfx1  = DEFO lr0 dep1 100. effx1 mod2 ;
TRAC (2 -5 2) dfx1 'TITR' 'Sxx stress component in the rebars' ;

* Damage variable in concrete beam:
vard1 = (tpas1 . 'VARIABLES_INTERNES' . if1) EXCO 'D' ;
conf0 = FORM ;
liso1 = PROG 0.1 'PAS' 0.1 0.9 0.99 ;
FORM (100. * dep1) ;
SI tpas1 . 'PROCEDURE_PERSO1' ;
TRAC liso1 vard1 mof1 'TITR' 'Damage in the concrete beam' ;
SINO ;
TRAC liso1 vard1 mod1 'TITR' 'Damage in the concrete beam' ;
FINS ;
```

```
220 FORM conf0 ;
221
222 FIN ;
```

# Appendix A

# Screenshots

```
$  * MOD1 = MODE Beam1 MECANIQUE ELASTIQUE ;
$  * list MOD1 ;



+------------------------------------------------------------------------------+
|                                                                              |
|          OBJET  MMODEL  CONTENANT      1 ZONE(S) ELEMENTAIRE(S)   2122345    |
|                                                                              |
+------------------------------------------------------------------------------+


          ZONE ELEMENTAIRE NUMERO      1 : IMO   2122359
          ------------------------------------------------

  POINTEUR SUR L'OBJET MAILLAGE :    2122072
  TYPE D'ELEMENT FINI  :     CUB8
  NOM DU CONSTITUANT   :              2122345
  FORMULATION          :     MECANIQUE
  MODELE DE MATERIAU   :     ELASTIQUE          ISOTROPE
  Liste des noms de composantes de DEPLACEM    -    Pointeur nomid:   2122366
         UX   UY   UZ
  Liste des noms de composantes de FORCES      -    Pointeur nomid:   2122373
         FX   FY   FZ
  Liste des noms de composantes de GRADIENT    -    Pointeur nomid:   2122380
         UX,X UX,Y UX,Z UY,X UY,Y UY,Z UZ,X UZ,Y UZ,Z
  Liste des noms de composantes de CONTRAIN    -    Pointeur nomid:   2122387
         SMXX SMYY SMZZ SMXY SMXZ SMYZ
  Liste des noms de composantes de DEFORMAT    -    Pointeur nomid:   2122394
         EPXX EPYY EPZZ GAXY GAXZ GAYZ
  Liste des noms de composantes de MATERIAU    -    Pointeur nomid:   2122401
         YOUN NU
         RHO  ALPH VISQ TREF TALP
  Liste des noms de composantes de TEMPERAT    -    Pointeur nomid:   2122415
         T
  Liste des noms de composantes de PRINCIPA    -    Pointeur nomid:   2122422
         SI11 SI22 SI33 COX1 COY1 COZ1 COX2 COY2 COZ2 COX3 COY3 COZ3
  Liste des noms de composantes de DEFINELA    -    Pointeur nomid:   2122450
         EIXX EIYY EIZZ GIXY GIXZ GIYZ

  numero de l element fini (nefmod):   14
  points integration epaisseur     :    8
  points de gauss pour masse       :    8
  points support de contraintes    :    8
  nombre de caracteristiques       :    0
  points de gauss pour rigidite    :    8
  numero de la formulation E.F(mfr):    1
  numero de  l element geometrique :   14
```

Figure A.1: Listing of the model MOD1 presented in 2.1.10, page 18.

```
$  * MAT1 = MATE MOD1 YOUN 30.e9 NU 0.2 ;
$  * list MAT1 ;



+------------------------------------------------------------------------------+
|                                                                              |
|          OBJET  MCHAML  CONTENANT      1 ZONE(S) ELEMENTAIRE(S)   2122674    |
|                                                                              |
|          TYPE : CARACTERISTIQUES                                             |
|          OPTION DE CALCUL   TRIDIMENSIONNEL                                  |
|                                                                              |
+------------------------------------------------------------------------------+


          ZONE ELEMENTAIRE NUMERO      1 : MCH   2122681
          ---------------------------------------------

  POINTEUR SUR L'OBJET MAILLAGE    2122072 : 'CUB8'

  NUMERO DE L'HARMONIQUE              0
  POINTEUR SUR LES POINTS SUPPORTS     2122527
  VALEURS DONNEES AUX POINTS DE GAUSS POUR LA RIGIDITE
  NOM DU CONSTITUANT                      2122345
  NOMBRE DE COMPOSANTES               2


   1-ERE COMPOSANTE - NOM : YOUN     - TYPE : REAL*8           mel   2122688

  CHAMP CONSTANT EGAL A    3.000E+10


   2-EME COMPOSANTE - NOM : NU       - TYPE : REAL*8           mel   2122695

  CHAMP CONSTANT EGAL A    2.000E-01



$  *
```

Figure A.2: Listing of the element field MAT1 presented in 2.1.10, page 18.

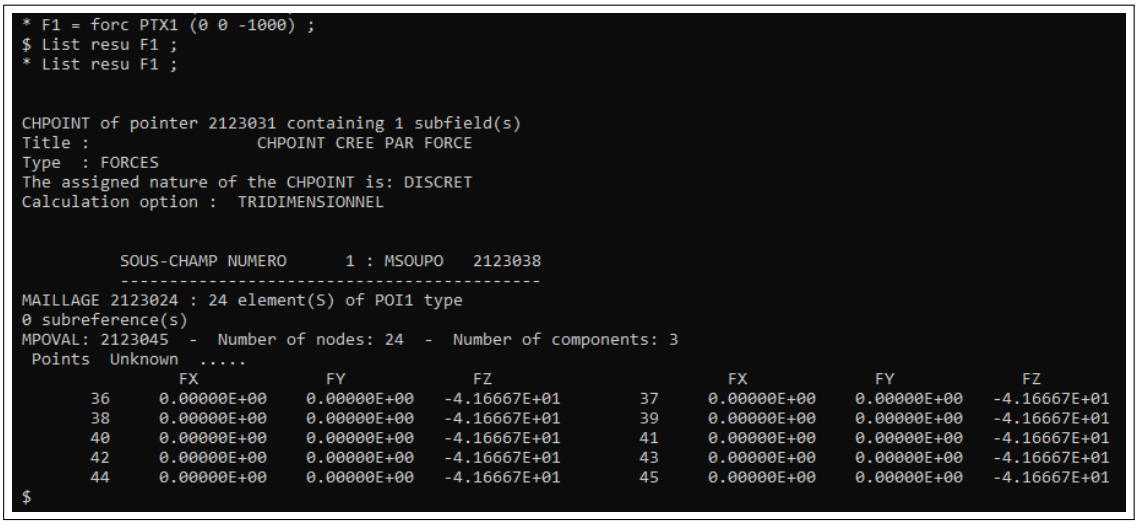
```
$  * PTX1 = ( beam1 coor 1) poin maxi ;
$  * list PTX1 ;
MAILLAGE 2122632 : 24 element(S) of POI1 type
0 subreference(s)
1st line  number element : 2nd color : 3rd... node(s)
          1       2       3       4       5       6       7       8       9      10
        DEFA    DEFA    DEFA    DEFA    DEFA    DEFA    DEFA    DEFA    DEFA    DEFA

         36      37      40      39      38      41      45      44      46      49
1st line  number element : 2nd color : 3rd... node(s)
         11      12      13      14      15      16      17      18      19      20
        DEFA    DEFA    DEFA    DEFA    DEFA    DEFA    DEFA    DEFA    DEFA    DEFA

         48      50      53      52      54      42      43      47      51      55
1st line  number element : 2nd color : 3rd... node(s)
         21      22      23      24
        DEFA    DEFA    DEFA    DEFA

         56      58      59      57
$  *
```

Figure A.3: Listing of the mesh PTX1 (see 2.1.11, page 18).

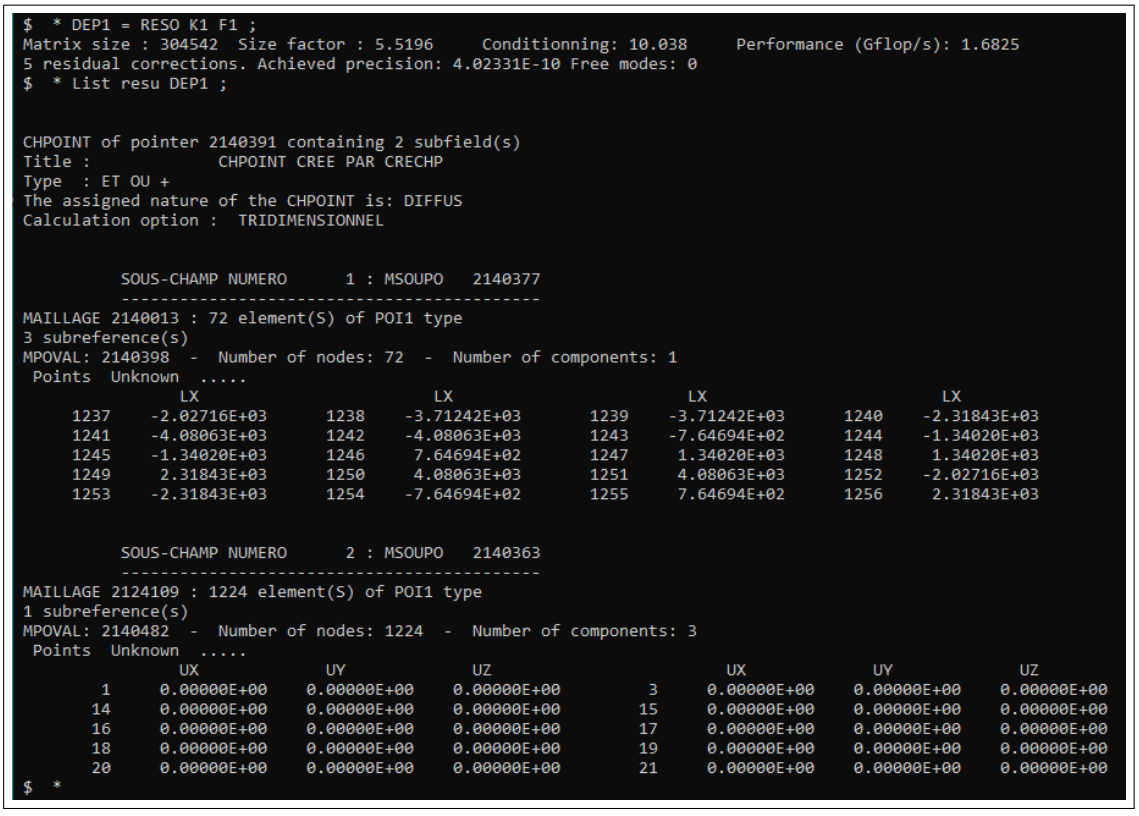


```
* F1 = forc PTX1 (0 0 -1000) ;
$ List resu F1 ;
* List resu F1 ;


CHPOINT of pointer 2123031 containing 1 subfield(s)
Title :                CHPOINT CREE PAR FORCE
Type  : FORCES
The assigned nature of the CHPOINT is: DISCRET
Calculation option :  TRIDIMENSIONNEL


          SOUS-CHAMP NUMERO      1 : MSOUPO   2123038
          -------------------------------------------
MAILLAGE 2123024 : 24 element(S) of POI1 type
0 subreference(s)
MPOVAL: 2123045  -  Number of nodes: 24  -  Number of components: 3
 Points  Unknown  .....
              FX             FY             FZ                      FX             FY             FZ
      36     0.00000E+00    0.00000E+00   -4.16667E+01       37     0.00000E+00    0.00000E+00   -4.16667E+01
      38     0.00000E+00    0.00000E+00   -4.16667E+01       39     0.00000E+00    0.00000E+00   -4.16667E+01
      40     0.00000E+00    0.00000E+00   -4.16667E+01       41     0.00000E+00    0.00000E+00   -4.16667E+01
      42     0.00000E+00    0.00000E+00   -4.16667E+01       43     0.00000E+00    0.00000E+00   -4.16667E+01
      44     0.00000E+00    0.00000E+00   -4.16667E+01       45     0.00000E+00    0.00000E+00   -4.16667E+01
$
```

Figure A.4: Listing of the "node field" of forces F1 (see 2.1.12, page 19).



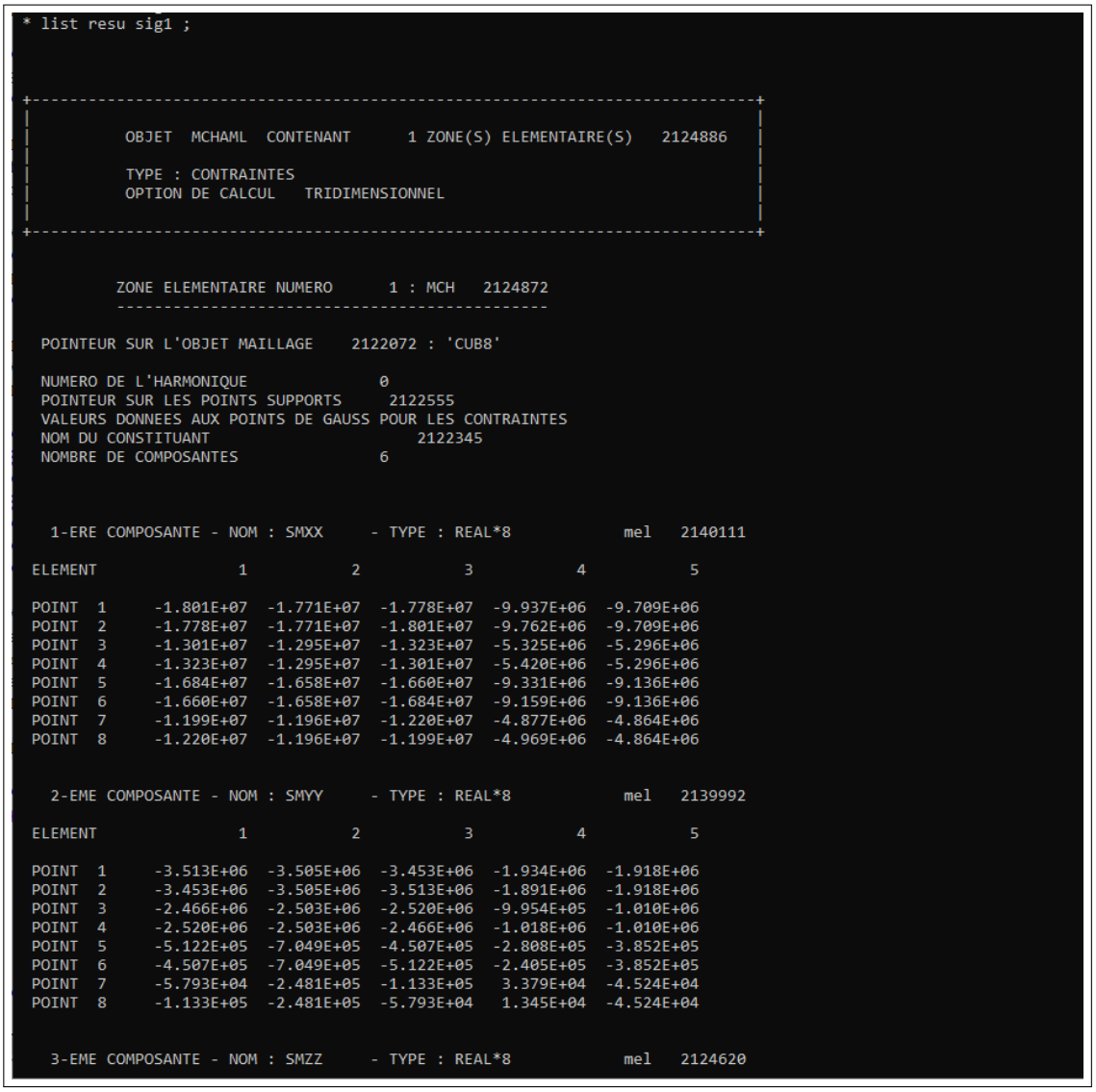
```
$  * DEP1 = RESO K1 F1 ;
Matrix size : 304542  Size factor : 5.5196     Conditionning: 10.038     Performance (Gflop/s): 1.6825
5 residual corrections. Achieved precision: 4.02331E-10 Free modes: 0
$  * List resu DEP1 ;


CHPOINT of pointer 2140391 containing 2 subfield(s)
Title :             CHPOINT CREE PAR CRECHP
Type  : ET OU +
The assigned nature of the CHPOINT is: DIFFUS
Calculation option :  TRIDIMENSIONNEL


          SOUS-CHAMP NUMERO      1 : MSOUPO   2140377
          -------------------------------------------
MAILLAGE 2140013 : 72 element(S) of POI1 type
3 subreference(s)
MPOVAL: 2140398  -  Number of nodes: 72  -  Number of components: 1
 Points  Unknown  .....
               LX                      LX                      LX                      LX
     1237    -2.02716E+03      1238    -3.71242E+03      1239    -3.71242E+03      1240    -2.31843E+03
     1241    -4.08063E+03      1242    -4.08063E+03      1243    -7.64694E+02      1244    -1.34020E+03
     1245    -1.34020E+03      1246     7.64694E+02      1247     1.34020E+03      1248     1.34020E+03
     1249     2.31843E+03      1250     4.08063E+03      1251     4.08063E+03      1252    -2.02716E+03
     1253    -2.31843E+03      1254    -7.64694E+02      1255     7.64694E+02      1256     2.31843E+03


          SOUS-CHAMP NUMERO      2 : MSOUPO   2140363
          -------------------------------------------
MAILLAGE 2124109 : 1224 element(S) of POI1 type
1 subreference(s)
MPOVAL: 2140482  -  Number of nodes: 1224  -  Number of components: 3
 Points  Unknown  .....
              UX             UY             UZ                      UX             UY             UZ
       1     0.00000E+00    0.00000E+00    0.00000E+00         3     0.00000E+00    0.00000E+00    0.00000E+00
      14     0.00000E+00    0.00000E+00    0.00000E+00        15     0.00000E+00    0.00000E+00    0.00000E+00
      16     0.00000E+00    0.00000E+00    0.00000E+00        17     0.00000E+00    0.00000E+00    0.00000E+00
      18     0.00000E+00    0.00000E+00    0.00000E+00        19     0.00000E+00    0.00000E+00    0.00000E+00
      20     0.00000E+00    0.00000E+00    0.00000E+00        21     0.00000E+00    0.00000E+00    0.00000E+00
$  *
```

Figure A.5: Listing of the "node field" of displacements DEP1 (see 2.1.13, page 20).



```
* list resu sig1 ;


+------------------------------------------------------------------------------+
|                                                                              |
|          OBJET  MCHAML  CONTENANT      1 ZONE(S) ELEMENTAIRE(S)   2124886    |
|                                                                              |
|          TYPE : CONTRAINTES                                                  |
|          OPTION DE CALCUL   TRIDIMENSIONNEL                                  |
|                                                                              |
+------------------------------------------------------------------------------+


         ZONE ELEMENTAIRE NUMERO      1 : MCH   2124872
         ------------------------------------------

  POINTEUR SUR L'OBJET MAILLAGE    2122072 : 'CUB8'

  NUMERO DE L'HARMONIQUE              0
  POINTEUR SUR LES POINTS SUPPORTS     2122555
  VALEURS DONNEES AUX POINTS DE GAUSS POUR LES CONTRAINTES
  NOM DU CONSTITUANT                      2122345
  NOMBRE DE COMPOSANTES               6


   1-ERE COMPOSANTE - NOM : SMXX     - TYPE : REAL*8            mel   2140111

 ELEMENT               1           2           3           4           5

 POINT  1     -1.801E+07  -1.771E+07  -1.778E+07  -9.937E+06  -9.709E+06
 POINT  2     -1.778E+07  -1.771E+07  -1.801E+07  -9.762E+06  -9.709E+06
 POINT  3     -1.301E+07  -1.295E+07  -1.323E+07  -5.325E+06  -5.296E+06
 POINT  4     -1.323E+07  -1.295E+07  -1.301E+07  -5.420E+06  -5.296E+06
 POINT  5     -1.684E+07  -1.658E+07  -1.660E+07  -9.331E+06  -9.136E+06
 POINT  6     -1.660E+07  -1.658E+07  -1.684E+07  -9.159E+06  -9.136E+06
 POINT  7     -1.199E+07  -1.196E+07  -1.220E+07  -4.877E+06  -4.864E+06
 POINT  8     -1.220E+07  -1.196E+07  -1.199E+07  -4.969E+06  -4.864E+06


   2-EME COMPOSANTE - NOM : SMYY     - TYPE : REAL*8            mel   2139992

 ELEMENT               1           2           3           4           5

 POINT  1     -3.513E+06  -3.505E+06  -3.453E+06  -1.934E+06  -1.918E+06
 POINT  2     -3.453E+06  -3.505E+06  -3.513E+06  -1.891E+06  -1.918E+06
 POINT  3     -2.466E+06  -2.503E+06  -2.520E+06  -9.954E+05  -1.010E+06
 POINT  4     -2.520E+06  -2.503E+06  -2.466E+06  -1.018E+06  -1.010E+06
 POINT  5     -5.122E+05  -7.049E+05  -4.507E+05  -2.808E+05  -3.852E+05
 POINT  6     -4.507E+05  -7.049E+05  -5.122E+05  -2.405E+05  -3.852E+05
 POINT  7     -5.793E+04  -2.481E+05  -1.133E+05   3.379E+04  -4.524E+04
 POINT  8     -1.133E+05  -2.481E+05  -5.793E+04   1.345E+04  -4.524E+04


   3-EME COMPOSANTE - NOM : SMZZ     - TYPE : REAL*8            mel   2124620
```

Figure A.6: Listing of the "element field" of stresses SIG1 (see page 23).

```
$  * List resu EVUZ1 ;
 Beam1 edges and PTX1 node set (in red)
EVOLUTION of pointer 2133125 composed of 2 elementary evolution(s)
of subtype REEL
Subevolution number 1 of color VERT
 CREE PAR EVOL CHPO
Variable in abscissa ABS
Variable in ordinate ( REEL )  UZ
Number of points : 51

      1    0.00000E+00   -1.66239E-02
      2    4.00000E-02   -1.61250E-02
      3    8.00000E-02   -1.56268E-02
      4    1.20000E-01   -1.51293E-02
      5    1.60000E-01   -1.46330E-02
      6    2.00000E-01   -1.41384E-02
      7    2.40000E-01   -1.36457E-02
      8    2.80000E-01   -1.31554E-02
      9    3.20000E-01   -1.26679E-02
     10    3.60000E-01   -1.21836E-02
Subevolution number 2 of color VERT
 POINTS NOMMES APPARTENANT A LA LIGNE
Variable in abscissa ABS
Variable in ordinate ( MARQ )  UZ
Number of points : 1

      1    2.00000E+00   P0
$  * tleg1 = tabl ;
```

Figure A.7: Listing of the "evolution" EVUZ1 (see page 25).

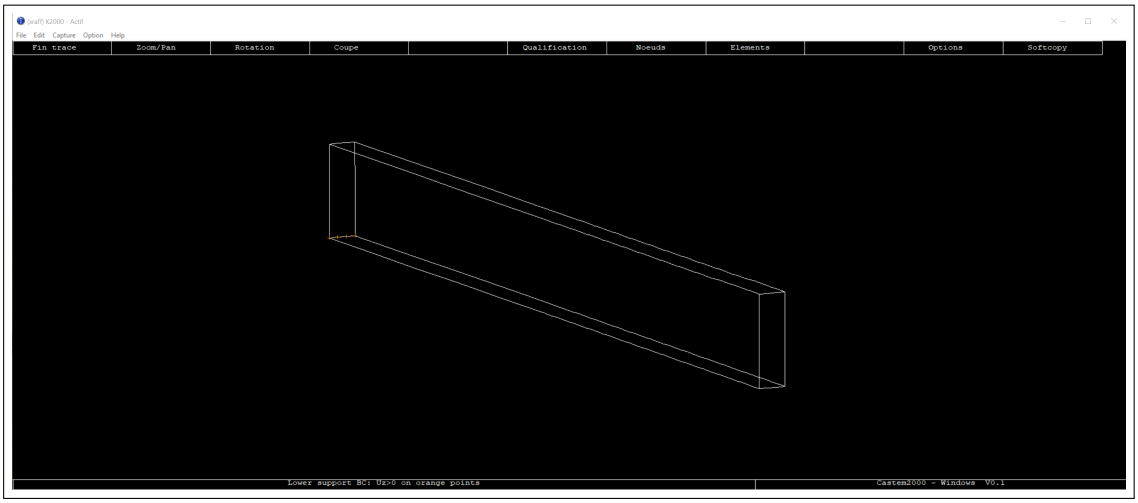



Figure A.8: Display of the points where BC1 is applied (see page **??**).

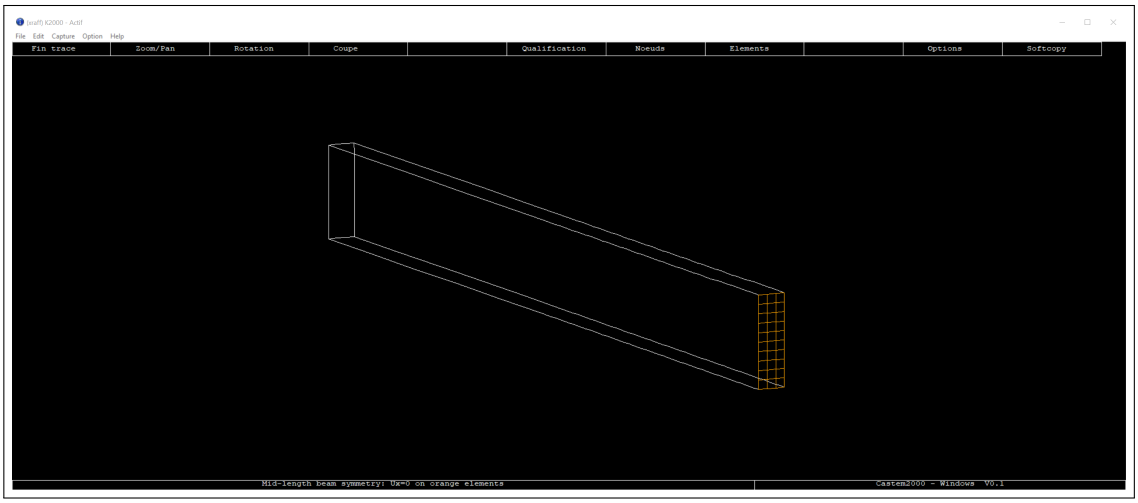



Figure A.9: Display of the surface mesh where BC3 is applied (see page **??**).

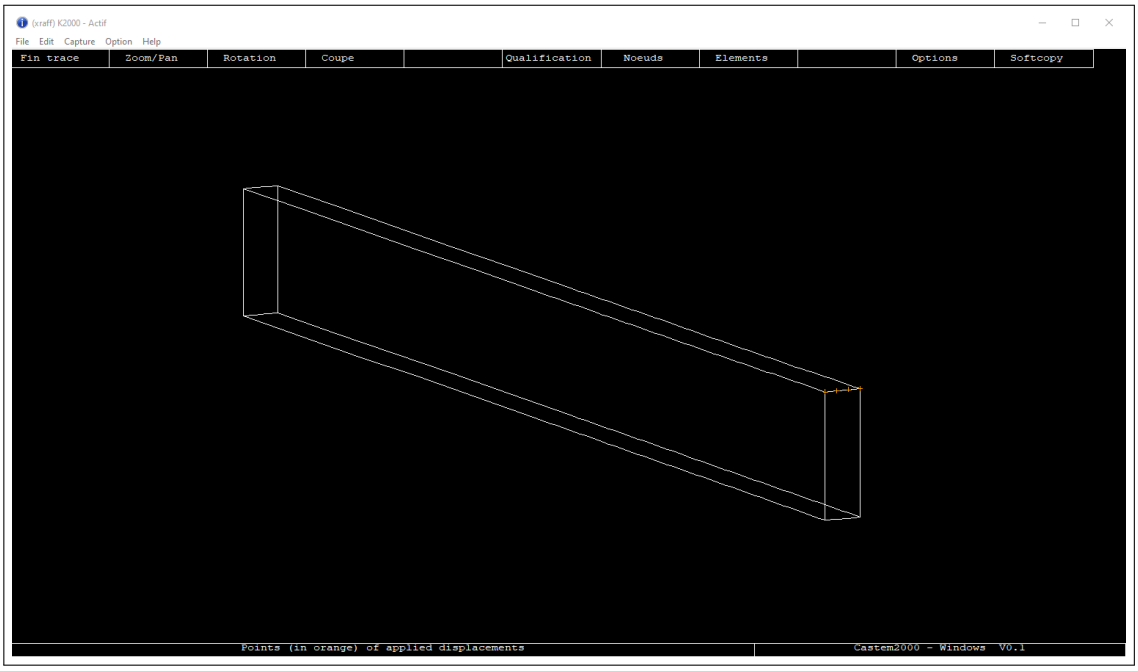



Figure A.10: Display of displacement application points (see page **??**).

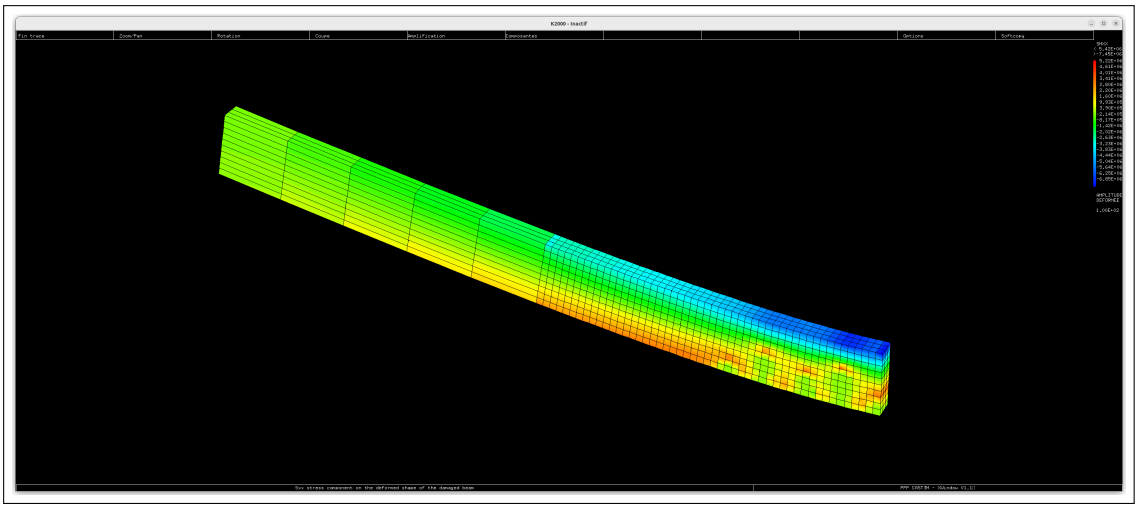


Figure A.11: Display of the $\sigma_{xx}$ component of the stress field on the deformed shape of the damaged beam (see page 52).

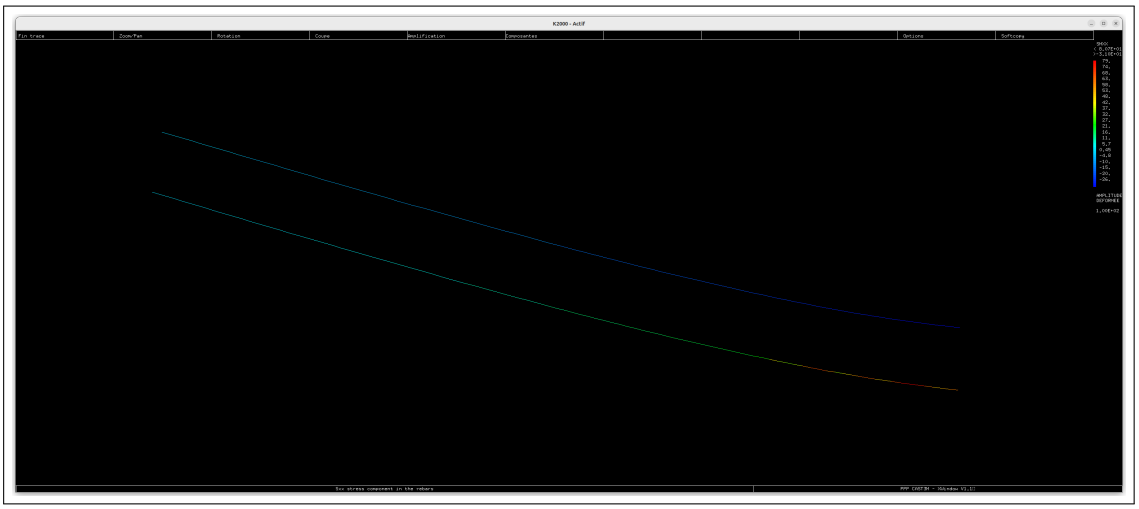


Figure A.12: Display of the longitudinal stresses in the rebars on the deformed configuration of the damaged beam (see page 53).

# Index


**Symbols**

'ACCRO' . . . . . 46
'BARR' . . . . . 43
'BLAN' . . . . . 52
'BLANC' . . . . . 22
'CACH' . . . . . 16, 52
'CONS' . . . . . 43
'CONTINUATION' . . . . . 51
'CONTRAINTES' . . . . . 51
'CUB8' . . . . . 9
'DEPL' . . . . . 18, 47
'DEPLACEMENTS' . . . . . 51
'DFIN' . . . . . 41
'DIFFUSION' . . . . . 18
'DIME' . . . . . 9
'DIMP' . . . . . 48
'DINI' . . . . . 41
'ECHO' . . . . . 16
'EFFX' . . . . . 53
'ELASTIQUE' . . . . . 18
'ELEM' . . . . . 9
'ENTIER' . . . . . 36
'ESTIMATION' . . . . . 51
'EXCEL' . . . . . 26
'FTRA' . . . . . 30
'GRANDS_DEPLACEMENTS . . . . . 49
'GRIL' 'POIN' 'GRIS' . . . . . 24
'HPP' . . . . . 49
'LEGE' . . . . . 24
'LOGIQUE' . . . . . 36
'MARQ S CROI' . . . . . 24
'MAXI' . . . . . 19
'MECANIQUE' . . . . . 18
'MODELE' . . . . . 55
'NU' . . . . . 18
'OPEN' . . . . . 30
'PASAPAS . . . . . 51
'PREDICTEUR' . . . . . 49
'PSC' . . . . . 30
'REACTIONS' . . . . . 51
'REACTUALISE' . . . . . 49
'RESU' . . . . . 16, 20
'ROUG' . . . . . 47, 52
'ROUGE' . . . . . 22
'SMXX' . . . . . 53
'SOUTYPE' . . . . . 51
'STRI' . . . . . 23
'TEMPS' . . . . . 51
'TEMPS_CALCULES' . . . . . 48
'THERMIQUE' . . . . . 18
'TITRE' . . . . . 16
'TRAC' . . . . . 30
'TRAC' 'PSC' . . . . . 56
'TRAN' . . . . . 16
'UX' . . . . . 18
'UY' . . . . . 18
'UZ' . . . . . 18
'Uz (m)' . . . . . 24
'VARIABLES_INTERNES' . . . . . 51
'VERT' . . . . . 24
'VTK' . . . . . 26
'WTABLE' . . . . . 51
'X' . . . . . 30
'XBOR' . . . . . 24
'XGRA' . . . . . 24
'YOUN' . . . . . 18
* . . . . . 41

**A**

appu . . . . . 23
ARET . . . . . 19, 23, 52
ARGU . . . . . 36

**B**

BLOQ . . . . . 18, 44

**C**

CHAI . . . . . 34, 47, 56
CHAN . . . . . 39
CHAR . . . . . 48
CHARMECA . . . . . 55
CHARTHER . . . . . 55
CHPO . . . . . 24
COMM . . . . . 28
COOR . . . . . 19
COUL . . . . . 19, 42

**D**

DEBP . . . . . 36
DEFO . . . . . 21, 51–53, 56
defo . . . . . 53
DENS . . . . . 12
DEPI . . . . . 47
DESS . . . . . 24, 31, 44
DIME . . . . . 34, 35, 52
DROI . . . . . 13–15
droi . . . . . 23

**E**

ELEM . . . . . 23
ENTI . . . . . 11
ENVE . . . . . 42
ERRE . . . . . 36
ET . . . . . 18, 19, 24, 34, 35, 41, 43, 46, 52
EVOL . . . . . 24, 44
EXCO . . . . . 47, 53
EXTR . . . . . 43, 55

**F**

FACE . . . . . 41, 45
factoria . . . . . 39
factorial . . . . . 37, 38
FIN . . . . . 35, 36
FINP . . . . . 36
FINS . . . . . 35
FINSI . . . . . 35
FLOT . . . . . 11
FORC . . . . . 19
FORM . . . . . 53, 54

**I**

INDE . . . . . 35
INFO . . . . . 11, 14, 15, 38, 39
INVE . . . . . 24
ITER . . . . . 35

**L**

LANG . . . . . 12
LECT . . . . . 34
LIRE . . . . . 26
LIST . . . . . 12, 14, 16, 20, 28, 30, 33, 37, 52
list . . . . . 52
LIST . . . . . 37, 54

**M**

MANU . . . . . 39
MATE . . . . . 18, 39, 43, 44
MAXI . . . . . 23
MAXI . . . . . 54
MESS . . . . . 33, 34
MINI . . . . . 23
MODE . . . . . 18, 39, 43
MOT . . . . . 33, 49
MOTS . . . . . 34
MULT . . . . . 36

**O**

OPTI . . . . . 9, 11, 12, 16, 24, 25, 38, 40, 56

**P**

PARATHER . . . . . 55
PASAPAS . . . . . 7, 48–51, 54–58
PECHE . . . . . 58
PERSO1 . . . . . 54–57
PERSO2 . . . . . 55
PLUS . . . . . 42
POIN . . . . . 19
PROG . . . . . 34, 56

**Q**

QUIT . . . . . 35, 36

**R**

REAC . . . . . 49
REDU . . . . . 53, 55
RELA . . . . . 44, 46
REPE . . . . . 35
RESO . . . . . 20
RESP . . . . . 36
REST . . . . . 25
RIGI . . . . . 20

**S**

SAUV . . . . . 24, 25
SI . . . . . 35
SIGM . . . . . 22
SINO . . . . . 35
SINON . . . . . 35
SORT . . . . . 26

**T**

TABL . . . . . 35
tabl . . . . . 24
TITR . . . . . 19, 52, 53
titr . . . . . 24
TRAC . . . . . 16, 19, 21–23, 30, 42, 47, 52–54, 56
TRAN . . . . . 15
TYPE . . . . . 35

**U**

UNPAS . . . . . 54



**V**

VECT . . . . . 47
VOLU . . . . . 16, 24