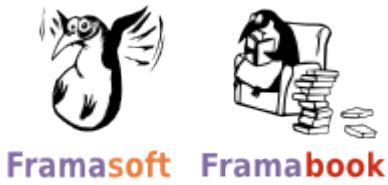
Encouragez les Framabooks !



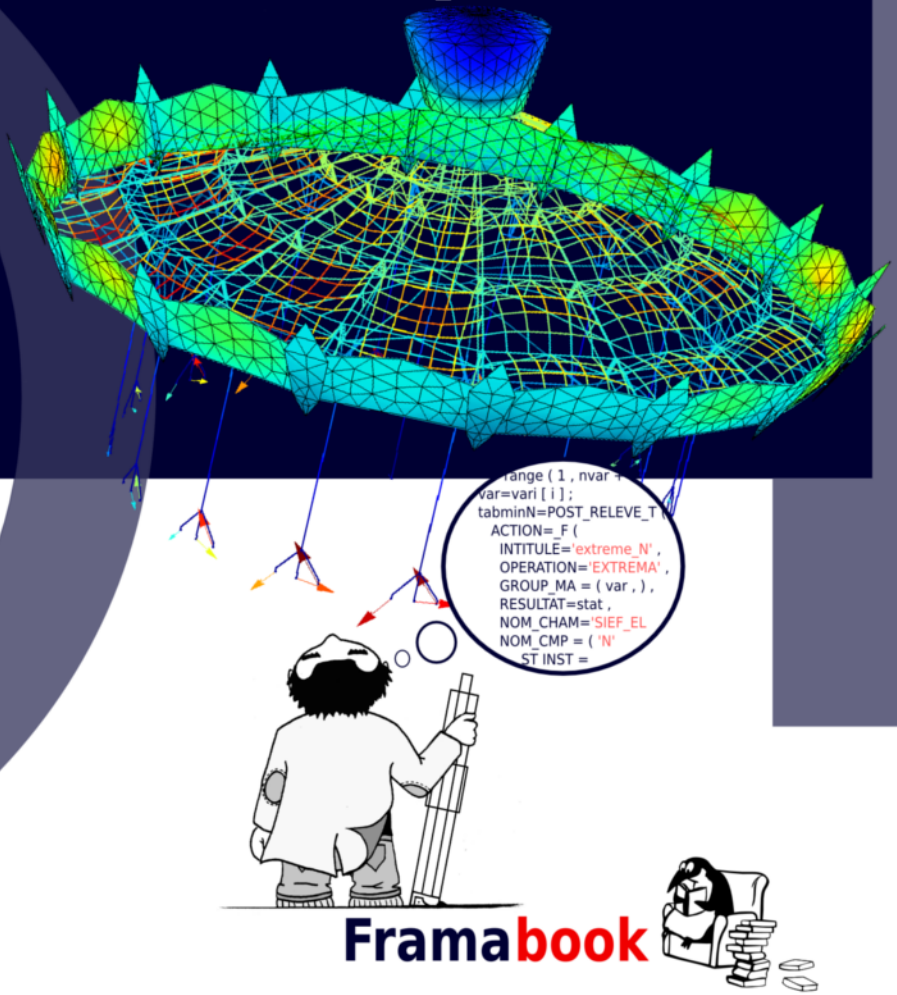
You can use **Unglue.it** to help to thank the creators for making *Beginning with Code_Aster* free. The amount is up to you.

Click here to thank the creators

Jean-Pierre Aubry

Beginning with *Code_Aster*

A PRACTICAL INTRODUCTION TO FINITE ELEMENT METHOD USING *Code_Aster, Gmsh* and *Salome*



Framabook

Jean-Pierre Aubry

Beginning with *Code_Aster*

A practical introduction to finite element method using *Code_Aster* Gmsh and Salome

Framabook

le pari du livre libre

Version 1.1.1

Framasoft a été créé en novembre 2001 par Alexis Kauffmann. En janvier 2004 une association éponyme a vu le jour pour soutenir le développement du réseau. Pour plus d'information sur Framasoft, consultez http://www.framasoft.net.

Se démarquant de l'édition classique, les Framabooks sont dits « livres libres » parce qu'ils sont placés sous une licence qui permet au lecteur de disposer des mêmes libertés qu'un utilisateur de logiciels libres. Les Framabooks s'inscrivent dans cette culture des biens communs qui, à l'instar de Wikipédia, favorise la création, le partage, la diffusion et l'appropriation collective de la connaissance. Le projet Framabook est coordonné par Christophe Masutti. Pour plus d'information, consultez http://framabook.org.

ISBN : 979-10-92674-03-3
Prix : 58 euros
Dépôt légal : décembre 2013, Framasoft (impr. lulu.com Raleigh, USA)
5, avenue Stephen Pichon – 75013 Paris
Pingouins : LL de Mars, Licence Art Libre
Couverture : création par Nadège Dauvergne, Licence CC By
Illustration de couverture par Pascal Galeppe
Mise en page avec LaTeX

Preface

I am very proud to introduce this book which is a perfect example of the Code_Aster Open-Source community vitality. I will take the opportunity to tell a story, in which I am pleased to have played my part, brilliantly continued by Pascal MIALON, *François* WAECKEL *and Christophe* DURAND.

Founder's goals

The development and release process started more than 25 years ago. Following a report of Yves BAMBERGER to the scientific council, Paul CASEAU, head of EDF R&D, decided in March 1988 to provide EDF needs a global answer.

. . . A unique code for mechanics

During the years 1975-1985, the deployment of the finite element method for mechanical analysis led to multiple software developments. The scattering of EDF R&D teams as well as the need for numerical tools dedicated to typical engineering subjects ended with numerous specific programs rather than one single general software.

Procurement, release and maintenance cost control showed the need, for EDF, for an involvement in a unique software integrating the previous developments. Started as a multidisciplinary project with voluminous specifications and a short timing, it became a great help in know-how management.

. . . A durable code

The need for a long run management of the project cases at EDF plants yielded heavy quality control requirements, regarding nuclear safety requirements. These projects were led by scattered and frequently renewed teams: the software should therefore play an important part regarding experience feedback in the long run.

. . . A code for expertise

The life span analysis of electricity power plants components requires to take into account: the loading history, the manufacturing process, the possible repairs. Thus, the required numerical models should respond to more demanding calculation hypothesis than the one used for classical engineering, systematically including: non-linear approaches, thermal effects, dynamic loading stresses, fluid structure interactions.

These models imply a R&D activity, whose results have to be submitted, as quickly as possible, to an industrial qualification when releasing the software.

Today it is an integrated computation system

. . . A solver

From a functional point of view, *Code_Aster* is a solver for mechanics: from a given geometric representation of the structure, the meshing, it implements the finite element method to produce result fields -deformation, stress, energy, material state-.

. . . For wider needs

But users have wider software needs: from CAD, to mathematical process and graphical result analysis in the end. Obtaining, ever and ever, more realistic input data -loadings, material parameters- requires that *Code_Aster* should be able to interact with other software dealing with the related physical phenomena. If a mechanical engineer may accept to

cope with several tools, they certainly expect a “seamless” software offer regarding service, interoperability, version and quality control.

. . . Pre and post integrated processing

The fact that the software integrated from the beginning numerous dedicated tools to the code itself, including mathematical processing, allowed at the time to capitalize a broad panel of engineering technologies as well as regulations and codes approach. This also enabled a “global” certification of the code while avoiding the use of external components with uncontrollable life cycle -commercial spreadsheet. . .-.

. . . From Python supervision to Salome-Meca

The initial architectural design turned out to be relevant and adaptable enough to allow the integration of new methodological input with no need of rebuilding. In 2000, Python was chosen to be the supervisor language and it increased the code modularity in dedicated tools and specific mathematical computation.

The present outcome of this approach is Salome-Meca: an integrated and complete GUI made available on the engineer workstation. With the same quality controlled software, the mechanical engineer can handle the whole simulation, from CAD to coupling with other physical solvers.

Developing the network . . .

. . . A durable process and agile software development

Very early what is now known as “agile software development” was settled down .

Needs are assessed through experience feedback sheets, and do not rely on any global specification. Each request follows its own development cycle, from requirements needs to final delivery, and do not depend on the other development cycles. The continuous update of the development version, available to any user, allows a quick feedback and enables improvement as well as debugging. Finally, developers, code architects and potential users may discuss continuously and more particularly at the stage of integration by the development team.

. . . A network supporting the industry innovation

This network structure, prefiguring the free software style contributes to the computational mechanics research. In twenty years, almost 200 developers and more than twenty doctoral students added their contribution to *Code_Aster* .

A numerical model is considered valid when it can be delivered to the operational teams in a qualified version. It has to fulfill three requirements: reliability, robustness, performance. *Code_Aster* reviewing procedures contribute to this goal through a set of requirements regression test cases, documentation, rules for system architecture. . .- as part of an incremental development process.

. . . Quality first

Documents tracing the code improvements enhance its quality. Apart from these everyday an independent validation occurs: the critical expertise by third parties. This type of reviews, along the versions, enhance the code qualified domain. This qualification, together with Quality Insurance, is essential when studies relating to nuclear safety are undertaken. Ultimately it does benefit to everyone.

Four major audits reinforced *Code_Aster* quality approach as well as its original network development strategy. Thus, thanks to the engineering services requirements and support, collective trust in the software patiently aroused. It is now well established at EDF and beyond.

Assisting internal users . . .

Code_Aster deployment has only possible by keeping a constant relationship between development teams and users.

The first major contribution to quality approach was to provide a user documentation with each new version. But also a theoretical justification of the models used in each verification test cases. This documentation is a great contribution greatly to EDF know-how in the mechanical field. The 20,000 pages current corpus is enhanced or reviewed with every new addition.

. . . Informing and sharing

The users' club, with its local correspondents, is the place where one can share experience and discuss with the development team. The most representative studies, displaying important issue or setting up advanced

modeling, are presented at the annual users' day as well as in the free *Code_Aster* information letter.

. . . Training

Hundreds of users follow annual training sessions, basic or more advanced ones, dedicated to dynamics, contact and non-linear analysis . . . The documentation broadness and the pedagogical dimension of more than 3,000 basic test cases allows efficient self-learning. The whole corpus of training documents is now accessible to the entire *Code_Aster* Open-Source Community.

. . . Listening and answering

Exchanges with users benefit of the use of a central main frame coupled with a cooperative system for experience feedback, particularly regarding the cases associated with confidential data.

A powerful simulation tool is nothing without the control and the development of a skill's network. Beyond the hotline, in house users have access to the expertise of R&D mechanical engineers for the implementation of complex studies.

For a wider distribution . . .

"You will not decide for yourself that you are good: others should have to tell you!" (Paul GODIN*, at project inception on January 2 th 1989)*

After a decade of development and three non public versions, releasing *Code_Aster* outside of the in-house user's circle was tempting. The appeal of an external recognition and possible new contributions supported this approach. In addition, valuing the industrial research results was fashionable at the end of the 90's.

. . .. Preparing code portability to prevent isolation

To reach the level of confidence and transparency suited to a tool used in nuclear safety, external distribution imposed itself. Preparing for distribution *Code_Aster*, a tool previously operated only in-house, on secure servers, required a significant number of proof tests to insure portability to other computers.

. . . Trying economic and commercial development

The commercial distribution of the operating and closed source version-was attempted in April, 1998. But this attempt imposed premature investments on the studies environment and above all developments in fields not closely related to our core business. Finally, the ”closed” nature of the code was in contradiction with its expertise goal.

The difficulty was also to find resellers who would accept to get involved in completely new software, particularly if they were already distributing one or more other software. This required from them a capacity that we failed to motivate, in an already very crowded market. This operation was rapidly stopped, in 1999.

. . . Evaluating the free software model

By the end of the twentieth century the free software model was becoming increasingly popular, whether for operating systems -Linux- or for internet development -Apache-. But application software were not yet concerned by this model. One first trigger was the Matra-Datavision choice to initiate, in 1999, the Open Cascade process which led to Salome development. At the same time INRIA [1] started the free distribution of SCILAB/Sicos. Thus we chose to evaluate the full implications of this model to an industrial simulation tool.

For a free diffusion . . .

This exhaustive evaluation of the free software model led to choose, in June 2001, the distribution under the GNU General Public License. The internet web site `www.code-aster.org` was opened on October, 19th 2001.

. . . Remaining close to internal users support

EDF did not intend to evolve to a software publisher. The requirement of maintaining, not increasing, the resources assigned by EDF to outside distribution implied *de facto* to give up the idea of any financial profit. The development team was at the service of the in-house engineering teams and had to remain so. It had also to contribute, as efficiently as possible, to the processing of the company sensitive dossiers.

[1] French national agency for computer science

. . . For a recognition by usage and a supplementary qualification

Beyond the 250 internal users, EDF wished to increase the credibility of its tools through a permanent confrontation with the main commercial tools and the evolution of the best industrial practices.

Open-Source allows a wide confrontation with the state of the art.

With more than the 5,000 annual downloads -20,000 for Salome-Meca- and through the public forum, the independent users community -industrial and academic- is rising . The Open-Source distribution allowed new proof tests and comparisons, a wider reading of the documentation as well as the detection of a few bugs. Several “benchmarks ”reinforced our confidence in the relevance of current models and the performance of the code.

. . . For spreading of competence

Several university teams run tutorial courses in finite elements or in mechanics linked to *Code_Aster*. Some service’s companies have a commercial activity in addition to their contract with EDF. The software talent pool has expended and is now durably established.

. . .For contribution collecting and cooperation building

The modular architecture of the code allows an easy integration of new modules and functionality. While following our quality criteria, significant contributions have already been integrated to the code.

Linus TORVALDS *promotes the free software model through the efficiency of the technical cooperation that it allows.*

Several academic partnerships have been developed. *Code_Aster* co-development agreements enable to develop, trough cost-sharing, common interest models , to guarantee their integration in the source code. They also serve as a foundation to collaborations in geomechanics, crack propagation, sliding contact, X-FEM. . . In 2012, a dozen of thesis had already been added on, and the process is still going on.

***Code_Aster*: a federative role for professional enhancement. . .**

Code_Aster had from the beginning a vocation for general simulation in mechanics.

. . . For all operations related to energy technology

EDF uses it today for modeling the behavior or pathology of its equipment:

- All components of the nuclear steam supply : pressure vessel, steam generators, primary motor-pump, primary and secondary circuits;
- The production equipment: turbo generator set and turbine components, towers and overhead power lines, both wind and hydro turbines;
- The civil engineering applications: pre-stressed concrete containment building, cooling towers, hydroelectric dams, nuclear waste storage sites.

The wealth of the available models -finite elements, behavior laws, analysis methods, post-processing- reflects this by itself.

. . . But also in unplanned areas

Taking advantage of the wide Open-Source deployment, *Code_Aster* has become by now an attractive industrial software. These means of dissemination have been used as an opportunity to show *Code_Aster* relevance in other mechanical simulation areas that were, for some, unexpected: the tectonic of geological layers, biomechanics, forming of steel or porcelain manufactured pieces, vibro-acoustic. . .

. . . For self-sustained development in digital simulation services

The possibilities offered by the GPL license have already allowed the emergence of services companies -training, assistance, development services, simulation deliveries. . .- in several countries. The progressive deployment of parallelism as well as its implementation in *Code_Aster*, particularly with the support of MUMPS Community, both allow these service providers to expand their offer to different organizations -SME, company of intermediate size . . .- with access to supercomputing centers HPC.

One remaining goal to realize would be to allow the emergence of distributors as added value resellers in various parts of the world.

. . . For the users community assistance

One of the latest avatars of the story, but not the last, was founding in July 2011 of *Code_Aster* Professional Network.

Code_Aster ProNet aims to increase *Code_Aster* Open-Source, and Salome-Meca, added values and make them better known. It allows links between the community actors beyond the technical exchanges on the forum.

Five prior modes of action were retained and are now shared by more than fifty members around the world, industrial organizations, research teams, service providers, teachers. . .:

- to create better quality multilateral exchanges -with EDF R&D and between members- by removing the limitations of a public and anonymous forum;
- to increase the members visibility on the various existing applications which have already been carried out and various usages;
- to disseminate insider information related to axes of evolution initiated by the members contributing to development, including EDF R&D ;
- to gather and structure common requests to services providers;
- to improve the collaborative development opportunities.

One to be continued!

The story has not ended yet and will continue to grow with the contributions of the new generations developers and users, after its twenty-fifth birthday in January 2014.

Let me conclude by warmly thanking Jean-Pierre AUBRY who has been using both his engineering and structural designer know-how to offer *Code_Aster* users' community a real learning guide for this specific software.

His high level practice within the forum can now be found here to help anyone discovering the software and also sometimes finite elements. His advices of best practice in the area of structural analysis will be very precious to every reader.

The way in which the numerical approach of structural computations is implanted in the different software is singular enough to require permanent explanations about what lies behind a series of "click".

In the *Code_Aster* command file explicitly writing everything you are doing is required. It will not only help you to remember what you did but it will also help anyone who will need to use the study results: that is traceability!

This guide is there to accompany you to enable you to join this adventure.

Jean-Raymond LÉVESQUE

Former member of *Code_Aster* Team (1989-2002)

Representative of *Code_Aster* ProNet

August 2013

Introduction

Code_Aster, acronym for Analysis of Structures and Thermomechanics for Studies and Research, is a general Finite Element Analysis software, coming from EDF (Électricité De France) R&D department.

It can handle mechanical, thermal and associated phenomena in all sort of analysis: linear statics, non-linear statics or dynamics, thermics and more.

Its development started in 1989. In 2001 EDF took the rather unusual decision, for a software of this size and scope, to put it under GNU GPL license.

Due to its numerous capabilities, *Code_Aster* is a very complex affair, and its somewhat unfriendly user interface makes the learning curve quite steep at the beginning.

The aim of this book is to ease up this steepness.

In itself *Code_Aster* does not provide any graphical interface for pre or post-processing, this task is left to third party software, and a few of them are also under GNU GPL licenses. This book introduces Gmsh and Salome for this task.

Last but not least, this book is in English, about a software whose native language is French, just as is the native language of the author. I hope the reader will forgive my poor level in Joyce's language.

Finally I have to express my thanks. To "La Machine"[1] for letting me apply, over many years, *Code_Aster* to many peculiar, and peculiar they were, engineering problems.

To Thomas de Soza, now in charge of EDF R&D *Code_Aster* core development team for a helpful, tedious yet uncompromising proof reading and suggestions.

Jean-Pierre AUBRY

Nantes, October 2011, November 2013.

[1] `www.lamachine.fr`, the web site helps to understand why a free software was accepted here!

Contents

CHAPTER 1

Foreword

1.1 How to read

The first chapters from chapter 3, to chapter 9, are meant to be read by the newbie user in a sequential order; these go from simple to more complicated examples concerning the geometry, the mesh and the model. At the same time we go from simple to more complicated analysis and post-processing tools, starting with a simple Salome-Meca analysis and finishing with a stand alone *Code_Aster* with ASTK and STANLEY post-processing.

The following chapters are more independent as we dive into non-linear analysis, 3D modeling with contact and friction. Their reading assumes that the fundamentals from the previous chapters have been understood. However the experienced user may find some useful hints, here or there, throughout the chapters.

We refer to the gigantic documentation just as little as necessary. For example we refer to `DEFI_MATERIAU` just enough to define the mate-

rial used in our examples not wandering into the 157 pages of U4.43.01 document[1].

Going through the examples on the computer needs to have the programs fully installed. If this is not the case, one needs to go to appendix E to learn how to, and do it!

1.2 What this book is

It is step by step introduction to the finite element analysis using *Code_Aster*.

In this book we take a few complete examples, from a practical engineer point of view, from the beginning to the solution.

It is limited to mechanical static analysis.

1.3 What this book is not

It is not a text book about mechanical engineering or structural design. Generally speaking, a successful finite element analysis of a structure, i.e. one giving a result without any runtime error or warning, is NO proof of a soundly designed structure!

It is not a text book about finite element theory, and I will not risk myself giving any reference in this matter in the bibliography section, particularly in English.

It is not a *Code_Aster* description of dynamic analysis like seismic response.

It is not a *Code_Aster* description of non purely mechanical analysis (hydraulics, thermics, coupling, heat induced stress like in welding).

It is not a collection of benchmarks trying to compare, to the last digit and with various types of meshing or modeling, the results of some problems with some well known analytical solutions.

[1] A large number of them referring to rather exotic cases, at least for the beginner!

1.4 What is *Code_Aster*

Code_Aster is a Finite Element Analysis engine, given the appropriate data files it will produce a set of result files. Used in this basic manner, one does not see anything on the screen, except a flow of lines in a terminal, and that's "all". But at the end of a successful run the problem is solved and this is the "all".

Except for the command file editor "Eficas", the problem manager tool "ASTK", and the post-processing tool "STANLEY"[1], *Code_Aster* does not provide any GUI.

The whole *Code_Aster* bundle is about 900 Mb on the hard drive.

1.5 What is Gmsh

"Gmsh: a three-dimensional finite element mesh generator with built-in pre- and post-processing facilities" by Christophe GEUZAINE and Jean-François REMACLE, is a private venture[2].

Gmsh can produce a geometry and convert it to a usable mesh. This is usually done in its own GUI, but could as well be done in a script only manner, with no graphical output.

Gmsh can also read, manipulate and display results in its GUI.

At less than 200 Mb on the hard drive Gmsh is a small boy.

1.6 What are Salome and Salome-Meca

"SALOME is an open-source software that provides a generic platform for pre- and post-Processing for numerical simulation. It is based on an open and flexible architecture made of reusable components." It is a joint venture from CEA[3], EDF, and OPEN CASCADE.

[1] All these three may well not be used at all.
[2] According to the authors "Gmsh" means nothing.
[3] "Commissariat à l'Énergie Atomique" French agency for nuclear power.

Salome[1] is a very complex piece of software including: a geometry module, a mesh module, some pot-processing modules, and much more.

It is also a very hefty guy filling almost 4 Gb on the hard drive.

Salome-Meca is a bundle which allows to run *Code_Aster* within Salome in a simple manner which allows to make studies in a single GUI!

1.7 Typographic

Menus or buttons more generally actions in the GUIs are typeset like this: Menu. *Code_Aster* reserved words, operators, function names, keywords, concepts are typeset like this: `MECA_STATIQUE`. File, or extension, names are typeset like this: *.comm*. And windows, or dialog boxes, titles as they appear on the screen are typeset like this: **Gmsh**.

"X" or "YOZ", in uppercase refer to a global direction or plane, while "x", in lower case, refers to a local direction i.e. in the local element coordinate system.

1.8 Software version

The example meshes and command files have been verified with the following versions:

- Gmsh up to 2.8.4;
- *Code_Aster* stable from 10.8 to 11.4;
- Salome-Meca 2013-2;
- Salome 6.6.0.

[1] Apart from being known as the queen of Calchis, Salome is the acronym for "Simulation Numérique par Architecture Logicielle en Open Source et à Méthodologie d'Évolution" which can be crudely translated into "Numerical simulation by means of open source software architecture and with methodological evolution".

Tests were conducted on different machines, all of them running GNU Linux[1]. In addition tests have been carried as well on openSUSE 12.3 in a VirtualBox®running in Windows 7®.

This book deals only with a standalone installation, on a single machine, where the programs are installed in some adequate directories, and the studies are run from a directory with read-write access for the regular user[2].

[1] SuSE 10 to openSUSE 12.3 flavors, the screen caps are from a customized “fvwm” desktop environment.

[2] It is strongly advised NOT to run the programs as *root*, with super user privileges

CHAPTER 2

Beginning with...

If everything seems to be going well, you have obviously overlooked something.

11th Murphy's law.

A study work-flow in *Code_Aster* is sufficiently different from most "black box" Finite Element Analysis codes to justify this chapter.

A finite element analysis is usually performed to foresee the mechanical[1] behavior of some structure, or part, at the design stage.

For this, we make an idealized model from the plans, sometimes preliminary sketches, of the structure. When we say idealized, we mean that every part is represented by components or elements which are understandable by the finite element program.

Then we apply some loads to the structure and check its behavior under the various loads and compare the behavior with requirements.

For example, for a walking bridge most standards or building codes, for instance Eurocode in Europe, may require:

[1] At least in the limited scope of this book.

- a maximum vertical deflection under the sole action of a given number of people on the bridge, the service load;
- a stress level below the yield stress under the action of 1.35 times the dead load plus 1.5 times the service load;
- the same with the addition of several wind loads with their own coefficients.

In addition:

- the concrete engineer may want to know the reaction forces on the ground for some well specified load cases;
- very often a buckling analysis is required;
- the engineer may want to look at natural frequency and mode shape.

How is this set up in *Code_Aster*?

2.1 Preparing the geometry and mesh

The geometry of the problem is prepared from the CAD file of the designer. At the very exception of some solid machine parts, a CAD drawing file always needs to be [deeply] modified to be transformed in a valid mesh.

In this book we draw and mesh the examples from scratch using either Gmsh or Salome. This means:

- outlining the borders of the structure, or the neutral fiber in case of beams, with points and lines;
- building the required surfaces and volumes from these lines;
- creating and naming groups of objects, e.g.:
  - all the lines supporting the same beam section;
  - all the surfaces having the same thickness;

- all the points, lines or surfaces carrying an identical load;
- all the points, bearing some ground fixation.

From this geometry we produce a mesh, which is a subdivision in elementary objects (point, edge, square, triangle, tetrahedron, etc) changing if necessary the overall or local density or size.

This mesh is saved in a *Code_Aster* understandable format, for instance the MED format.

2.2 Preparing the command file

The mesh is only a topological entity[1], we need to instruct *Code_Aster* what to make of it in order to solve a physical problem and output the results.

This is done with a command file, the so called *.comm* file which is essentially a flow of operations. It is written in *Code_Aster* language[2].

The minimum blocks in this file are roughly:

- reading and modifying the mesh;
- assigning finite elements to it;
- defining the properties of the materials which are used;
- assigning the materials to the model;
- assigning geometric properties to the structural element, (shell thickness, beam section, etc.);
- setting boundary conditions and loads;
- choosing the adequate analysis type and solving;
- calculating the forces, stress, strain or more;
- writing the results in files, in ASCII and binary format.

[1] It consists of nodes, elements and groups.
[2] Which is just a package of specific commands interpreted in Python.

I will allow myself a little digression here: in *Code_Aster* a command file must be written as an ASCII text while most commercial software hide the creation of this command file behind numerous mouse clicks in no less numerous dialog boxes popping out, here and there, on the screen[a].

Code_Aster behavior calls for some more forethought from the user compared to the "click factories"[b], and is thus a much better tool for learning what all this business of "finite element calculation" is made of inside.

[a] Although for most of them this file exists and can be edited by hand!
[b] "Usines à clics" to retain the terminology used by C. DURAND former *Code_Aster* development manager.

2.3 Launching the calculation

Next is the actual execution of the study. This can be done in various ways which are [almost] all[3] described in this book:

- in the GUI of Salome-Meca, easy and simple;
- in the ASTK interface, less easy, more powerful;
- on the command line with 'as_run', useful for scripting.

2.4 Viewing the results

Once the calculation is done[4] we are able to read the results in their ASCII format. And, more pleasing, to display colorful views on the screen in Gmsh or in the Salome Post-Pro or ParaVis modules.

Like...

[3] As far as the basic ones are concerned.
[4] Don't panic if it does not work on the first go, after many years of practice it hardly ever works on the first run for me!

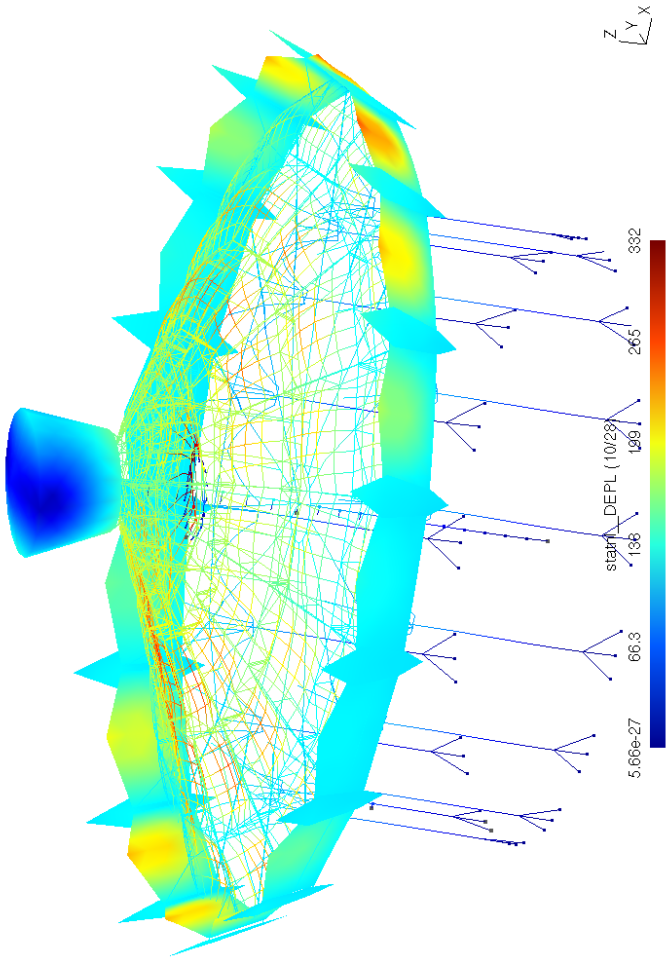


FIGURE 2.1: Post-processing view of 'Ile de Nantes Carousel' top structure, with beams, plates and cables, built in steel wood glass and canvas, 22 m diameter

CHAPTER 3

Drawing and meshing a simple frame

In this first chapter, we draw a simple structure in Gmsh and make a mesh from the geometry.

3.1 Drawing the frame with Gmsh

As a first example we study an A frame[1], 1 m high, with a 2 m span, with one load under the form of a 10 kg mass at three quarter chord of the span and another load of 100 N, vertical downwards at the quarter chord. This frame is sketched in figure 3.1.

Note: the structure is symmetrical in geometry not in mass.

The first thing to be done is to create a directory for the problem, anywhere we have read-write permissions, we name this directory *frame1*.

Now let's launch Gmsh, we should have something looking like figure 3.2:

[1] It really is an inverted U shape, and could support a swing for the kids in the garden.

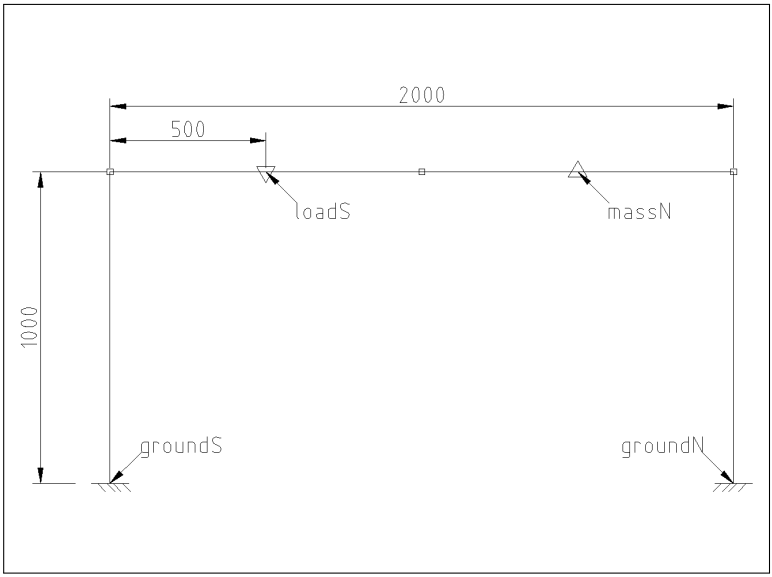


FIGURE 3.1: Sketch of frame1

- one large window, named **untitled.geo**;
- a menu bar on the top;
- a status bar at the bottom;
- a little trihedron at the bottom right;
- a "Modules" tree on the left-hand side.

From the main window choose the menu File > Save As... and save the file in the directory *frame1* recently created, name it *frame1.geo*. Push Yes in the next dialog box. We can notice that the file name has not changed in the **Gmsh** window title, we must open the *frame1.geo*, file through the File > Open... menu.

That's an important feature of Gmsh. When we do Save As..., Gmsh saves a copy but does not switch to this newly created file, it keeps working on the current file, as important is the fact that every change made

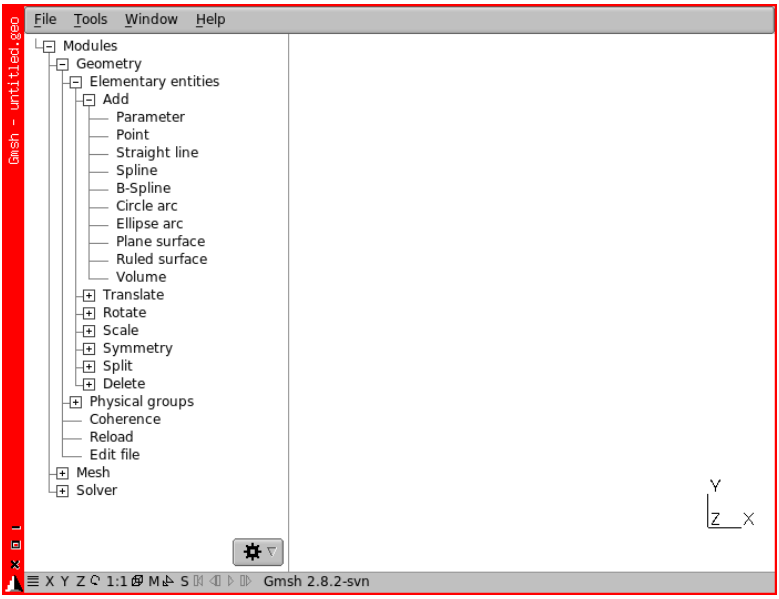


FIGURE 3.2: Gmsh, with some modules expanded

in the GUI is saved on the fly in the *.geo* file. This behavior may look strange to the beginner used to common spreadsheet and text processor, but once understood, we wonder how we would do without it. This is the common behavior of almost all database processing programs[1].

In the tree, under - Modules:

- push on the button Geometry, then Elementary entities;
- then Add > Point, another little windows pops up;
- type in the coordinates x=0, in X coordinate box, y=-1000, z=0;
- in the box Prescribed mesh element size at point enter 100;
- push Add.

[1] Likewise there is no Revert menu option, the way is to edit the.geo file.

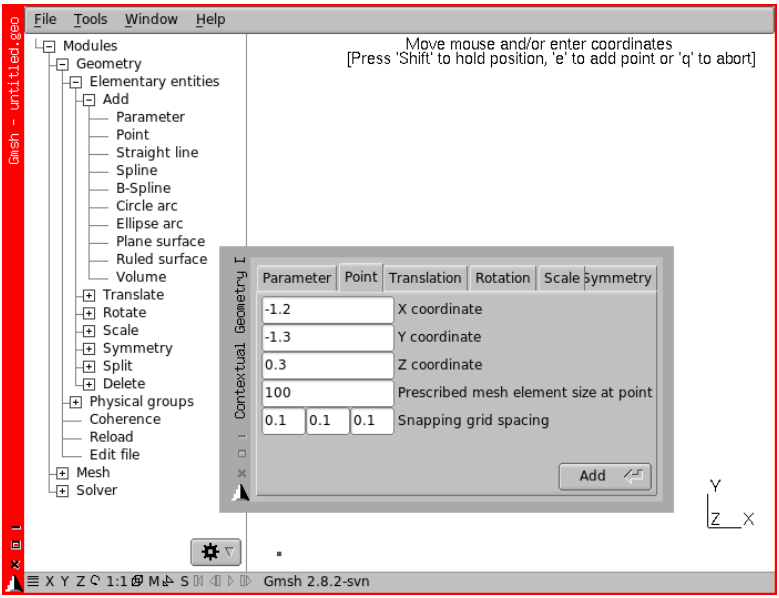


FIGURE 3.3: Creating a Point in Gmsh

We can notice some instructions about what can be done in the context displayed at the top center of the main window. We can also notice at the bottom of the **Gmsh** main window, the status bar reminding us what is the active command. We can see the newly created point as a little square box in Gmsh main window. Now let's open the file *frame1.geo* with our favorite text editor[1]. We can see something like this:

```
// Gmsh project created on Mon Nov  18 08:32:45 2013
cl1=100;
Point(1) = {0, -1000, 0, cl1};
```

Let's enter a new point, in Gmsh, with x=0, y=-1000, z=1000 then push Add. In the text editor we can see some warning that the source file has been changed, refresh the text window, a new line appears:

[1] As far as I am concerned, it's Kate.

```
// Gmsh project created on Mon Nov  18 08:32:45 2013
cl1=100;
Point(1) = {0, -1000, 0, cl1};
Point(2) = {0, -1000, 1000, cl1};
```

In the text editor add a new line like this:

```
Point(3) = {0, -500, 1000, cl1};
```

Save the file, then in Gmsh, push on Geometry » Reload, the new point appears in the main window.

Switching from the text editor to Gmsh GUI is the real way to do things efficiently!

In Tools menu choose Options, then the entry Geometry, by checking or unchecking the Point numbers option we can trigger the display of the numbering of the points. Complete the geometry with a point:

```
Point(4) = {0, 0, 1000, cl1};
```

Now in the Geometry module we push on the button Elementary entities » Add » New » Straight line, with the mouse we connect the points by choosing the 2 end points of each line, while looking carefully at the request in the top center of the screen.

Then in Geometry » Elementary entities » Symmetry » Line fill the dialog box with A=0, B=1, C=0, D=0 [1], and we pick all the lines with the mouse and type e. The structure has been duplicated by symmetry about the XOZ plane and looks like figure 3.4, depending on the visibility toggles.

Note: new points are automatically created and there is no duplicate point at Point 4. We can have a look in the text editor to see how this is done. And the **Gmsh** window looks like figure 3.4

[1] The symmetry is defined by a vector, A, B, C being the 3 components (in X, Y, Z) of the vector normal to the plane and D the distance in space from the plane to the origin of the global axis, in short the 4 parameters of the plane equation in space!

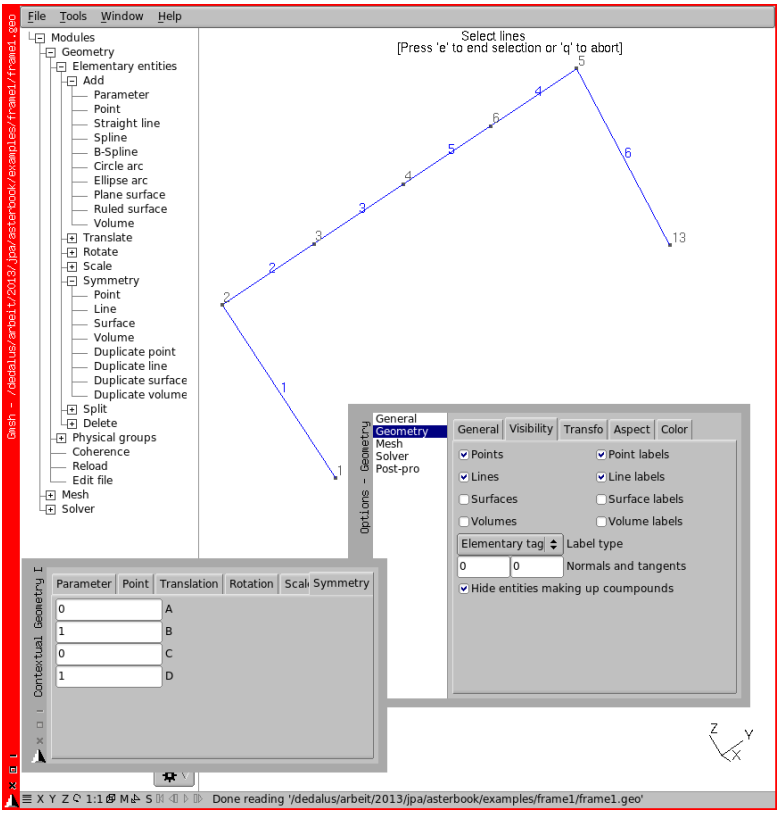


FIGURE 3.4: Geometric entities in Gmsh, with symmetry dialog box

Now we put together in 'Groups' the geometric entities which share some properties[1], in the tree Geometry » Physical groups » Add » Line, pick with the mouse the four lines being part of the frame top bar, once this is done type e, like in figure 3.5.

We can notice a new line appended in the text editor:

```
Physical Line(7) = {2, 3, 5, 4};
```

[1] In Gmsh groups are called Physical

The actual digit may be different! Edit it so it becomes:

```
Physical Line("topbeam") = {2, 3, 5, 4};
```

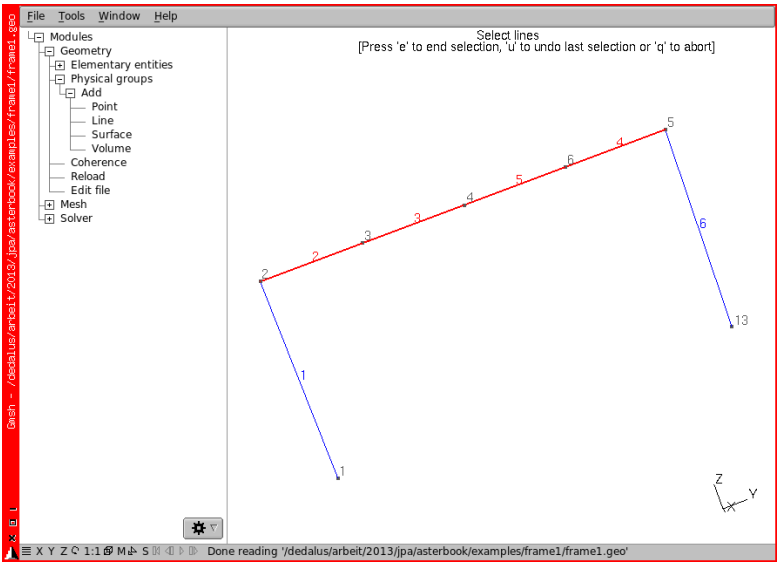


FIGURE 3.5: 4 Lines selected to make a Physical

Before we continue a very important warning: within a mesh file which is to be processed later by *Code_Aster* we must not use group names longer than 24 characters[1] .

This gives the name 'topbeam' to the group formed by the four lines 2, 3, 5, 4. We keep going on either from the GUI or from the text editor until the groups looks like below, notice we have also groups of points. The final *.geo* file looks like this[2]:

```
// Gmsh project created on Mon Nov  18 08:32:45 2013
cl1=100;
Point(1) = {0, -1000, 0, cl1};
Point(2) = {0, -1000, 1000, cl1};
Point(3) = {0, -500, 1000, cl1};
```

[1] With versions earlier than 11.3.10 it used to be 8 characters, a major improvement!
[2] // at the beginning of a line means this line is a comment in Gmsh, just like in C language.

```
Point(4) = {0, 0, 1000, cl1};
Line(1) = {1, 2};
Line(2) = {2, 3};
Line(3) = {3, 4};
Symmetry {0, 1, 0, 0} {
  Duplicata { Line{2, 3, 1}; }
}
Physical Line("topbeam") = {2, 3, 5, 4};
Physical Line("mast") = {1, 6};
Physical Point("groundS") = {1};
Physical Point("groundN") = {13};
Physical Point("loadS") = {3};
Physical Point("massN") = {6};
```

Some Gmsh hints:

- At the left end of the status bar there are several buttons;
  - X, Y, Z set the view from the selected axis, pushing ⇧ at the same time reverses the axis;
  - S means snapping, sometimes we have to deactivate it, if it was activated by error (if activated it appears in red);
  - a few other buttons whose use is explained in the balloon help.
- To select multiple items at once, we push ctrl and draw a bounding box with the mouse, any entity having a bit of itself within the bounding box is selected.

3.2 Meshing with Gmsh

At this stage, we have only geometric entities, we have to transform them into mesh entities.

In the tree push Mesh 〉 1D, the model is meshed like in figure 3.6.

In the **Options** window toggle the boxes in Visibility for Mesh and Geometry to see what has been created.

In the text editor we can see an entry at line 2: 'cl1=100', this entry 'cl1' is repeated as the fourth entry at every node. This is the elementary mesh size, named as 'Characteristic length' in Gmsh which is applied around this given point. Change cl1=500 in this line, save in the text editor, reload in Gmsh and mesh again, we can see quite a coarser mesh.

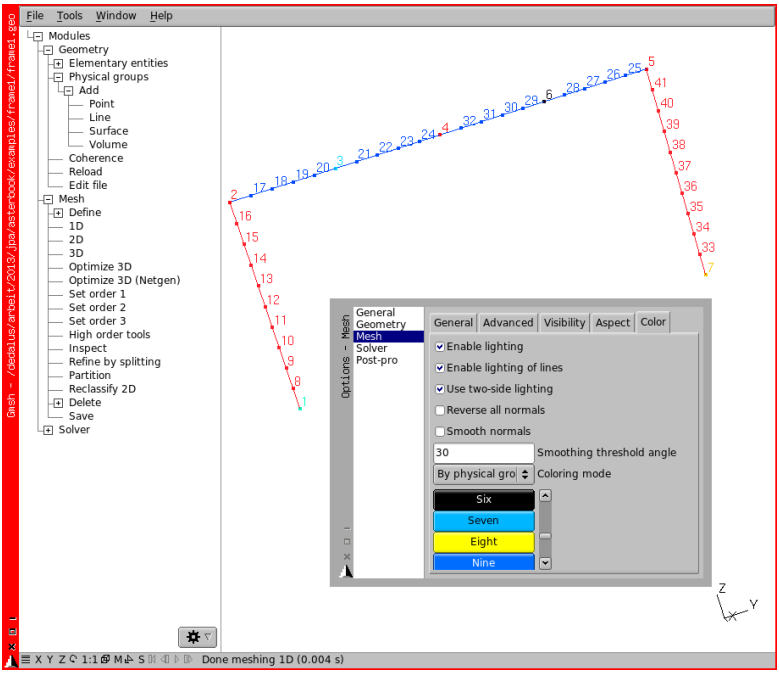


FIGURE 3.6: This is the mesh, with node numbers and Options > Color > Coloring mode set to By physical group and a bit of tweaking with the colors for the elements

We can just as well push Mesh > Define > Elements size at points in the tree, then fill the Value with any number let's say 10 and pick up one of the point, then do the meshing. The mesh is refined around this point.

Now, to save the mesh we do, menu File > Save As..., in the Format pull down list choose MED File (*.med) and save the file as *frame1.med*. A small dialog box named **MED Options** pops up, like at the top of figure 3.7:

Leave unchecked the box Save all (ignore physical groups) so as to save the groups created in the mesh.

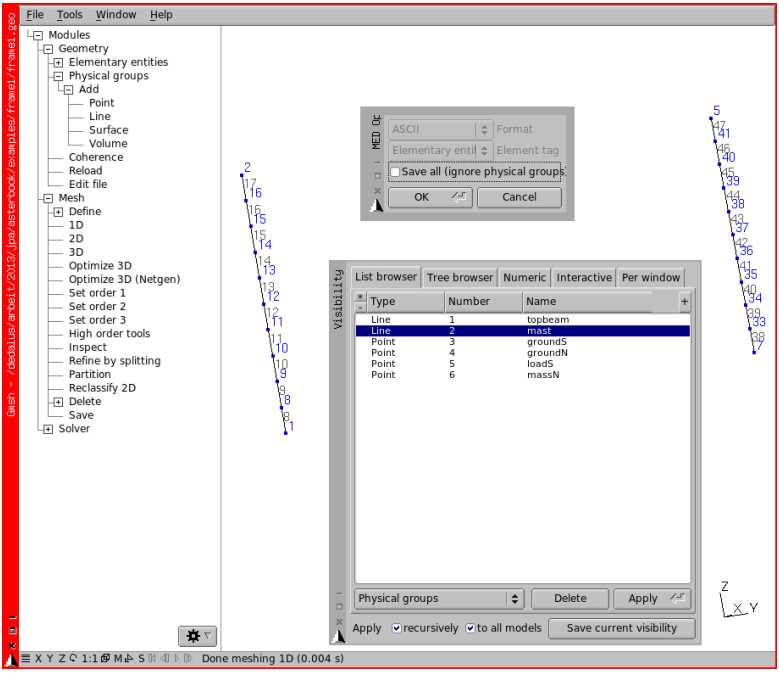


FIGURE 3.7: Only Physical 'mast', 'massN' and 'loadS' are set to visible, with node and line numbers, and coloring by elements

More about that:

- If we leave this box unchecked all the elements belonging to groups are translated to the *.med* file, and *ONLY* these elements, Points elements (or nodes) as well. This means we have to create groups for everything we need later.
- If we check this box all the elements are translated, all of them, but *WITHOUT* any group definition.

This is very important, as when we do the meshing Gmsh meshes all the entities it founds without any distinction, it is only at save time that the real mesh is saved as stated above.

One more hint about using Gmsh, go to the menu Tools » Visibility in the list browser tab at the bottom, pull down to Physicals groups, here we can

play with the visibility groups by groups to see if things look like what we want, like in figure 3.7.

To finish with, pushing ctrl + L displays the 'Message Console' which is a log, and appears at the bottom of the main window, and can be re-sized.

3.3 Drawing and meshing it with Salome-Meca

The drawing and meshing job we have just done in Gmsh can also take place in Salome-Meca[1], we do that in appendix D.

3.4 Calculating it with Salome-Meca

In this first example, we are going to run the study in Salome-Meca. Salome-Meca is used here as a front end to automate tasks for *Code_Aster*, we describe here only the basics to get a result for this example.

In order to calculate, Salome-Meca needs as input:

- a mesh file, we have just made it,
- a command file **.comm*, we are just going do this in chapter 4.

[1] Or in a stand alone version of Salome.

CHAPTER 4

Creating the command file

Never make anything simple and efficient when a way can be found to make it complex and wonderful.

13th Murphy's law.

In this second chapter, we write a command file to study the behavior of the structure we drew and meshed in chapter 3.

We command to study the behavior under a static linear analysis, with several load cases.

And we command to produce the results files.

- First but important remark: in this first example we start straight away with a multiple load case, solved at once, study.
- Second remark: we describe the file, step by step, regardless of the tools used to produce it[1].

[1] Section 4.19 gives a quick glance at Eficas.

- Third remark: in this book we show the command files with a small indent scheme rather than in the classical *Code_Aster* or Eficas manner with its very large indent[1].
- Fourth remark: all the concepts we create in the command files, in this book, are typeset in lowercase, we leave uppercase to the exclusive use of *Code_Aster* reserved words[2].

Now let's have a look, bit by bit, at the commented command file for the *frame1* study.

Note: lines beginning with # are comments[3]. Many things are explained directly in the code with commented lines at the right place. A line like, #U4.21.01 means: the following command or operator is described in the U4.21.01 document in *Code_Aster* documentation available on `http://www.code-aster.org`.

And before going any further a very important warning: within a command file, we must not use concept names[4] longer than 8 characters, which is not much!

4.1 Beginning with `DEBUT()`

Not much to say here, but in more demanding examples the `DEBUT()` procedure may take some arguments, however any *.comm* file must begin with this procedure, whose documentation is U4.11.01.

```
#U4.11.02
DEBUT();
```

[1] *Code_Aster* command files do not require any mandatory indent at all as long as we do not use conventional Python inside them, however a command must start at the beginning of a line without any indent.

[2] *Code_Aster* is case sensitive: 'TRUE', 'true' and 'truE' point to 3 different objects while 'True' is a Python reserved word. And a concept can be named 'modele', while 'MODELE' is a *Code_Aster* reserved word.

[3] More generally the *.comm* file follows the Python syntax.

[4] The names created on the left hand side on the "=" sign.

4.2 Reading and modifying the mesh

Here, we read the mesh which by default is assigned the Fortran unit LU 20 (Logical Unit 20). The `INFO_MED=2` line provides a more verbose mode of what is read, the output being in the *.mess* file. This feature is quite useful for checking that *Code_Aster* is actually reading what we expect. And of course we read a file in MED format, the one we saved previously[1] .

```
#U4.21.01
mesh=LIRE_MAILLAGE(
    INFO=1,
    #INFO_MED=2,
    UNITE=20,
    FORMAT='MED',
);
```

Now we create some groups within the mesh.

With `CREA_GROUP_MA=_F(NOM='TOUT',TOUT='OUI',)`, we create a group named `'TOUT'` which contains all the elements which are in the mesh[2].

With `CREA_GROUP_NO=_F(TOUT_GROUP_MA='OUI',)`, we create a group of nodes for every group of element, each one of these groups contains all the nodes belonging to the parent element, and bears the same name. This is very useful with MED file imported from Gmsh because Physical Points (groups) are translated as groups of elements, `GROUP_MA`[3], in the MED file. Thus we can, later on, apply boundary conditions to nodes.

[1] *Code_Aster* can read other format:

- ASTER (.mail) format, mostly used in the test cases, U4.21.01;
- IDEAS (.unv) format with `PRE_IDEAS`, U7.01.01;
- Gmsh (.msh) format with `PRE_GMSH`, U7.01.31.

[2] This may be useful but we do not use it in this command file.

[3] Point element, `POI1`.

```
#U4.22.01
mesh=DEFI_GROUP(
    reuse =mesh,
    MAILLAGE=mesh,
    CREA_GROUP_MA=_F(NOM='TOUT',TOUT='OUI',),
    CREA_GROUP_NO=_F(TOUT_GROUP_MA='OUI',),
);
```

With `CREA_GROUP_NO=_F(GROUP_MA='mast',)`, we would have done the same only for the 'mast' group. Many other useful things can be done in this command, for example using the boolean `UNION` of groups to simplify their manipulation later on. We can also have a look at the commands: `CREA_MAILLAGE`, `ASSE_MAILLAGE`, `MODI_MAILLAGE...` in the generous *Code_Aster* documentation.

After that is a little trick that saves the modified mesh so it can be checked within Salome or Gmsh. However, this line is only translated if the study is run within ASTK, not in Salome-Meca, some more about this later...

```
IMPR_RESU(
  FORMAT='MED',
  UNITE=71,
  RESU=_F(MAILLAGE=mesh,),
);
```

4.3 Making a finite element model from the mesh

Here, we transform the mesh in a proper finite element model by assigning some properties to the mesh elements, as explained in U4.41.01[1]. For example:

- we assign, `AFFE=`, to the mesh groups

  `GROUP_MA=('topbeam','mast',);`
- the type of beam, `MODELISATION='POU_D_T';`
- telling as well that we deal with a mechanical behavior, `PHENOMENE='MECANIQUE';`

[1] In *Code_Aster* the `MAILLAGE` contains only the topology of the mesh and nothing else.

as explained in U3.11.01.

On the next line, we assign to the mesh group 'massN', in fact it is a point, the properties of a discrete element, as explained in U3.11.02 so as to put a mass on it later on.

```
#U4.41.01
model=AFFE_MODELE(
    MAILLAGE=mesh,
    AFFE=(
        _F(
          GROUP_MA=('topbeam','mast',),
          PHENOMENE='MECANIQUE',
          MODELISATION='POU_D_T',
        ),
        _F(
          GROUP_MA=('massN',),
          PHENOMENE='MECANIQUE',
          MODELISATION='DIS_T',
        ),
    ),
);
```

Here the mesh is transformed in a model in the most straightforward manner, the whole mesh at once. We may also not transform some of the mesh elements, thus excluding them from the analysis, just to see what would happen if they were not there, quite useful in preliminary design!

4.4 Defining materials

With this operator, we define a material, with its properties.

```
#U4.43.01
steel=DEFI_MATERIAU(ELAS=_F(E=210000.,NU=0.3,RHO=8e-9),);
```

As we are performing a simple linear elastic analysis only the Young modulus `E` and the Poisson ratio `NU` are needed. We also add the mass density `RHO` as we want to load the frame with its own weight[1].

[1] The mass density needs to be given in t/mm^3, as we use mm as length unit and N as force unit, more about unit system in chapter 6.1 .

4.5 Assigning materials to elements

Here we assign to the beam elements groups the material properties 'steel'.

Note: the discrete element, 'massN', is not assigned a material, this is not necessary, but *Code_Aster* does not complain if we assign one.

```
#U4.43.03
material=AFFE_MATERIAU(
    MAILLAGE=mesh,
    AFFE=_F(GROUP_MA=('topbeam','mast',), MATER=steel,),
);
```

4.6 Giving properties to elements

With beam, and in general with structural elements, this is a rather tricky part.

Here, we define a set, or concept, named 'elemcar' and assign some physical properties to the model elements, this is described in U4.42.01, and this document is most important and should be read carefully!

We begin with the beam properties, with some alternatives which are commented.

```
#U4.42.01
elemcar=AFFE_CARA_ELEM(
    MODELE=model,
    POUTRE=(
        #the vertical members are rectangular section
        #(40x20 mm) with a thickness of 1.5 mm
        _F(
            GROUP_MA=('mast',),
            SECTION='RECTANGLE',
            CARA=('HY','HZ','EP',),
            VALE=(40, 20, 1.5,),
        ),
        #same with the horizontal bar
        _F(
            GROUP_MA=('topbeam',),SECTION='RECTANGLE',
            CARA=('HY','HZ','EP',),VALE=(40, 20, 1.5,),
        ),
        #next lines would have produced the same section properties
        #_F(
            #GROUP_MA=('topbeam',),SECTION='GENERALE',
            #CARA=(
```

```
                #'A','IY','IZ','AY','AZ','EY','EZ',
                #'JX','RY','RZ','RT',
            #),
            #VALE=(
                #171, 11518, 34908, 1.5, 1.5, 0, 0,
                #26700, 20, 10, 12,
            #),
        #),
    ),
```

This next section deals about orientation of the beams, for the moment everything is commented and we review that in detail in chapter 6.3 .

```
    #in the next lines we would give the 'mast' group
    #the same orientation as the top beam
    #leave it commented at first
    #ORIENTATION=_F(
        #GROUP_MA=('mast',),
        #CARA='VECT_Y',
        #VALE=(1.0, 0.0, 0.0,),
    #),
    #and in the next ones we can rotate
    #to the 'topbeam' along its axis,
    #leave it commented at first
    #ORIENTATION=_F(
        #GROUP_MA=('topbeam',),
        #CARA='ANGL_VRIL',
        #VALE=90.0,
    #),
```

In this last section, we set the properties of the mass element.

```
    #in the next line we give to the discrete element
    #the property of a point mass
    #(CARA='M_T_D_N'), and give it
    #the value of 0.01 tonnes e.g. 10 kg
    DISCRET=(
        _F(GROUP_MA='massN', CARA='M_T_D_N',VALE=(.01),),
        #following block set stiffness of point element 'massN'
        #to null stiffness
        #although this is not necessary,
        #commenting this block would raise a warning,
        #unimportant in this case
        _F(
            GROUP_MA=('massN',),
            CARA='K_T_D_N',
            VALE=(0, 0, 0,),
            REPERE='GLOBAL',
     ),
  ),
);
```

4.7 Setting boundary conditions

Now, we assign the boundary conditions and loads, in several sets as explained in U4.44.01, again, this document is also most important and should be read carefully! In this first set we fix in all 6 DOFs at the bottom of the masts.

```
#U4.44.01
ground=AFFE_CHAR_MECA(
    MODELE=model,
    DDL_IMPO=_F(
        GROUP_NO=('groundS','groundN',),
        DX=0,DY=0,DZ=0,DRX=0,DRY=0,DRZ=0,
    ),
);
```

I strongly advise not to mix fixations and loads in a boundary condition set and to split the loads in as many elementary AFFE_CHAR_MECA as is logical.

DDL is the french acronym for DOF.

4.8 Setting loadings

In a second set, we apply the gravity, PESANTEUR, to the beam groups and also to the discrete element. GRAVITE, the acceleration of gravity is rounded off to $10000mm/s^2 = 10m/s^2$ so as to produce a 100 N load, symmetrical to the force load, with due allowance to the multiplier used later on.

Despite its name, “pesanteur” meaning “gravity”, this keyword may be used to apply any uniform acceleration, in any direction, to any group of a model.

```
selfwght=AFFE_CHAR_MECA(
    MODELE=model,
    PESANTEUR =_F(
        GRAVITE=10000,
        DIRECTION=(0, 0, -1),
        GROUP_MA=('topbeam','mast','massN',),
    ),
);
```

In the third set, we apply a vertical force of 135 N to the node at the first quarter of the top bar.

```
cc=AFFE_CHAR_MECA(
    MODELE=model,
    FORCE_NODALE=_F(GROUP_NO=('loadS',),FZ=-135,),
 );
```

And in the fourth set we apply a distributed vertical force of 0.1 N per unit length (here mm) to the top bar.

```
cr=AFFE_CHAR_MECA(
    MODELE=model,
    #FORCE_POUTRE=_F(GROUP_MA=('topbeam',),FZ=-0.1,),
    FORCE_NODALE=_F(GROUP_NO=('topbeam',),FZ=-0.1*2000/17,),
    #17 is the number of nodes in the group 'topbeam'
);
```

In the above lines we have commented the line with `FORCE_POUTRE` as *Code_Aster* cannot yet calculate with more than one distributed load on beam elements! However it knows how to in a single `AFFE_CHAR_MECA`[1].

This is a very annoying limitation. The work around is to replace it with an equivalent nodal force applied to the `GROUP_NO=('topbeam',)` which we have created earlier with `DEFI_GROUP` and which contains all the nodes of `GROUP_MA=('topbeam',)`.
This equivalent nodal force is the distributed load multiplied by the length of the beam elements [2].

4.9 Stepping for the load case

Here we define some step functions which are applied to the loads.

For example, for the gravity force 'selfwght', the function 'selfw_m' is applied, 0 at instant 2, 1.35 at instant 3 and for all instant after 3, except at instant 6 where it drops down to 0.

[1] More about this in chapter 19.1 .

[2] A trick would be to have *Code_Aster* calculate the exact nodal forces depending on the element length, this is very feasible with some Python coding.

For the nodal force 'cc', the function 'cc_m' is applied , 0 at instant 3, 1 at instant 4 and 5, with 0 at any instant before 3, and again 0 at instant 6.

Finally for the distributed force 'cr', the function 'cr_m' is applied , 0 at instant 4, 1.5 at instant 5, dropping down to 1 at instant 6 where it is acting alone, with 0 at any instant before 4[1].

```
selfw_m=DEFI_FONCTION(
    NOM_PARA='INST',
    VALE=(2,0, 3,1.35, 5,1.35, 6,0),
    PROL_GAUCHE='CONSTANT',PROL_DROITE='CONSTANT',
);
cc_m=DEFI_FONCTION(
    NOM_PARA='INST',
    VALE=(3,0, 4,1, 5,1, 6,0),
    PROL_GAUCHE='CONSTANT',PROL_DROITE='CONSTANT',
);
cr_m=DEFI_FONCTION(
    NOM_PARA='INST',
    VALE=(4,0, 5,1.5, 6,1,),
    PROL_GAUCHE='CONSTANT',PROL_DROITE='CONSTANT',
);
```

4.10 Stepping for the solution

Here we define a step function which is applied to the calculation, we calculate every single instant from 2, `DEBUT=2.0`, to 6, `JUSQU_A=6`, with a step of 1, `PAS=1.0`

```
liste=DEFI_LIST_REEL(
    DEBUT=2.0,
    INTERVALLE=_F(JUSQU_A=6,PAS=1.0,),
);
```

We must understand what is done just above. We perform the calculation at five steps or instants, `INST` in the *Code_Aster* jargon, from 2 to 6, for every integer:

- at `INST` 2, we have no load at all[2];

[1] The word instant must be taken here with some restriction, it just only means step, it is not related to time (in time unit, seconds) but the same word, `INST`, is used by *Code_Aster* to specify the step in static analysis and a real instant, in seconds, in a dynamic analysis.

[2] We start at 2 but we might have started at 0, -5 or 1.342! In linear static the choice is completely arbitrary and left to the user.

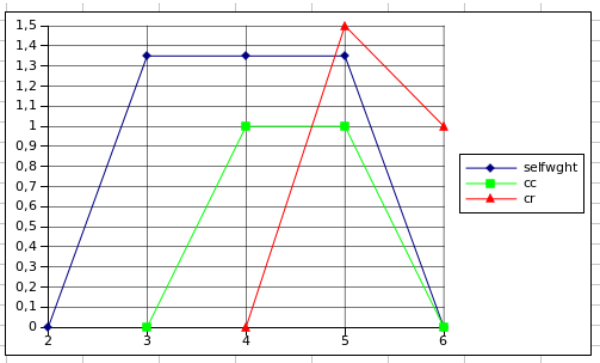


FIGURE 4.1: Load steps

- at INST 3, 1.35 times the self weight load;
- at INST 4, 1.35 times the self weight load, plus 1 times the nodal load 'cc';
- at INST 5, 1.35 times the self weight load, plus 1 times the nodal load 'cc', plus 1.5 times the "distributed" load 'cr';
- at INST 6, 1 times the "distributed" load 'cr' is acting alone.

This is illustrated in figure 4.1

4.11 Analysing it

After having defined loads and material now is the time for the main calculation. We conduct here a linear static calculation and therefore we use MECA_STATIQUE:

- with the previously defined model, MODELE=model;
- the defined material set, CHAM_MATER=material;
- the properties, CARA_ELEM=elemcar;
- the boundary conditions and loads defined under EXCIT=...

- at all the instants prescribed in `LIST_INST=liste`;
- and store the results in the concept `stat`.

```
#U4.51.01
stat=MECA_STATIQUE(
    MODELE=model,
    CHAM_MATER=material,
    CARA_ELEM=elemcar,
    #with the load, or boundary condition defined in EXCIT
    #with the applied step function where needed
    EXCIT=(
        _F(CHARGE=ground,),
        _F(CHARGE=selfwght,FONC_MULT=selfw_m,),
        _F(CHARGE=cc,TYPE_CHARGE='FIXE',FONC_MULT=cc_m,),
        _F(CHARGE=cr,TYPE_CHARGE='FIXE',FONC_MULT=cr_m,),
    ),

    #the calculation is made for all instant in this list
    LIST_INST=liste,
    #we can give a title to this study
    #TITRE='my_title'
);
```

If `FONC_MULT` had been omitted in `EXCIT` and `LIST_INST` omitted as well we would have run the solution for one load case being the sum of all the `EXCIT` instances.

4.12 Calculating results

For now the concept `stat` contains only the displacement at nodes, as well as the forces at Gauss points, we must enhance it with some useful results on elements.

Enhance! that's why we use the keyword `reuse`, which is curiously written in lower case.

```
#U4.80.01
stat=CALC_CHAMP(
  reuse =stat,RESULTAT=stat,
  CONTRAINTE=(
    'SIEF_ELNO','SIPO_ELNO','SIPM_ELNO',
  ),
  FORCE=('REAC_NODA',),
);
```

For the versions earlier than 11, commands to calculate stresses and forces were a bit different, this is the subject of chapter 18.7 .

4.13 Calculating and printing the mass of the model

The next lines calculate the structural mass of the given groups and put the results in the *.resu* file in a tabular format as explained in U4.81.22. This should always be done to check the consistency of the model and calculation.

```
masse=POST_ELEM(
    RESULTAT =stat,
    MODELE=model,
    MASS_INER=_F(GROUP_MA=('topbeam','mast','massN',),),
    TITRE= 'masse'
);
#U4.91.03
IMPR_TABLE (
  TABLE=masse,
  FORMAT_R='1PE12.3',
)
```

I reckon that checking the calculated mass with the estimates made otherwise is the prime test of the model validity. In structures made of beams this calls for an increased mass density of the materials so as to cope with all the unmodeled bits and pieces which contribute nonetheless to the loading of the structure by their own weight.

`FORMAT_R='1PE12.3'` prints the numbers with twelve characters, 3 digits after the decimal point and one digit to the left of the decimal point, for example $-2.762E+03$ while `FORMAT_R='E12.3'` would print $-0.276E+04$.

4.14 Printing the reactions

Checking the reactions against what is expected is just as well very important, so, in the next lines we calculate the sum of the reactions and put the results in the *.resu* file, in a tabular format, and in the conventional format.

```
#U4.81.21
sum_reac=POST_RELEVE_T(
  ACTION=_F(
    INTITULE='sum reactions',
    GROUP_NO=('groundS','groundN',),
    RESULTAT=stat,
    NOM_CHAM='REAC_NODA',
    TOUT_ORDRE='OUI',
    RESULTANTE=('DX','DY','DZ',),
    MOMENT=('DRX','DRY','DRZ',),
    POINT=(0,0,0,),
    OPERATION='EXTRACTION',
    ),
);
IMPR_TABLE (TABLE=sum_reac,FORMAT_R='1PE12.3',)
```

This first abstract of code puts the sum of the individual reaction on the groups of node 'groundS', 'groundN', in a table named 'sum_reac' with the `OPERATION='EXTRACTION'` keyword. Then the table is printed with `IMPR_TABLE`.

`MOMENT=('DRX','DRY','DRZ',)`, computes the moment of this sum of reaction about the point whose coordinates are stated in `POINT=(x,y,z,)` coordinates.

Using `RESULTANTE=('DX','DY','DZ','DRX','DRY','DRZ')`, would have printed the sum of the individual reaction force as well as the sum of the individual reaction moment, the sum of the individual moment being not very meaningful!

```
#then in tabular format per group of node
reac1=POST_RELEVE_T(
    ACTION=_F(
        INTITULE='reactionsS',
        GROUP_NO=('groundS',),
        RESULTAT=stat,
        NOM_CHAM='REAC_NODA',
        TOUT_ORDRE='OUI',
        RESULTANTE=('DX','DY','DZ'),
        OPERATION='EXTRACTION',
    ),
);
#very simple form
#IMPR_TABLE (TABLE=reac1,)

#or more detailed with field separator
IMPR_TABLE(
    TABLE=reac1,
    FORMAT='TABLEAU',
```

```
    UNITE=8, #this is also the default value
    #whichever separator that suits the needs
    SEPARATEUR=' * ',
    FORMAT_R='1PE12.3',
    TITRE='reaction_1',
    INFO=2,
);

reac2=POST_RELEVE_T(
    ACTION=_F(
        INTITULE='reactionsN',
        GROUP_NO=('groundN',),
        RESULTAT=stat,
        NOM_CHAM='REAC_NODA',
        TOUT_ORDRE='OUI',
        RESULTANTE=('DX','DY','DZ'),
        OPERATION='EXTRACTION',
    ),
);
IMPR_TABLE (TABLE=reac2,FORMAT_R='1PE12.3',)

#now we print the individual reactions
#in the .resu file in RESULTAT format
#U4.91.01
IMPR_RESU(
    MODELE=model,
    FORMAT='RESULTAT',
    RESU=_F(
        NOM_CHAM='REAC_NODA',
        GROUP_NO=('groundS','groundN',),
        RESULTAT=stat,
        FORMAT_R='1PE12.3',
    ),
);
```

4.15 Printing some key values

It is a wise idea to print in the *.resu* file some key values, here we print:

- the minimum and maximum value of the vertical component of the displacement in the 'topbeam' group;
- the minimum and maximum value of the axial force in the 'mast' group of elements;
- the minimum and maximum value of the bending moments in the two beam groups.

This print out gives a quick glance at the overall basic results as a check of what was expected. It can also be used to compare with the values displayed in the post-processor views!

```
IMPR_RESU(
    MODELE=model,
    FORMAT='RESULTAT',
    RESU=(
        _F(
          RESULTAT=stat,
            NOM_CHAM='DEPL',
            NOM_CMP='DZ',
            GROUP_MA=('topbeam',),
            FORMAT_R='1PE12.3',
            VALE_MAX='OUI',VALE_MIN='OUI',
        ),
        _F(
          RESULTAT=stat,
            NOM_CHAM='SIEF_ELNO',
            NOM_CMP='N',
            GROUP_MA=('mast',),
            FORMAT_R='1PE12.3',
            VALE_MAX='OUI',VALE_MIN='OUI',
        ),
        _F(
          RESULTAT=stat,
            NOM_CHAM='SIEF_ELNO',
            NOM_CMP=('MFY','MFZ',),
            GROUP_MA=('topbeam',),
            FORMAT_R='1PE12.3',
            VALE_MAX='OUI',VALE_MIN='OUI',
        ),
    ),
);
```

4.16 Printing some others results in ASCII file *.resu*

Then we put some stress results, all components for every element in the specified groups in the *.resu* file.

```
IMPR_RESU(
    MODELE=model,
    FORMAT='RESULTAT',
    RESU=(
        _F(
            RESULTAT=stat,
            NOM_CHAM='SIPO_ELNO',
            GROUP_MA=('topbeam','mast',),
```

```
            FORMAT_R='1PE12.3',
        ),
    ),
);
```

This is a rather restricted set printed to ASCII file, we may want more or different fields according to the study.

4.17 Printing results for graphical viewing, MED file

Finally we put in a *.med* file the results we want to be displayed, in graphical format, in the Post-Pro or ParaVis modules of Salome-Meca or in the Post-pro module of Gmsh.

```
#U7.05.21
IMPR_RESU(
    FORMAT='MED',
    UNITE=80,
    RESU=_F(
        #following lines print on the named groups,
        GROUP_MA=('topbeam','mast',),
        RESULTAT=stat,
        NOM_CHAM=(
            'DEPL',
            'SIEF_ELNO',
            'SIPO_ELNO',
            'SIPM_ELNO',
            'REAC_NODA',
        ),
    ),
);
```

4.18 Ending the command file with `FIN()`

Well not exactly finally, as the *.comm* file must be ended with the procedure `FIN()`, which like the procedure `DEBUT()` would take some arguments in a more complicated study.

```
#U4.11.02
FIN();
```

4.19 Preparing the command file with Eficas

In the above sections we described step by step the command file regardless of the way it is prepared.

As far as I am concerned I write it with a text editor, Kate[1].

However the *Code_Aster* package provides a useful tool for editing the *.comm* file, called Eficas, it provides integrated syntax check capabilities, it also provides a choice of allowable keywords for every command or operator.

We can launch it within Salome-Meca, in the Aster module and then in the menu Aster > Tools > Run Eficas. Figure 4.2 shows an **Eficas** window with *frame3.comm* loaded.

The main draw back is that as soon as we have some Python in the *.comm* file Eficas cannot be used to edit the file, making it a useful tool at learning stage but not so much for serious studies.

The acronym EFICAS means, in french "Editeur de FIchier de Commandes ASter" which translates into "Aster command file editor" but it is also a pun with the french word "efficace" meaning "efficient"!

[1] Gedit, SciTE, Geany, Emacs, or any text editor does the job just as well. Depending on the editor we use we may give it some "syntaxic coloration", for a more comfortable reading. Usually Python syntax hightlighting is just enough.

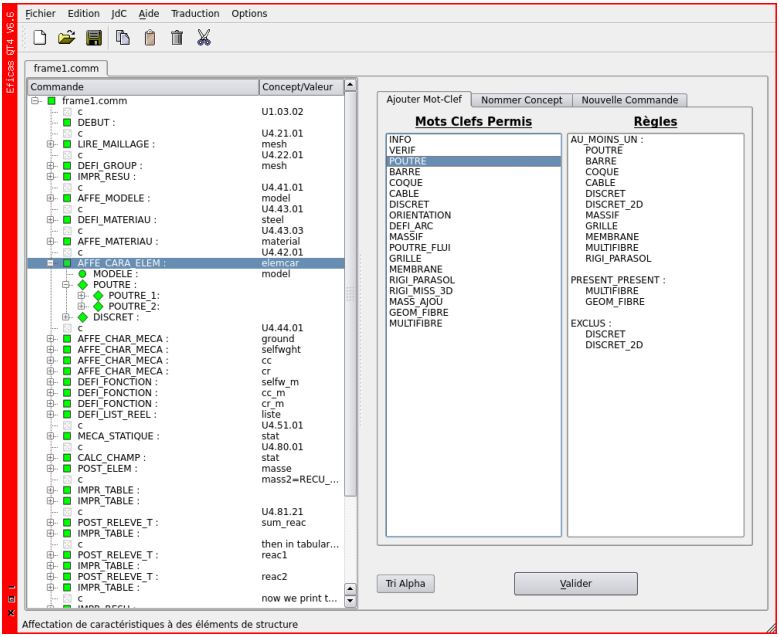


FIGURE 4.2: Eficas in action

CHAPTER 5

Solving in Salome-Meca

It works better if you plug it in.

Sattinger's law.

In this third chapter, we assemble the mesh file produced in chapter 3 with the command file produced in chapter 4 in a Salome-Meca study.

We run this study and look at the results in the Post-Pro module of Salome-Meca.

5.1 Putting it together

We have now a working *.comm* file saved in the study's directory, we can launch Salome-Meca. Once this is done:

- we first need to create a study with File > New;
- then we save it with File > SaveAs... under the name *frame1.hdf* in the study directory;

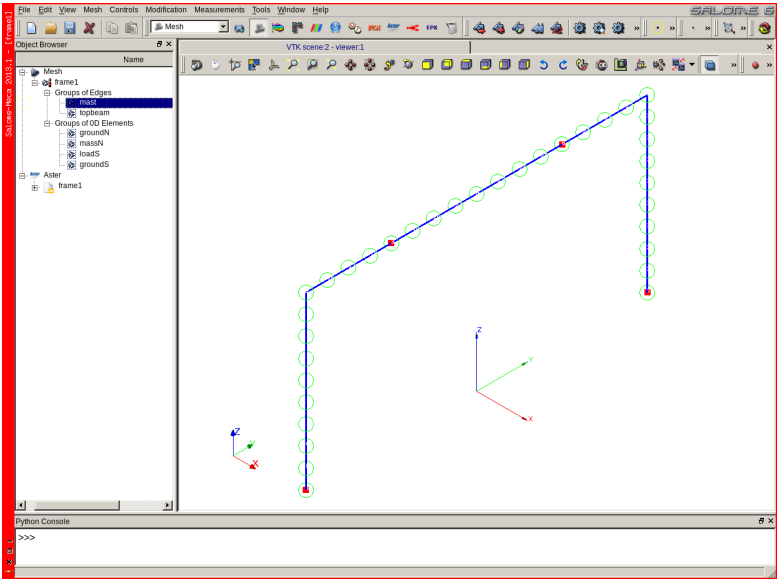


FIGURE 5.1: Mesh imported in Salome-Meca

- we go in the pull down list to Mesh module;
- then menu File > Import > MED file and after the appropriate navigation we open *frame1.med*;
- in the browser on the left-hand side click on the + sign/button along Mesh, then RMB click on frame1 > Show.

We can now see the mesh previously created, like in figure 5.1, with the 1D elements in blue, 0D in red and nodes in a green circle[1].

We can play a bit with the various possibilities in the menus or by RMB clicking on entities.

- Then we switch to Aster module using the pull down list to Aster module;

[1] To get this exact view it is necessary to tweak a bit in File > Preferences... > Mesh.

- then menu Aster > Add study case, a new window pops up, looking like figure 5.2;
- name the case as 'frame1';
- choose from disk in the Command file sub menu, then with the icon just on the right choose the *frame1.comm* file;
- do the same with the MESH to choose *frame1.med* .

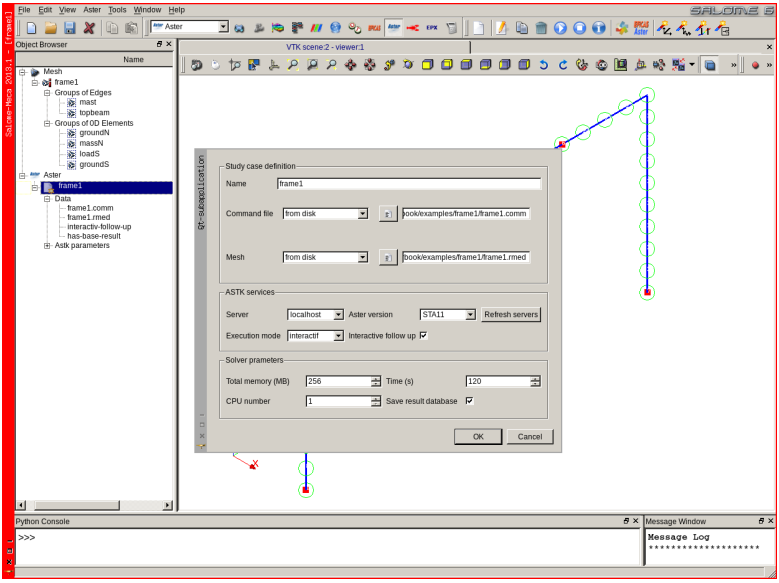


FIGURE 5.2: Setting the study parameters

All the other options can be left as they are by default, the meaning of Total memory and Time is self explanatory, the values can be changed if needed.

The check box Save result database can be unchecked at this stage, leaving it does not do any harm[1].

[1] We are going to see later how helpful is the database.

- Click OK;
- in the browser click on the + sign along Aster;
- then RMB click on frame1 > Run.
- within a few seconds a little green "checked" sign should appear to the left of 'frame1' in the browser, and a RMB click on frame1 > Status should pop up an **Information** box stating an 'OK', otherwise see chapter 6.4 .

5.2 Viewing the results with Salome-Meca

A new entry named Post-Pro has been created in the browser[1]:

1. click on the + sign along it;
2. RMB click on frame1.med;
3. on the + sign left of mesh;
4. on the + sign left of Fields;
5. on the + sign left of stat_DEPL;
6. RMB click on 4 > Activate Post-Pro Module;
7. again on 4;
8. choose Deformed Shape and Scalar Map;
9. and finally click OK on the next box.

and we can see a nice picture, like figure 5.3, of the deformed shape of the structure under the load at `INST` 4[2].

We can do the same at any other instant.

[1] Post-Pro does not exist anymore in version 7 of Salome.
[2] I have changed a bit the defaults settings to print readable a view on a white background.

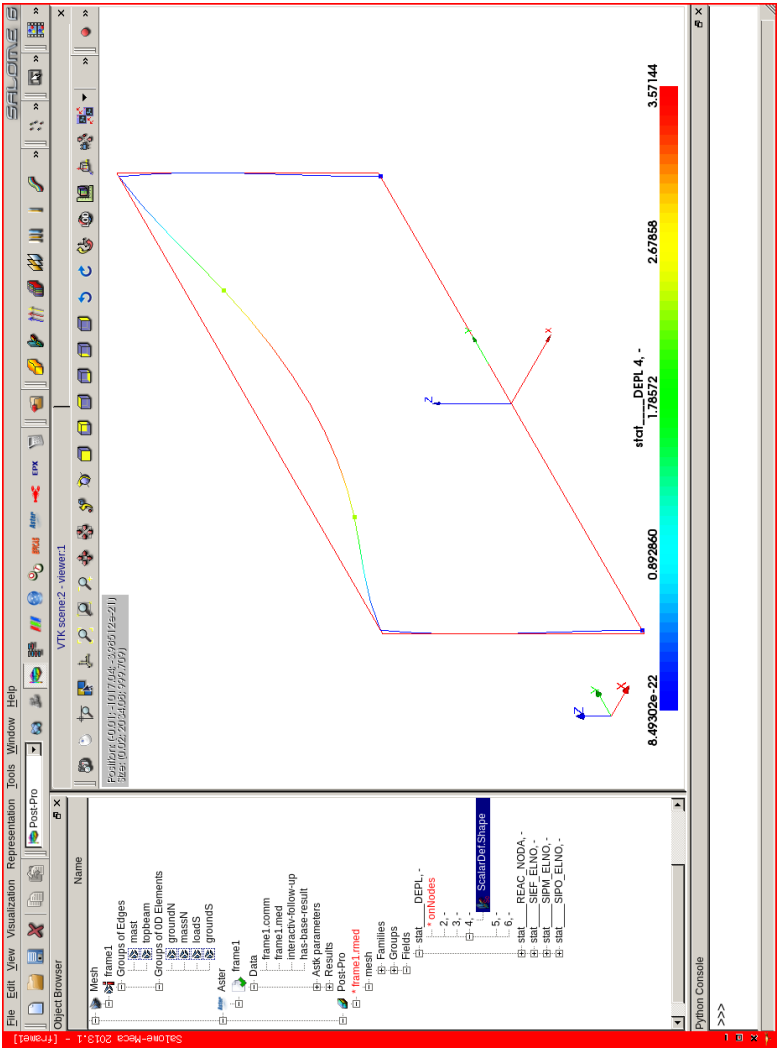


FIGURE 5.3: Deformed shape with values, and bounding box in red

Now let's try to display SIEF_ELNO component MFY on a Scalar map, like figure 5.4.

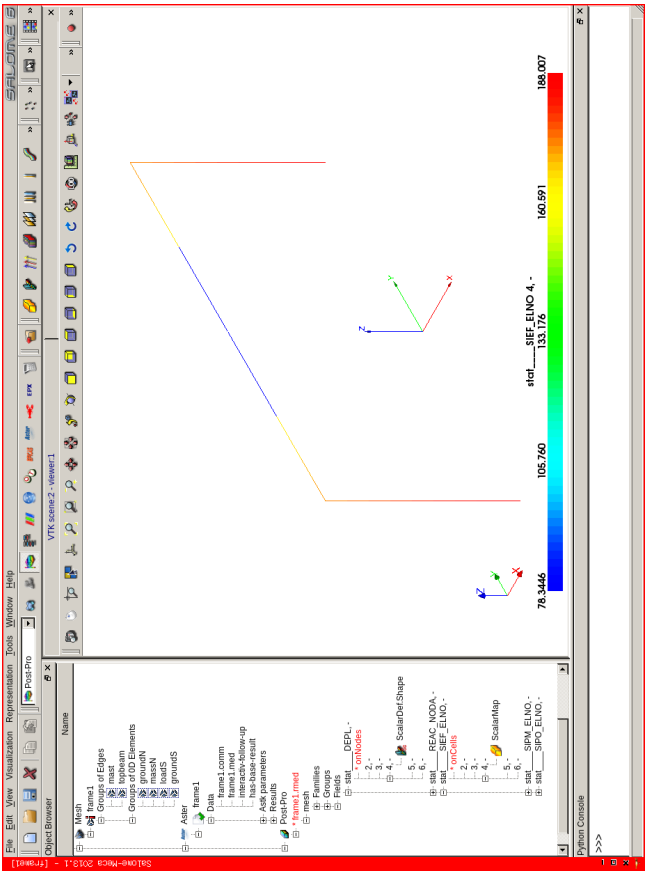


FIGURE 5.4: Display of the field SIEF_ELNO, (it's written in the Scalar Bar), component MFY, but alas nothing tells us about the component in the bar!

Then with the vector of REAC_NODA on top of it like figure 5.5.

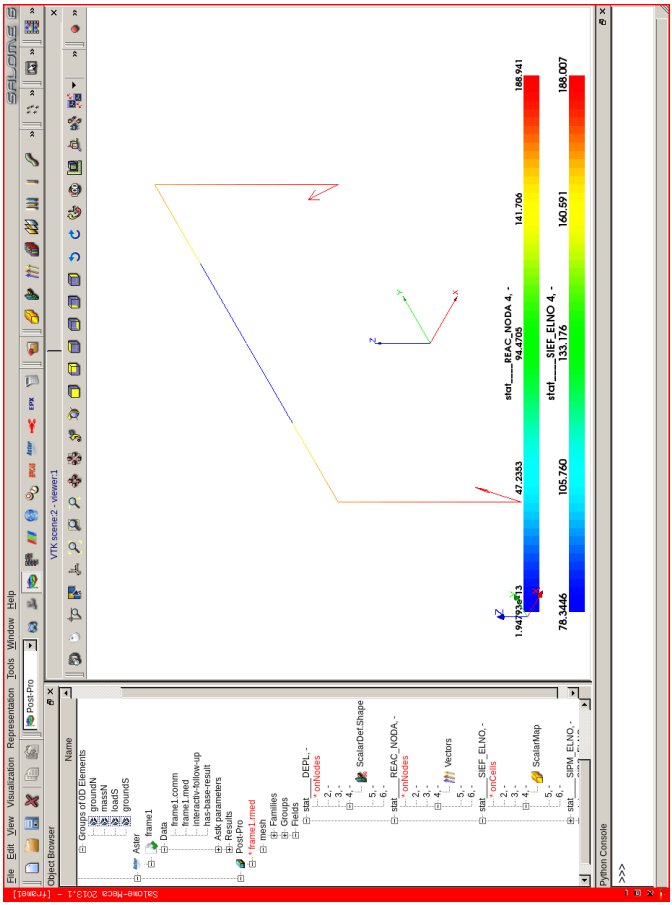


FIGURE 5.5: Value of the field SIEF_ELNO, with the Reaction vector on top

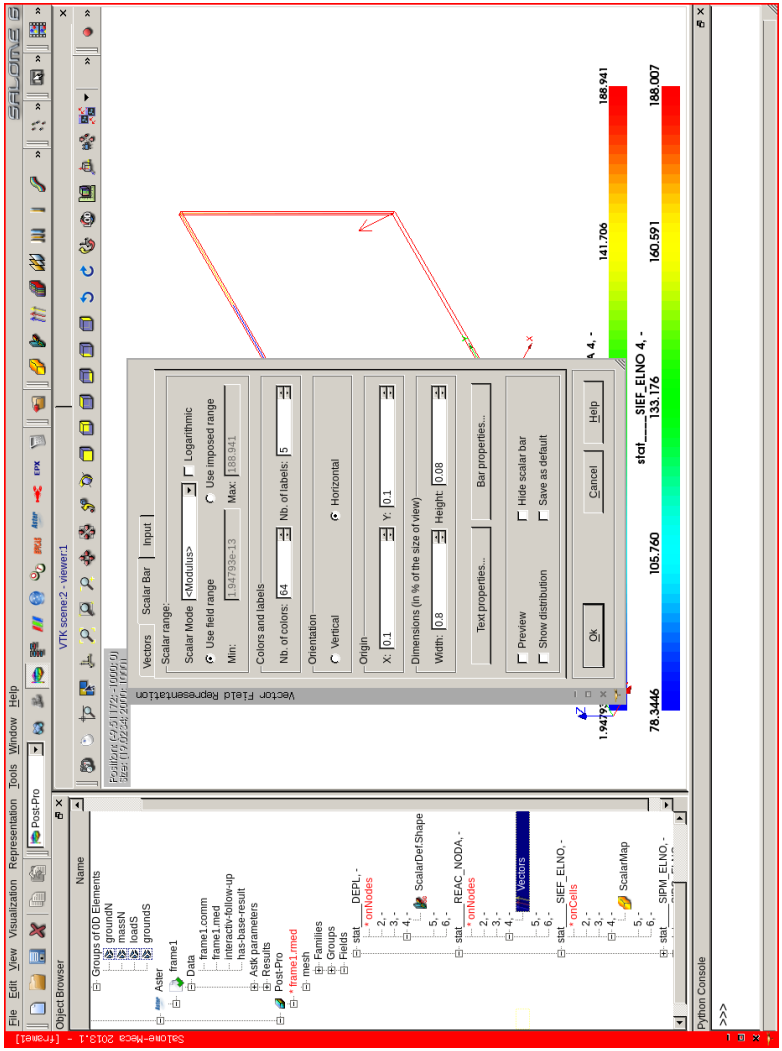


FIGURE 5.6: Modifying the appearance

To access the dialog box allowing us to change the appearance of the displayed picture we can RMB click in the image title in the browser and choose Edit..., or RMB click on its picture in the graphical window (the activated object shows within a red bounding box frame) and choose Edit....

This dialog box, figure 5.6 displays its **Scalar Bar** tab, allows us to change the appearance of almost anything appearing in a Post-pro window:

- scale of deformed shape;
- range of value in the Scalar Bar;
- as well as its position and dimensions;
- we can also select the groups to be displayed;
- and much more...

5.3 Sophisticating the display

We can also 'pick' a value from the screen. For this, we first need to have one result selected in the browser tree, then in the menu select View > Windows > Selection, and a dialog box like in figure 5.7 should appear.

Here the `SIEF_ELNO` fields is displayed with the Scalar Bar showing the `N` component.

In the **Selection** dialog box we choose the Cell tab and click on one element, some values appear in the **Selection** dialog box.

Here the element selected is at the bottom of the mast[1], we can check that the first value of the vector '170.91' is in agreement with the graphical display.

However, this nice feature is not of much interest for beam results as only the first 3 components of the tensor (or vector) are shown, which means that it is impossible to access a component like `MFY`, this is rather disappointing!

[1] Highlighted in purple.

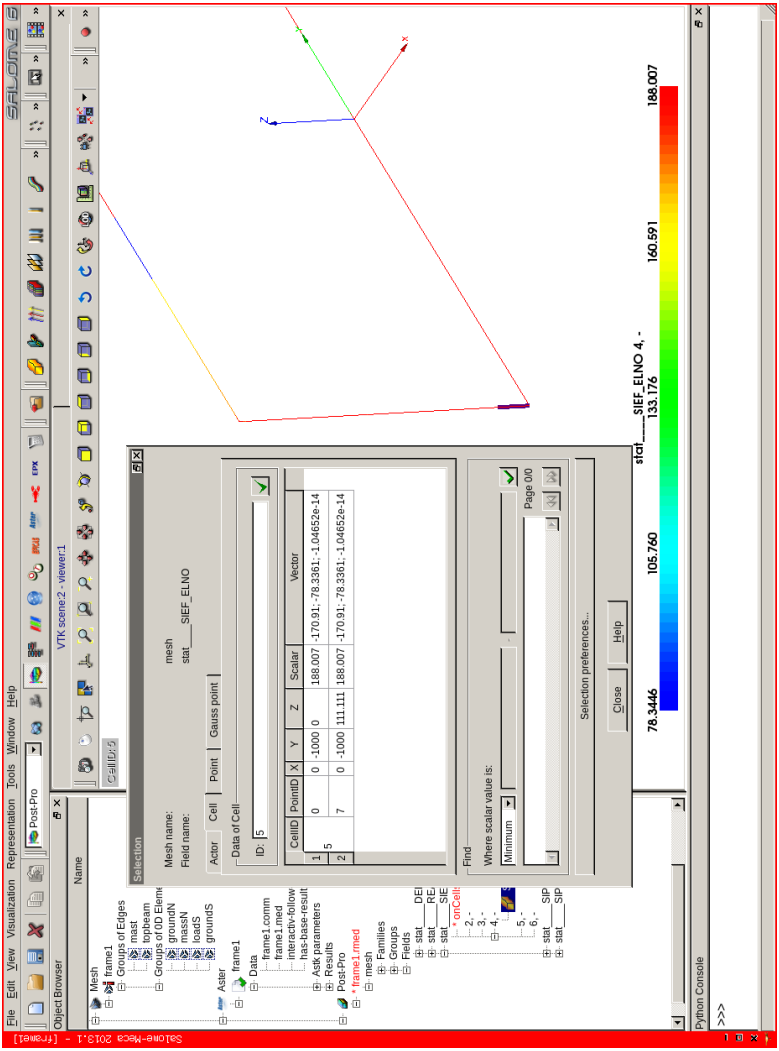


FIGURE 5.7: 'Picking ' a result from the screen

Maybe of interest in some case, we can display a tensor value, like SMFY on a deformed shape, for that:

- in the tree choose stat___DEPL;
- choose the instant we want;
- RMB click, Edit;
- then select Deformed Shape and Scalar Map;
- in the Scalar Field pull down to stat___SIPO_ELNO;
- in Scalar Bar:
  - in the Scalar Mode;
  - pull down to the component SMFY;
- push OK.

And the 'ScalarDef.Shape' results display should look like figure 5.8.

5.4 Looking at ASCII results

5.4.1 Printing RESULTAT

It is easy to read the *.resu* file with any text editor.

First is the printout of the reaction sum:

```
#
#------------------------------------------------------------------------------
#
#ASTER 11.03.22 CONCEPT sum_reac CALCULE LE 25/06/2013 A 06:58:46 DE TYPE
#TABLE_SDASTER
 INTITULE          RESU      NOM_CHAM          NUME_ORDRE     INST           RESULT_X
RESULT_Y       RESULT_Z       MOMENT_X       MOMENT_Y       MOMENT_Z
 sum reactions             REAC_NODA                        1       2.000E+00      0.000E+00
0.000E+00      0.000E+00       0.000E+00      0.000E+00      0.000E+00
 sum reactions             REAC_NODA                        2       3.000E+00      8.834E-27
8.811E-12      2.089E+02       6.750E+04     -1.692E-11     -3.061E-24
 sum reactions             REAC_NODA                        3       4.000E+00      1.516E-26
1.137E-13      3.439E+02      -8.731E-11     -2.932E-11     -3.292E-24
 sum reactions             REAC_NODA                        4       5.000E+00      2.913E-26
-2.274E-13      6.439E+02      -1.746E-10     -5.634E-11     -6.151E-24
 sum reactions             REAC_NODA                        5       6.000E+00      9.137E-27
-1.137E-13      2.000E+02      -7.276E-11     -1.801E-11     -1.954E-24
```

Then the printout of the maximum displacement, component DZ, for the 'topbeam' group:

```
 ---------------------------------------------------------------------------
 ASTER 11.03.22 CONCEPT stat CALCULE LE 25/06/2013 A 06:58:46 DE TYPE EVOL_ELAS

                    ENTITES TOPOLOGIQUES SELECTIONNEES
 GROUP_MA : topbeam


 ======>

 ------>
 CHAMP AUX NOEUDS DE NOM SYMBOLIQUE  DEPL
 NUMERO D'ORDRE: 1 INST:      2.000E+00

 LA VALEUR MAXIMALE DE DZ        EST     0.000E+00 EN    17 NOEUD(S) : N2
 LA VALEUR MINIMALE DE DZ        EST     0.000E+00 EN    17 NOEUD(S) : N2

 ======>

 ------>
 CHAMP AUX NOEUDS DE NOM SYMBOLIQUE  DEPL
 NUMERO D'ORDRE: 2 INST:      3.000E+00

 LA VALEUR MAXIMALE DE DZ        EST    -1.534E-03 EN     1 NOEUD(S) : N2
 LA VALEUR MINIMALE DE DZ        EST    -2.116E+00 EN     1 NOEUD(S) : N26

 ======>

 ------>
 CHAMP AUX NOEUDS DE NOM SYMBOLIQUE  DEPL
 NUMERO D'ORDRE: 3 INST:      4.000E+00

 LA VALEUR MAXIMALE DE DZ        EST    -4.531E-03 EN     1 NOEUD(S) : N5
 LA VALEUR MINIMALE DE DZ        EST    -3.571E+00 EN     1 NOEUD(S) : N4

 ======>

 ------>
 CHAMP AUX NOEUDS DE NOM SYMBOLIQUE  DEPL
 NUMERO D'ORDRE: 4 INST:      5.000E+00

 LA VALEUR MAXIMALE DE DZ        EST    -8.708E-03 EN     1 NOEUD(S) : N5
 LA VALEUR MINIMALE DE DZ        EST    -6.772E+00 EN     1 NOEUD(S) : N4

 ======>

 ------>
 CHAMP AUX NOEUDS DE NOM SYMBOLIQUE  DEPL
 NUMERO D'ORDRE: 5 INST:      6.000E+00

 LA VALEUR MAXIMALE DE DZ        EST    -2.785E-03 EN     1 NOEUD(S) : N5
 LA VALEUR MINIMALE DE DZ        EST    -2.133E+00 EN     1 NOEUD(S) : N4
```

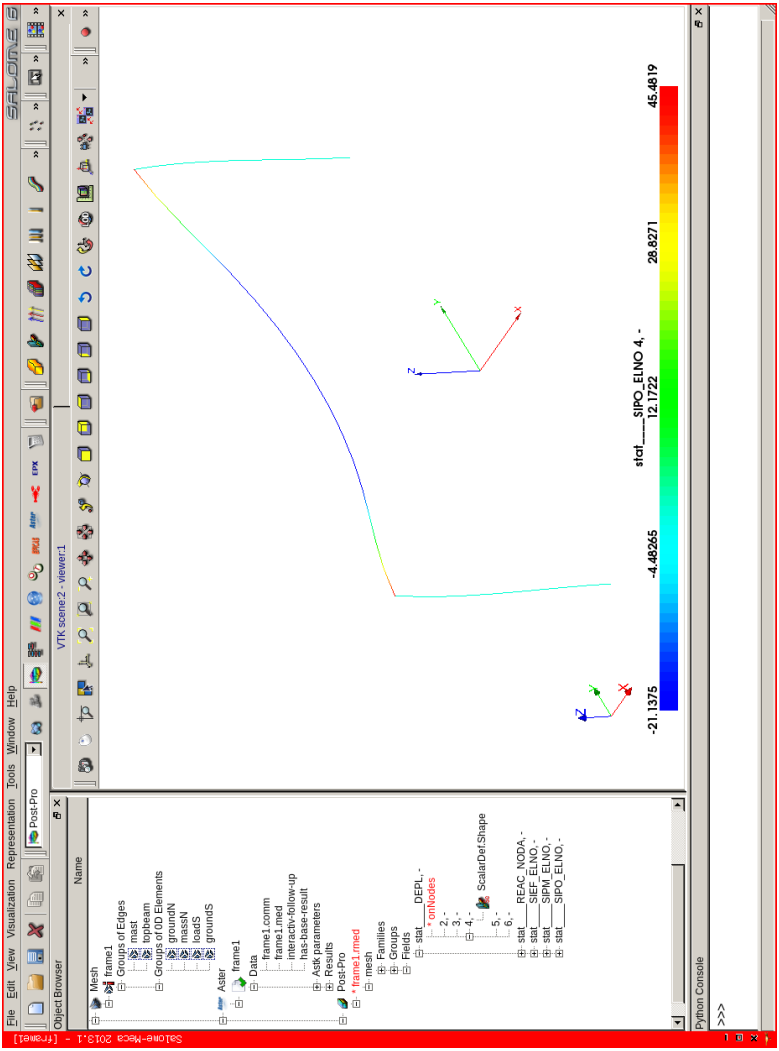


FIGURE 5.8: SMFY value on deformed shape

5.4.2 Printing results in TABLE

We can print results with IMPR_TABLE or with IMPR_RESU... FORMAT='RESULTAT'. The printed results have, of course, the same value as long as we print the same thing; the appearance may be more pleasing in RESULTAT or in TABLE mode, this a matter of taste.

TABLE makes it easier to export to a spreadsheet. And for this a clever use of the keyword SEPARATEUR allows more exporting flexibility to the output file.

The object TABLE allows many manipulations with some Python code which may be very helpful. Later in this book there are a few examples of tables manipulated with Python.

5.5 Verifying the results

What we have here is not quite a simple problem which could be easily solved by hand, we would need to apply the virtual work theory, leading to a lengthy process of solving a set of 3 equations with 3 unknowns variables[1].

However a first critical look at the ASCII results shows:

- at INST=2, reactions are null, as expected there is no load;
- at INST=3, the sum of the vertical reactions is equal to 13.5 times the mass (with due allowance to units and the value of 10 we used for g), the reactions are not symmetrical as expected as a 10 kg mass is sitting at a quarter of the length of the top beam, on 'massN', again as expected;
- at INST=4, the sum of the reactions is equal to 13.5 times the mass plus the 135 N force on 'loadS', the reactions are now symmetrical as the 135 N force balances the 10 kg mass, multiplied by the 1.35 factor, again as expected;

[1] This is what *Code_Aster* does, with a few more equations!

- at `INST=5`, the sum of the reactions is increased by 300 N, that's the "distributed load" applied on 'topbeam' with its 1.5 multiplier, again as expected;
- and at `INST=6`, the sum of the reactions is just 200 N, balancing the "distributed load" applied on 'topbeam' with its 1.0 multiplier, all good.

And we should always perform this type of simple checkup, and if it is not "as expected", something is wrong somewhere and should be sorted out. The following tables summarize what are the right[1] results for *frame1* example:

Code_Aster INST			2	3	4
maximum vertical displacement	DZ	mm	0	-2.116	-3.571
'topbeam' max bending moment	MFY	Nmm	0	3.54E+04	5.24E+04
'topbeam' min bending moment	MFY	Nmm	0	-2.45E+04	-2.43E+04
'groundS' vertical reaction	DZ	N	0	64.33	171.94
'groundN' vertical reaction	DZ	N	0	144.54	171.94
'groundS' moment reaction	DRX	Nmm	0	-2.07E+04	-2.60E+04
'groundN' moment reaction	DRX	Nmm	0	8.04E+03	2.60E+04

Code_Aster INST			5	6
maximum vertical displacement	DZ	mm	-6.772	-2.133
'topbeam' max bending moment	MFY	Nmm	9.57E+04	2.88E+04
'topbeam' min bending moment	MFY	Nmm	-5.16E+04	-1.82E+04
'groundS' vertical reaction	DZ	N	321.94	100.00
'groundN' vertical reaction	DZ	N	321.94	100.00
'groundS' moment reaction	DRX	Nmm	-4.74E+04	-1.43E+04
'groundN' moment reaction	DRX	Nmm	4.74E+04	1.43E+04

5.6 Spotting a mistake?

Taking a closer look at figure 5.4 we can see a $5.24E4$ value for the bending moment `MFY` in the 'topbeam' at the corner where it joins the mast, but the value of the same field `MFY` seems to be rather different, in fact it is null, in the 'mast' at the same corner. As the connection is rigid shouldn't we have the same value in each member?

Have we spotted a mistake in the calculation?

[1] Right? More exactly the one calculated on my computer!

YES but NO!

If we display the value of MFZ we can see the same value of $5.24E4$ for the bending moment MFZ in the 'mast'. Figure 5.9 shows the two result values on the same picture, MFY at the bottom and MFZ at the top.

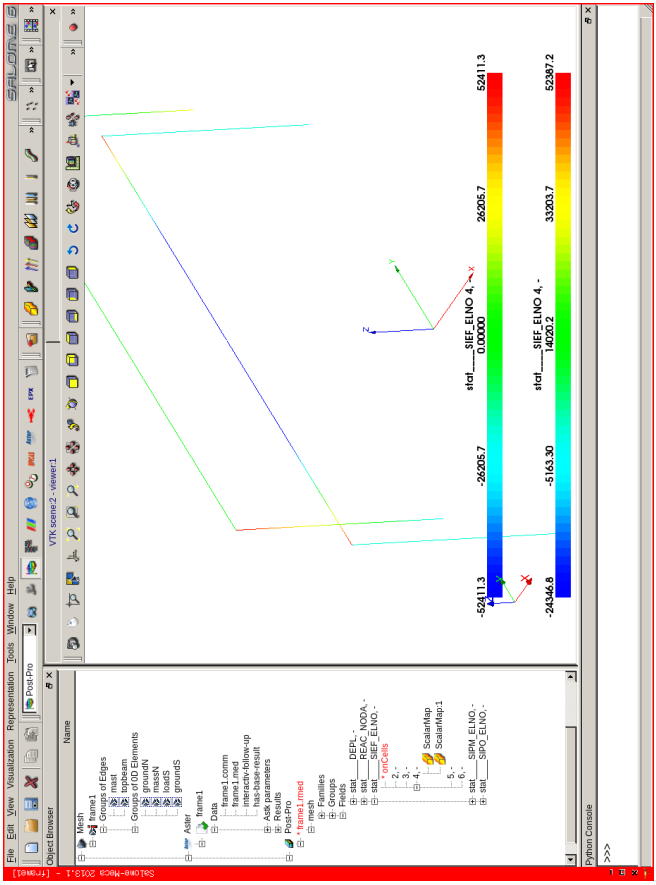


FIGURE 5.9: MFY in 'topbeam' equals MFZ in 'mast' at corner

The answer is that the local axis of the 'topbeam' and 'mast' groups are not oriented in the same direction, this is explained in chapter 6.3 .

CHAPTER 6

Understanding some details

Dimensions will always be expressed in the least useable terms. For example, velocity will be expressed in furlongs/fortnight.

Murphy's law.

Before refining the model and the analysis we make a pause to go deeper into some details like:

- units;
- some *Code_Aster* jargon related to stress results;
- beam elements orientation;
- understanding the various messages of the *.mess* file;
- the "Overwriting" rule.

6.1 Dealing with units

The previous model has its lengths in millimeters, its forces in Newton and its time in seconds. In such a system we need to express the mass in

tons and the mass density in t/mm^3. These oddities, though common in the world of engineering are necessary to form an homogeneous system of units.

In *frame1* example: the volume of the element is calculated by *Code_Aster* in cubic mm which multiplied by the mass density gives us mass in ton, which again multiplied by an acceleration, which is implied in mm per square seconds, gives us kilograms multiplied by meters divided by squared seconds also known as Newton. Then the forces results are in Newton, N, the moments results in $N.mm$ and the stresses in N/mm^2.

All this to say that *Code_Aster* is not aware of the units we use, given a set of units as entries it produces results in a set of homogeneous units.

The following table summaries the two mostly used mechanical engineering sets of units, "mm.t.s" and "SI", also known as "ISO", with a few typical values [1]:

	physical quantity	dimension	mm.t.s	SI
base units	length	L	mm	m
	mass	M	t	kg
	time	T	s	s
	temperature	°	K	K
derived units	angle	1	rad	rad
	frequency	T^{-1}	Hz	Hz
	force	$\mathrm{M.L.T}^{-2}$	N	N
	pressure, stress	$\mathrm{M.L}^{-1}\mathrm{.T}^{-2}$	$\mathrm{N/mm}^2$	$\mathrm{N/m}^2$
	mass density	$\mathrm{M.L}^{-3}$	$\mathrm{t/mm}^3$	$\mathrm{kg/m}^3$
example, steel	mass density	$\mathrm{M.L}^{-3}$	7.80E-09	7800
example, gravity	acceleration	$\mathrm{L.T}^{-2}$	9810	9.81
example, steel	Young modulus	$\mathrm{M.L}^{-1}\mathrm{.T}^{-2}$	210000	2.10E11

[1] For temperature the base unit is Kelvin degree, everyday Celsius degree scale uses the same increment, but Farenheit does not.

6.2 Understanding SIEF, SIPO, SIPM...

The fields DEPL and SIEF_ELGA are calculated even if we do not request it in CALC_CHAMP. Calculation and/or printing of any field can be restricted to one or more of its component with NOM_CMP.

Field DEPL means displacement.

The field SIEF_ELGA means SIgma (stress) or EFfort (force or moment), per ELement at GAuss points.

The field SIEF_ELNO means SIgma (stress) or EFfort (force or moment), per ELement at NOde.

With due allowance to stress or effort these two fields are meaningful whatever the element.

For beams, the field SIEF_ELNO means EFfort (force or moment), per ELement at NOdes. From a practical point of view it contains the normal force, N, the 2 shear forces, VY and VZ, and the 3 moments, MT, MFY and MFZ in the beam, in its LOCAL axis[1].

The field EFGE_ELNO means EFfort (force or moment), GEneralised, per ELement at NOdes, in the element local axis, it contains the same components as SIEF_ELNO and for any practical purpose shows the same things. The same applies at GAuss points for EFGE_ELGA.

For beams[2], the field SIPO_ELNO means Stress (SIgma POutre), per ELement at end NOdes. This is the stress produced by any of the three forces and moment defined in SIEF_ELNO if they were acting alone on the beam section. For example SMFY is the stress due to the single bending moment MFY, it is computed from MFY above and the beam characteristics, defined in AFFE_CARA_ELEM...POUTRE[3].

SIPM_ELNO[4] (SIgma Poutre Maximum) gives the component of the stress in the direction of the last 2 characters of the component name. The most important SIXX, can be seen as the extremum, maximum or

[1] For plates it would be NXX (with the unit of N/mm), NXY, NXY, MXX... in the local element axis. And for 3D elements, pure stresses SIXX, SIYY, SIZZ, SIXY... in the global axis .

[2] And for beams only, this field is restricted to beams.

[3] Here, $\mathtt{SMFY} = \frac{\mathtt{MFY}}{\mathtt{IY}} \times \mathtt{RZ}$.

[4] Same remark as above.

minimum, normal stress, i.e. the addition of the normal stresses due to the normal force and the two bending moments.

For beam elements, another important note is: the stress due to the torsional moment is only true if the "Saint Venant" conditions are achieved, that is to say there is no warping of the section. Which is more and more false as the section differs from a round and closed one to an open one, and/or if the warping is restrained by boundary conditions[1]. The actual stress may then be much higher by a factor which can be ten or more, the problem then cannot strictly be solved by this type of beam analysis. This is a classical problem of stress analysis described in many text books. Using solid elements connected to beams can be an alternative, this is described in chapter 14.2 .

Another important note: Salome Post-Pro proposes as the first choice of a field the option modulus, which is the "modulus" of the vector formed by the sum of the first three components. For displacement this is the sum of the first three, `DX, DY, DZ` and is thus a meaningful value[2]. For some other fields like `SIEF_ELNO` or `SIPO_ELNO` this modulus has therefore no engineering meaning and should never be looked at as a serious information.

For plates, the fields `SIEF_ELGA` and `SIEF_ELNO` give 9 components, the first three being the normal forces, `'N..'`, the next three the moments, `'M..'`, and the last three the shear forces, `'V..'`, in the element `LOCAL` axis.

Document U2.01.05, named "Stresses, forces and strain", is a useful reference regarding this matter.

[1] Some type of beam elements like POU_D_TG take warping restrain into account.
[2] Just as well for reactions.

6.3 Orienting beam elements

As we have seen in chapter 5.6 , not understanding the orientation of beam local axis may lead: at best to improper result interpretation, at worst to completely erroneous results.

In this example the 'topbeam' group has its local Y axis lying in the XOY global plane. It is the same for the 'mast' group but since the elements are strictly lying on the global Z axis, the local Y axis is strictly lying in the global Y axis. Figure 6.1 shows the beam orientation as defined in the original *.geo* and *.comm* files.

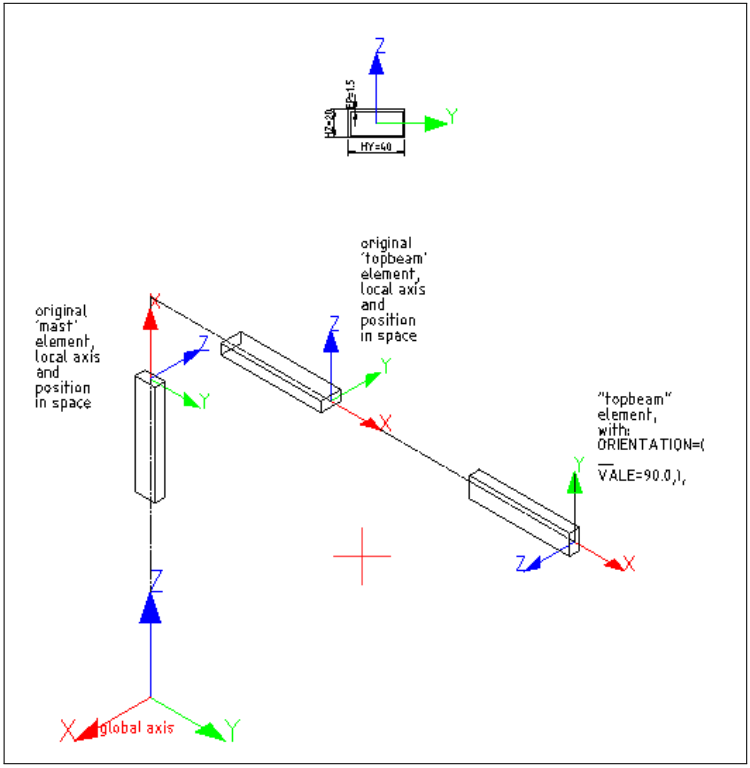


FIGURE 6.1: Original orientation and orienting the 'topbeam' elements

To modify this, we search for these lines in the *frame1.comm*:

```
#ORIENTATION=_F(
      #GROUP_MA=('mast',),
      #CARA='VECT_Y',
      #VALE=(1.0, 0.0, 0.0,),
  #),
```

Un-comment them, this puts the local y axis of mast beam pointing in the global X direction and produces a continuous `MFY` in 'mast' and 'topbeam' groups.

Search again for these other lines in the *frame1.comm*:

```
  #ORIENTATION=_F(
      #GROUP_MA=('topbeam',),
      #CARA='ANGL_VRIL',
      #VALE=90.0,
  #),
```

Un-comment them and change `VALE` to 90.0[1], so it becomes:

```
ORIENTATION=_F(
  GROUP_MA=('topbeam',),
  CARA='ANGL_VRIL',
  VALE=90.0,
),
```

If we run the calculation, we can see a reduced maximum displacement. In fact the rectangular section of the top bar (group 'topbeam') was originally lying on its flat side and we have turned it 90 degrees along its own axis so it now lies vertically, just like in the figure 6.1.

The rule for the orientation of the local axis of beams[2] in*Code_Aster* is very simple:

- the local x axis lies along the beam;
- the local y axis lies by default in the global XOY plane;
- and the local z completes the trihedron.

[1] Here the angles are given in degrees, somewhere else in radians! One has to be careful.
[2] Or any line element.

Here “by default” means: the keyword `ORIENTATION` is not specified in the *.comm* file for the given group.

If the beam is strictly parallel to the global Z axis, the beam y local axis is then exactly coincident with the global Y axis.

If we have a circular array of vertical beams along the Z axis around a circle in the XOY plane and want them to have a rectangular section pointing towards the center of the circle, the trick is to offset them slightly from vertical, just enough so that the tangent of the angle is not null.

In the following figures, we have a few cases depicting beam orientation[1]:

- In figure 6.2, a circular array of three beams of rectangular section, lying “on their flat” at an angle of 30° on the global horizontal plane XOY, without any `ORIENTATION` keyword.

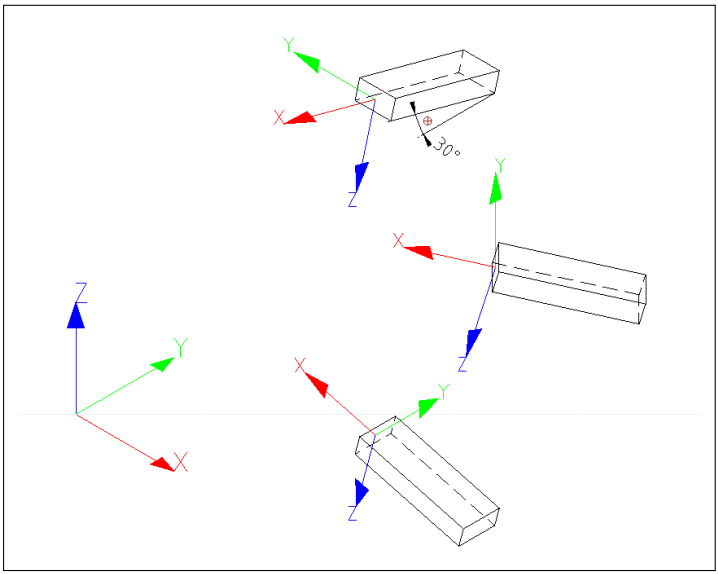


FIGURE 6.2: Circular array of beam, y local axis lying in the global XOY plane

[1] In these figures the global axis Z is vertical

- In figure 6.3, a similar array with the beams lying exactly vertical, *strictly* parallel to Z global axis, without any `ORIENTATION` keyword, note the local y axis pointing in Y global axis.

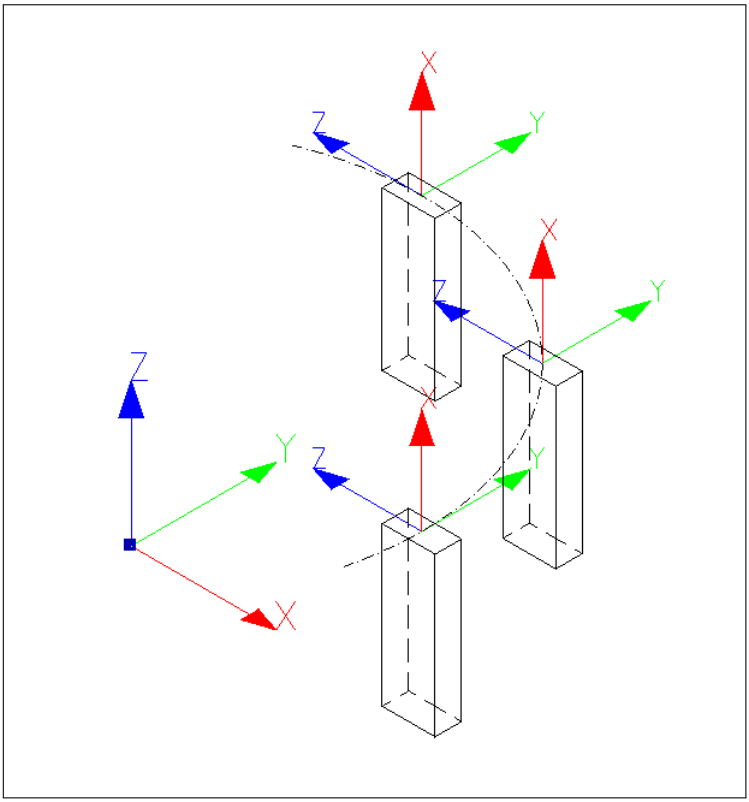


FIGURE 6.3: Circular array of beam, , local x axis strictly vertical, y local axis lying in the global XOY plane and pointing in Y direction

- Figure 6.4, is almost like figure 6.3 but the x local axis is turned a bit so as to be off vertical[1], note the local axis z pointing towards the center of the global coordinate system.

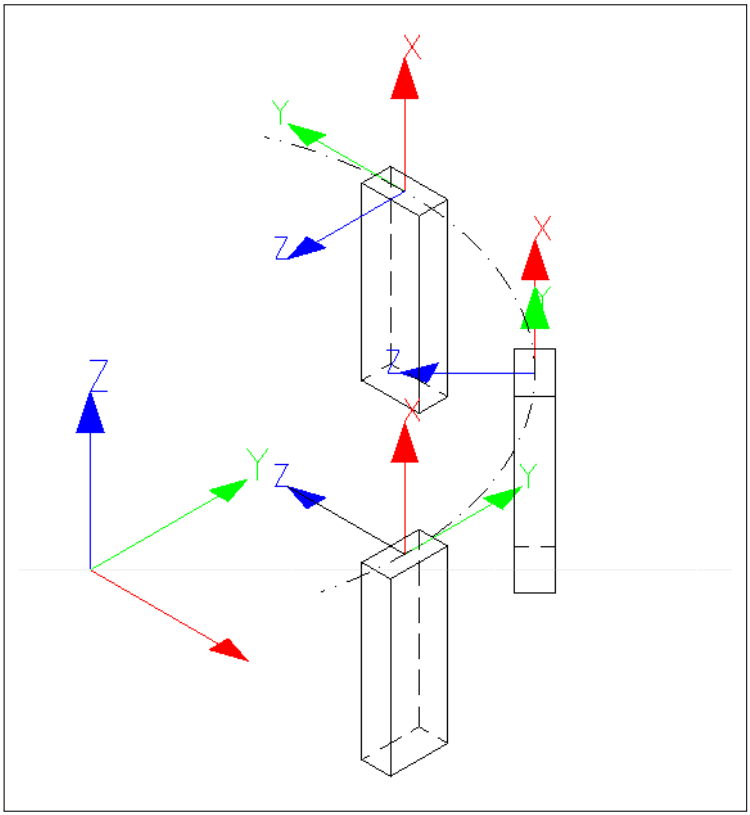


FIGURE 6.4: Circular array of beam, local x axis slightly offset from vertical z local axis lying in the global XOY plane and pointing towards the array's center

[1] The beams are not exactly parallel.

- Figure 6.5 shows almost the same case as n figure 6.2, but the third beam at the top of the figure is rotated by 90° with the keywords `ORIENTATION=_F(....,CARA='ANGL_VRIL', VALE=90.0,),.`

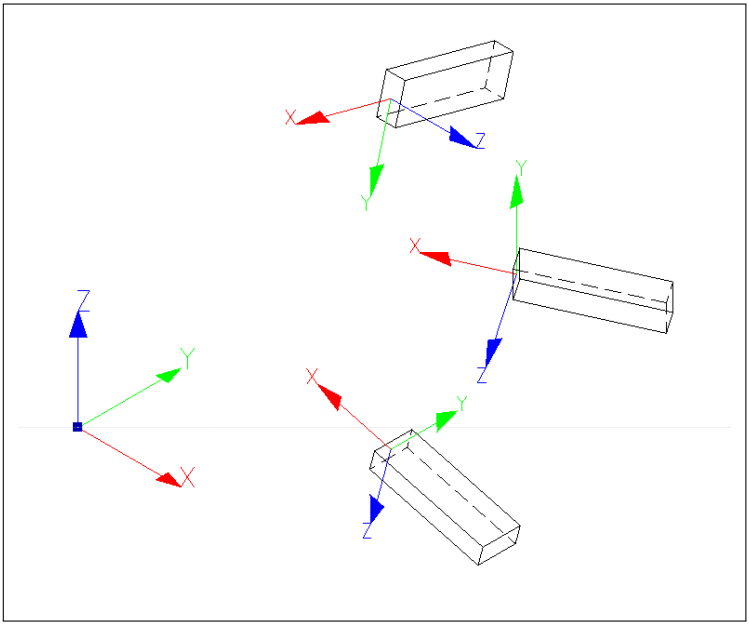


FIGURE 6.5: One beam rotated 90° with `ORIENTATION=_F(.... CARA='ANGL_VRIL', VALE=90.0,)`

- In figure 6.6, the two beams occupy an identical position in 3D space but the order of their nodes is inverted.

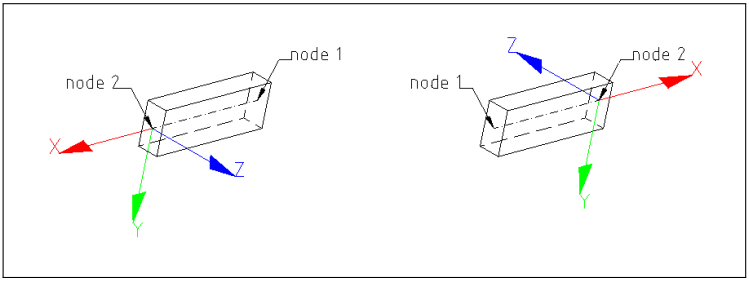


FIGURE 6.6: Node order reversed

As a general rule I advise to build the mesh with:

- global Z is the vertical, earth gravity direction, pointing upward;
- global X is the principal direction of the mesh, the direction of motion for a mobile object (road vehicle, airplane or ship);
- global Y completes the trihedron.

Another way to define the orientation of beam element, or group of beam element, is to use the following syntax:

```
ORIENTATION=_F(
  GROUP_MA=('topbeam',),
  CARA='VECT_Y',
  VALE=(0.0, 0.0, 1.0),
),
```

which orients the local Y axis in the direction of the global Z axis.

Again document U4.42.01 gives all the instructions, and more keywords, on how to change the orientation of the local axis.

When we perform any change of orientation of one beam, we of course change its local axis, this has to be kept in mind to understand the forces and stresses results. This applies also to the forces applied along the beam

if defined with the keyword REPERE='LOCAL', and may lead to exactly the reverse of what we expected!

Once we have fully understood the principles of beam orientation in *Code_Aster* and applied them in the mesh and groups of element we may well find it is hardly ever necessary to use anything but the keyword ANGL_VRIL to model any practical model. However when we are in doubt about some orientation it is always a good idea to perform a "dummy" analysis with loads and boundary conditions restricted to the very area raising the doubt and to check the results at various orientations with a quick hand calculation[1].

6.4 Finding it out when things go wrong

In this first example all went well and we got a result at the first try[2]. Let's say this is an exception. We, more than often, have some kinds of errors in the early stages. The only way around is to look at the *.mess* file, the different kinds of errors are quite explicitly described and the *.comm* file can be corrected accordingly.

At the beginning of a project there, most probably, are many syntax errors[3], and it can make debugging a rather tedious process. Even when the calculation gives a result there may be warnings in the *.mess* file. As is usually stated in the warning itself we must understand what it means before taking the results for granted.

Here is the end of the *.mess* file in case of success:

```
EXECUTION_CODE_ASTER_EXIT_9671=0
```

=0 means no error, no warning. 9671 is the job ID, in case of problems we find some files with this ID in *$HOME/flasheur* directory.

Here is the end of a *.mess* file with a typical Syntax Error:

[1] In chapter 17.2 we explain how to draw in a graphical window, Gmsh or Salome, vectors showing the local beam axis.
[2] The guy, or girl, who typed all this command file without any error is a lucky one!
[3] All the more so as we proceed without Eficas.

```
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! erreur de syntaxe,
  Erreur de nom:
  name 'DEFI_GROUPE' is not defined ligne 11                              !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
```

We should have written `DEFI_GROUP` instead of `DEFI_GROUPE`.

If we have `=1` instead of `=0` at the end of the file, we must look higher in the file until we find something like this:

```
!-------------------------------------------------------------------!
! <EXCEPTION>  <MODELISA7_75>                                       !
!                                                                   !
! le GROUP_NO  couron  ne fait pas partie du maillage :             !
! maillage                                                          !
!-------------------------------------------------------------------!
```

The error description always begins with a < character. Here some command refers to the group ’couron’ which is not part of the mesh. We do not go any further in the description of errors and warnings this is well documented in the *.mess* file.

And our ingeniousness in raising errors does not seem to have any limit.

A final note to point out is: if a fatal error is briefly repeated at the end of the *.mess* file, a more complete explanation is situated earlier in the same file, just by the command that generated the fatal error[1].

6.5 Understanding the “Overwriting” rule

This rule governs how keyword repetitions are handled in the *.comm* file, we give here some practical examples. This topic is explained in U1.03.00[2].

In this first example, we give to the group ’massN’, at first the property of a beam -which is wrong- and in a second sentence the property of a discrete element, only the second one is applied, luckily it is what

[1] The “just by” word must be taken with a grain of salt, as for a `PAR_LOT='NON'` run the error message may not pop up exactly where it is relevant, though it is descriptive enough.

[2] The translation of the french “Surcharge” by “Overload” in the machine translation of the English version of the documentation does not look satisfactory to me, and I prefer the word “Overwriting”.

we wanted. Swapping the two sentences would have raised an error as 'massN' which is here a point element, cannot be a beam!

```
model=AFFE_MODELE(
    MAILLAGE=mesh,
    AFFE=(
        _F(
          GROUP_MA=('topbeam','mast','massN',),
          PHENOMENE='MECANIQUE',
          MODELISATION='POU_D_E',
        ),
        _F(
          GROUP_MA=('massN',),
          PHENOMENE='MECANIQUE',
          MODELISATION='DIS_T',
        ),
    ),
);
```

In this second example, we specify twice, a load on the same element (here a node).

```
cc=AFFE_CHAR_MECA(
    MODELE=model,
    FORCE_NODALE=(
    _F(GROUP_NO=('loadS',),FZ=-100,),
    _F(GROUP_NO=('loadS',),FX=2000,FZ=-1000,),
  ),
 );
```

The effective load taken into account in the calculation is `FX=2000, FZ=-1000`:

- the second instance `FZ=-1000` REPLACES the first one `FZ=-100`[1];
- the `FX=2000` of the second sentence is added to the load case[2].

We should always remember clearly this feature when specifying loads within a single `AFFE_CHAR_MECA`.

[1] Some finite element codes would have made the sum resulting in `FZ=-1100`.
[2] As it replaces a non existent item!

CHAPTER 7

Adding end release to the top bar

In this chapter, we slightly modify the first structure, adding end release to the top beam.

We use some parametric scripting capabilities of Gmsh.

And look at the results in the ParaVis module of Salome.

It is obvious from the results of the preceding example that the top bar is rigidly linked to the top of the masts. What, if we want a rotation free joint? *Code_Aster* does not provide any built-in end release option at the end of a beam element. What we do is create a short (10 mm) line element in the mesh at this connection. Then we give it the properties of a discrete element `K_TR_D_L` as specified in U4.42.01. Appendix C 3.1 deals in more details with these elements.

7.1 Using parametric scripting in Gmsh

We take the opportunity to make a new *.geo* file and mesh, using some scripting capabilities of Gmsh to first of all create a new mesh. Here is the file:

```
cl1=100;
max=3;
pas=2;
ly=1000; //half length of frame along y
hz=1000; //height of frame along z
dly=10; //length of hinge along y
Point(1) = {0, 0, 1000, cl1};
//create Point
For i In {0:max:pas}
  Point(10+i) = {0,ly/2*(i-1), hz, cl1};
  Point(20+i) = {0,(ly-dly)*(i-1), hz, cl1};
  Point(30+i) = {0,ly*(i-1), hz, cl1};
  Point(40+i) = {0,ly*(i-1), 0, cl1};
EndFor
//create Line
top[]={}; //initialize list for top
hinge[]={}; //idem for hinge
mast[]={};
For i In {0:max:pas}
  Line(10+i) = {1, 10+i};
  top[]+=10+i; //add element in top list
  Line(20+i) = {10+i, 20+i};
  top[]+=20+i;
  Line(30+i) = {20+i, 30+i};
  hinge[]+=30+i; //add element in hinge list
  Line(40+i) = {30+i, 40+i};
  mast[]+=40+i;
EndFor
//makes  the group 'topbeam' with the list top,
Physical Line("topbeam") = top[];
Physical Line("hinge") =hinge[];
Physical Line("mast") = mast[];
Physical Point("groundS") = {40};
Physical Point("groundN") = {42};
Physical Point("loadS") = {10};
Physical Point("massN") = {12};
```

The first lines set the values of some control variables[1], then some of the frame dimensions which are used in the next loops to create the points. It should be noted that the points are named in a discontinuous order, this is

[1] The hinge length is very small and is hardly visible in the graphical window at a normal scale, figure 7.1 is a zoom view on the discrete element n° 30 in black, with mast in red and top beam in green.

allowed in Gmsh[1]. Next, we initialize some lists which we fill up in the next loop while creating the lines, to finally put these lists in Physical.

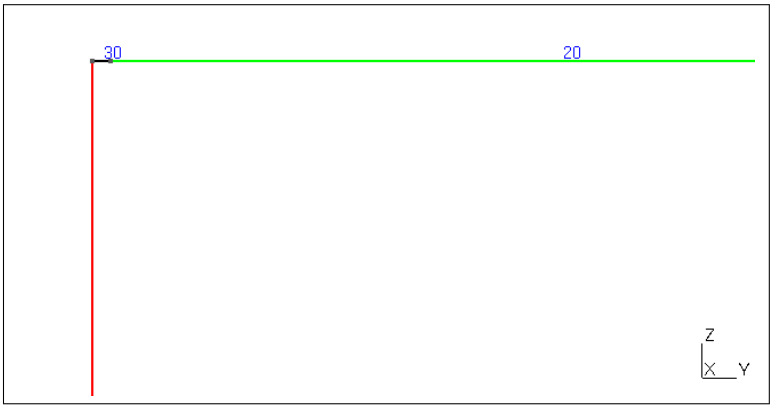


FIGURE 7.1: Zoom on the discrete line, n° 30, in black

There is a feature in the loop creating the lines, it is the outward orientation of the created geometrical lines. In the **Gmsh** graphical window, if we set the Tangents field at a value of 100, like in figure 7.2, we can see the lines of the 'topbeam' oriented outwards. If we set a load with FORCE_POUTRE, with values related to element orientation like N[2] the force pulls outwards from the center, it maybe what we want. If not the loop for creating the lines should be written differently[3].

However the *Code_Aster* MODI_MAILLAGE ... ORIE_LIGNE comes in handy to reorient line elements on request.

[1] This allows us to use some kind of "logical" numbering.
[2] In tension along the beam.
[3] We wrote it this way here so as to illustrate the feature.

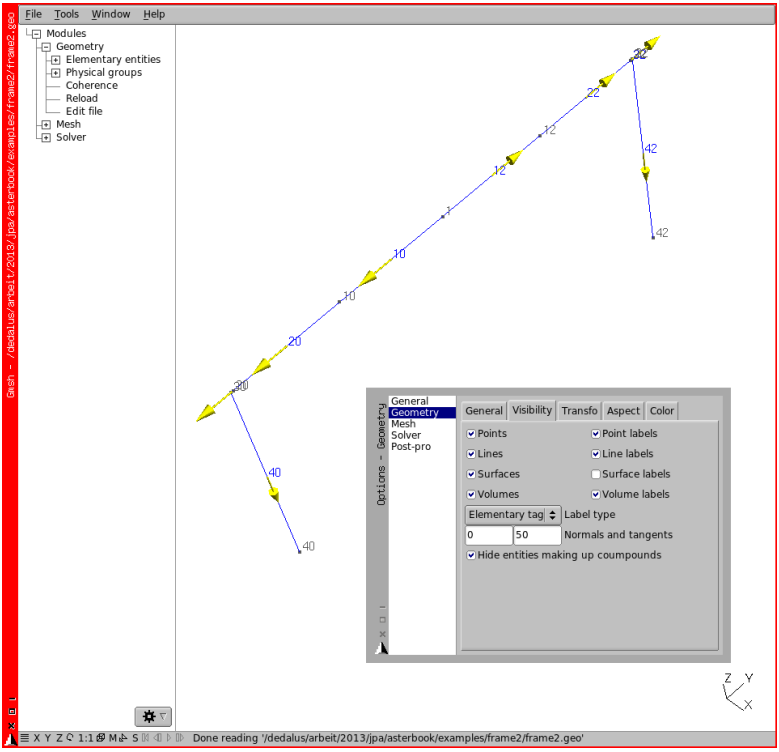


FIGURE 7.2: 'frame2' with line orientation

7.2 Modifying the *.comm* file

As the line element may not be oriented exactly as we want we first reorient the top beam along the y global axis with a*Code_Aster* command:

```
#U4.23.04
mesh=MODI_MAILLAGE(
  MAILLAGE=mesh,
    reuse =mesh,
    ORIE_LIGNE=(
    #next lines reorient all the line element on 'topbeam'
    #along a vector lying along y global axis
    #with origin at node 'masseN
    _F(
```

```
      GROUP_MA='topbeam',
      VECT_TANG=(0, 1, 0,),
      GROUP_NO='massN',
    ),
    #next lines raise an error as the elements in 'mast'
    #are not all connected
    #we would have needed 2 groups 'mastN' and 'mastS'
    #like wise for the 'hinge' group
    #_F(
      #GROUP_MA='mast',
      #VECT_TANG=(0, 0, 1,),
      #GROUP_NO='groundN',
    #),
    ),
);
```

This is not strictly necessary for this simple example, however it comes useful with more demanding ones. Note: the elements within a group must be connected so as to apply this operator, if not, we have to split in more groups, which can become a bit tedious.

Secondly we modify `AFFE_MODELE`.

As said earlier we are going to use a discrete element `'K_TR_D_L'` from U4.42.01. in this type of element:

- `K` stands for stiffness;
- `TR` for Translation and Rotation;
- `D` for diagonal only matrix (only 6 terms);
- and `L` for line (the element being a `SEG2` mesh element).

```
model=AFFE_MODELE(
    MAILLAGE=mesh,
    AFFE=(
        ......................
        _F(
          GROUP_MA=('hinge',),PHENOMENE='MECANIQUE',
          MODELISATION='DIS_TR',
        ),
    ),
);
```

Then the ad-hoc lines to add in `AFFE_CARA_ELEM` are so:

```
DISCRET=(.........
  #here we define the hinges as an element
  #with very low stiffness 1e1 in rotation around
  #the X axis in GLOBAL coordinates
      _F(
          GROUP_MA=('hinge',),
          CARA='K_TR_D_L',
          VALE=(1e6,1e6,1e6,0,1e9,1e9,),
          REPERE='GLOBAL',
    ),
),
.......................
```

The order of the six value is:

- stiffness in translation along X, Y, Z;
- stiffness in rotation around X, Y, Z;

we give relatively high value to all of them but for the rotation around X which is set to null[1].

X being here in 'GLOBAL' coordinate. In 'LOCAL' it would have been a free rotation around Y with the specified 'ORIENTATION'[2].

Note: the orientation of this discrete line element follows exactly the same rules as the one described earlier for beams.

7.3 Solving

This part is the same as for *frame1* example, see chapter 3.

We should just check, once the calculation is made, that the top bar is really articulated, that is to say no moments are transmitted to the masts. In fact a very small moment is transmitted due to the offset of the vertical load, the length of the hinge, on top of the mast.

[1] Or a very small value.

[2] Appendix C 3.1 provides some formulas to hand calculate the matrix rigidity coefficients of a line member.

7.4 Viewing the results in the ParaVis module

As ParaVis is going to replace Post-Pro as a result display module in Salome, from version 7 upward, and Salome-Meca, it is time to learn a few basics about how to use it. We first go in the ParaVis module of Salome:

- select the file to open, either with File » OpenParaViewFile... or with the Open File Icon; 
- in the **PipeLine Browser**, if this tear off window is not visible open it with View » Windows » PipeLine Browser[1];
- in the **Properties** window:
  - in Supports tick the elements we want, here we leave everything ticked;
  - in Fields tick stat_DEPL;
  - push the green colored Apply button, the frame appears in the window;
- push on the Warp By Vector icon, lying in the middle of the View » Toolbars » Common tool bar; 
- eventually fill the Scale Factor field in the **Properties** window with a convenient value, here 10, and push Apply[2];
- in the **Display** window, on the Color by pull down list, probably set to Solid Color on a first run, select stat_DEPL_Vector[3];
- on the pull down list just to the right pull to Magnitude;
- toggle Color Legend Visibility icon , sitting at the left of this toolbar; 
- in the Time toolbar[4] choose the INST we want;
- push the Rescale to Data Range icon, *this has to be done all the time to be certain of what is displayed on the screen*! 

[1] The useful working windows of ParaVis are hidden in View » Windows.
[2] Or in the Object Inspector type the Scale Factor, here 10, and push Apply
[3] This is also available through a tool bar hidden in View » Toolbars » Active Variable Controls
[4] This one is in View » Toolbars » Current Time Controls

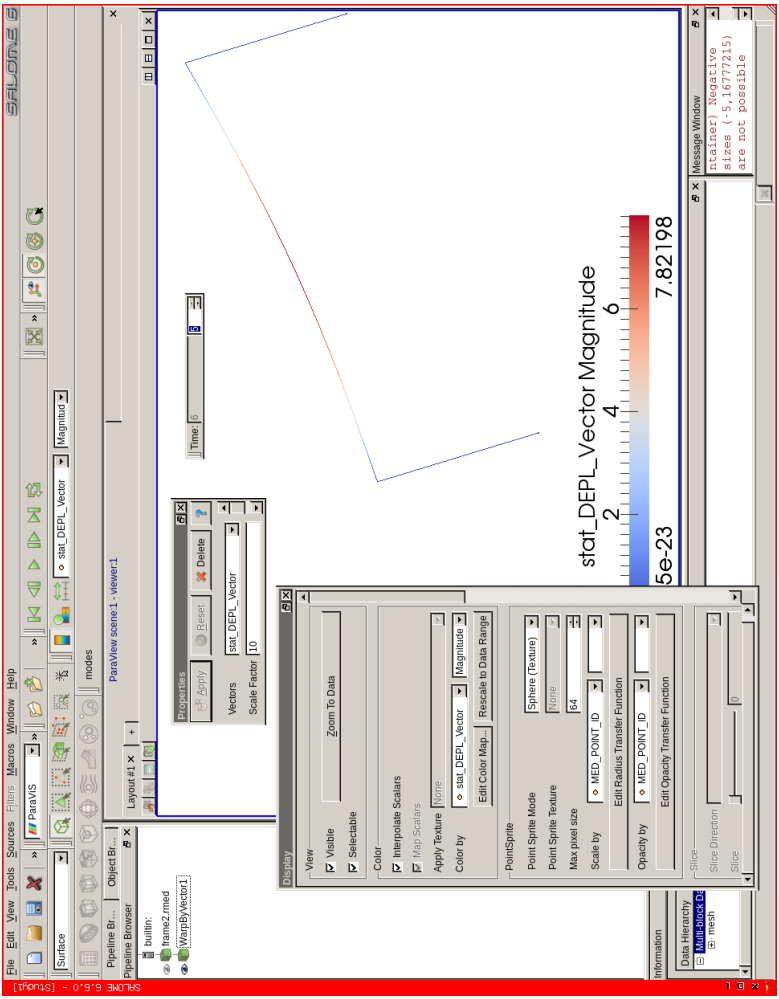


FIGURE 7.3: ParaVis view of displacement

At the end of this rather lengthy process, we get a view like figure 7.3 at INST 6[1].

There may be quite a bit of searching around to find out where are the windows or the tool bars in ParaVis, it is very important to make room for the tool bars, that otherwise may be hidden out of the screen on the right hand side or partially superposed one on top of the other.

And now to display some forces in the beams:

- in the Pipeline Browser select again the main result;
- in the **Properties** window:
  - in Supports tick the elements we want, here 'mast' and 'top-beam';
  - in Fields tick stat_SIEF_ELNO;
  - push the green colored Apply button;
- in the menu select Filters » Integration Points » Elno Mesh an entry appears in the **PipeLine Browser**, make it visible;
- push again the green colored Apply button in the window **Properties**;
- on the pull down list, probably named Solid Color select stat_SIEF_ELNO;
- on the pull down list just to the right select MFY;
- toggle Color Legend Visibility icon, sitting at the left of this toolbar;
- in the Time toobar[2] choose the INST we want;
- push the Rescale to Data Range icon this has to be done all the time to be certain of what one is looking at on the screen!

At the end of this process we get a view like figure 7.4, at INST 6, showing the bending moment around the local y axis.

[1] I changed some of the visibility settings in File » Preferences... » ParaVis and by pushing the icon Edit Color Map to change the defaults values which do not render a nice image on a paper print.

[2] This one in View » Toolbars » Current Time Controls

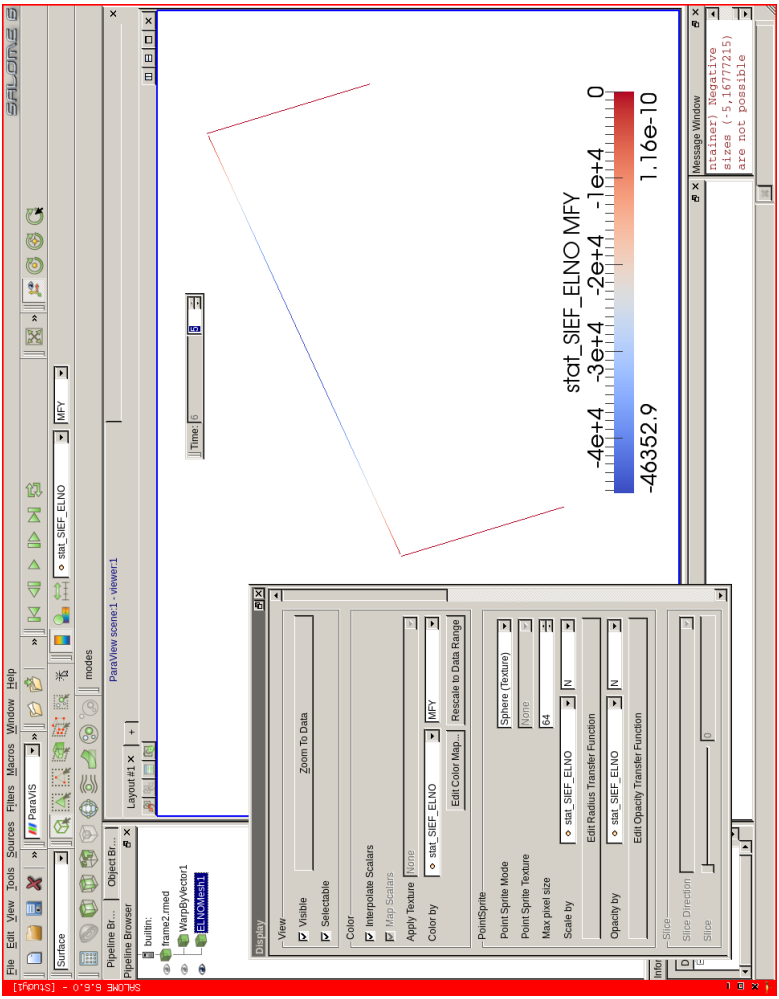


FIGURE 7.4: ParaVis view of bending moment

7.5 Looking at ASCII results

This example, with the pinned end top beam, provides a very easy way to check up the result, for example the deflection for a simply supported beam subject to a distributed load is known as being:

$$DZ = \frac{5}{384}\frac{pL^4}{EI}$$

which in this case turns out to be:

$$DZ = \frac{5}{384} \times \frac{0.1 \times 1980^4}{210000 \times 11518} = 8.27$$

which compares to the printed results at `INST=6`

```
 ---------------------------------------------------------------------------------
 ASTER 11.03.22 CONCEPT stat CALCULE LE 25/06/2013 A 12:12:55 DE TYPE EVOL_ELAS

                    ENTITES TOPOLOGIQUES SELECTIONNEES
 GROUP_MA : topbeam


 ======>

 ------>
 CHAMP AUX NOEUDS DE NOM SYMBOLIQUE  DEPL
 NUMERO D'ORDRE: 5 INST:      6.000E+00

 LA VALEUR MAXIMALE DE DZ       EST    -2.885E-03 EN     2 NOEUD(S) : N4
 LA VALEUR MINIMALE DE DZ       EST    -7.822E+00 EN     1 NOEUD(S) : N1
```

Likewise the maximum bending moment is:

$$MFY = \frac{pL^2}{8}$$

which in this case turns out to be[1]:

$$MFY = \frac{0.1 \times 1980^2}{8} = 49005$$

which compares to the printed results at `INST=6`

[1] The question arises whether to use a length of 2000, overall length, or 1980, length excluding the end hinges.

```
 ASTER 11.03.22 CONCEPT stat CALCULE LE 25/06/2013 A 12:12:55 DE TYPE EVOL_ELAS

                    ENTITES TOPOLOGIQUES SELECTIONNEES
 GROUP_MA : topbeam


 ======>

 ------>
 CHAMP PAR ELEMENT AUX NOEUDS DE NOM SYMBOLIQUE  SIEF_ELNO
 NUMERO D'ORDRE: 5 INST:      6.000E+00

 LA VALEUR MAXIMALE DE MFY              EST      1.164E-10 EN     1 MAILLE(S) : M20
 LA VALEUR MAXIMALE DE MFZ              EST      3.073E-13 EN     1 MAILLE(S) : M16
 LA VALEUR MINIMALE DE MFY              EST     -4.635E+04 EN     1 MAILLE(S) : M5
 LA VALEUR MINIMALE DE MFZ              EST     -3.073E-13 EN     1 MAILLE(S) : M20
```

The [not so] slight discrepancies can be practically put onto the account of the fact that the load is not exactly a distributed load[1].

Generally, this kind of hand calculation is done to provide a rough estimate of a result value to ensure the calculation validity.

If we want to perform a bench mark of *Code_Aster*, we can:

- isolate the 'topbeam' as a single span beam pinned in between 2 fixed supports (this can be dine by adding the group oh node 'mast' to the boundary condition 'ground',
- apply in 'cr' a true distributed load with,

  `FORCE_POUTRE=_F(GROUP_MA=('topbeam',),FZ=-0.1,);`
- run the solution with only that load case;

and we will find an exact agreement of the *Code_Aster* results values with the text book values.

7.6 Using an alternative option with beam elements

Instead of using a `K_TR_D_L` for the 'hinge' group we may use a beam element with a null, or near null, value for the moment of inertia on the right axis. We then should not forget to assign with elements a beam element model and assign them a material as well. Here are the command file abstracts doing that.

[1] A larger number of nodes on 'mast' group would produce a closer result.

Model section:

```
model=AFFE_MODELE(
    MAILLAGE=mesh,
    AFFE=(
        _F(
          GROUP_MA=('topbeam','mast','hinge',),
          PHENOMENE='MECANIQUE',
          MODELISATION='POU_D_T',
        ),
        _F(
          GROUP_MA=('massN',),
          PHENOMENE='MECANIQUE',
          MODELISATION='DIS_T',
        ),
        #_F(
          #GROUP_MA=('hinge',),
          PHENOMENE='MECANIQUE',
          #MODELISATION='DIS_TR',
        #),
    ),
);
```

Material section:

```
material=AFFE_MATERIAU(
    MAILLAGE=mesh,
    AFFE=_F(
      GROUP_MA=('topbeam','mast','hinge',),
      MATER=steel,
    ),
);
```

Element properties section:

```
elemcar=AFFE_CARA_ELEM(
    MODELE=model,
    POUTRE=(
        #the vertical members are rectangular section
        #(40x20 mm) with a thickness of 1.5 mm
        _F(
            GROUP_MA=('mast',),SECTION='RECTANGLE',
            CARA=('HY','HZ','EP',),VALE=(40, 20, 1.5,),
        ),
        #same with the horizontal bar
        _F(
            GROUP_MA=('topbeam',),SECTION='RECTANGLE',
            CARA=('HY','HZ','EP',),VALE=(40, 20, 1.5,),
        ),
        _F(
            GROUP_MA=('hinge',),SECTION='GENERALE',
            CARA=(
```

```
                'A','IY','IZ','AY','AZ','EY','EZ',
                'JX','RY','RZ','RT',
            ),
            VALE=(
                171, 0.1, 34908, 1.5, 1.5, 0, 0,
                26700, 20, 10, 12,
            ),
        ),
    ),
  #orientation would be here if necessary
  DISCRET=(
    _F(GROUP_MA='massN', CARA='M_T_D_N',VALE=(.01),),
  ),
);
```

As we use a null or near null for one value of moment of inertia[1] it is of course meaningless to try to calculate the related bending stress in these elements, but the forces and moments are meaningful!

Last but not least using short beam elements or discrete elements is not strictly equivalent, chapter 3.1 provides a deeper insight into discrete stiffness elements.

[1] However the values chosen for IZ or JX are true for the full tubular section and are thus highly questionable.

CHAPTER 8

Making an highway sign

In this chapter, we modify the structure by adding some other beam members and some plating making it much like a highway gantry signal.

After the drawing and meshing of the plates we modify the command file for the pates.

And in the end we look at the graphical results in ParaVis and in Gmsh.

For this study, we modify the geometry by adding a second top bar 200 mm below the first one, joining these two bars by 5 vertical members and filling the gaps with a steel plate, leaving the hinged joints in between the masts and the top panel[1]. Thus our frame looks very much like a motorway signal frame. And we modify the last load 'cr' into 'cv' representative of the wind blowing onto the top panel.

Figure 8.1 shows the finished structure.

[1] The joints being still modeled with discrete elements.

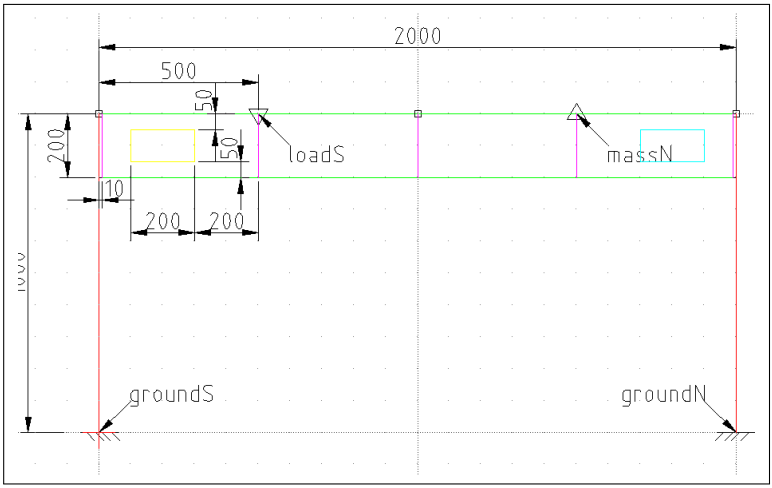


FIGURE 8.1: Sketch of frame3

8.1 Creating geometry and mesh in Gmsh

```
cls=25; //small characteristic length for the surfaces
clb=100; //coarser elsewhere
max=3;
pas=2;
ly=1000; //half length of frame along y
hz=1000; //height of frame along z
hz1=200;
dly=10; //length of hinge along y
Point(1) = {0, 0, hz, cls};
Point(101) = {0, 0, hz-hz1, cls};
//create Point
For i In {0:max:pas}
  Point(10+i) = {0,ly/2*(i-1), hz, cls};
  Point(20+i) = {0,(ly-dly)*(i-1), hz, cls};
  Point(30+i) = {0,ly*(i-1), hz, clb};
  Point(40+i) = {0,ly*(i-1), hz-hz1, clb};
  Point(50+i) = {0,ly*(i-1), 0, clb};
  //next point in middle of inner surfaces
  Point(60+i) = {0,ly/4*(i-1), hz-hz1/2, cls};
  //next points for bottom bar
  Point(110+i) = {0,ly/2*(i-1), hz-hz1, cls};
  Point(120+i) = {0,(ly-dly)*(i-1), hz-hz1, cls};
  //next points for hollows in outside panels
```

```
  Point(210+i) = {0,ly/2*(i-1)*1.4, hz-hz1/4, cls};
  Point(220+i) = {0,ly/2*(i-1)*1.6, hz-hz1/4, cls};
  Point(230+i) = {0,ly/2*(i-1)*1.4, hz-hz1*3/4, cls};
  Point(240+i) = {0,ly/2*(i-1)*1.6, hz-hz1*3/4, cls};
EndFor
```

```
//create Line
top[]={}; //initialize list for top
hinge[]={}; //idem for hinge
mast[]={};
vertb[]={};
panelN[]={};
panelS[]={};
For i In {0:max:pas}
  Line(10+i) = {1, 10+i};
  top[]+=10+i; //add element in top list
  Line(20+i) = {10+i, 20+i};
  top[]+=20+i;
  Line(110+i) = {101, 110+i};
  top[]+=110+i;
  Line(120+i) = {110+i, 120+i};
  top[]+=120+i;
  Line(30+i) = {20+i, 30+i};
  hinge[]+=30+i; //add element in hinge list
  Line(130+i) = {120+i, 40+i};
  hinge[]+=130+i;
  Line(40+i) = {30+i, 40+i};
  mast[]+=40+i;
  Line(50+i) = {40+i, 50+i};
  mast[]+=50+i;
  Line(210+i) = {10+i, 110+i};
  vertb[]+=210+i;
  Line(220+i) = {20+i, 120+i};
  vertb[]+=220+i;
  //next lines for the hollows on outside panels
  Line(250+i) = {210+i, 230+i};
  Line(260+i) = {230+i, 240+i};
  Line(270+i) = {220+i, 240+i};
  Line(280+i) = {210+i, 220+i};
EndFor
Line(201) = {1, 101};
vertb[]+=201;
//makes sure the hinge line are not split in several elements
Transfinite Line {hinge[]} = 2 Using Progression 1;
```

Up to here, it is just like the previous example, with more beams, now come the tricky part, creating the surface panels.

```
//create surface
For i In {0:max:pas}
  //Loop and Surface on the interior
```

```
  Line Loop(310+i) = {201, 110+i, -(210+i), -(10+i)};
  Plane Surface(311+i) = {310+i};
  //Loop and Surface on the exterior
  Line Loop(320+i) = {210+i, 120+i, -(220+i), -(20+i)};
  Line Loop(320+i+1) = {280+i, 270+i, -(260+i), -(250+i)};
  //hollowed surface with two loops
  Plane Surface(325+i) = {320+i, 320+i+1};
  //inside surface on the inner loop
  //minus sign for coherent normals
  Plane Surface(335+i) = {-(320+i+1)};
EndFor
For i In {0:max:pas}
  //forces the surface mesh to pass through the point
  Point{60+i} In Surface{311+i};
EndFor
a[]={311, 325, 335};
panelS[]=a[];
//changing next line to a[]={-313, -327};
//would reverse the normal but only in the saved mesh!!
//not visible in the geam GUI
a[]={313, 327};
panelN[]=a[];
Recombine Surface {panelN[]};
//Recombine Surface {panelS[]};
```

We create the surfaces within a loop, as we can see in figure 8.2 the normals are oriented differently, changing `a[]={313, 327}` to `a[]={-313, -327}`, with the minus sign would put all the normals in the same direction, however this is not visible in Gmsh GUI until one read a mesh file! For this, in Gmsh File > Save Mesh creates a *.msh* file which can the be opened again and looks like figure 8.3.

We also create a hollow surface with two loops, first the outer one, then the inner one[1].

In the end, we specify `Recombine Surface panelN[];`, for one of the group so as to mesh it with quadrilateral elements while `panelS` is meshed only with triangles.

[1] There may be as many loops as there are hollows.

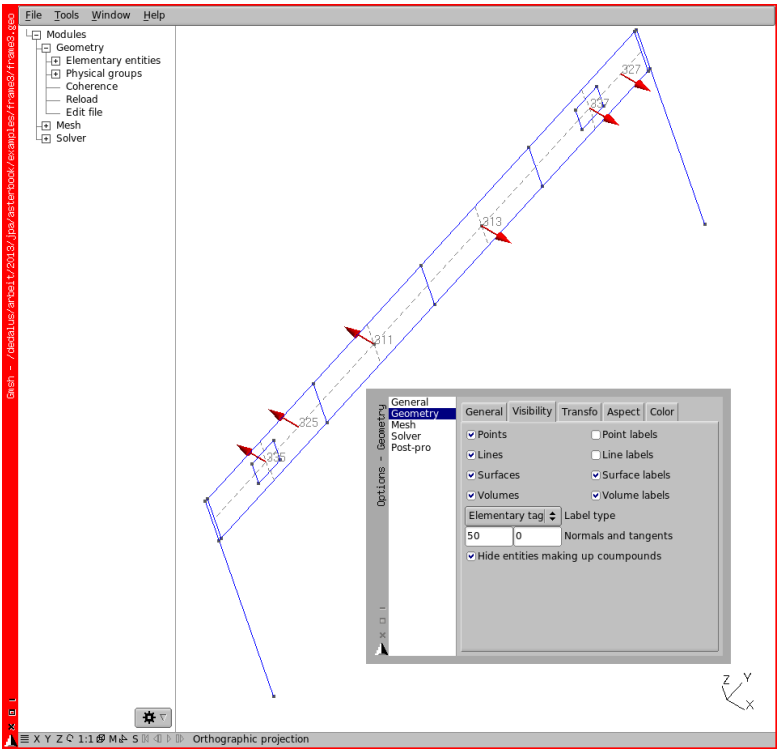


FIGURE 8.2: Orientation of normals from Geometry

Now we create two groups for the surfaces on either side.

```
//makes  the group 'topbeam' with the list top,
Physical Line("topbeam") = top[];
Physical Line("hinge") =hinge[];
Physical Line("mast") = mast[];
Physical Line("vertb") = vertb[];
Physical Surface("panelN") = panelN[];
Physical Surface("panelS") = panelS[];
Physical Point("groundS") = {50};
Physical Point("groundN") = {52};
Physical Point("loadS") = {10};
Physical Point("massN") = {12};
Physical Point("oripanel") = {60};

Color Cyan{ Surface{panelN[]};}
Color Yellow{ Surface{panelS[]};}
```

```
Color Green { Line {top[]};}
Color Red { Line {mast[]};}
Color Purple { Line {vertb[]};}
Color Black { Line {hinge[]};}
```

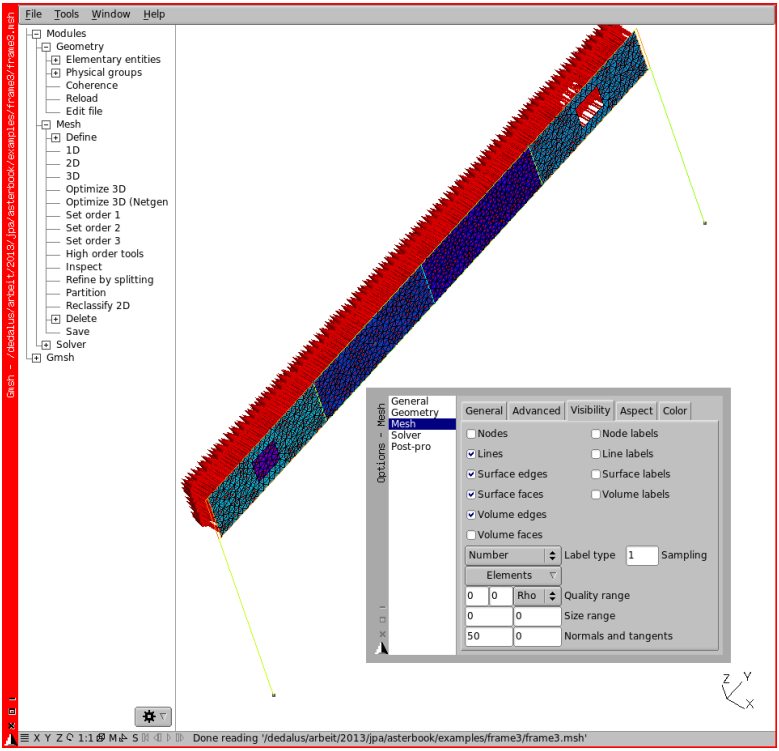


FIGURE 8.3: Orientation of normals from Mesh saved as *frame3.msh* and re-opened, with a[]=-313, -327

This last part creates the groups and assigns some color to the mesh. To look at the orientations of the Surface in Gmsh, in the menu Tools > Options > Geometry > Visibility, we enter a realistic value in the Normals field. Once meshed, with the extra 2D, the mesh looks like 8.3[1].

[1] For the color to appear properly on mesh line elements Lines must be unchecked in Tools > Options > Geometry > Visibility.

The physical point 'oripanel' is created to be used later as a reference point to set the normal orientation in the command file.

We used the `Point In Surface` command to force one Point to become a Node belonging to the Surface, the command `Line In Surface` would do the same to force all nodes belonging to a Line to be nodes on the Surface[1].

Several notes:

- The rendering of the surface is possible only on the mesh not on the geometry.
- Creating line loop by hand in the text editor must be done with an extreme care as it is easy to create an inverted loop on which Gmsh crashes after an error message.

8.2 Commanding for plate elements

Producing a *.comm* file for plates is quite straightforward for the first part.

```
DEBUT();

mesh=LIRE_MAILLAGE( INFO=1,UNITE=20,FORMAT='MED',);

mesh=DEFI_GROUP(
  MAILLAGE=mesh,
    reuse =mesh,
    CREA_GROUP_MA=(
        _F(NOM='panel',UNION=('panelN','panelS',),),
        _F(NOM='TOUT',TOUT='OUI',),
    ),
    CREA_GROUP_NO=(_F(TOUT_GROUP_MA='OUI',),),
);

mesh=MODI_MAILLAGE(
  MAILLAGE=mesh,
    reuse =mesh,
    #U4.23.04
    ORIE_NORM_COQUE=(
        _F(
          GROUP_MA=('panel',),
          VECT_NORM=(1.0, 0, 0),
```

[1] If the Point(s) or Line(s) do not lie exactly in the plane of a Plane Surface a distorted mesh is created.

```
            GROUP_NO='oripanel',
        ),
        #_F(GROUP_MA=('panelN',),),
        #_F(GROUP_MA=('panelS',),),
    ),
);
```

We create a new group which the UNION of the two panel groups created in Gmsh.With ORIE_NORM_COQUE, we reorient the normals of all the elements of this newly created group, in the direction given by VECT_NORM with its origin on the node GROUP_NO='oripanel'.

Then the assignation part.

```
model=AFFE_MODELE(
MAILLAGE=mesh,
    AFFE=(
        _F(
            GROUP_MA=('topbeam','vertb','mast',),
            PHENOMENE='MECANIQUE',MODELISATION='POU_D_T',
      ),
        _F(
            GROUP_MA=('massN',),PHENOMENE='MECANIQUE',
            MODELISATION='DIS_T',
      ),
        _F(
            GROUP_MA=('hinge',),PHENOMENE='MECANIQUE',
            MODELISATION='DIS_TR',
      ),
        #here is the modeling of plate element
        _F(
            GROUP_MA=('panel',),PHENOMENE='MECANIQUE',
            MODELISATION='DKT',
      ),
    ),
);
steel=DEFI_MATERIAU(ELAS=_F(E=210000.,NU=0.3,RHO=8e-9),);

material=AFFE_MATERIAU(
    MAILLAGE=mesh,
    AFFE=_F(
        GROUP_MA=('topbeam','vertb','mast','panel'),
        MATER=steel,
    ),
);
```

```
elemcar=AFFE_CARA_ELEM(
    MODELE=model,
    POUTRE=(
        _F(
            GROUP_MA=('mast',),
```

```
            SECTION='RECTANGLE',CARA=('HY','HZ','EP',),
            VALE=(40, 20,1.5,),
        ),
        _F(
            GROUP_MA=('topbeam',),
            SECTION='RECTANGLE',CARA=('HY','HZ','EP',),
            VALE=(40, 20, 1.5,),
        ),
        _F(
            GROUP_MA=('vertb'),
            SECTION='RECTANGLE',CARA=('HY','HZ','EP',),
            VALE=(20, 20, 1.5,),
        ),
    ),
    DISCRET=(
        _F(GROUP_MA='massN', CARA='M_T_D_N',VALE=(.01),),
        _F(
            GROUP_MA='hinge',
            CARA='K_TR_D_L',VALE=(1e6,1e6,1e6,1e1,1e9,1e9),
            REPERE='GLOBAL'
        ),
    ),
    #the plate is given a thickness of 3 or 1
    #and the orientation of the element is defined
    #by VECTEUR see U4.42.01
    COQUE=(
        _F(GROUP_MA='panelN',EPAIS=3.0,VECTEUR=(0,1,0),),
        _F(GROUP_MA='panelS',EPAIS=1.0,VECTEUR=(0,1,0),),
    ),
);
```

We apply the `panel` properties to the `UNION` group `panel` but we give different thickness to the groups `panelN` and `panelS`[1].

`VECTEUR` is a vector whose projection on the element plane, sets the local x axis of the coordinate system of the element. It is used to compute the principal stresses[2], it should not be normal to **ANY** element in the group[3]! In our case it is a Y (global axis) oriented vector, lying perfectly in the plane of all the elements, it is not always so easy!

We should not make a confusion between the normal vector and the vector defined just above. Just like any structural element, beam or discrete, a plate, or a shell, needs an associated local coordinate system in which:

- local z axis is the normal vector, defined by the mesh topology itself;

[1] And this 1 to 3 different thickness is visible in the flexural stress in the panel.
[2] For anisotropic plates it is also used to define the anisotropic directions.
[3] If this happens *Code_Aster* raises an error, and the vector needs another definition or the group needs to be split into several others.

- local x axis is the projection of the above defined, VECTEUR, vector on the plane tangent to the element at its barycenter, this is unknown by the mesh topology and*has to be defined*;
- local y axis completes the trihedron.

Figure 8.4 illustrates the matter.

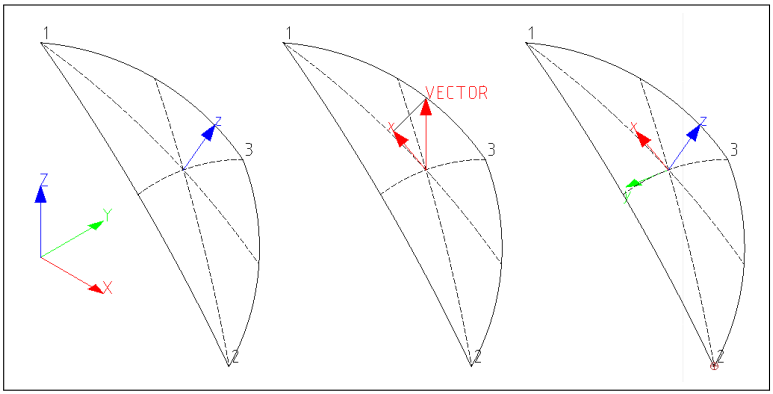


FIGURE 8.4: Shell element local axis

With:

1. triangular shell element (1,2,3), with local z vector, normal, as given from mesher or ORIE_NORM_COQUE;
2. VECTOR given as VECTEUR=(0,0,1), projected on element gives local x axis;
3. complete local axis trihedron.

The boundary conditions are as follows.

```
ground=AFFE_CHAR_MECA(
    MODELE=model,
    DDL_IMPO=_F(
        GROUP_NO=('groundS','groundN',),
        DX=0,DY=0,DZ=0,DRX=0,DRY=0,DRZ=0,
    ),
);
```

```
selfwght=AFFE_CHAR_MECA(
    MODELE=model,
    PESANTEUR=_F(
        GRAVITE=10000,DIRECTION=(0,0,-1),
        GROUP_MA=('topbeam','vertb','mast','massN','panel',),
    ),
);

cc=AFFE_CHAR_MECA(
    MODELE=model,
    FORCE_NODALE=_F(GROUP_NO=('loadS',),FZ=-100,),
);

cv=AFFE_CHAR_MECA(
    MODELE=model,
    #here we define a pressure on the plate elements
    FORCE_COQUE=_F(GROUP_MA=('panel'),PRES=0.01,),
    #next line would have meant a distributed force along x
    #equivalent if normals are in the right direction
    #FORCE_COQUE=_F(GROUP_MA=('panel'),FX=-0.01,),
);
```

PRES is a uniform pressure, acting along the normal of the element, but in the opposite direction, this can raise many errors or give unexpected results if the normals are not oriented as we think they are! For this kind of study I would rather use FORCE_COQUE, but for learning from the mistakes let's try with F3.

Stepping for the load case and solving.

```
selfw_m=DEFI_FONCTION(
    NOM_PARA='INST',
    VALE=(2,0, 3,1.35, 5,1.35, 6,0,),
    PROL_DROITE='CONSTANT',
);
cc_m=DEFI_FONCTION(
    NOM_PARA='INST',
    VALE=(3,0, 4,1, 5,1, 6,0,),
    PROL_GAUCHE='CONSTANT',PROL_DROITE='CONSTANT',
);
cv_m=DEFI_FONCTION(
    NOM_PARA='INST',
    VALE=(4,0, 5,1.5, 6,1),
    PROL_GAUCHE='CONSTANT',PROL_DROITE='CONSTANT',
);

liste=DEFI_LIST_REEL(
    DEBUT=2.0,
    INTERVALLE=_F(JUSQU_A=6,PAS=1.0,),
);
```

```
stat=MECA_STATIQUE(
    MODELE=model,
    CHAM_MATER=material,
    CARA_ELEM=elemcar,
    EXCIT=(
        _F(CHARGE=ground,),
        _F(CHARGE=selfwght,FONC_MULT=selfw_m,),
        _F(CHARGE=cc,TYPE_CHARGE='FIXE',FONC_MULT=cc_m,),
        _F(CHARGE=cv,TYPE_CHARGE='FIXE',FONC_MULT=cv_m,),
    ),
    LIST_INST=liste,
);
```

```
stat=CALC_CHAMP(
    reuse =stat,RESULTAT=stat,
    CONTRAINTE=(
        'SIEF_ELNO',
        'SIPO_ELNO',
        'SIPM_ELNO',
        'SIGM_ELNO',
    ),
    FORCE=('REAC_NODA'),
);
```

In a single call of CALC_CHAMP we calculate forces, stresses and nodal reactions[1].

Keyword SIGM_ELNO is here to calculate the stress in the plate element, but only in the neutral plane.

And the way to post-process the plate elements with the stress and criteria on the top face is as follow:

```
stat2=POST_CHAMP(
  RESULTAT=stat,
    GROUP_MA=('panel',),
    EXTR_COQUE=_F(
        NUME_COUCHE=1,
        NIVE_COUCHE='SUP',
        NOM_CHAM=('SIGM_ELNO',),
    ),
);

statsup=CALC_CHAMP(
    RESULTAT=stat2,
    GROUP_MA=('panel',),
```

[1] Though under CONTRAINTE, SIEF_ELNO for beams is a force, or moment, and not a stress.

```
    CRITERES=('SIEQ_ELNO',),
);
```

At first the concept 'stat' holds the results for all layers. When one needs to compute on a particular layer, it is necessary to extract in a new concept, `stat2`, the field on the element per node `NOM_CHAM=('SIGM_ELNO',)`, with `POST_CHAMP`, here on `NUME_COUCHE=1`[1], as there is only one layer, on the top face `NIVE_COUCHE='SUP'`. Before using `CALC_CHAMP` again to compute a field at node `CRITERES=('SIEQ_ELNO')` in the final concept `statsup`.

With this, we have calculated the stresses in the plate elements in the top face `NIVE_COUCHE='SUP'` together with stresses at nodes, like the well known "von Mises" equivalent stress[2].

```
masse=POST_ELEM(
    RESULTAT =stat,
    MODELE=model,
    MASS_INER=_F(
        GROUP_MA=('topbeam','mast','massN','panelN','panelS'),
    ),
    TITRE= 'masse'
);
```

This, now well known, bit to calculate and print the mass of the model.

This previous section about calculating forces and stresses in elements is only valid in 11.3, or higher, versions of *Code_Aster*. For version 10.8 the syntax was different and is described in chapter 18.7 .

8.3 Printing the results

Printing the ASCII results is just like the previous example, however we print the maximum and minimum for `SIEQ_NOEU, VMIS` on the top face of the plate elements to check the displayed values.

[1] "Couche", in french, means Layer, and "Nive" is an abstract for "Niveau" which means Level.

[2] If the difference in between the neutral plate of the plate and the top face does not jump to the eye, add a bottom face calculation with `INF` to better show the bending behavior in 'panel'.

```
IMPR_RESU(
    MODELE=model,
    FORMAT='RESULTAT',
    RESU=(
        _F(
          RESULTAT=statsup,
            NOM_CHAM='SIEQ_NOEU',
            NOM_CMP=('VMIS',),
            GROUP_MA=('panelN',),
            FORMAT_R='1PE12.3',
            VALE_MAX='OUI',VALE_MIN='OUI',
            INST=6.0,
        ),
    _F(
          RESULTAT=statsup,
            NOM_CHAM='SIEQ_NOEU',
            NOM_CMP=('VMIS',),
            GROUP_MA=('panelS',),
            FORMAT_R='1PE12.3',
            VALE_MAX='OUI',VALE_MIN='OUI',
            INST=6.0,
        ),
    ),
);
```

And now, we are going to prepare a .med result file slightly different from the previous ones, and read this file with Salome-Meca module ParaVis and Gmsh.

```
IMPR_RESU(
    FORMAT='MED',
    UNITE=80,
    RESU=(
        _F(
            GROUP_MA=('topbeam','vertb','mast','panel',),
            RESULTAT=stat,NOM_CHAM=('DEPL',),
        ),
        _F(
            GROUP_MA=('topbeam','vertb','mast',),
            RESULTAT=stat,NOM_CHAM=('SIPO_ELNO',),
        ),
        _F(
            GROUP_MA=('topbeam','vertb','mast',),
            RESULTAT=stat,NOM_CHAM=('SIPO_ELNO',),
            NOM_CMP=('SMFY',),NOM_CHAM_MED='smfy',
        ),
        _F(
            GROUP_MA=('topbeam','vertb','mast','panel',),
            RESULTAT=stat,NOM_CHAM=('SIPM_ELNO',),
            NOM_CMP=('SIXX',),NOM_CHAM_MED='sixx',
        ),
        _F(
```

```
                GROUP_MA=('panel',),RESULTAT=statsup,
                NOM_CHAM='SIEQ_NOEU',NOM_CMP='VMIS',
                NOM_CHAM_MED='vmis',
            ),
            _F(
                GROUP_MA=('panel',),RESULTAT=statsup,
                NOM_CHAM='SIEQ_NOEU',
            ),
        ),
    );

    FIN();
```

Printing is split in many groups of lines, each one of them performing one and only one specific task.

- First group of lines prints DEPL, the standard displacement field.
- Second group of lines prints SIPO_ELNO the standard beam stresses field, without specific options, the group 'panel' being excluded of the print out[1].
- Third group of lines prints SIPO_ELNO restricted to NOM_CMP=('SMFY',) giving it the name, NOM_CHAM_MED, 'smfy' in the med file.
- Fourth group of lines prints SIPM_ELNO restricted to NOM_CMP=('SIXX',) giving it the name, NOM_CHAM_MED, 'sixx' in the med file.
- Fifth group of lines prints SIEQ_NOEU restricted to NOM_CMP=('VMIS',) for the group 'panel' only[2] and for the result concept 'statsup', giving it the name, NOM_CHAM_MED, 'vmis' in the med file.
- And finally the sixth group of lines prints SIEQ_NOEU as standard, with all the components.

The reason for these restricted outputs is that most of the post-processors do their own averaging or mixture on field from a tensor value

[1] It is not wise to print results specific to beams on plates.
[2] Likewise it is not wise to print results specific to plates on beams.

calculated by *Code_Aster* and the results for a selected component within a standard field may be wrong[1].

8.4 Viewing the results in ParaVis

We do not deal with the display of displacement which is just like as explained in the previous chapter.

Here we display the stresses in beams:

- in the **Properties** window;
  - in Supports tick the elements we want, here 'mast', 'vertb' and 'topbeam';
  - in Fields tick stat_SIPO_ELNO;
  - push the green colored Apply button;
- in the menu select Filters > Integration Points > Elno Mesh an entry appears in the PipeLine Browser, make it visible;
- on the pull down list, probably named Solid Color select _SIPO_ELNO;
- on the pull down list just to the right select SMFY;
- toggle Color Legend Visibility icon, sitting at the left of this toolbar;
- in the Time toolbar[2] choose the INST we want;
- push the Rescale to Data Range icon, *this has to be done all the time to be certain of what-is displayed at on the screen*!

At the end of this process, we get the view like figure 8.5, at INST 6 [3], showing the stress due to the bending moment around the local y axis .

Displaying stresses is much simpler, like a displacement, as it is not necessary to go through the Filters > Integration Points > Elno Mesh step. And we can get a view like figure 8.6 showing the von Mises criteria in the panels at INST 6.

[1] By a small amount due to some averaging algorithm, however they are not wrong with a component directly extracted from the *Code_Aster* calculated results.

[2] This one in View > Toolbars > Current Time Controls

[3] ParaVis is using yet another numbering scheme of the instants!

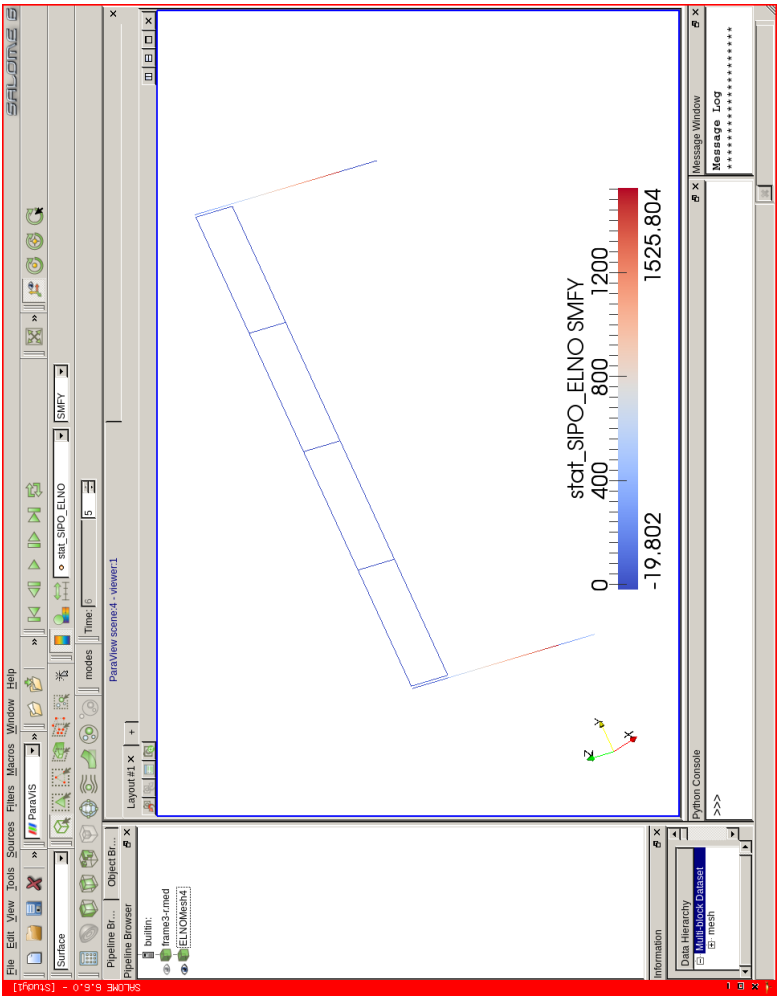


FIGURE 8.5: ParaVis view of SIPO_ELNO, SMFY component

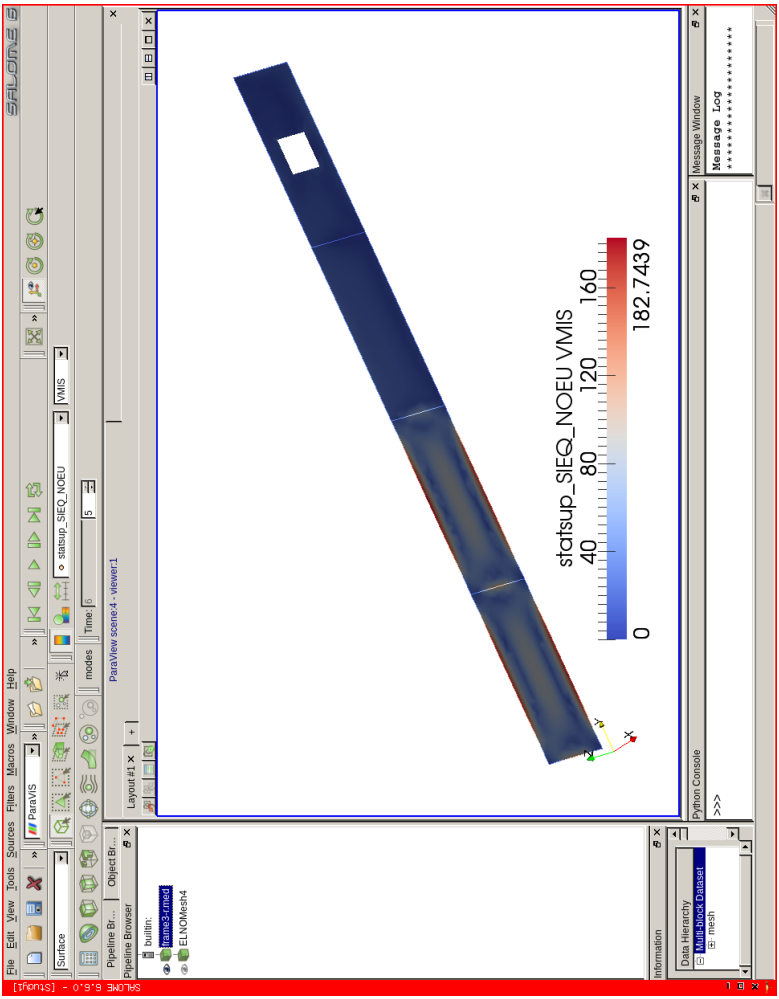


FIGURE 8.6: ParaVis view of SIEQ_NOEU, VMIS component

8.5 Viewing the results in Gmsh

We can open the med result file n Gmsh. On the left-hand side in the tree, we can see under Post_processing 6 lines for every one of the 6 fields we saved in the med file. Likewise there are 6 scalar bars in the graphic window.

In the tree untick the values until only stat__DEPL remains ticked and is the only visible field in the graphic window.

Only the ticked views in the tree are visible in the graphic window.

Push the little arrow on the right-hand side to the menu Options which opens a dialog window, this window allows to make change to the display in the graphic window.

First of all it is a good idea to go in Tools » Options » Mesh » Visibility and uncheck everything so the post-processing view is not polluted by some mesh views[1]. Pushing the keys < Alt> + M also hides the mesh, or push the 'M' in the status bar.

To choose the step, we go to the General tab and increase or decrease the list right of Time step, either by typing a value or using the - and + buttons. Whatever was the step numbering scheme chosen in *Code_Aster* Gmsh always starts at 0, likewise the views are also numbered from 0.

8.5.1 Displaying displacement

To view the deformed shape:

1. in the Visibility tab pull the lower left list to Force vector;
2. in the Aspect tab pull the Vector display list to Displacement;
3. choose a significant value for Displacement factor.

Figure 8.7 shows the deformed shape[2].

[1] Though sometimes it is useful to see the mesh.

[2] Note the view restricted to some groups in **Visibility** dialog box, and the mesh has been made visible for lines.

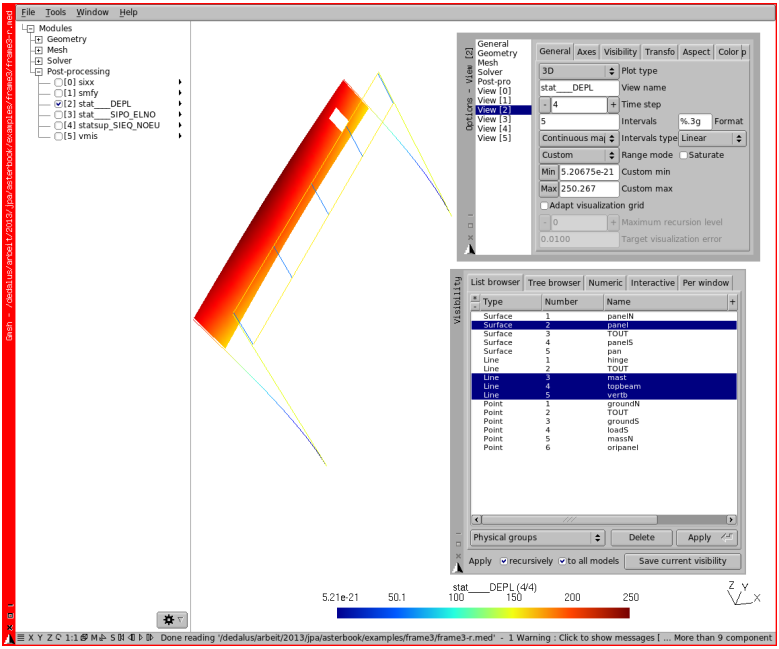


FIGURE 8.7: Gmsh view of displacement at INST 6, superimposed on the undisplaced mesh

To view individual components:

1. in the Visibility tab pull the lower left list to Force scalar;
2. in the box immediately to the right type in the field Id (Id is 0 for DX, 1, is for DY, and so on);
3. in the General tab pull the list Range mode to Custom;
4. and push the Min and Max buttons to refresh the display with the proper component values.

To view numerical values:

1. in the General tab pull the list right of Interval types to Numeric values;

2. in the Aspect tab pull the Glyph location list to Vertex to display the values at the nodes.

8.5.2 Displaying stress in beam element

Select and make visible the view stat__SIPO_ELNO Here the view of the field as Vector display is meaningless! To view individual components:

1. in the Visibility tab, pull the lower left list to Force scalar;
2. in the box immediately to the right, type in the field Id (Id is 0 for SN, 3 is for SMT, 4 is for SMFY, and so on);
3. in the General tab pull the list Range mode to Custom;
4. and push the Min and Max buttons to refresh the display with the proper component values.

To view numerical values:

1. in the General tab pull the list right of Interval types to Numeric values;
2. in the Aspect tab pull the Glyph location list to Vertex to display the values at the nodes, which sometimes may produce a very cluttered view, or Barycenter to display the values in the middle of the element.

To restrict the view to some groups:

1. in the Gmsh menu Tools choose Visibility;
2. in the Physical groups options restrict the views to whichever groups we want.

Figure 8.8 shows value of the field SIPO_ELNO component SMFY.

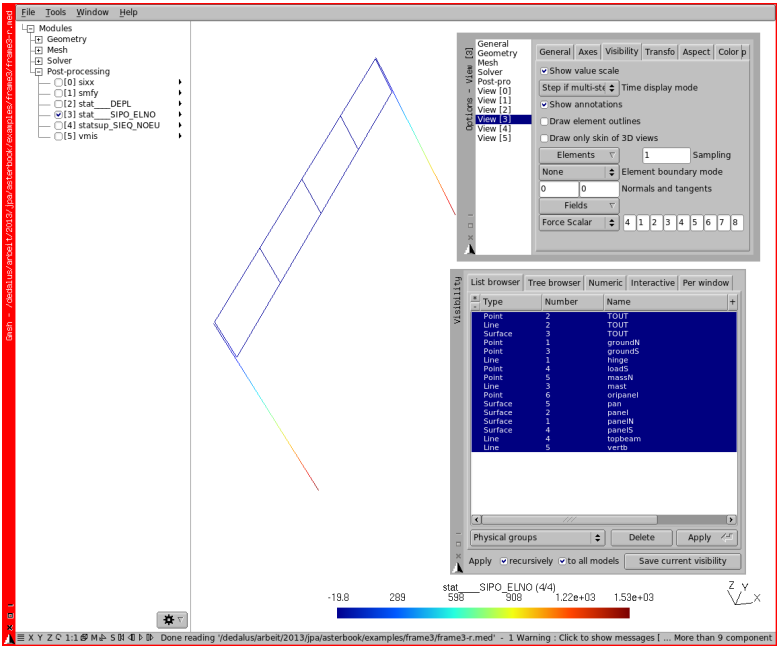


FIGURE 8.8: Gmsh view of SIPO_ELNO, SMFY at INST 6

8.5.3 Displaying stress in plate element

Select and make visible the view stat__SIEQ_NOEU, and proceed along just as for the stresses in beam elements.

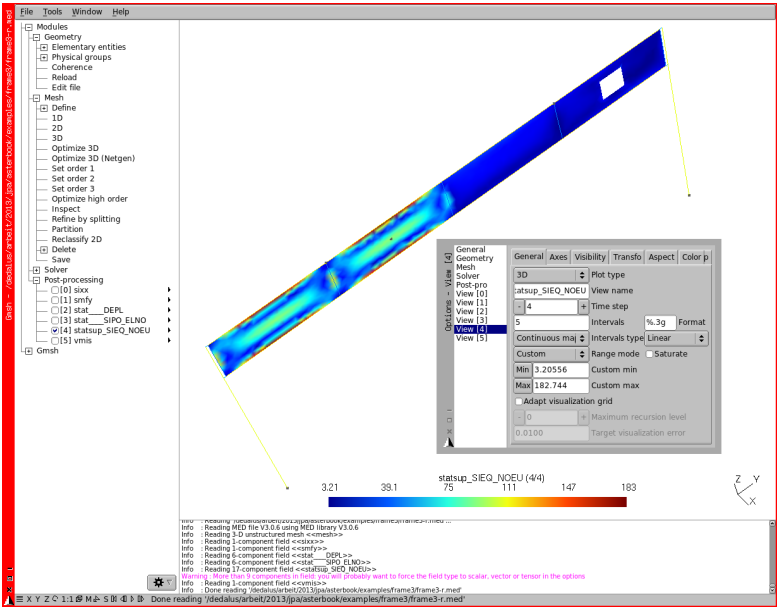


FIGURE 8.9: Gmsh view of `SIEQ_NOEU, VMIS component`

8.5.4 Displaying stress of a named field

In the .comm field we specified some results like this:

```
IMPR_RESU(
    FORMAT='MED',
    UNITE=80,
    RESU=(
        ........................
        _F(
            GROUP_MA=('panel',),RESULTAT=statsup,
            NOM_CHAM='SIEQ_NOEU',NOM_CMP='VMIS',
            NOM_CHAM_MED='vmis',
        ),
    ),
);
```

here the field contains only one component and the view in Gmsh is best made with:

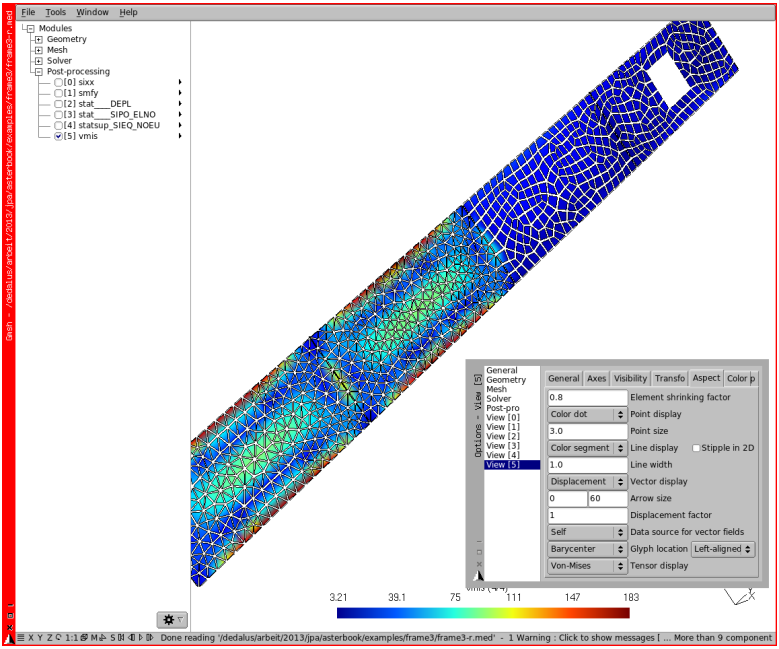


FIGURE 8.10: Gmsh "exploded" view of 'vmis' at INST 6

1. in the Visibility tab pull the lower left list to Original Field;
2. and proceed along as above.

Figure 8.10 shows value of the field 'vmis' MED field with, in dialog box Options:

- tab Visibility with Draw elements outlines ticked on;
- tab Aspect with Element shrinking factor set to 0.8;

to display a well known "exploded" or "shrunk" appearance!

8.5.5 Displaying more...

More tips about the use of Gmsh in post-processing are given in appendix B.

CHAPTER 9

Stiffening it with rods

In this chapter, we add to the structure of chapter 8 a lateral stiffening against lateral load under the shape of rod shrouds.

We take the opportunity to run the study directly in ASTK, without the help of Salome-Meca.

And we look at the results directly in STANLEY.

Back to engineer problems. We may consider this structure needs stiffening against horizontal loads, so we add four stiffening rods downwards from the top of the masts. These rods are what is called `BARRE` in *Code_Aster* jargon. These elements transmit axial forces, either tension or compression, but no end moments. Of course the real building must be designed and built according to this feature. To handle correctly these elements, *Code_Aster*, like any finite element code, requires one single element along the rod length.

Figure 9.1 shows the finished structure with the stiffening rods.

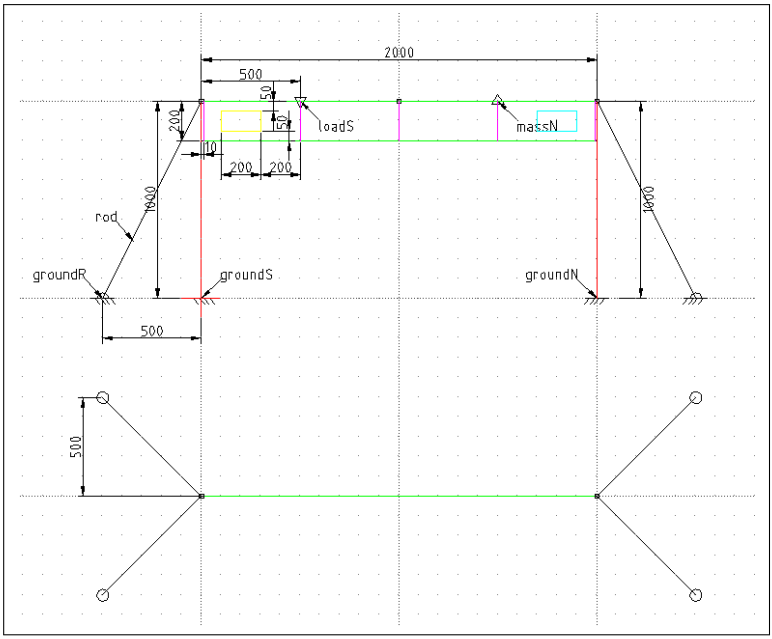


FIGURE 9.1: Sketch of frame4

9.1 Modifying in Gmsh

To create the ground points, the following is added to the points loop.

```
Point(550+i) = {500,(ly+500)*(i-1), 0, cls};
Point(560+i) = {-500,(ly+500)*(i-1), 0, cls};
```

In the loop creating the lines, we add[1]:

```
Line(550+i) = {550+i, 20+i};
rod[]+=550+i;
```

[1] And here we can see the advantages of the "logical" numbering scheme we use.

```
    Line(560+i) = {560+i, 30+i};
    rod[]+=560+i;
```

And:

```
Transfinite Line {rod[]} = 2 Using Progression 1;
Physical Line("rod") = rod[];
Physical Point("groundR") = {550, 552, 560, 562};
```

to ensure the rods are meshed as a single element, and finally a group is created for both the rods and the ground points. Once meshed the model looks like figure 9.2, and we save it as *frame4.med*

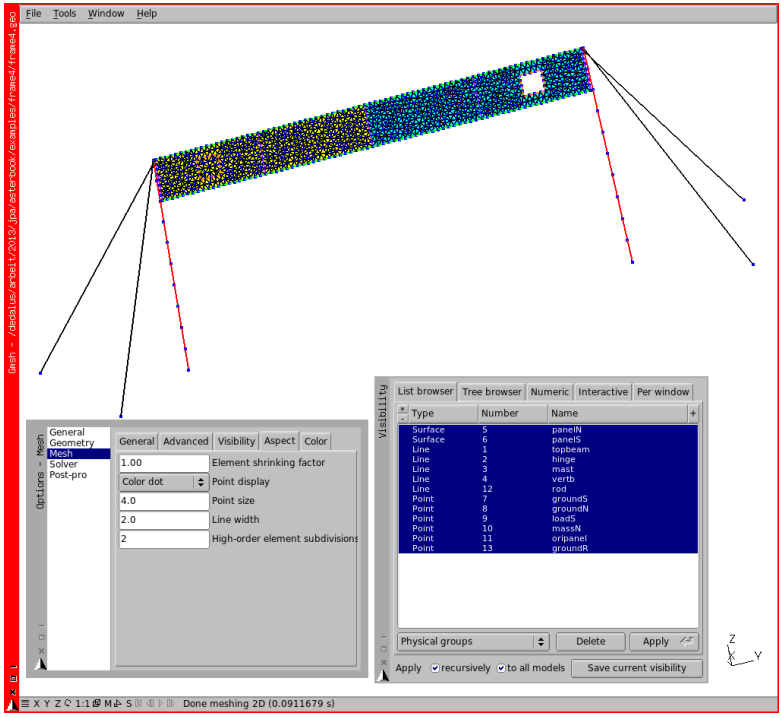


FIGURE 9.2: Motorway signal frame meshed with rod support, visibility restrained to Physicals.

9.2 Enhancing the command file

We add the following lines[1]:

```
model=AFFE_MODELE(
    MAILLAGE=meshf,
    AFFE=(
         ..................
        _F(
            GROUP_MA=('rod',),
            PHENOMENE='MECANIQUE',MODELISATION='BARRE',
        ),
         ..............................

elemcar=AFFE_CARA_ELEM(
    MODELE=model,
     ......................
    BARRE=_F(
        GROUP_MA=('rod',),SECTION='CERCLE',
        CARA=('R','EP',),VALE=(16,1.5,),
    ),
     ................................

ground=AFFE_CHAR_MECA(
    MODELE=model,
    DDL_IMPO=(
     ....................
        _F( GROUP_NO=('groundR',),DX=0,DY=0,DZ=0,),
);
```

A node belonging to a BARRE carries only translational DOFs, that's why the boundary condition misses DRX, DRY and DRZ.

The part for print out may be modified with this addition as it is a good idea to verify the traction or compression force load in the bar[2].

```
IMPR_RESU(
    MODELE=model, FORMAT='MED', UNITE=80,
    RESU=(
         ......................
        _F(
            GROUP_MA=('rod',), RESULTAT=stat,
            NOM_CHAM=('SIEF_ELNO',),NOM_CMP='N',
            NOM_CHAM_MED='force_in_rod',
        ),
         ................................
```

Finally, we also save this file as *frame4.comm*

[1] Not forgetting to add 'rod' in material assignment
[2] The same type of entry could be put in the .resu file.

9.3 Introducing ASTK for the analysis

Here, we run the study in a more sophisticated tool named ASTK. ASTK, whose name is the contraction of ASTER and TK, is a powerful tool allowing to put together very complex studies in a relatively simple graphical interface.

ASTK may be launched from Salome-Meca or from a stand alone *Code_Aster* installation[1]. From **Salome-Meca**, in the browser, RMB click on Aster - frame4 then choose Export to ASTK.

The ASTK handling is well described in U1.04.00.

Once launched a window like figure 9.3 appears[2].

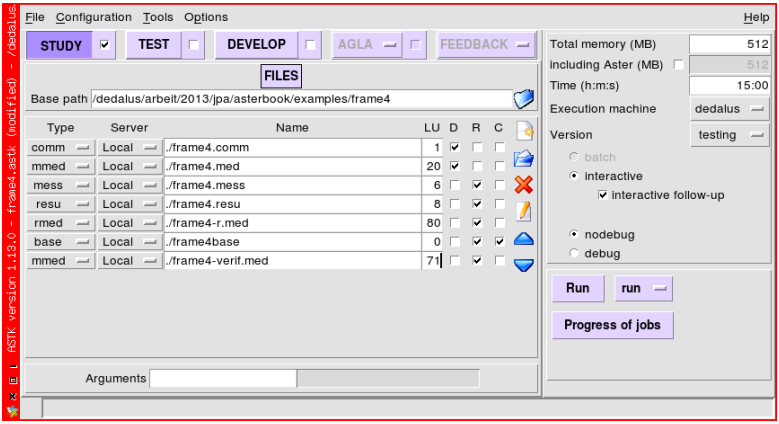


FIGURE 9.3: ASTK window

On the top part, right of Base path field, click on the icon looking like a directory, then navigate until reaching the study directory and select it.

In the main part of the window, we can see the files of the study. And a stack of icon on the right. Let's click on the top one to create a new line.

On this line, we create an entity to retrieve the verification mesh, referenced in the first lines of the *.comm* file:

[1] Where it stands in *$ASTER_ROOT/bin*.

[2] Here it is filled up, more or less the way it looks like following an export from Salome-Meca, otherwise it is empty.

- Pull down the name list on the extreme left and choose mmed to specify it as a med file.
- Name it *./frame4verif.med*.
- In the column LU, Logical Unit change 20 to 71, that's what we have specified in the *.comm* file.
- Uncheck the column D, standing for data and check the column R, standing for results as we want to write in this file.

We also create a new line with type base, named 'frame4base', LU=0 and with R and C[1] ticked to retrieve a database with the results.

Tick interactive follow-up and push the button Run. A terminal window should pop up. Once the terminal has disappeared, have a look in the *.mess* file to see if all went well.

ASTK is a powerful tool offering a more sophisticated handling of jobs, like chaining multiple *.comm* file and multiple output files and much more...
However we have to go back to Salome-Meca[2] or Gmsh to open the *.med* results file to see what the results looks like.

A few more notes about ASTK:

- on the right side of the **ASTK** window, we can change the Total memory and the Time allocated to the job;
- the file extensions are not mandatory at all and we can use whatever we like, or none at all[3];
- except for 1, 6 and 8 the LU may be also chosen as one desires.

[1] C stands for compressed.
[2] Or a stand alone version of Salome.
[3] Like any file in the Unix world!

9.4 Using STANLEY, a quick approach to post-processing

Once the calculation is successfully finished we can RMB click on the line of the database and choose Open with... > Post-processing using Code_Aster (Stanley) a window like figure 9.4 pops up. A data base enables the restart of a calculation where it was left off.

STANLEY is a very powerful tool to view results, particularly in the early stages of a study.

The document U4.81.31 explains the use of STANLEY.

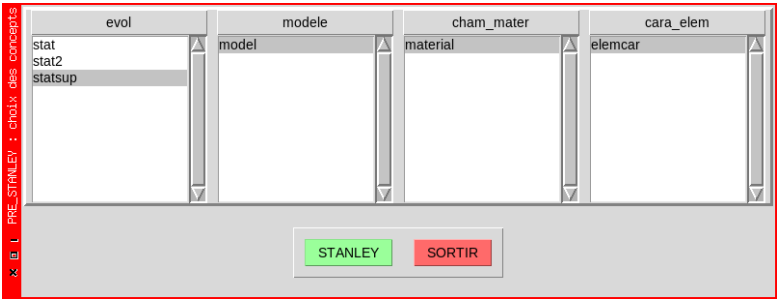


FIGURE 9.4: STANLEY window

At first, we may have to set some parameters in the menu Parametres - Editer, toggle the Mode switch to Gmsh/Xmgrace, push the button OK, figure 9.5.

We could have left the Mode on Salome, but it's more fun to start in Gmsh mode[1]!

Then select the same item as in figure 9.6, push STANLEY:

- on the left most column named Champs (fields) select SIEQ_NOEU.
- On the next column named Composantes (components) select VMIS;

[1] It is also the default, and only way to proceed if you have a stand alone *Code_Aster* install without Salome

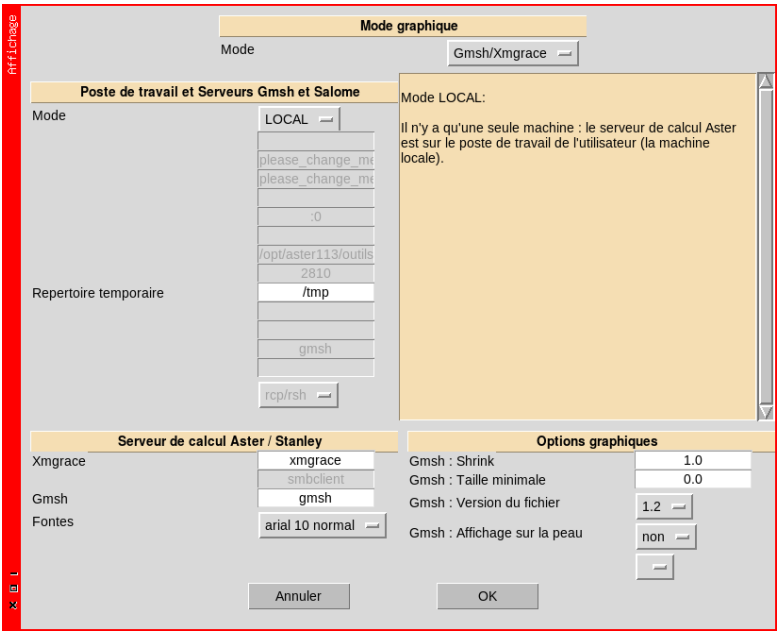


FIGURE 9.5: STANLEY parameters set to Gmsh/Xmgrace" mode

- on the next column named Entites Geometriques (geometric entities) select panel (2D);
- on the right most column named Ordres select 5 (this is the load case at the last instant, number 6 in our case)[1].

The window now looks like figure 9.6. On the extreme right the traffic light is green, we can push TRACER.

Had the traffic light been orange we would have had to push CALCULER so as to calculate the field and turn the light to green.

Had the light been red, then the requirements could not be met and the field could not be calculated.

In our case the light is green. Let's go, push TRACER!

[1] A LMB click on the Ordres opens a large list of options to change to, among which Inst.

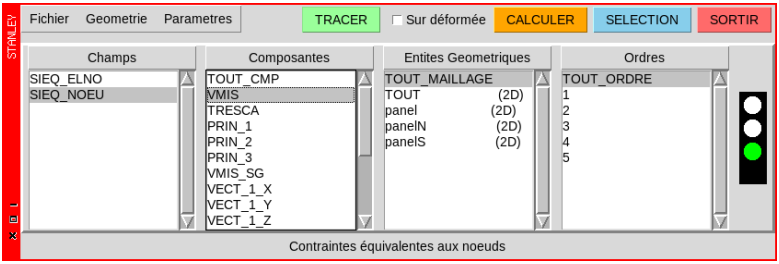


FIGURE 9.6: STANLEY window, selection made

Hum! nothing happens? Then read in appendix E 5.3 . Once corrected or if all go well then a window like figure 9.7 appears.

I am cheating a bit, as at first the selected view is the XOZ plane, and our model has a null y dimension, so we have to turn it a bit, to view something[1].

A useful hint for Post-processing view in Gmsh is as follows:

- in the **Gmsh** command window, we choose the view we want in the view list;
- push the arrow on the right;
- then Options - Color;
- then uncheck Enable lighting.

With this the color mapping of results is independent of any light source, otherwise it may become unreadable in areas which are in the shade from the light source.

In the **Gmsh** command window, menu File » Save Default Options saves this setting for all further Gmsh work.

One annoying drawback of Gmsh post-processing with STANLEY is that the groups of 1D elements like beams or rods are not present as groups in the file, while the groups of 2D elements are.

[1] The Gmsh windows appearance is also a little different from what we saw previously as the version of Gmsh embedded in*Code_Aster* is not the last one.

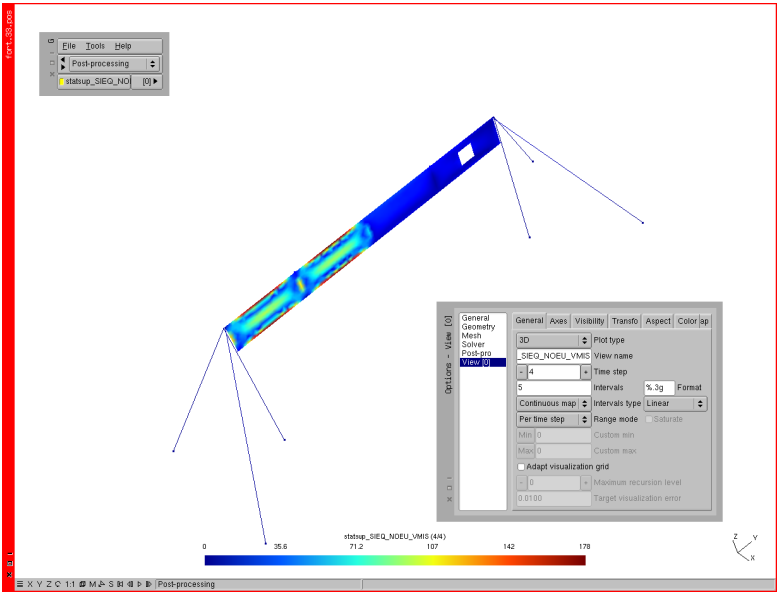


FIGURE 9.7: Von Mises criteria in 'panel' element in Gmsh Post-pro View

There is a lot more to play-with concerning the display appearance in Gmsh, this can be set by pressing the arrow on the right of the View , then Options, a lot of them are available there.

In the Options of Gmsh, if we play with the - + button on the list named Time step we notice that Gmsh has its own stepping, always starting at 0, with steps one by one, whatever the stepping of `INST` stated in the *.comm* file is.

And a final remark, Stanley can be called within a *.comm* file just by writing the line `STANLEY()` just after the calculation, `MECA_STATIQUE` here, as the fields do not need to have been calculated before. And in this case we have to quit STANLEY by pushing SORTIR for the *.mess* and *.resu* and all the results files to be saved on disk.

And last but not least STANLEY may be called by a RMB click on the database even in case of an aborted calculation[1] allowing us to have a look at the results just before the crash, quite useful to guess was went wrong!

[1] As this happens all too often in a non-linear calculation.

CHAPTER 10

Replacing rods, by cables, first step in non-linear

In this chapter, we replace the rods of chapter 9 by cables.

And this implies a non linear analysis which we setup.

10.1 Replacing rod by cables

We can see, in the previous model, some rods being in tension while the others are in compression. These rods when in compression may fail by buckling at a very modest load[1], in order to take this kind of failure into account a sound approach is to replace them by wire ropes.

If we want to replace the rod elements by wire rope elements or `CABLE`[2], we need to perform a non-linear analysis. Indeed a cable does

[1] More about buckling in chapter 15.2 .

[2] This is how this element is known in the *Code_Aster* jargon.

not support any compressive load: its behavior is therefore non-linear and a dedicated constitutive law has to be used.

Switching to a non-linear simulation is quite easy.

Starting from *frame4.geo*, we create a *frame5.geo* file replacing the group name 'rod' by 'cable', for the sake of clarity. We also change the 'Transfinite' command allowing to mesh the cables with 20 elements, and not a single one, along their length. Then, we update the mesh and save it as *frame5.med*.

Roughly speaking a non-linear analysis is performed in multiple calculation steps in order to diminish the effects of the non-linearities. At each step the geometry may be updated to account for large displacements, the stiffness of each element may change due to non-linear constitutive behavior. This usually makes non-linear calculations more CPU-intensive.

In the case of cables, this stiffness update causes them to be kind of "eliminated" when they are in compression (near-zero stiffness). There are many parameters available in non-linear analysis, here, we just only scratch the surface of what is available.

One last remark before starting: a linear calculation always gives a result, but it may sometimes be irrelevant, for example with displacements exceeding the model dimensions. This is because the calculation is made only once on the un-deformed shape with the assumption of linear elastic behavior. That's why a result should always be questioned and the hypothesis used in the analysis verified. On the contrary non-linear calculations may fail and stop at one step in the middle with a singular matrix or a lack of convergence, in this case we need to tune some parameters and try it again. A close look at the *.mess* file is here of great help. Non-linear analysis may become a rather tedious involvement.

10.2 Switching to non-linear analysis

Non-linear calculation requires some changes to the command file, we describe them now.

```
model=AFFE_MODELE(
    MAILLAGE=mesh,
    AFFE=(
    ........
    #here is the modelling of cable element
        _F(
            GROUP_MA=('cable',),
            PHENOMENE='MECANIQUE',MODELISATION='CABLE',
        ),
    ),
);
```

Most structures including cables require some pre-tension being applied in the cable, here, we do it by cooling them[1].

Firstly, we describe a temperature field, set to zero on the whole model.

```
temper1=CREA_CHAMP(
    TYPE_CHAM='NOEU_TEMP_R',
    MODELE=model,
    OPERATION='AFFE',
    AFFE=(
        _F(TOUT='OUI',NOM_CMP='TEMP',VALE=0.,),
    ),
);
```

Secondly, we describe another temperature field, set to -100° C on the cables and to zero on the rest of the model.

```
temper2=CREA_CHAMP(
  MODELE=model,
    TYPE_CHAM='NOEU_TEMP_R',
    OPERATION='AFFE',
    AFFE=(
        _F(TOUT='OUI',NOM_CMP='TEMP',VALE=0.,),
        _F(GROUP_MA=('cable',),NOM_CMP='TEMP',VALE=-100.,),
    ),
);
```

From these two fields we create a thermals results, indexed by the time variable, `INST`:

- from `INST` = -1, which is before the start of the actual calculation, to instant `INST` = 1, the null temperature field is applied;

[1] This is a very common practice with most finite elements codes.

- from INST = 1 to INST = 2, a temperature ramp is created using the second field defined below, this cools the cable creating the pre-tension.
- finally from INST 2 to INST 7, which is beyond the actual calculation range, we apply a constant temperature field in order to maintain the pre-load.

```
ltemp2=DEFI_LIST_REEL(
    DEBUT=2.0,
    INTERVALLE=_F(JUSQU_A=7.0,PAS=1.0,),
);

evtemp=CREA_RESU(
  TYPE_RESU='EVOL_VARC',
    NOM_CHAM='TEMP',
    OPERATION='AFFE',
    AFFE=(
        _F(CHAM_GD=temper1,INST= (-1, 0, 1,),),
        _F(CHAM_GD=temper2,LIST_INST=ltemp2,),
    ),
);
```

Strictly speaking this simple model would have converged without any pre-load in the cable and the real construction itself would not need a significant pre-load considering the load direction and level.

Moreover the time at which the pre-load is introduced in the calculation is not very realistic. In true life we would first erect the structure then tighten the cables and afterward submit it to working loads, so the proper order for applying loads might be:

1. gravity load;
2. thermal load to tighten the cables;
3. service load.

This makes an excellent exercise after having read this book!

The coefficient of thermal expansion for the cable is set to $\alpha = 12 \times 10^{-6}\mathrm{K}^{-1}$.

With a temperature increase $\Delta T = -100°\mathrm{K}$.

This yields a strain $\frac{\Delta L}{L} = \alpha times \Delta T = -0.0012$.

And, if the cable is constrained, a force:

$$F = \frac{\Delta L}{L} \times E \times SECTION = -1200 \text{ N}.$$

And a normal stress of $\sigma_N = \frac{F}{SECTION} = 120 \text{ N.mm}^{-2}$.

The cable material needs a special treatment as it is supposed to have no stiffness in compression[1], it also needs a proper assignment of temperature behavior.

Also, although in steel, the cable is of wired construction, so we use an "apparent" modulus of elasticity which is somewhat less than solid steel[2].

```
#U4.43.01
steel=DEFI_MATERIAU(
    ELAS=_F(E=210000.,NU=0.3,RHO=8e-9,ALPHA=12e-6,),
);
msteel=DEFI_MATERIAU(
    ELAS=_F(E=100000,NU=0.3,RHO=8e-9,ALPHA=12e-6,),
    CABLE=_F(EC_SUR_E=1.E-4,),
);
```

Then we assign the material to the model.

```
material=AFFE_MATERIAU(
    MAILLAGE=mesh,
    AFFE=(
        _F(
            GROUP_MA=('topbeam','vertb','mast','panel',),
            MATER=steel,
        ),
        #material to cables
        _F(GROUP_MA=('cable',),MATER=msteel,),
    ),
    #here we apply the temperature that makes the pretension
    AFFE_VARC=_F(
        TOUT='OUI',
        NOM_VARC='TEMP',
        EVOL=evtemp,
        VALE_REF=0.0,
    ),
```

[1] Since this is not possible and would yield a singular matrix, a cable is assumed to have a fraction of the tension stiffness while in compression, `EC_SUR_E` stands for E in Compression divided by E.

[2] The value used here, $100000 N.mm^{-2}$ holds true for quite a low quality cable.

```
);                              ....
```

Note: the temperature is applied to the whole model in this section, TOUT='OUI', and this is done with NOM_VARC='TEMP' in the material assignment.

And finally the cable section, only the section is required, N_INIT=10.0 is a numerical pre-tension in the cable so the calculation is possible and has no effects on the results.

```
elemcar=AFFE_CARA_ELEM(
    MODELE=model,
    ..................
    CABLE=_F(
        GROUP_MA=('cable',),
        N_INIT=10.0,
        SECTION=(10,),
    ),                              ....
```

The non-linear analysis in*Code_Aster* allows many options and parameters to be tweaked. Here is an example of the commands which solve our problem.

```
#we may need tweaking with PAS until the problem converges
#here the problem is almost linear so a large PAS is OK
liste=DEFI_LIST_REEL(
    DEBUT=1.0,
    INTERVALLE=_F(JUSQU_A=6,PAS=0.5,),
);
#we may also want a more restricted list
# at which to print results
listresu=DEFI_LIST_REEL(
    DEBUT=1.0,
    INTERVALLE=_F(JUSQU_A=6,PAS=1.0,),
);
```

We start calculation and printing results at INST 1 to be able to see the temperature and cable pre-loading in at post-processing

```
#here is the non-linear analysis see
#U4.51.03
#U4.51.11
statnl=STAT_NON_LINE(
    MODELE=model,
    CHAM_MATER=material,
    CARA_ELEM=elemcar,
    EXCIT=(
        _F(CHARGE=ground,),
        _F(CHARGE=selfwght,FONC_MULT=selfw_m,),
```

```
        _F(CHARGE=cc,TYPE_CHARGE='FIXE_CSTE',FONC_MULT=cc_m,),
        _F(CHARGE=cv,TYPE_CHARGE='FIXE_CSTE',FONC_MULT=cv_m,),
    ),
    #the beam and plates parts are allowed to deform in
    # 'PETIT' kinematics, e.g. small perturbations
    COMP_INCR=_F(
        RELATION='ELAS',DEFORMATION='PETIT',
        GROUP_MA=('topbeam','vertb','mast','panel'),
    ),
    #the cables parts are allowed to deform in
    # 'GROT_GDEP' kinematics,
    # e.g. large rotations, large displacements
    COMP_ELAS=_F(
        RELATION='CABLE',DEFORMATION='GROT_GDEP',
        GROUP_MA=('cable',),
    ),
    INCREMENT=_F(LIST_INST=liste,),
    #the resolution method
    NEWTON=_F(
        PREDICTION='TANGENTE',
        MATRICE='TANGENTE',
        REAC_ITER=1,
    ),
    #this numerical trick may speed things up
    RECH_LINEAIRE=_F(),
    #how do we consider the calculation is finished at every step
    #that is a sort of quality criteria as well
    CONVERGENCE=_F(
        RESI_GLOB_RELA=1e-4,
        ITER_GLOB_MAXI=300,
    ),
);
```

With the relevant options for the calculations of results for our study :

```
#result concept on the whole model
statnl=CALC_CHAMP(
  reuse =statnl,
  RESULTAT=statnl,
  CONTRAINTE=(
    'SIEF_ELNO',
    'SIPO_ELNO',
    #'SIPM_ELNO', is non available in non-linear
    'SIGM_ELNO',
  ),
  FORCE=('REAC_NODA',),
);

#result concept on the face of the panel
stat2=POST_CHAMP(
  RESULTAT=statnl,
    GROUP_MA=('panel',),
    EXTR_COQUE=_F(
        NUME_COUCHE=1,
```

```
        NIVE_COUCHE='SUP',
        NOM_CHAM=('SIGM_ELNO',),
    ),
);

statsup=CALC_CHAMP(
    RESULTAT=stat2,
    GROUP_MA=('panel',),
    CRITERES=('SIEQ_ELNO','SIEQ_NOEU',),
);
```

10.3 Printing results

Printing the result is no different of what we have shown previously, keeping in mind that we can print only what's been calculated before.

However we want to see the temperature applied in the different members, so we add this section in the *.med* printing.

```
IMPR_RESU(
    FORMAT='MED',
    UNITE=80,
    RESU=(
    _F(
          RESULTAT=evtemp,
            NOM_CHAM='TEMP',
            LIST_INST=listresu,
        ),
        ..................
    ),
);
```

And we can use the restricted list of results by introducing this keyword `LIST_INST=listresu,` wherever we want in `POST_RELEVE_T` or `IMPR_RESU`

The results displays as follows, figure 10.1.

A look at the *.mess* file shows us that the calculation is done in hardly more than one iteration at each step which means that the behavior of the structure is almost linear for the given loads. This is what we expected anyway, here the non-linear calculation is made to take into account the cable behavior. And we can actually see that the load carried by the cable, under the wind load, is quite different from the one carried previously by the rods.

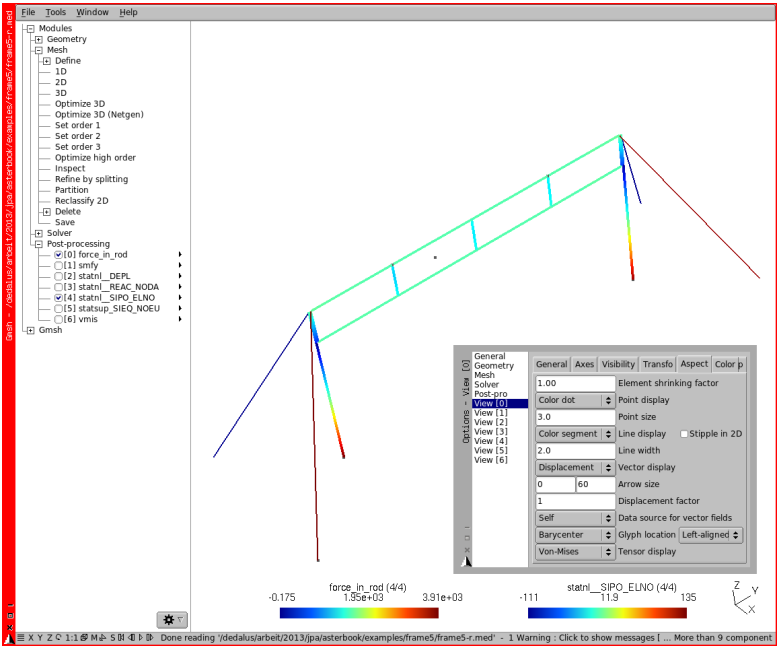


FIGURE 10.1: Results with cables

Precisely the rods in he previous example carried a tension on the windward side and a compression on the leeward of about the same magnitude, at about 2000 N, for the wind load. When the windward cables carry here a tension of 3900 N, and the leeward ones, at -0.18 N are just slack, the pre-tension being 1140 N. Lowering the pre-tension would produce a no load in the leeward cables.

10.4 A variation in CREA_RESU

Creating the 'evtemp' concept with CREA_RESU can also be written like this:

```
evtemp=CREA_RESU(
  TYPE_RESU='EVOL_VARC',
    NOM_CHAM='TEMP',
```

```
    OPERATION='AFFE',
    AFFE=(
    _F(CHAM_GD=temper1,INST=-1.0,),
    _F(CHAM_GD=temper1,INST=1.0,),
    _F(CHAM_GD=temper2,INST=2.0,),
    _F(CHAM_GD=temper2,INST=7.0,),
    ),
);
```

The intermediate values being interpolated in between the given ones[1].

[1] The first alternative gives a more understandable display of 'evtemp', in Gmsh for a *.med* file.

CHAPTER 11

Cycling on a cable

In this chapter, we remove all the top structure to replace it by a cable spanned in between the two masts.

We let a clown cycle along this cable.

Once the analysis is performed we look at the graphical results and animate them on the screen.

And we also produce some time dependent plots with XmGrace, which gives us the opportunity to work with tables.

11.1 Replacing the top bar by a cable

We now model a structure with two vertical masts, each one supported by 2 angled cable shrouds and with an horizontal cable extending in-between the two mast tops. The geometry looks like figure 11.1.

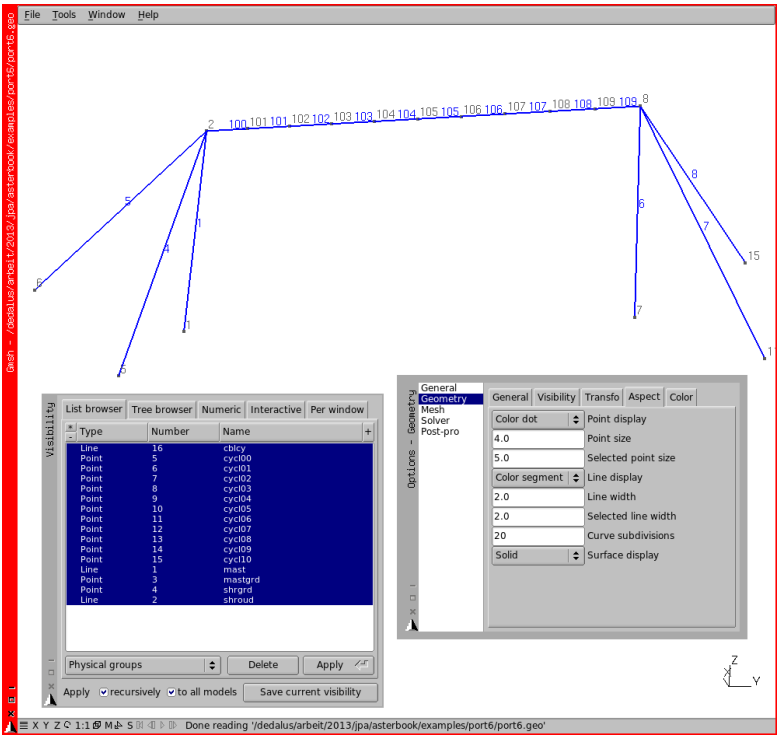


FIGURE 11.1: Geometry ready for cycling

And the corresponding *.geo* file:

```
cl1=100;
Point(1) = {0, -1000, 0, cl1};
Point(2) = {0, -1000, 1000, cl1};
Point(5) = {-500, -1500, 0, cl1};
Point(6) = {500, -1500, 0, cl1};
Line(1) = {1, 2};
Line(4) = {5, 2};
Line(5) = {6, 2};
Symmetry {0, 1, 0, 0} {
  Duplicata { Line{1, 4, 5}; }
}

Physical Line("mast") = {1, 6};
Physical Line("shroud") = {4, 5, 7, 8};
Physical Point("mastgrd") = {1, 7};
Physical Point("shrgrd") = {5, 6, 11, 15};
```

And its second part building the top cable geometry:

```
//this loop creates the points along the top cable
//notice that Point 100 doubles with Point 2,
// Point 110 with Point 8
For i In {0:10:1}
    Point(i+100)={0,-1000+200*i,1000,cl1};
EndFor
//this loop creates the top cable section
For i In {0:9:1}
    Line(i+100)={i+100, i+100+1};
EndFor
//this loops creates individual Physical Point along the
//cable each one with a "logical" name
For i In {0:10}
  Physical Point (Sprintf("cycl%02g",i)) = {i+100};
EndFor
//this loop creates the Physical Line
//describing the top cable
lg[]={};
For i In {0:9}
  lg[]+=i+100;
EndFor
Physical Line ("cblcy") = lg[];
Transfinite Line {lg[]} = 0 Using Progression 1;
//This line removes the double points at geometry level
Coherence;
//This line removes doubles Nodes, once meshed,
//experiment the difference!!
//Coherence Mesh;
```

Notice the loop creating the points 100 to 109, along the top cable, together with the lines joining these points. Notice also the loop creating one group of Physical Point with an indexed name, like 'cycl01', for each one of the above points[1].

And also the 'Coherence' command removing the double points, as well as its fellow 'Coherence Mesh' which does not do exactly the same thing.

Experiment with it to grasp the difference !

Once meshed our structure looks like figure 11.2:

[1] Using the C alike command 'Sprintf'.

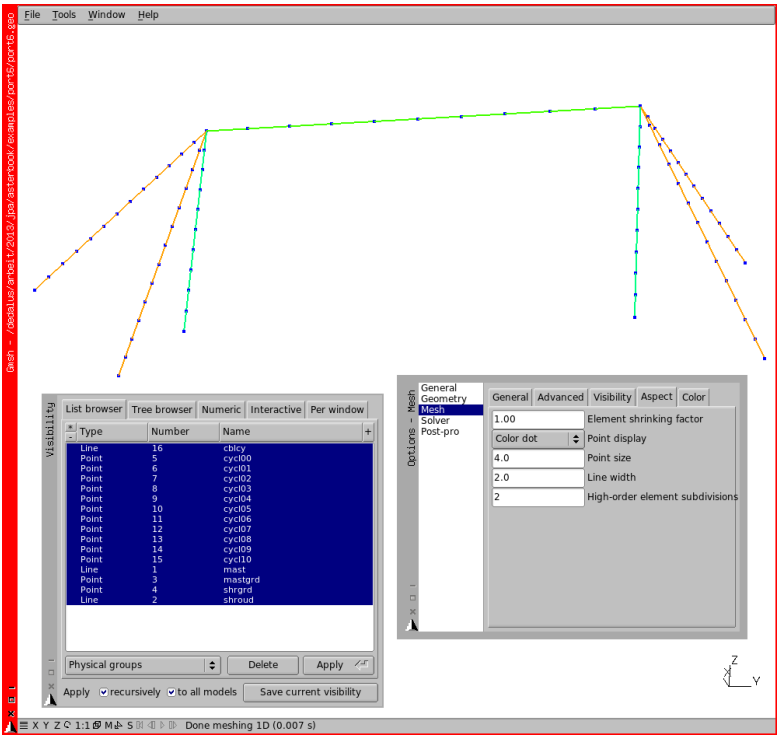


FIGURE 11.2: Structure meshed, ready for analysis

11.2 Cycling on the cable, like a clown!

Now, we look at the behavior of this model with a clown cycling along the cable. This is done by applying the load in a "sawtooth" manner on the node groups previously created, in relation to time.

Here again the so called "instants" are only steps and not a real time, expressed in seconds. We just take into account the fact that the clown load is changing place, regardless of the speed at which the clown is moving since this would be dynamics, outside the scope of this book[1].

[1] The static analysis carried here is valid for the slow speed implied here.

Note: as we apply the clown load on a single node, we suppose he is using a single wheel vehicle, easier for us, trickier for him !

11.2.1 Commanding for solution

We give the first part of the command file with little commentary as all the concepts used here have been reviewed in the previous chapters.

Firstly, mesh reading, manipulating and model making.

```
DEBUT(
    #next line is only useful for more sophisticated Python call
    #within the .comm file
    #U4.11.1
    #PAR_LOT='NON',
);

mesh=LIRE_MAILLAGE(
    INFO=1,
    #INFO_MED=2,
    UNITE=20,FORMAT='MED',
);

mesh=DEFI_GROUP(
    reuse =mesh,MAILLAGE=mesh,
    CREA_GROUP_MA=_F(NOM='TOUT',TOUT='OUI',),
    CREA_GROUP_NO=(_F(TOUT_GROUP_MA='OUI',),),
);

IMPR_RESU(FORMAT='MED', UNITE=71,  RESU=_F(MAILLAGE=mesh,),);

model=AFFE_MODELE(
    MAILLAGE=mesh,
    AFFE=(
        _F(
            GROUP_MA=('mast',),PHENOMENE='MECANIQUE',
            MODELISATION='POU_D_T',
        ),
        _F(
            GROUP_MA=('cblcy','shroud',),
            PHENOMENE='MECANIQUE',
            MODELISATION='CABLE',
        ),
    ),
);
```

Secondly, setting the temperature fields so as to pre-load the cables.

```
#putting pre-load in the vertical cables 'shroud' is enough
#the top cable 'cblcy' is pre-loaded by the shroud
#and the relatively low stiffness of the 'mast'
temper=CREA_CHAMP(
  MODELE=model,
    TYPE_CHAM='NOEU_TEMP_R',
    OPERATION='AFFE',
    AFFE=(
    #here we benefit of the Overwriting rule
    _F(TOUT='OUI',NOM_CMP='TEMP',VALE=0.,),
    _F(GROUP_MA=('shroud',),NOM_CMP='TEMP',VALE=-10.0,),
    ),
);

#from this field we create a thermal result
ltemp=DEFI_LIST_REEL(
    DEBUT=-1.0,
    INTERVALLE=_F(JUSQU_A=11.0,PAS=1.0,),
);

evtemp=CREA_RESU(
  TYPE_RESU='EVOL_VARC',
  NOM_CHAM='TEMP',
    OPERATION='AFFE',
    AFFE=(
        _F(CHAM_GD=temper,LIST_INST=ltemp,),
    ),
);
```

Note how we take advantage of the "Overwriting rule", described in chapter 6.5 , in superimposing a temperature of -10°C, on 'shroud' group only, after having set 'TOUT' at 0°C.

Thirdly, defining the materials and setting the element's properties.

```
steel=DEFI_MATERIAU(ELAS=_F(
    E=210000.,NU=0.3,RHO=8e-9,ALPHA=12e-6),
);

cablst=DEFI_MATERIAU(
    ELAS=_F(E=100000,NU=0.3,RHO=8e-9,ALPHA=12e-6,),
    CABLE=_F(EC_SUR_E=1.E-4,),
);

material=AFFE_MATERIAU(
    MAILLAGE=mesh,
    AFFE=(
        _F(GROUP_MA=('mast',), MATER=steel,),
        _F(
            GROUP_MA=('cblcy','shroud',),
            MATER=cablst,
        ),
```

```
    ),
    AFFE_VARC=_F(
        TOUT='OUI',
        NOM_VARC='TEMP',
        EVOL=evtemp,
        VALE_REF=0.0,
    ),
);

elemcar=AFFE_CARA_ELEM(
    MODELE=model,
    POUTRE=_F(
        GROUP_MA=('mast',),
        SECTION='RECTANGLE',
        CARA=('HY','HZ','EP',),
        VALE=(100, 50, 5,),
    ),
    #mast section is increased compared to other examples,
    #it would not withstand the load
    ORIENTATION=_F(
        GROUP_MA=('mast',),CARA='ANGL_VRIL', VALE=90.0,
    ),
    CABLE=_F(
        GROUP_MA=('cblcy','shroud',),
        N_INIT=10.0,
        SECTION=(10,),
    ),
);
```

Fourthly, setting boundary conditions and gravity load.

```
ground=AFFE_CHAR_MECA(
    MODELE=model,
    DDL_IMPO=(
        _F(
            GROUP_NO=('mastgrd',),
            DX=0,DY=0,DZ=0,DRX=0,DRY=0,DRZ=0,
        ),
        _F(GROUP_NO=('shrgrd',),DX=0,DY=0,DZ=0,),
    ),
);

selfwght=AFFE_CHAR_MECA(
    MODELE=model,
    PESANTEUR =_F(
        GRAVITE=10000,DIRECTION=(0,0,-1),
        GROUP_MA=('mast','cblcy','shroud',),
    ),
);
```

And know the part applying the cycling load, as a traveling point load along the 9 nodes previously created on the cable.

```
#in the next line we apply a vertical load of 100 N on each
#of the group of nodes
#(1 node in each group here) on the cable
#with a sawtooth style time stepping on the load
#so as to mimic a rolling load
#this is done in a Python loop, note mandatory indents in the loop
iter=9;
lc=[None]*(iter+1);
lcm=[None]*(iter+1);
for i in range (1,iter+1):
    grpno='cycl%02g' %i;
    lc[i]=AFFE_CHAR_MECA(
        MODELE=model,
        FORCE_NODALE=_F(GROUP_NO=(grpno,),FZ=-100.),
    );
    lcm[i]=DEFI_FONCTION(
        NOM_PARA='INST',
        VALE=(i-1.0, i,1, i+1.0,),
        PROL_GAUCHE='CONSTANT',
        PROL_DROITE='CONSTANT',
    );

selfw_m=DEFI_FONCTION(
    NOM_PARA='INST',
    VALE=(0.1, 10.1.),
    PROL_GAUCHE='CONSTANT',
    PROL_DROITE='CONSTANT',
);
```

Here is a block with some list definition, each one of them serves a special purpose:

- the first one, 'liste' is used as an argument by INCREMENT, the calculation[1] is performed at every single step described in this list, it may need to be adapted until the problem converges;
- the second one, 'listresu' is used as an argument by many IMPR_RESU, it tells which steps are printed in the results files, it is entirely arbitrary and has no influence on the problem solving, to reduce the size of the files this list should be restricted to useful values;
- the third one, 'listarchiv' is used as an argument by the keyword ARCHIVAGE, it tells which steps are saved in the 'data base', it has a direct influence on the 'data base' size and is entirely to the user's choice.

[1] A linear system solution.

```
liste=DEFI_LIST_REEL(
  #we start at the beginning of the preload
    DEBUT=-1.0,
    INTERVALLE=(
        _F(JUSQU_A=0.0,PAS=0.2,),
        _F(JUSQU_A=10.0,PAS=0.2,),
    ),
);

#we also want a more restricted list
#at which to print results
#only the INST not the NUME_ORDRE
from -1.0 as we want to see the preload
#and only up to 5 as the problem is symmetrical
listresu=DEFI_LIST_REEL(
    DEBUT=-1.0,
    INTERVALLE=_F(JUSQU_A=5,PAS=1.0,),
);

#the following list would restrict the instant
#written in the data base hence it's size
listarchivc=DEFI_LIST_REEL(
    DEBUT=-1.0,
    INTERVALLE=_F(JUSQU_A=10,PAS=1.0,),
);
```

Now, we create the cycling load case:

```
#Python loop to create the argument 'loadr' passed to 'EXCIT'
#loadr is actually a list, or a tuple!
#first the grounded DOF, then the gravity,
#and the various cycling load
loadr=[];
loadr.append(  _F(CHARGE=ground,),  );
loadr.append(  _F(CHARGE=selfwght,FONC_MULT=selfw_m,),  );
for i in range (1,iter+1):
    loadr.append(  _F(
        CHARGE=lc[i],
        TYPE_CHARGE='FIXE_CSTE',FONC_MULT=lcm[i],),
);
```

This is just one of the several available ways to create the argument 'loadr' in Python which is a rich language. A close look at the *.mess* file helps us to understand how*Code_Aster* translates this bit of Python code.

```
statnl=STAT_NON_LINE(
    MODELE=model,
    CHAM_MATER=material,
    CARA_ELEM=elemcar,
```

```
    EXCIT=loadr,
    COMP_INCR=_F(
        RELATION='ELAS',DEFORMATION='PETIT',
        GROUP_MA=('mast',),
    ),
    COMP_ELAS=_F(
        RELATION='CABLE',DEFORMATION='GROT_GDEP',
        GROUP_MA=('cblcy','shroud',),
    ),
    INCREMENT=_F(LIST_INST=liste,),
    NEWTON=_F(
        PREDICTION='TANGENTE',
        MATRICE='TANGENTE',
        REAC_ITER=1,
    ),
    RECH_LINEAIRE=_F(),
    CONVERGENCE=_F(
        RESI_GLOB_RELA=1e-4,
        ITER_GLOB_MAXI=300,
    ),
    ARCHIVAGE=_F(LIST_INST=listarchiv,),
);
```

```
statnl=CALC_CHAMP(
    reuse =statnl,
    RESULTAT=statnl,
    CONTRAINTE=(
        'SIEF_ELNO',
        'SIPO_ELNO',
    ),
    FORCE=('REAC_NODA'),
);
```

11.2.2 Commanding for results

We are know going to print some results in ASCII format, firstly the usual sum of reactions and individual reactions.

```
sum_reac=POST_RELEVE_T(
    ACTION=_F(
        INTITULE='sum reactions',
        GROUP_NO=('mastgrd','shrgrd',),
        RESULTAT=statnl,
        NOM_CHAM='REAC_NODA',
        RESULTANTE=('DX','DY','DZ'),
        OPERATION='EXTRACTION',
        LIST_INST=listresu,
        #another way to restrict the useful printing
        #INST=(0, 1, 2, 3, 4, 5, 6, 7,),
```

```
        #a good way to produce a lot of
        #maybe useless information
        #TOUT_ORDRE='OUI',
    ),
);
IMPR_TABLE (TABLE=sum_reac,)

IMPR_RESU(
    MODELE=model,
    FORMAT='RESULTAT',
    RESU=_F(
        NOM_CHAM='REAC_NODA',
        GROUP_NO=('mastgrd','shrgrd',),
        RESULTAT=statnl,
        LIST_INST=listresu,
    ),
);
```

Followed by some key values of forces.

```
IMPR_RESU(
    MODELE=model,
    FORMAT='RESULTAT',
    RESU=(
        _F(
            NOM_CHAM='SIEF_ELNO',
            GROUP_MA=('mast','shroud','cblcy',),
            RESULTAT=statnl,
            LIST_INST=listresu,
            VALE_MAX='OUI',VALE_MIN='OUI',
        ),
        _F(
            NOM_CHAM='SIEF_ELNO',
            GROUP_MA=('shroud',),
            RESULTAT=statnl,
            LIST_INST=listresu,
            VALE_MAX='OUI',VALE_MIN='OUI',
        ),
        _F(
            NOM_CHAM='SIEF_ELNO',
            GROUP_MA=('cblcy',),
            RESULTAT=statnl,
            LIST_INST=listresu,
            VALE_MAX='OUI',VALE_MIN='OUI',
        ),
    ),
);
```

And in a *.med* file.

```
IMPR_RESU(
    FORMAT='MED', UNITE=80,
    RESU=(
```

```
        _F(
            GROUP_MA=('mast','shroud','cblcy'),
            RESULTAT=statnl,
            NOM_CHAM=('DEPL','SIEF_ELNO',),
            LIST_INST=listresu,
        ),
        _F(
            GROUP_MA=('mast',),
            RESULTAT=statnl,
            NOM_CHAM=('SIPO_ELNO',),
            LIST_INST=listresu,
        ),
    ),
);
```

11.2.3 Creating time dependent plots

Up to know, we have only produced numerical or graphical result files with step by step results, in the next section, we produce a graphical plot of some results, displacement or forces, versus "time"[1].

This section must be read carefully and well understood as the production of such a plot is a rather low level code approach in *Code_Aster*. Yet it reveals a very powerful one!

Firstly, we extract the vertical displacement of the point in the middle of the cable, 'dz5' and at the point at one tenth of the length, 'dz1'.

```
#next lines are how to prepare and save a plot for XmGrace
#here we make a table with the displacement
#in z direction of the point 'cycl05'
#in the middle of the cable in function of the time
dz5=POST_RELEVE_T(
    ACTION=_F(
        OPERATION='EXTRACTION',
        INTITULE='displ_middle',
        RESULTAT=statnl,
        NOM_CHAM='DEPL',
        TOUT_ORDRE='OUI',
        GROUP_NO='cycl05',
        RESULTANTE=('DZ',),
        MOYE_NOEUD='NON',
    ),
    TITRE='displacement middle of cable',
);
#next line line to print it the .resu file, not really useful here,
#just to see what it looks like
IMPR_TABLE (TABLE=dz5,)
```

[1] With the same restriction as above.

```
#the same for point 1
dz1=POST_RELEVE_T(
    ACTION=_F(
        OPERATION='EXTRACTION',
        INTITULE='displ_near_end',
        RESULTAT=statnl,
        NOM_CHAM='DEPL',
        TOUT_ORDRE='OUI',
        GROUP_NO='cycl01',
        RESULTANTE=('DZ',),
        MOYE_NOEUD='NON',
    ),
    TITRE='displacement near end of cable',
);
```

Secondly, we extract the extreme values of the force in the top cable.

```
#here we prepare a table with the EXTREMA
#of the normal force in cable elements
forcec=POST_RELEVE_T(
    ACTION=_F(
        OPERATION='EXTREMA',
        INTITULE='force_in_cable',
        RESULTAT=statnl,
        NOM_CHAM='SIEF_ELNO',
        NOM_CMP='N',
        TOUT_ORDRE='OUI',
        GROUP_MA='cblcy',
    ),
    TITRE='force_in_cable',
);
#first print out
IMPR_TABLE (TABLE=forcec,)
```

Thirdly, we keep only the positive, tensional, values.

```
#then we restrict it to the maximum + tension value
forcec=CALC_TABLE(
    TABLE=forcec,reuse=forcec,
    ACTION=_F(
        OPERATION='FILTRE',NOM_PARA='EXTREMA',VALE_K='MAX',
    ),
    TITRE='max_tension_in_cable',
);
#second print out
IMPR_TABLE (TABLE=forcec,)
```

To get a nicer plot we want to scale these values, dividing them by ten.

```
#here we define a function, divide by ten
byten=FORMULE(NOM_PARA='VALE',VALE='VALE/10',),

#then we divide the tension value by ten for a nicer plot
forcec=CALC_TABLE(
    TABLE=forcec,reuse=forcec,
    ACTION=_F(
        OPERATION='OPER',
        FORMULE=byten,
        NOM_PARA='VALE10',
    ),
    TITRE='max_tension_in_cable/10',
);
#third print out
IMPR_TABLE (TABLE=forcec,)

##the same could be done with a single call to CALC_TABLE
#byten=FORMULE(NOM_PARA='VALE',VALE='VALE/10',),
#forcec=CALC_TABLE(
    #TABLE=forcec,reuse=forcec,
    #ACTION=(
        #_F(OPERATION='FILTRE',NOM_PARA='EXTREMA',VALE_K='MAX',),
        #_F(OPERATION='OPER',FORMULE=byten,NOM_PARA='VALE10',),
    #),
#);
```

And, we create the functions for the plot:

- to have `INST`, against `NUME_ORDRE` as abscissa of the XmGrace plot;

```
#here is a function that loads in a tuple:
#x = value time step instant, time = value time step instant
time=RECU_FONCTION(
    TABLE=dz5,  #could have been dz1
          #as the parameters are the same
    PARA_X='NUME_ORDRE',
    PARA_Y='INST',
);
```

- to have `DZ` for 'dz5', against' `NUME_ORDRE` as one ordinate of the XmGrace plot;

```
#here a function that loads in a tuple:
#x = value tnum order,
#deltaZ5 = Z displacement in the middle of the cable
deltaZ5=RECU_FONCTION(
  TABLE=dz5,
  PARA_X='NUME_ORDRE',
  PARA_Y='DZ',
```

```
);
```

- to have the same for 'dz1;

```
#the same near the end of the cable
deltaZ1=RECU_FONCTION(
  TABLE=dz1,
  PARA_X='NUME_ORDRE',
  PARA_Y='DZ',
);
```

- to have the tension in the cable as another ordinate of the XmGrace plot.

```
#the same for the tension in the cable
fNc=RECU_FONCTION(
  TABLE=forcec,
  PARA_X='NUME_ORDRE',
  PARA_Y='VALE10',
);
```

Finally, this next section assembles the whole plot, with some enhancements like titles, legends and scales[1].

```
#here we print these functions in an XmGrace format file
#we need an entry in ASTK for that file with LU=29,
#maybe extension .agr
IMPR_FONCTION(
    FORMAT='XMGRACE',
    UNITE=29,
    TITRE='displacement and force ',
    BORNE_X=(0,10),
    #restrict to nice ordinates
    BORNE_Y=(-80,100),
    #restrict to useful abscissa
    GRILLE_X=1,
    GRILLE_Y=10,
    LEGENDE_X='time (s)',
    LEGENDE_Y='displacement (mm) or force (N*10)',
    COURBE=(
        _F(FONC_X=time,FONC_Y=deltaZ5,LEGENDE='dZ5',),
        _F(FONC_X=time,FONC_Y=deltaZ1,LEGENDE='dZ1',),
        _F(FONC_X=time_num,FONC_Y=fNc,LEGENDE='tension',),
    ),
);

STANLEY()
```

[1] There could be many more arguments.

```
FIN()
```

11.2.4 Concluding about this command file

To sum it up what has to be noted in this file is the use of one single Python loop to create the loads , `lc[i]`, of the clown cycling along the cable.

With one line, within the loop, we are able to create 10 load cases!

Another loop appends all the load case in a single Python list, or tuple, `loadr`, used as an argument to `EXCIT` in `STAT_NON_LINE`.

Finally, we use the list `listresu` to limit the printing to the round numbered `INST`.

11.3 Viewing results

Figure 11.3 is a view of the displacement at `INST` 2 and 5, in Gmsh. To obtain the 2 superimposed views, we proceed like this:

- we make view[0] the active and only visible, set the proper parameters:
  - OPTIONS » Aspect » Displacement factor to 10;
  - button Min and Max in General tab;
  - Tools » Visibility with only line groups visible;
  - set Time step to 5;
- RMB on the arrow right of view[0], Alias » View with Options, this creates a view[3];
- return in view[0] and set Time step to 2;
- button Min and Max in General tab;
- in the Axes tab set Axes mode to Full grid, just to show a scaled bounding box [1].

[1] Of course this is not necessary at all.

We can animate one displacement, the classical 4 buttons (Rewind, Step backward, Play/Pause Step forward) are in the status bar at the bottom of the **Gmsh** window. We can even record this animation: in File > Save as... we can choose Format: Movie -MPEG (*.mpg) to record a movie with quite a few options available.

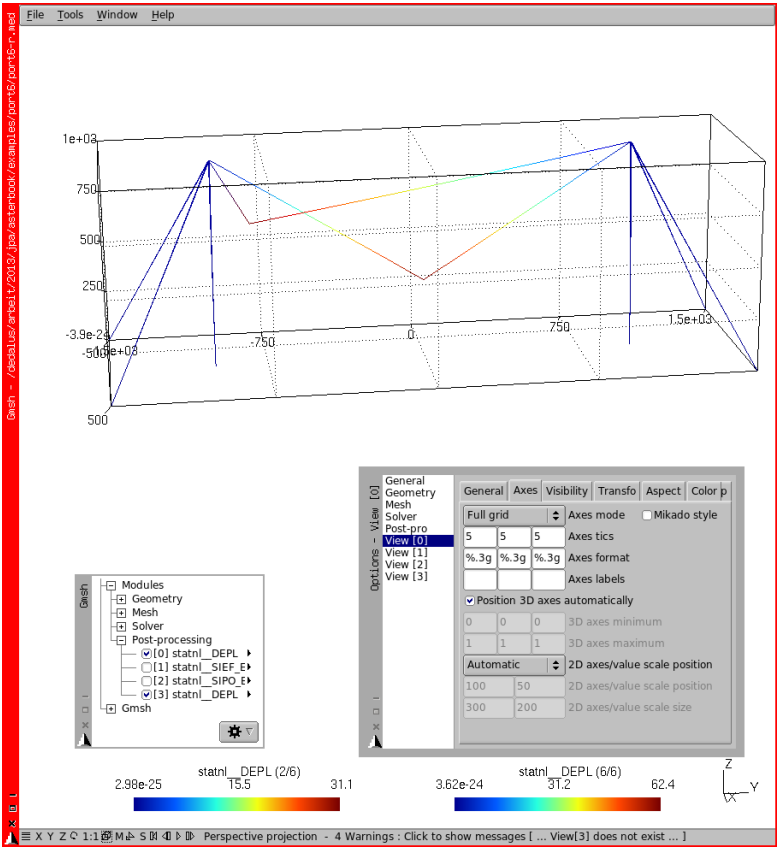


FIGURE 11.3: Deformed shape at INST 1 and 5

11.4 Plotting results with XmGrace

XmGrace is a GNU plotting program coming within the*Code_Aster* package[1].

If we run the study in ASTK and let the **STANLEY** window appear:

- in the first column on the left-hand side titled, in French, Champs, we choose DEPL;
- in the second column titled Composantes, we choose DZ;
- in the third column Entites Geometriques, we LMB click the title to toggle it to Courbes and choose cycl01 and cycl05, this is how we previously named the first and the middle nodes on which the clown is cycling.

We leave the last right column Ordres as it is to plot the curve on the whole set of orders, the **STANLEY** window looks like figure 11.4.

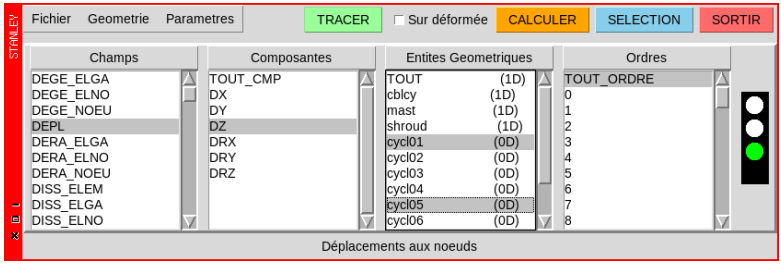


FIGURE 11.4: STANLEY ready for plotting in XmGrace

A gentle push on the TRACER button and the **XmGrace** window like figure 11.5 appears with the plot of the vertical displacement of the two nodes with respect to time or more precisely NUME_ORDRE as stated along the abscissa axis.

The strange appearance on the left hand side of the plots comes from the fact that the load does start later than NUME_ORDRE=0. To change

[1] More information is available here http://plasma-gate.weizmann.ac.il/Grace/

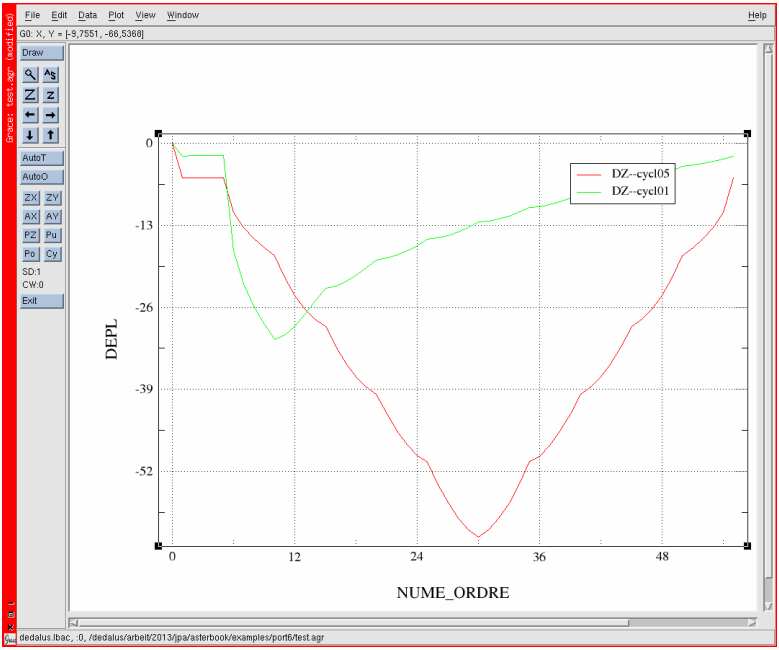


FIGURE 11.5: Plot in XmGrace within STANLEY

NUME_ORDRE to INST, which is more meaningful in our case, we have to RMB click the last column and select INST[1].

A click on the Stanley menu Geometrie > Ajout Point would have allowed us to plot on another point, created on the fly by giving its coordinates. If these coordinates do not point to a node in the structure the produced plot is meaningless!

[1] The very long list popping up shows us the vast possibilities of STANLEY in post-processing.

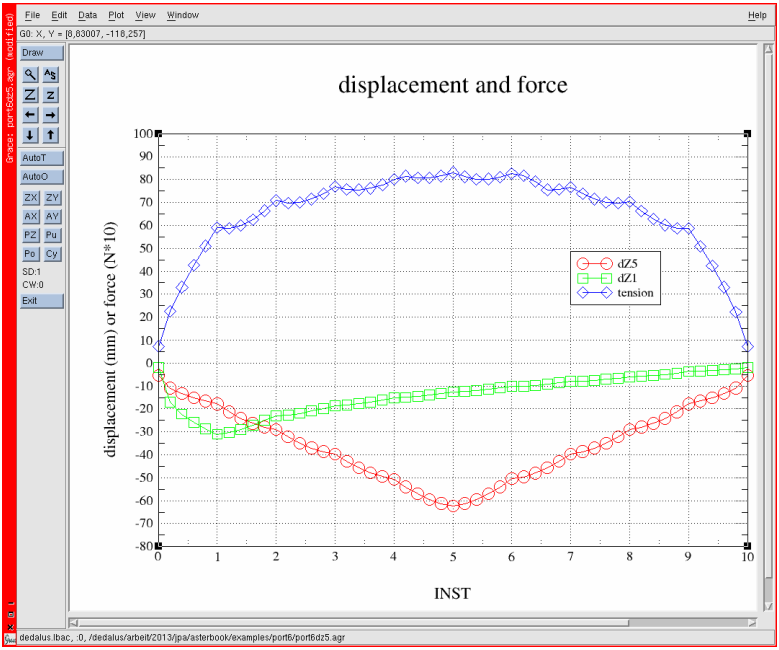


FIGURE 11.6: Combined plot

As we have saved an XmGrace file, in LU 29, with extension *.agr* we can also open it with XmGrace outside of STANLEY. This plot is shown in figure 11.6. The saved plot is somewhat more sophisticated than the one produced from STANLEY and may be customized even more. This is described in the operator IMPR_FUNCTION U4.33.01 as well as manual U2.51.01.

This plot shows at its bottom the displacement of the node in the middle of the cable (dZ5), in red, and the displacement of a node at 10% of the length of the cable (dZ1), in green, with respect to time. These displacements have of course a negative value. In blue, the value of the maximum tension in the cable is also displayed. Note: the force scale is multiplied by 10, so as to make a nicer plot.

11.5 Verifying some results

It is quite easy to do a quick check on the results, the formula for a cable with a point load in the middle is well known[1], we have:

$$\tan\theta - \sin\theta = \frac{W}{2EA}, \text{ or if } \theta < 12°, \theta = (\frac{W}{EA})^{\frac{1}{3}}$$

where θ is the slope of the cable at the ends, W the point load, E the cable's Young modulus, A its section and L its length, which gives here:

$$\theta = (\frac{100}{100000 \times 10})^{\frac{1}{3}} = 0.046rad = 2.6°$$

the vertical displacement being:

$$\delta = \frac{L}{2}\tan\theta = 46mm$$

compared to $62mm$ calculated by *Code_Aster*; and the tension in the cable:

$$T = \frac{W}{2\sin\theta} = 1078N$$

compared to $496N$ calculated by *Code_Aster*. The discrepancy is quite large but can be explained by the low pre-load in the 'shroud' cables with which the top of the masts move inward by $1.13mm$. Setting the temperature from $-10°$ to $-100°$ would give *Code_Aster* results almost like the formula.

However one of the real design problem in this type of cable vehicle may occur when the vehicle reaches the end of the cable the slope may become steep and the cable should not foul some part of the structure at this stage.

Here, we are self powered and when the clown reaches point 9 the deflection of the cable becomes $31mm$ which for a length to go of $200mm$ gives a slope of around 16%, we must ensure that this guy has got good legs and that the wheel does not slip on the cable!

[1] Again it can be found in [Roark], precisely in TABLE 12.

11.6 Working with tables

To build the entries for the XmGrace plot, we use several operators like POST_RELEVE_T or CALC_TABLE, let's have a closer look of what we actually do.

In the call to POST_RELEVE_T, we ask the code to extract the EXTREMA value of SIEF_ELNO component N, the tension, in the element forming the cycling cable.

As we can see in the following listing (commented as "#first print out" in the command file) we have 2 values, MAX or MIN at each NUME_ORDRE.

```
#force in cable
 INTITULE         RESU     NOM_CHAM         NUME_ORDRE
EXTREMA  MAILLE   NOEUD    CMP      VALE
 force in cable   statnl   SIEF_ELNO
0 MAX      M82      N2       N          0.00000E+00
 force in cable   statnl   SIEF_ELNO
0 MIN      M82      N2       N          0.00000E+00
 force in cable   statnl   SIEF_ELNO
0 MAX_ABS  M82      N2       N          0.00000E+00
 force in cable   statnl   SIEF_ELNO
0 MIN_ABS  M82      N2       N          0.00000E+00
 force in cable   statnl   SIEF_ELNO
1 MAX      M82      N2       N          4.99876E+02
 force in cable   statnl   SIEF_ELNO
1 MIN      M88      N14      N          4.63404E+02
 force in cable   statnl   SIEF_ELNO
1 MAX_ABS  M82      N2       N          4.99876E+02
 force in cable   statnl   SIEF_ELNO
1 MIN_ABS  M88      N14      N          4.63404E+02
.............................
```

MIN entry could have been negative, e.g. compressive, if we had not chosen a cable element.

In the first call to CALC_TABLE, we clean the table to keep only the MAX tensile value, resulting in the "#second print out" shown below. Only this MAX is retained under VALE.

```
#force in cable
#FILTRE -> NOM_PARA: EXTREMA          CRIT_COMP: EQ
VALE: ('MAX',)
 INTITULE         RESU     NOM_CHAM         NUME_ORDRE
EXTREMA  MAILLE   NOEUD    CMP      VALE
 force in cable   statnl   SIEF_ELNO
0 MAX      M82      N2       N          0.00000E+00
```

```
 force in cable   statnl   SIEF_ELNO
1 MAX      M82      N2       N          4.99876E+02
 ..............................
```

The value of the tensile force are so large compared to the displacement that the XmGrace plot would not look nice. That's why in the second call to CALC_TABLE, we add another column which contains the value of the tensile force divided by ten, we name this new column 'VALE10', and we use this one in the plot.

```
#force in cable
#FILTRE -> NOM_PARA: EXTREMA          CRIT_COMP: EQ
VALE: ('MAX',)
 INTITULE          RESU     NOM_CHAM          NUME_ORDRE
EXTREMA  MAILLE   NOEUD    CMP      VALE          VALE10
 force in cable   statnl   SIEF_ELNO
0 MAX      M82      N2       N          0.00000E+00   0.00000E+00
 ..............................
```

This is just an introduction to the many options in POST_RELEVE_T and CALC_TABLE, they are fully described U4.81.21 and U4.33.03, and we are dealing again with this matter in chapter 18.1 .

CHAPTER 12

Going solid, with contact, a bi-linear spring

In this chapter, with a 2 parts model, we cope with various problems:

- 3D solid meshing and modeling;
- with a quadratic mesh;
- with various options to link the two parts together: glued, glued with free rotation, contact and contact with friction;
- at the same time we run five analysis within a single command file;
- and we run the study without any post processing but saving the results in a base.

12.1 Introducing the study

In this chapter, we study the behavior of what can be seen as a bi-linear leaf spring[1], of rather simple design, shown in figure 12.2. One male part,

[1] The name "leaf" may not be very suitable, as this "leaf" is thicker than it is wide.

which we call 'part1' is a thick plate, in magenta, carrying a pin, in red, at the left-hand end, this 'part1' is solidly fixed at its right-hand end. Two female parts, 'part2' with a hole receiving the pin and solidly fixed at their left-hand end. There is a little radial play, 0.5 mm, of the pin within the hole.

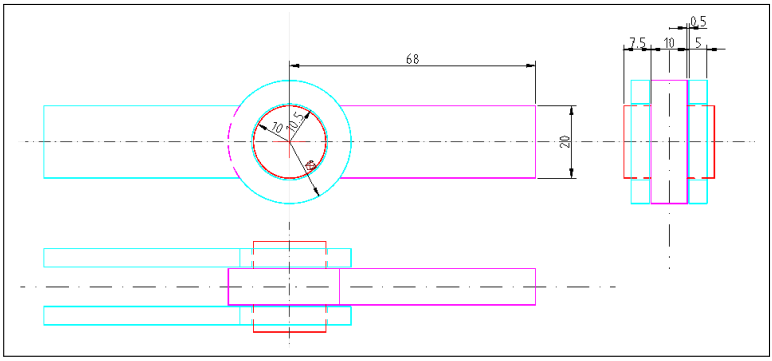


FIGURE 12.1: Dimensional sketch of bi-linear spring

We study the behavior of this model subjected to a vertical load on the pin, under various hypothesis:

1. 'part1' is not linked at all to 'part2' and takes the whole load;
2. both part are solidly 'glued' together around the pin and behave like one single continuous part taking the load;
3. relative rotation is freed around the pin, but not translation;
4. 'part1' moves down until the pin touches the hole face, and then both parts share the load;
5. the same as above, but with friction along the vertical faces, just like there was a nut on the pin, tightened as required, and in this case load sharing starts at the very beginning, before the pin comes in contact of the hole face, which may never happen if the tightening pressure is high enough.

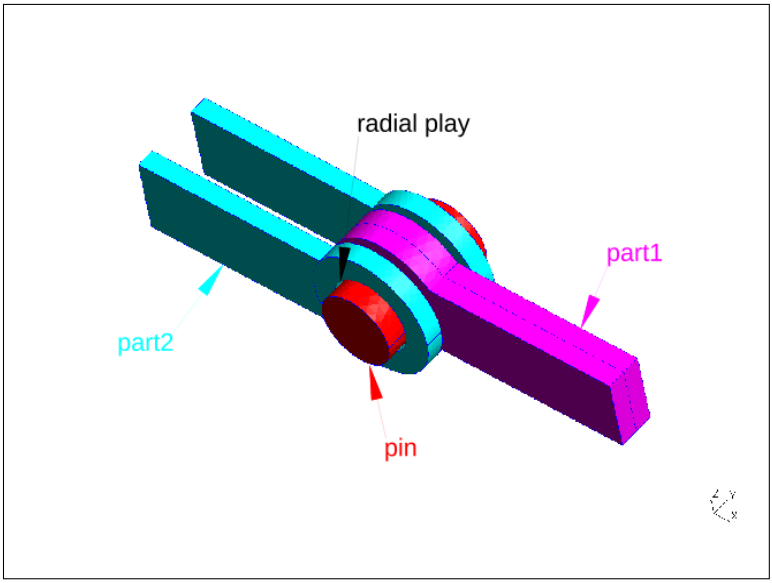


FIGURE 12.2: View of the full model (slightly exploded)

All this is carried at once, within a single run of *Code_Aster*. However we split the study in two parts:

1. putting the study together and solving;
2. independent post-processing.

For this kind of study the solution may take hours, while post-processing calculations are a matter of minutes. We calculate the solution, store it in a database, and call a post-processing *.comm* file to calculate and print out the results. This post-processing command file can be tailored without having to re-run the solution. This is known as `POURSUITE` in the *Code_Aster* jargon. We also see how to calculate the solution on a quadratic mesh but projecting the result on a linear[1] simplified mesh.

We also use the symmetry of the model, and the fact that the load acts in the plane of symmetry to mesh only half of the model.

[1] Here linear means non-quadratic as explained in chapter16.4 .

12.2 Meshing 'part1'

The *.geo* file for Gmsh is as below:

```
//'part1' geometry
cl1=2.5; //characteristic length, for meshing
t1=5; //thickness of half part
r1=10.0; //radius of pin
r2=17.0; //outside radius of eye

Point(1) = {0, -t1, 0, cl1}; //base point
Point(11) = {0, -t1, -r1, cl1};
Point(12) = {r1*Sin(2*Pi/3), -t1, r1+r1*Cos(2*Pi/3), cl1};
Point(13) = {r1*Sin(4*Pi/3), -t1, r1+r1*Cos(4*Pi/3), cl1};
Point(21) = {Sqrt(r2*r2-r1*r1), -t1, r1, cl1};
Point(22) = {-r2, -t1, 0, cl1};
Point(23) = {Sqrt(r2*r2-r1*r1), -t1, -r1, cl1};
Point(24) = {4*r2, -t1, r1, cl1};
Point(25) = {4*r2, -t1, -r1, cl1};
Circle(1) = {11, 1, 12};
Circle(2) = {12, 1, 13};
Circle(3) = {13, 1, 11};
Circle(4) = {21, 1, 22};
Circle(5) = {22, 1, 23};
Line(6) = {21, 24};
Line(7) = {23, 25};
Line(8) = {25, 24};
```

```
//extrusion of lines to create surfaces
//this line would have only extruded the geometry
//fix1s[]  = { Extrude {0, t1, 0} {Line{-8};} };
//this line line extrude the geometry and the associated
//mesh have 3 divisions along the extrusion length
fix1s[]  = { Extrude {0, t1, 0} {Line{-8};Layers {3};} };
edge1s[]  = { Extrude {0, t1, 0} {Line{6, -4, -5, -7};
    Layers {3};} };
pin1s[]  = { Extrude {0, -1.5*t1, 0} {Line{1, 2, 3};
    Layers {5};} };

//creation of plane surface
Line Loop(41) = {2, 3, 1};
Plane Surface(42) = {41};
Line Loop(43) = {6, -8, -7, -5, -4};
Plane Surface(44) = {-43, 41};

//extrusion of plane surfaces into volume
//as the bounding surface are layered
//it is not necessary to use layer for extruding the volume
part1v[]  = { Extrude  {0, t1, 0} {Surface{42, 44}; } };
pin1v[]  = { Extrude  {0, -1.5*t1, 0} {Surface{42};} };

// this point is used to compute and plot displacement
```

```
Point{30} In Surface{61};
//only the items necessary in the calculation are grouped

Physical Point("load1p") = {30};
Physical Surface("bear1s") = {44};
Physical Surface("sym1s") = {103};
Physical Surface("pin1s") = {pin1s[]};
Physical Surface("fix1s") = {fix1s[]};
Physical Surface("load1s") = {61};
Physical Volume("part1v") = {part1v[]};
Physical Volume("pin1v") = {pin1v[]};
```

```
//coloring of mesh
//all surfaces in Cyan
Color Cyan{
  Surface{12, 16, 20, 24, 28, 32, 36, 40, 42, 44, 52, 56,
      60, 61, 103, 120};
}
//then
Color Orange{ Surface{44}; }
Color Black{ Surface{61}; }
Color Green{ Surface{103}; }
Color Red{ Surface{pin1s[]}; }
Color Blue{ Surface{fix1s[]}; }
Color Magenta{ Volume {part1v[], pin1v[] };}
```

It is entirely parametric, with the parameters on the first lines. Once meshed it looks like figure 12.3. Note also how 'Surface' are created by 'Extrude' of 'Line', and 'Volume' by 'Extrude' of 'Surface'. The command 'Extrude' takes the argument 'Layers' to tell how many elements are created along the length of the extrusion at mesh time.

One remark: we divide the circle in three arcs as Gmsh cannot draw an arc whose angle is equal to or greater than π[1].

At the end of the file is the group building and naming, the last character stands as "v" for volume, "s" for surface and "p" for point.

The last lines are for coloring the mesh, which gives:

- green and black for the plane of symmetry;
- the load is applied onto 'load1s' which is black;
- the ground fixation is on 'fix1s' which is blue;

[1] This not a Gmsh restriction but basic geometry: given the center and 2 points there is an infinite number of arcs in 3D space if the three points are aligned (angle $= \pi$), and 2 arcs in the other case, the one with the angle $< \pi$ is drawn.

- orange for the plane which receives contact in the 'bolted' joint, 'bear1s';
- while the pin shows in red.

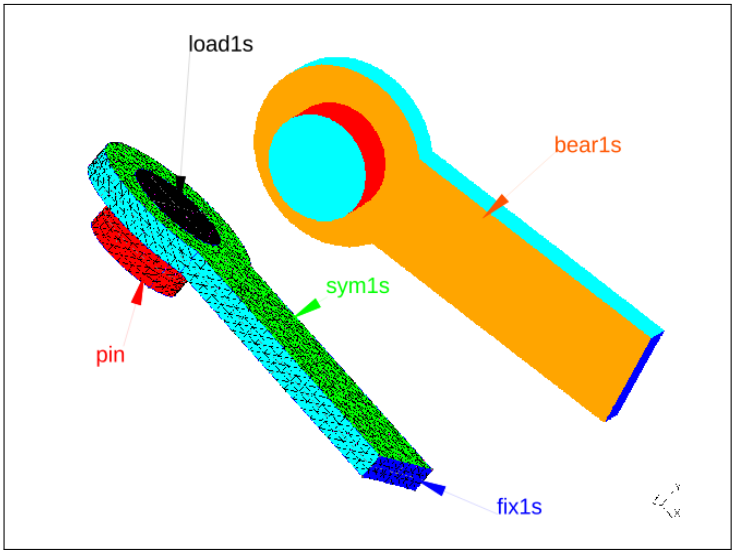


FIGURE 12.3: 'part1' meshed, with and without the element's' outline

12.3 Meshing 'part2'

The *.geo* file for Gmsh looks like this:

```
//'part2' geometry
cl1=2.0; //characteristic length, for meshing
off1=5.5; //y offset from part 1
t1=5; //thickness of half part
r1=10.5; //inside radius of eye
r2=17.0; //outside radius of eye

Point(1) = {0, -off1, 0, cl1};
Point(11) = {0, -off1, -r1, cl1};
Point(12) = {-r1*Sin(2*Pi/3), -off1, r1+r1*Cos(2*Pi/3), cl1};
Point(13) = {-r1*Sin(4*Pi/3), -off1, r1+r1*Cos(4*Pi/3), cl1};
Point(21) = {-Sqrt(r2*r2-r1*r1), -off1, r1, cl1};
```

```
Point(22) = {r2, -off1, 0, cl1};
Point(23) = {-Sqrt(r2*r2-r1*r1), -off1, -r1, cl1};
Point(24) = {-4*r2, -off1, r1, cl1};
Point(25) = {-4*r2, -off1, -r1, cl1};
Circle(1) = {11, 1, 12};
Circle(2) = {12, 1, 13};
Circle(3) = {13, 1, 11};
Circle(4) = {21, 1, 22};
Circle(5) = {22, 1, 23};
Line(6) = {21, 24};
Line(7) = {23, 25};
Line(8) = {25, 24};
Circle(9) = {21, 1, 23};
```

```
//extrusion of lines to create surfaces
fix2s[]  = { Extrude {0, -t1, 0} {Line{-8};Layers {3};} };
edge2s[]  = { Extrude {0, -t1, 0} {Line{6, -4, -5, -7, 9};
    Layers {3};} };
hole2s[]  = { Extrude {0, -t1, 0} {Line{-1, -2, -3};
    Layers {3};} };

//creation of plane surface
Line Loop(106) = {4, 5, -9};
Line Loop(107) = {2, 3, 1};
Plane Surface(108) = {106, 107};
Line Loop(109) = {6, -8, -7, -9};
Plane Surface(110) = {109};

//extrusion of plane surface into volume
part2v[]  = { Extrude  {0, -t1, 0} {Surface{-108, 110}; } };

Physical Point ("move2p") = {11};
Physical Surface("bear2s") = {108};
Physical Surface("hole2s") = {hole2s[]};
Physical Surface("fix2s") = {fix2s[]};
Physical Surface("pres2s") = {142};
Physical Volume("part2v") = {part2v[]};
```

```
color of mesh
//all surfaces in Cyan
Color Cyan{
   Surface{17, 21, 25, 29, 33, 37, 41, 45, 108, 110,
        142, 164};
}
//then
Color Orange{ Surface{108}; }
Color Purple{ Surface{142}; }
Color Black { Surface{hole2s[]}; }
Color Blue{ Surface{fix2s[]}; }
Color Magenta{ Volume {part2v[]};}
```

And the mesh is shown in figure 12.4 . Note: there is a gap of 0.5 mm in the Y direction in between the two parts.

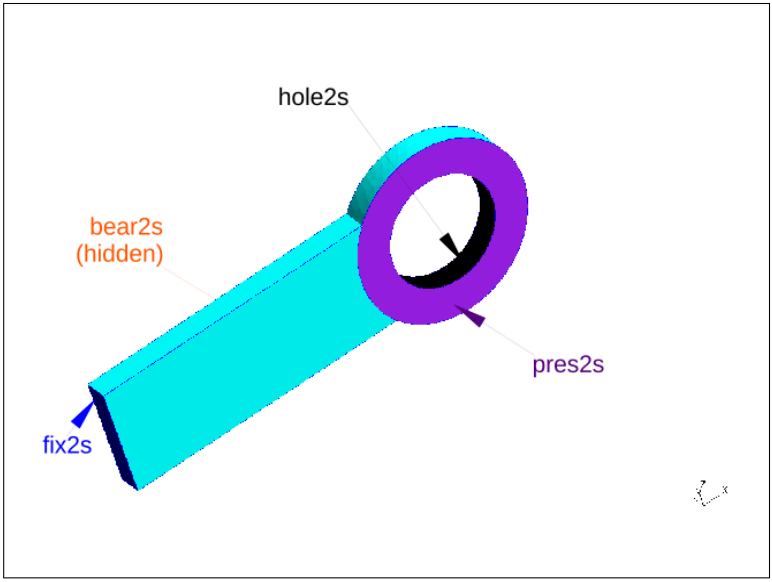


FIGURE 12.4: 'part2' meshed

The same remarks as for part1 apply:

- the ground fixation is on 'fix2s' which is blue;
- the face of the hole 'hole2s' is black;
- 'bear2s', the plane which applies contact in the "bolted" joint is orange, it is hidden on figure 12.4;
- while the "nut" pressure is applied on 'pres2s' which shows in purple.

We create a 3D quadratic mesh on both parts with Mesh » 3D and Mesh » Set order » 2. We keep these meshes quite coarse to get a solution in a relatively short time. Any serious industrial study, trying to optimize the part, particularly for stress, would, at first, show a more complicated shape, and require much finer a mesh in the strategic areas.

12.4 Commanding for the solution

We introduce the command file step by step.

12.4.1 Reading and manipulating the meshes

```
DEBUT();
#in group naming
#last char s stands for surface,
#v for volume and p for point
#last but one char stands for the part number

#we read the med file for each part
part1=LIRE_MAILLAGE(UNITE=20,FORMAT='MED',);
part2=LIRE_MAILLAGE(UNITE=21,FORMAT='MED',);

#U4.23.03
#we assemble the two meshes into a new one
mesh12=ASSE_MAILLAGE(
    MAILLAGE_1=part1,
    MAILLAGE_2=part2,
    OPERATION='SUPERPOSE',
);
```

We read the two meshes from to different Logical Units, we assemble them in one single new mesh with ASSE_MAILLAGE.

```
#re orient the normal for the face groups,
#with the Gmsh design as seen earlier this should be ok
#but one never knows
mesh12=MODI_MAILLAGE(
    reuse =mesh12,
    MAILLAGE=mesh12,
    ORIE_PEAU_3D=(
        _F(GROUP_MA='sym1s',),
        _F(GROUP_MA='pin1s',),
        _F(GROUP_MA='fix1s',),
        _F(GROUP_MA='hole2s',),
        _F(GROUP_MA='fix2s',),
        _F(GROUP_MA='load1s',),
    ),
);

#define groups of nodes on the new mesh
#should be here, not before,
#to compute and print a correct value for reaction
mesh12=DEFI_GROUP(
    reuse =mesh12,
```

```
    MAILLAGE=mesh12,

    #CREA_GROUP_MA=_F(NOM='TOUT',TOUT='OUI',),

    CREA_GROUP_NO=(
        _F(GROUP_MA='fix2s',),
        _F(GROUP_MA='fix1s',),
        _F(GROUP_MA='sym1s',),
        _F(GROUP_MA='load1p',),
        _F(GROUP_MA='move2p',),
    ),
);
```

Then, we reorient the normal to the surface elements to be sure they point in the same outward direction within each group. We create groups of node where necessary, for post processing, at the boundary conditions for example.

```
#make a copy of mesh12
qmesh=COPIER(CONCEPT=mesh12);

#convert the quadratic mesh  to linear mesh
#onto which the results will be projected
lmesh=CREA_MAILLAGE(
    MAILLAGE=qmesh,
    QUAD_LINE=_F(TOUT='OUI',),
);

#alternatively to run the problem on a linear mesh
#lmesh=COPIER(CONCEPT=qmesh);
#which should written like this in version 11.2 and earlier
lmesh=CREA_MAILLAGE(MAILLAGE=qmesh, COPIE=_F(),);

#to have a look at the different meshes
IMPR_RESU(FORMAT='MED', UNITE=71,  RESU=_F(MAILLAGE=qmesh,),);
IMPR_RESU(FORMAT='MED', UNITE=72,  RESU=_F(MAILLAGE=lmesh,),);

#make a quadratic model
qmod=AFFE_MODELE(
    MAILLAGE=qmesh,
    AFFE=_F(TOUT='OUI',PHENOMENE='MECANIQUE',MODELISATION='3D',),
);

#and a linear model
lmod=AFFE_MODELE(
    MAILLAGE=lmesh,
    AFFE=_F(TOUT='OUI',PHENOMENE='MECANIQUE',MODELISATION='3D',),
);

steel=DEFI_MATERIAU(ELAS=_F(E=2.1e5,NU=0.3,),);

mate=AFFE_MATERIAU(
    MAILLAGE=qmesh,
    AFFE=_F(TOUT='OUI',MATER=steel,),
```

```
);
```

Finally, we create the quadratic mesh, named 'qmesh', as a copy[1] of 'mesh12', make a model out of it and assign it a material.

We also create a linear mesh, named 'lmesh' onto which the the graphical results will be projected, with PROJ_CHAMP, and make a model out of it.

12.4.2 Setting the boundary conditions

We describe the boundary conditions and loadings which are used throughout the various calculations.

```
#set the boundary fixing for part1
#'encastre' on the right-hand end
#no Y displacement for the symmetrical model
fix1=AFFE_CHAR_MECA(
    MODELE=qmod,
    DDL_IMPO=(
        _F(GROUP_MA=('sym1s','load1s',),DY=0.0,),
        _F(GROUP_MA='fix1s',DX=0.0,DY=0.0,DZ=0.0,),
);

#set the boundary fixing for part2
#'encastre' on the left-hand end
fix2=AFFE_CHAR_MECA(
    MODELE=qmod,
    DDL_IMPO=_F(GROUP_MA='fix2s',DX=0.0,DY=0.0,DZ=0.0,),
);
```

This is straightforward and does not need any comment, however note that 3D elements do not bear rotational degrees of freedom, thus neither DRX, DRY nor DRZ.

12.4.3 Gluing the two parts around the pin

```
#u4.44.01
#we 'glue' volume 'part2' to the face of the pin 'pin1s'
glue=AFFE_CHAR_MECA(
    MODELE=qmod,
    LIAISON_MAIL=_F(
        GROUP_MA_MAIT='part2v',
        GROUP_MA_ESCL='pin1s',
```

[1] Copy the mesh is not necessary at all and is introduced here to illustrate its use.

```
    ),
);
```

The LIAISON_MAIL, allows to "glue together" two groups of elements. In this case, we ask for the GROUP_MA_ESCL='pin1s', the group of surface elements to be glued in translation and rotation to GROUP_MA_MAIT='part2v', the group of volume of all 'part2'[1]. Which part should be master and which one slave is explained in the documentation.

12.4.4 Relieving rotation around the pin

```
#same thing but only
#the displacement normal to the contact face are glued
#the two part can rotate one in each other
#and slide along the pin axis
#but their respective axis remain identical
#they cannot move one to each other in x and z directions
freerot=AFFE_CHAR_MECA(
    MODELE=qmod,
    #just as above but rotation is allowed
    LIAISON_MAIL=_F(
        GROUP_MA_MAIT='part2v',
        GROUP_MA_ESCL='pin1s',
        DDL_MAIT='DNOR',
        DDL_ESCL='DNOR',
    ),
);
```

To relieve the rotation, we use the keywords DDL_MAIT='DNOR' and DDL_ESCL='DNOR' in LIAISON_MAIL, this means: we allowing relative displacement, but the normal distance to the two surfaces remains constant.

12.4.5 Setting the contact conditions around the pin

```
#define contact in-between the pin and the hole
#as soon as the pin touches the hole surface
#it pushes the hole and part2 with it
contact=DEFI_CONTACT(
    MODELE=qmod,
    FORMULATION='DISCRETE',
    #'hole'overlaps 'pin' so should be 'MAIT'
```

[1] MAIT, "maître" in french stands for master and ESCL, "esclave", for slave.

```
    #or 'hole' is bigger than 'pin' so should be 'MAIT'
    #U2.04.04
    ZONE=_F(
        GROUP_MA_MAIT='hole2s',
        GROUP_MA_ESCL='pin1s',
    ),
);
```

Here, the master and slave are group of surface elements, with the same remarks about which is which[1]. There are many choices regarding the selection of a contact solver, FORMULATION, here again one should read the documentation, and try out!

Concerning contact, U2.04.04 is the first introductory document to be read.

12.4.6 Setting the contact conditions around the pin, with friction

This first part sets the contact with Coulomb friction.

```
contact5=DEFI_CONTACT(
  MODELE=qmod,
  FORMULATION='CONTINUE',
  FROTTEMENT='COULOMB',
    ZONE=(
      #first the same vertical contact as before
      #no friction COULOMB=0.0
      _F(
          GROUP_MA_MAIT='hole2s',
          GROUP_MA_ESCL='pin1s',
          COULOMB=0.0,
      ),
      #second the fiction on the vertical planes
      #rather low friction coefficient
      #COULOMB=0.1
      _F(
          GROUP_MA_MAIT='bear1s',
          GROUP_MA_ESCL='bear2s',
          COULOMB=0.1,
      ),
  ),
);
```

The difference between these two contact definitions must be noted:

[1] Here 'hole2s' which overlaps 'pin2s' should be master.

- as we want to allow friction in-between the two bodies `FORMULATION='CONTINUE', FROTTEMENT='COULOMB'` has to be used;
- although we use two contact `ZONE` they must share these same parameters;
- we set the frictional coefficient, `'COULOMB',` at 0.1, which is rather a low -slippery- value for steel on steel contact.

For friction to take place it is necessary to tighten the bolt, we do this below. We specify this tightening to be done from `INST` -5 to 0, which is before the vertical load is applied[1]. It stricto sensu is a pre-load.

We apply this load using `FORCE_FACE` applying a distributed load on the face of a 3D volume, the unit is force/area.

```
#pressure load to tighten the joint
pres=AFFE_CHAR_MECA(
    MODELE=qmod,
    FORCE_FACE=_F(GROUP_MA='pres2s',FY=12,),
);

#tightening bolt load steps for non-linear analysis
pres_m=DEFI_FONCTION(
    NOM_PARA='INST',VALE=(-5,0, 0,1),
    PROL_GAUCHE='CONSTANT',PROL_DROITE='CONSTANT',
);
```

This $12N/mm^2$ pressure applied on a $\pi(17.5^2 - 10^2) = 562mm^2$ area gives a total load of $6739N$, which multiplied by the frictional coefficient gives a vertical resisting force of $674N$ which in turn is about 8.6% of the ultimate vertical load $7854N$[2].

12.4.7 Setting the vertical load

```
#vertical load on the pin in part1 on the plane of symmetry
load=AFFE_CHAR_MECA(
  MODELE=qmod,
  FORCE_FACE=_F(GROUP_MA='load1s',FZ=-25.0,);
  #total force is 25*pi*10^2=7854
```

[1] Just as in true life, the bolt would be tightened before loading the spring.

[2] This very modest friction will not restrain displacement very much, except at the beginning of the loading, as we can see at solution time.

```
);

#load steps for non-linear analysis
load_m=DEFI_FONCTION(
    NOM_PARA='INST',VALE=(0,0, 5,1),
    PROL_GAUCHE='CONSTANT',PROL_DROITE='CONSTANT',
);
```

The load is set as a force per unit of area on the group 'load1s', together with a ramp for the non-linear contact solution.

12.4.8 Setting for the five solutions

We solve the study, and later, post-process the results within a Python loop.

We initialize the loop used in the post-precessing and set the 5 solution parameters.

```
#iter is the number of solutions
iter=5;
result=[None]*(iter+1);
```

For case 1, 'part1' is free from 'part2' and is loaded alone.

```
#case one
#part1 alone with the load
#no 'LIAISON' nor contact with part2
result[1]=MECA_STATIQUE(
    MODELE=qmod,
    CHAM_MATER=mate,
    EXCIT=(
        _F(CHARGE=fix2,),
        _F(CHARGE=fix1,),
        _F(CHARGE=load,),
    ),
);
```

For case 2, 'part1' and 'part2' are solidly glued together.

```
#case two
#part1 and part2 glued together
result[2]=MECA_STATIQUE(
    MODELE=qmod,
    CHAM_MATER=mate,
    EXCIT=(
        _F(CHARGE=fix2,),
```

```
        _F(CHARGE=fix1,),
        _F(CHARGE=glue,),
        _F(CHARGE=load,),
    ),
);
```

For case 3, rotation is freed around 'part1' and 'part2'.

```
#case three
#part1 and part2 free to rotate at joint
result[3]=MECA_STATIQUE(
    MODELE=qmod,
    CHAM_MATER=mate,
    EXCIT=(
        _F(CHARGE=fix2,),
        _F(CHARGE=fix1,),
        _F(CHARGE=freerot,),
        _F(CHARGE=load,),
    ),
);
```

For case 4, 'part1' comes in contact with 'part2'.

```
#case four
#part1  coming into contact with part2
linst=DEFI_LIST_REEL(
    DEBUT=0.0,
    INTERVALLE=_F(JUSQU_A=5.0,PAS=1.0,),
);

result[4]=STAT_NON_LINE(
    MODELE=qmod,
    CHAM_MATER=mate,
    EXCIT=(
        _F(CHARGE=fix2,),
        _F(CHARGE=fix1,),
        _F(
          CHARGE=load,
          TYPE_CHARGE='FIXE_CSTE',FONC_MULT=load_m,
         ),
    ),
    CONTACT=contact,
    COMP_INCR=_F(
        RELATION='ELAS',DEFORMATION='PETIT',
        GROUP_MA=('pin1v','part1v','part2v',),
    ),
    INCREMENT=_F(LIST_INST=linst,),
    NEWTON=_F(
        PREDICTION='TANGENTE',
        MATRICE='TANGENTE',
        REAC_ITER=1,
    ),
```

```
    CONVERGENCE=_F(RESI_GLOB_RELA=1e-4,ITER_GLOB_MAXI=30,),
);
```

For case 5, we add some friction along the mating vertical surfaces. For this we need to perform the calculation from the beginning of the pre-load, so we create a new 'listinst'.

```
linst5=DEFI_LIST_REEL(
    DEBUT=-5.0,
    INTERVALLE=(
    _F(JUSQU_A=0.0,PAS=5.0,),
    _F(JUSQU_A=3.75,PAS=1.25,),
    _F(JUSQU_A=5.0,PAS=0.625/2,),
  ),
);
```

This rather sophisticated list of steps does not come from any devilish theory but from trial and error, looking at the convergence table in the *.mess* file:

- it very strongly depends on the pressure load 'pres', values of 10 or 15 require very different stepping schemes;
- it depends also on the mesh, the set given here works for the linear mesh;
- sometimes the solution is reached with largish steps like `JUSQU_A=0.0,PAS=5.0` in one single step, for the pressure load, in this example[1] .

```
#case five
#part1  coming in contact with part2
#friction taken into account
result[5]=STAT_NON_LINE(
    MODELE=qmod,
    CHAM_MATER=mate,
    EXCIT=(
        _F(CHARGE=fix1,),
        _F(CHARGE=fix2,),
        _F(
            CHARGE=pres,
            TYPE_CHARGE='FIXE_CSTE',FONC_MULT=pres_m,
        ),
        _F(
```

[1] Which, according to *Code_Aster* guru Thomas de Soza: "That does not surprise me, tangential contact should be avoided as it raises an indeterminacy. By applying the pre-load in one step one sets a sheer contact".

```
            CHARGE=load,
            TYPE_CHARGE='FIXE_CSTE',FONC_MULT=load_m,
        ),
    ),
    CONTACT=contact5,
    COMP_INCR=_F(
        RELATION='ELAS',DEFORMATION='PETIT',
        GROUP_MA=('pin1v','part1v','part2v',),
    ),
    INCREMENT=_F(LIST_INST=linst5,),
    NEWTON=_F(
        PREDICTION='TANGENTE',
        MATRICE='TANGENTE',
        REAC_ITER=1,
    ),
    CONVERGENCE=_F(RESI_GLOB_RELA=1e-4,ITER_GLOB_MAXI=50,),
);

STANLEY();

FIN();
```

We have created a tuple holding five 'result[]'. `STANLEY()` call is here so we can have a first look at the results.

With minor alterations, we could have also run the problem from a linear mesh built in Gmsh. Moreover in some more complicated problems some groups can be quadratic while the rest of the mesh is linear.

12.5 Running the study

Now, we can launch the study within ASTK not forgetting to create a 'base', where to save the results[1].

[1] At this stage 'R' (result) should be ticked, not 'D' (data) as we create the base for the first time , 'C' for compressed may be ticked.

CHAPTER 13

Post-processing the spring

In this chapter, we write a command file to post-process the base created in chapter 12.

We project the results of the calculation on a simpler linear mesh.

We look at the results and create some time dependent plots.

All with a significant use of Python.

13.1 Commanding for Post-processing

13.1.1 Preliminaries

We do the post-processing in `POURSUITE`, that is to say: we read the database previously built and enhance the results it contains. This way we can tailor the post-processing *.comm* file until we get a satisfactory set of information without having to re-run the study.

```
#U4.11.03
POURSUITE(
  #may be required in more complex problems,
  #not here
  #PAR_LOT='NON',
);

#some definition for solution loop
#qres is the tuple for the quadratic solutions
#sr1 and sr2 are the tuples for the sums
#of reaction at the fixing of part1 and part2
#and sr12 the sum of both
qres=[None]*(iter+1);
sr1=[None]*(iter+1);
sr2=[None]*(iter+1);
sr12=[None]*(iter+1);
```

While a normal command file starts with `DEBUT()`, here we start with `POURSUITE()`. Then, we initialize some tuples to hold the results.

And we start the post-processing inside the loop over the result[].

```
for i in range (1,iter+1):
  result[i]=CALC_CHAMP(
    reuse =result[i],
    RESULTAT=result[i],
    CONTRAINTE='SIGM_ELNO',
    FORCE='REAC_NODA',
    CRITERES=(
      #'SIEQ_ELNO',
      'SIEQ_NOEU',
      #'SIEQ_ELGA',
    ),
  );
```

Printing displacement of typical loaded node, and stresses in elements, at nodes.

```
  IMPR_RESU(
    FORMAT='RESULTAT',
    RESU=(
      _F(
        RESULTAT=result[i],
        GROUP_NO=('load1p',),
        NOM_CHAM='DEPL',NOM_CMP=('DZ',),
        FORMAT_R='1PE12.3',
      ),
      _F(
        RESULTAT=result[i],
        NOM_CHAM='SIEQ_NOEU',NOM_CMP=('VMIS',),
        FORMAT_R='1PE12.3',
```

```
        VALE_MAX='OUI',
      ),
      _F(
        RESULTAT=result[i],
        GROUP_NO=('fix1s',),
        NOM_CHAM='SIEQ_NOEU',NOM_CMP=('VMIS',),
        FORMAT_R='1PE12.3',
        VALE_MAX='OUI',
      ),
    ),
  );
```

And reactions at end supports.

```
    #computing and printing the sum of reaction
    #for part1 and part2
    sr1[i]=POST_RELEVE_T(
        ACTION=_F(
            INTITULE='reac1',
            GROUP_NO=('fix1s',),
            RESULTAT=result[i],
            NOM_CHAM='REAC_NODA',TOUT_ORDRE='OUI',
            RESULTANTE=('DX','DY','DZ'),
            OPERATION='EXTRACTION',
        ),
    );
    IMPR_TABLE (TABLE=sr1[i],)
    sr2[i]=POST_RELEVE_T(
        ACTION=_F(
            INTITULE='reac2',
            GROUP_NO=('fix2s',),
            RESULTAT=result[i],
            NOM_CHAM='REAC_NODA',TOUT_ORDRE='OUI',
            RESULTANTE=('DX','DY','DZ'),
            OPERATION='EXTRACTION',
        ),
    );
    IMPR_TABLE (TABLE=sr2[i],)
    sr12[i]=POST_RELEVE_T(
        ACTION=_F(
            INTITULE='reac12',
            GROUP_NO=('fix1s','fix2s',),
            RESULTAT=result[i],
            NOM_CHAM='REAC_NODA',TOUT_ORDRE='OUI',
            RESULTANTE=('DX','DY','DZ'),
            OPERATION='EXTRACTION',
        ),
    );
    IMPR_TABLE (TABLE=sr12[i],)
```

13.1.2 Creating the MED result file

```
#project the quadratic model qmod result[i]
#onto the low definition linear model lmod
#name this result qres[i]
qres[i]=PROJ_CHAMP(
    RESULTAT=result[i],MODELE_1=qmod,MODELE_2=lmod,
);

IMPR_RESU(
    FORMAT='MED',
    RESU=(
        _F(RESULTAT=qres[i],NOM_CHAM=('DEPL',),),
        _F(RESULTAT=qres[i],NOM_CHAM=('SIEF_ELNO',),),
        _F(
            RESULTAT=qres[i],
            NOM_CHAM=('SIEQ_NOEU',),NOM_CMP=('VMIS',),
        ),
        #_F(
            #RESULTAT=qres[i],
            #whatever else we want
        #),
    ),
);
#here ends the Python loop
```

Projecting the quadratic mesh onto the original mesh and printing results in a a med file.

13.1.3 Creating a plot of some results

In his section, we look in details at case 4 and 5 with contact. We want to see what happens to the displacement of the pin, at what time it makes contact with the hole and what are the reactions for each part. This is done in a XmGrace readable graphic created here[1].

```
#here we define a function, multiply by 10000
by10000=FORMULE(NOM_PARA='DZ',VALE='DZ*10000',),

and initialize some tuplez
dz1=[None]*(iter+1);
dz2=[None]*(iter+1);
rz1=[None]*(iter+1);
rz2=[None]*(iter+1);
rz12=[None]*(iter+1);
loadt=[None]*(iter+1);
```

[1] However we create almost empty plot values for case 1 to 3 in the same time.

```
dtaZ1=[None]*(iter+1);
dtaZ2=[None]*(iter+1);
rcZ1=[None]*(iter+1);
rcZ2=[None]*(iter+1);
```

These tuples, if used as an argument in XmGrace, should be no more than 8 characters long, including the trailing "_" plus the index, this is not much!

In this first part, we extract the displacement, in the global Z direction, of node 'load1p' on 'part1'.

```
for i in range (1,iter+1):
  #next lines are how to prepare and save a plot for XmGrace
  #one on each case
  #here we make a table with the displacement
  #in z direction of the point 'load1p'
  #in the centre of the pin on the plane of symmetry
  dz1[i]=POST_RELEVE_T(
    ACTION=_F(
      OPERATION='EXTRACTION',
      INTITULE='displ_part1',
      RESULTAT=qres[i],
      NOM_CHAM='DEPL',
      TOUT_ORDRE='OUI',
      GROUP_NO='load1p',
      RESULTANTE=('DZ',),
      MOYE_NOEUD='NON',
    ),
    TITRE='displacement of pin center',
  );
  #print in
  IMPR_TABLE (TABLE=dz1[i],)

  dz1[i]=CALC_TABLE(
    TABLE=dz1[i],reuse=dz1[i],
    ACTION=_F(
      OPERATION='OPER',FORMULE=by10000,NOM_PARA='Z10000',
    ),
  );
  IMPR_TABLE (TABLE=dz1[i],)
```

In this second part, we do the same for node 'move2p' on 'part2'.

```
  #here we make a table with the displacement
  #in z direction of the point 'move2p'
  #on top of part2 above the hole
  dz2[i]=POST_RELEVE_T(
    ACTION=_F(
      OPERATION='EXTRACTION',
      INTITULE='displ_part2',
      RESULTAT=qres[i],
```

```
      NOM_CHAM='DEPL',
      TOUT_ORDRE='OUI',
      GROUP_NO='move2p',
      RESULTANTE=('DZ',),
      MOYE_NOEUD='NON',
    ),
    TITRE='displacement of part2 above pin',
  );
  #print in
  IMPR_TABLE (TABLE=dz2[i],),

  dz2[i]=CALC_TABLE(
    TABLE=dz2[i],reuse=dz2[i],
    ACTION=_F(
      OPERATION='OPER',FORMULE=by10000,NOM_PARA='Z10000',
    ),
  );
  IMPR_TABLE (TABLE=dz2[i],),
IMPR_TABLE (TABLE=dz2,)
```

And in the third part, we extract the reactions at the ends of 'part1' and 'part2' as well as their sums.

```
  rz1[i]=POST_RELEVE_T(
    ACTION=_F(
      OPERATION='EXTRACTION',
      INTITULE='reaction 1',
      RESULTAT=qres[i],
      NOM_CHAM='REAC_NODA',
      TOUT_ORDRE='OUI',
      GROUP_NO='fix1s',
      RESULTANTE=('DZ',),
      MOYE_NOEUD='NON',
    ),
    TITRE='reaction 1',
  );


  rz2[i]=POST_RELEVE_T(
    ACTION=_F(
      OPERATION='EXTRACTION',
      INTITULE='reaction 2',
      RESULTAT=qres[i],
      NOM_CHAM='REAC_NODA',
      TOUT_ORDRE='OUI',
      GROUP_NO='fix2s',
      RESULTANTE=('DZ',),
      MOYE_NOEUD='NON',
    ),
    TITRE='reaction 2',
  );

  rz12[i]=POST_RELEVE_T(
    ACTION=_F(
```

```
      OPERATION='EXTRACTION',
      INTITULE='reaction 12',
      RESULTAT=qres[i],
      NOM_CHAM='REAC_NODA',
      TOUT_ORDRE='OUI',
      GROUP_NO=('fix1s','fix2s',),
      RESULTANTE=('DZ',),
      MOYE_NOEUD='NON',
    ),
    TITRE='reaction 1+2',
);
```

Then, we create some functions of the values we want to plot with respect to INST.

```
#here a function that loads in a tuple:
#x = value time step instant,
#loadt = applied load, as sum of reactions
loadt[i]=RECU_FONCTION(
  TABLE=rz12[i],
  PARA_X='INST',
  PARA_Y='DZ',
);
#here a function that loads in a tuple:
#x = value time step instant,
#deltaZ1 = Z displacement of part 1
dtaZ1[i]=RECU_FONCTION(
  TABLE=dz1[i],
  PARA_X='INST',
  PARA_Y='Z10000',
);

dtaZ2[i]=RECU_FONCTION(
  TABLE=dz2[i],
  PARA_X='INST',
  PARA_Y='Z10000',
);

rcZ1[i]=RECU_FONCTION(
  TABLE=rz1[i],
  PARA_X='INST',
  PARA_Y='DZ',
);

rcZ2[i]=RECU_FONCTION(
  TABLE=rz2[i],
  PARA_X='INST',
  PARA_Y='DZ',
);
```

To create the plots of displacements and reactions for case 4 and 5 in XmGrace.

```
  #here we print these functions in  XmGrace format files
  #numbering is such than case 4 is in LU 29, case 5 in LU 30
  #we have to have  entry in ASTK for these files
  #type dat in ASTK maybe extension .agr
  IMPR_FONCTION(
    FORMAT='XMGRACE',
    UNITE=25+i,
    TITRE='displacement and reaction ',
    BORNE_X=(0,8000),
    BORNE_Y=(-10000,8000),
    GRILLE_X=(1000),
    GRILLE_Y=(1000),
    LEGENDE_X='applied load (N)',
    LEGENDE_Y='displacement (mm/10000) or force (N)',
    COURBE=(
      _F(FONC_X=loadt[i],FONC_Y=dtaZ1[i],LEGENDE='dz1',),
      _F(FONC_X=loadt[i],FONC_Y=dtaZ2[i],LEGENDE='dz2',),
      _F(FONC_X=loadt[i],FONC_Y=rcZ1[i],LEGENDE='reacz1',),
      _F(FONC_X=loadt[i],FONC_Y=rcZ2[i],LEGENDE='reacz2',),
    ),
  );
  #end of loop
```

One remark here: IMPR_FONCTION is executed for case 1 to 5 but files are created in ASTK only for case 4 and 5 (LU=29 and LU=30), see figure 13.1, so only these last two cases are printed, this is not very optimized!

And the next part makes a plot of the displacement of 'load1p' node for all five cases.

```
IMPR_FONCTION(
  FORMAT='XMGRACE',
  UNITE=31,
  TITRE='compare displacement',
  BORNE_X=(0,8000),
  BORNE_Y=(0,-10000),
  GRILLE_X=(1000),
  GRILLE_Y=(1000),
  LEGENDE_X='applied load (N)',
  LEGENDE_Y='compared displacement (mm/10000) or force (N)',
  COURBE=(
    _F(FONC_X=loadt[1],FONC_Y=dtaZ1[1],LEGENDE='dz1, case 1',),
    _F(FONC_X=loadt[2],FONC_Y=dtaZ1[2],LEGENDE='dz1, case 2',),
    _F(FONC_X=loadt[3],FONC_Y=dtaZ1[3],LEGENDE='dz1, case 3',),
    _F(FONC_X=loadt[4],FONC_Y=dtaZ1[4],LEGENDE='dz1, case 4',),
    _F(FONC_X=loadt[5],FONC_Y=dtaZ1[5],LEGENDE='dz1, case 5',),
  ),
);
```

And finally, we need to kill the previously created concepts. If we want to run this command file later on we will have to write them again, as we save the base we have to make allowance for writing them again.

```
#we have to kill the CONCEPT if we want to run this command file
#more than one time
#first the ones lying within a loop
for i in range (1,iter+1):
  DETRUIRE(CONCEPT=_F(NOM=(sr1[i],sr2[i],sr12[i],qres[i],),),);
  DETRUIRE(CONCEPT=_F(NOM=(dz1[i],dz2[i],rz1[i],rz2[i],),),);
  DETRUIRE(CONCEPT=_F(NOM=(rz12[i],loadt[i],),),);
  DETRUIRE(CONCEPT=_F(NOM=(dtaZ1[i],dtaZ2[i],rcZ1[i],rcZ2[i],),),);
#note the Python indent

#then the one outside of a loop
DETRUIRE(CONCEPT=_F(NOM=by10000),);

STANLEY();

FIN();
```

13.2 Running the post processing

Figure 13.1 is a view of the ASTK setup to run the post-processing.

We use the same database as previously which this time is set for 'D' data. and 'R' results.

13.3 Viewing deformed shape for all cases

Here are the views, in Gmsh, of the deformed shape for all cases, at the last instant for the non linear cases, the first three of them with a displacement factor of 5. The two last ones, with contact, are shown with a a displacement factor of 1. As using a factor different from 1, in these cases, for the displacements will lead to an erroneous picture, e.g. here a displacement of 1 mm of 'part1' induces a displacement of 0.5 mm only on part 2 [1]!

Finally the views are from a quadratic mesh computation except the view for case 5 coming from a linear one.

[1] This is an illustration of the non linear behavior in case of contact (unilateral boundary condition).

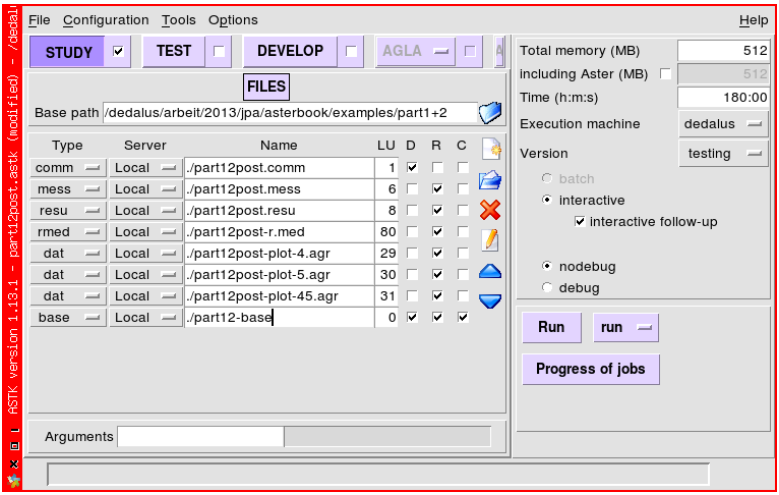


FIGURE 13.1: ASTK windows for Post-processing

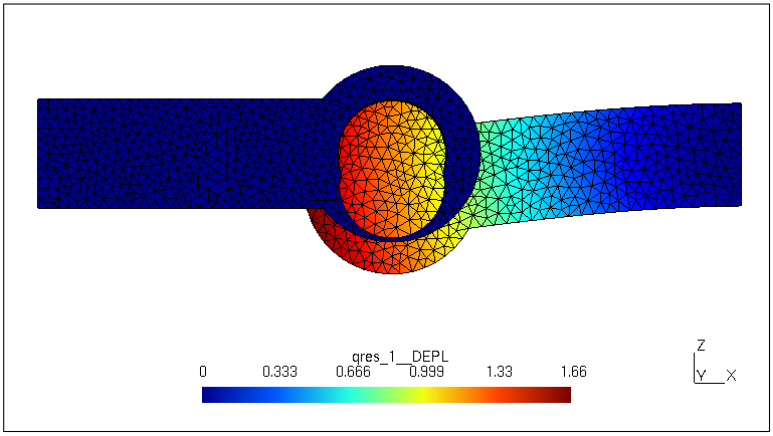


FIGURE 13.2: 'part1' loaded on its own is the only deformed part, it is penetrating inside 'part2'

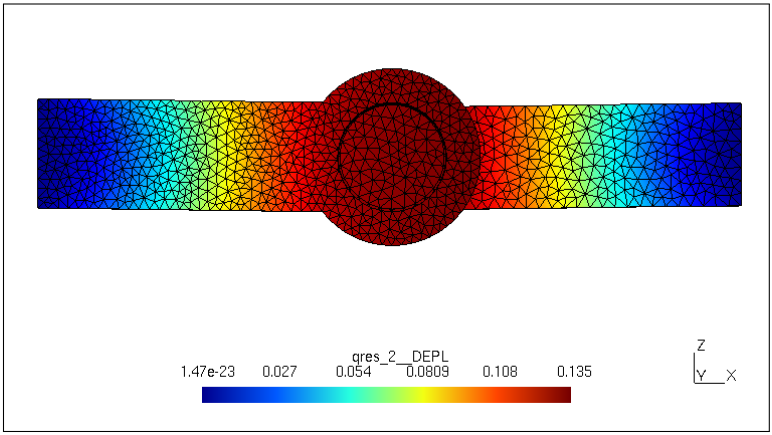


FIGURE 13.3: 'part1' is "glued" 'to part 2'

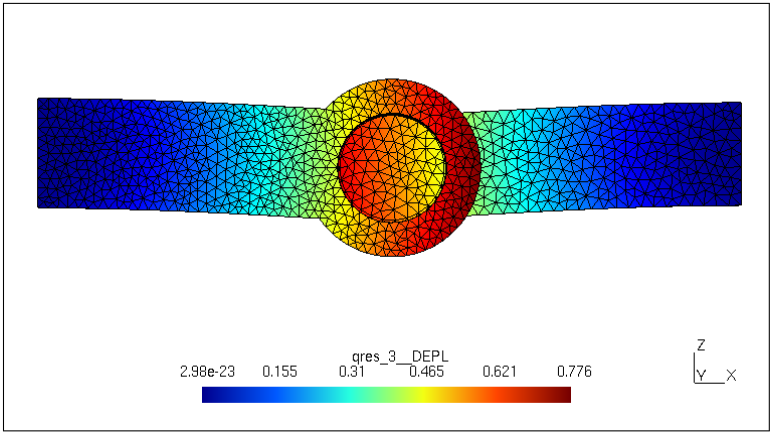


FIGURE 13.4: Rotation is freed around the pin

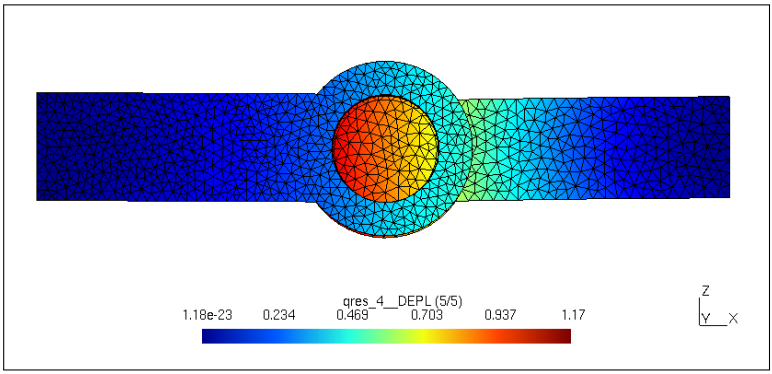


FIGURE 13.5: 'part1' comes in contact with 'part2' (displacement factor = 1)

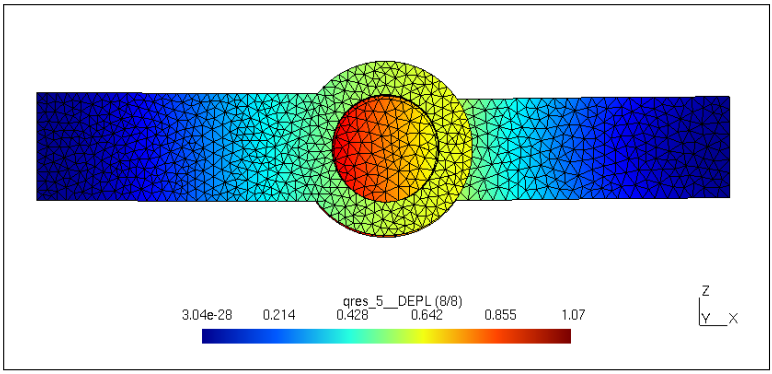


FIGURE 13.6: Here with some friction (displacement factor = 1, calculation from a linear mesh)

13.4 Numerical results

case	sol type	max dspl, dZ1	reac, 'part1'	reac, 'part2'
1	quadratic	-1.23	7854	0
2		-0.13	3802	4052
3		-0.57	3663	4191
4		-0.87	5536	2318
5		NA	NA	NA
1	linear	-1.16	7783	0
2		-0.13	3783	4000
3		-0.46	3692	4091
4		-0.82	5515	2268
5		-0.79	5520	2264

We should note that the displacement given above is the vertical displacement of the node 'load1p' at the center of the loaded area in 'part1' while the previous screen views show the displacement vector, everywhere in the model, this is not exactly the same thing.

As we pointed out in 12.4.6 , the estimated value of the frictional vertical resisting force is about 8.6% of the vertical load, we can see in the above 'linear mesh' results that the displacement for case 5 is somewhat less, as it shows a decrease of 3.7%, here the estimates is of a softer assembly than the finite element calculation.

13.5 Checking the results

For all cases we consider the part 1 or 2 as a beam of $b = 5mm$ width by $h = 20mm$ height, which gives a moment of inertia:

$I = \frac{b \times h^3}{12} = 3333mm^4$, a section modulus $Z = \frac{I}{\frac{h}{2}} = 333mm^3$ and an equivalent point load $F = 7854N$

For the first result the right part, 'part1' is a cantilever beam of $L = 68mm$, hence the displacement comes out at:

$$dz = \frac{FL^3}{3EI} = \frac{7854 \times 68^3}{3 \times 210000 \times 3333} = 1.17mm$$

compared to*Code_Aster* calculated $1.23mm$ for case one;

and a maximum normal stress of:

$$\sigma = \frac{M}{Z} = \frac{FL}{Z} = \frac{7854 \times 68}{333} = 1584N/mm^2$$

compared to*Code_Aster* calculated $2031N/mm^2$;

For the second result the assembly of 'part1' and part2' glued together acts a beam fixed at both ends of $L = 136mm$, hence the displacement comes out at:

$$dz = \frac{FL^3}{192EI} = \frac{7854 \times 136^3}{192 \times 210000 \times 3333} = 0.15mm$$

compared to *Code_Aster* calculated $0.134mm$ for case two;

and a maximum normal stress of:

$$\sigma = \frac{M}{Z} = \frac{FL}{8Z} = \frac{7854 \times 136}{8 \times 333} = 396N/mm^2$$

compared to *Code_Aster* calculated $468N/mm^2$.

The discrepancy is coming from the approximation we are making here. The values for the other results lying obviously in between these two extremes.

This technique, to hand calculate two analytical approximate values for a problem, one, optimistic, the second, pessimistic, is known as "bracketing" and should be used as much as possible to ensure the validity of a finite element result's value!

13.6 Looking at some plots

Creating the plot in the command file is quite a lengthy job, and we are a bit eager to look at it, figure 13.7 shows the plot for case 4, [almost] straight out of the box[1].

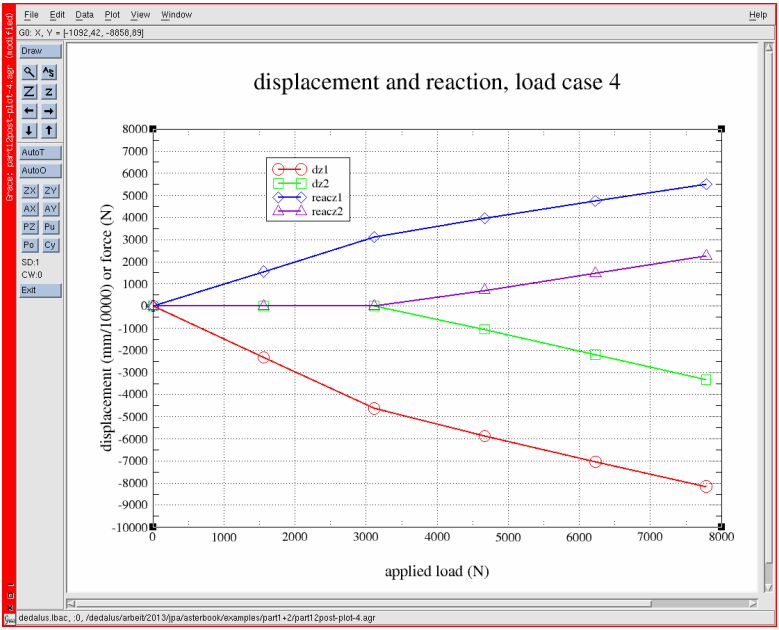


FIGURE 13.7: XmGrace plot for case 4

The bi-linear behavior of the model is clearly visible here.

[1] I changed one color, thickness of lines and legend position.

And the plot for case 5 in figure 13.8:

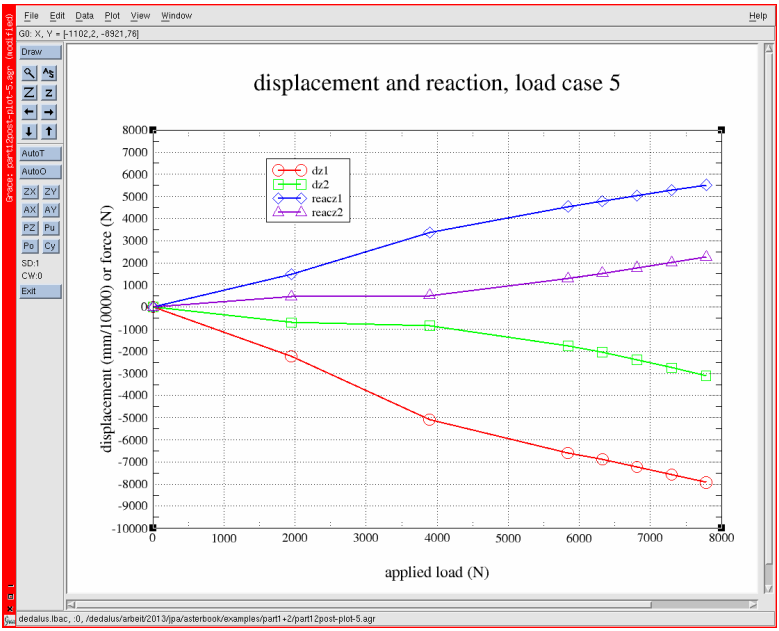


FIGURE 13.8: XmGrace plot for case 5

The friction influence is also clearly visible.

And finally the compared displacements for all five cases is shown in figure 13.9. For case 1 to 3 it is only a single point at the load of $7854N$.

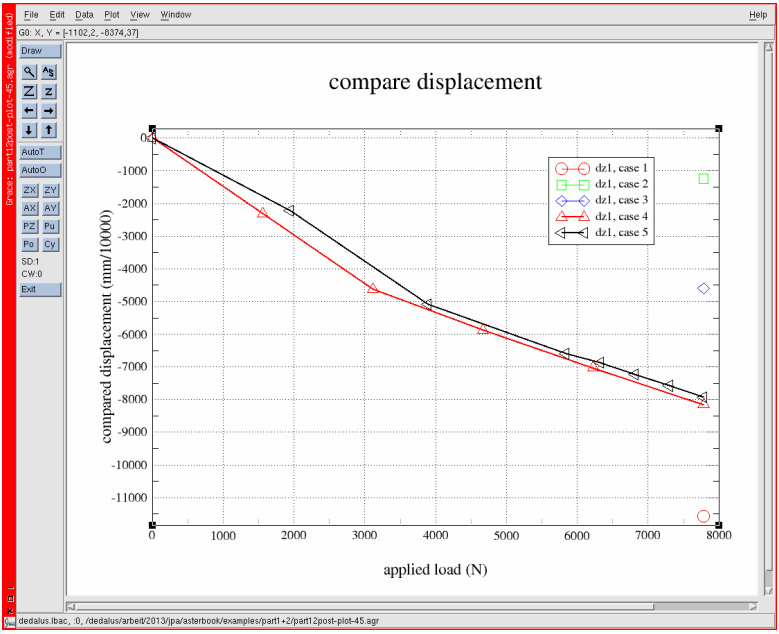


FIGURE 13.9: XmGrace plot comparing Z displacement of point 'load1p'

However this analysis is of course only valid in the elastic range. Figure 13.10 shows the von Mises criteria at the last instant for case 4, the maximum value, at $991 N/mm^2$, is rather above the yield stress[1], which should warn the engineer!

[1] Not to speak of fatigue allowance in a spring design.

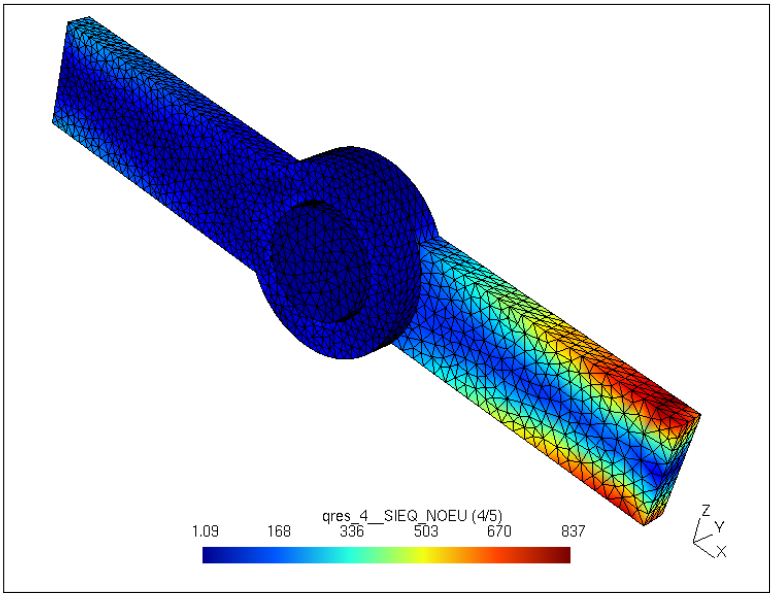


FIGURE 13.10: Von Mises criteria for case4 at last INST.

CHAPTER 14

Introducing plastic analysis, and more...

In the first part of this chapter, we study the plastic behavior of one of the part of chapter 12.

In the second part, we substitute a length of beam section to one portion of the solid part.

14.1 Running an Elasto-plastic analysis

We have seen in the previous example that the stresses in the components of the model were exceeding the elastic limit, or yield stress, it is of course not the proper behavior for a spring. Nevertheless we describe in this chapter how we can compute and display this behavior for 'part1' alone of the previous example.

As far as the mesh is concerned we just simply use the mesh created for part1 in our previous example.

This calculation is made on a simple linear mesh, it does not really make sense to launch a quadratic mesh refinement until the problem is solved at first with a linear mesh and the result do not show real deficiencies.

14.1.1 Initializing the mesh

Nothing new here.

```
DEBUT();

mesh=LIRE_MAILLAGE(UNITE=20,FORMAT='MED',);

mesh1=MODI_MAILLAGE(
    reuse =mesh1,
    MAILLAGE=mesh1,
    ORIE_PEAU_3D=(
        _F(GROUP_MA='fix1s',),
        _F(GROUP_MA='load1s',),
    ),
);

mesh1=DEFI_GROUP(
    reuse =mesh1,
    MAILLAGE=mesh1,
    CREA_GROUP_MA=_F(N
    CREA_GROUP_NO=(NOM='TOUT',TOUT='OUI',),
        _F(GROUP_MA='fix1s',),
        _F(GROUP_MA='load1s',),
  ),
);

IMPR_RESU(FORMAT='MED', UNITE=71, RESU=_F(MAILLAGE=mesh1,),);

mod1=AFFE_MODELE(
    MAILLAGE=mesh1,
    AFFE=_F(TOUT='OUI',PHENOMENE='MECANIQUE',MODELISATION='3D',),
);
```

14.1.2 Creating the non-linear material

Here, we are dealing with a non linearity related to the material, so we have to describe, within a function, the deformation or strain, of the material versus the stress. Here, we use a rather simple one for a mild steel whose yield limit is $240N/mm^2$, the strain is 20% for a stress of $380N/mm^2$ and 40% for a stress of $420N/mm^2$. Following the curve given in figure 14.1.

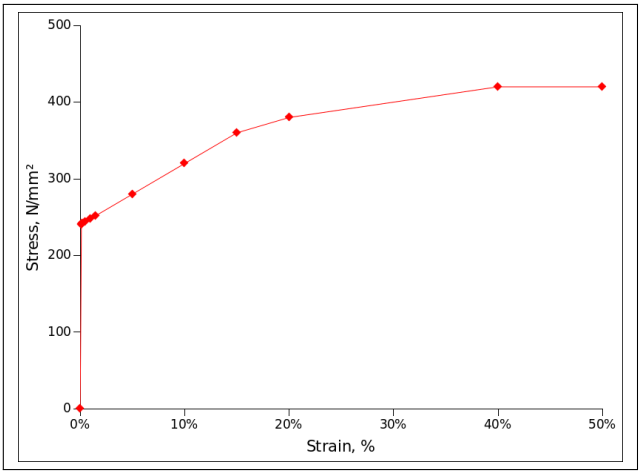


FIGURE 14.1: Steel elastoplastic stress/strain curve

```
s235elpl=DEFI_FONCTION(
    NOM_PARA='EPSI',NOM_RESU='SIGMA',
    VALE=(
        #there should not be a point at 0.0, 0.0
        #0.0000, 0.0
        0.00114, 240.0,
        0.0012, 241.0,
        0.0020, 241.6,
        0.0050, 244.0,
        0.0100, 248.0,
        0.0150, 252.0,
        0.0500, 280.0,
        0.1000, 320.0,
        0.1500, 360.0,
        0.2000, 380.0,
        0.4000, 420.0,
    ),
    INTERPOL='LIN',
    PROL_DROITE='LINEAIRE',
    PROL_GAUCHE='CONSTANT',
);
```

Note that there should not be an explicit point at $(0,0)$ the first couple of point: `'EPSI'` (ϵ) , `'SIGMA'` (σ) marks the frontier of the elastic domain and should verify $\frac{\sigma}{\epsilon} = E$ where E is the young modulus given in DEFI_MATERIAU.

Note also that we may need quite closely spaced points after the sharp bend of the yield limit so the solver does not get lost[1].

Finally, the keyword `PROL_DROITE='CONSTANT',` ensure a constant strain after a stress of $420N/mm^2$.

14.1.3 Setting model and BC

```
steel=DEFI_MATERIAU(
    ELAS=_F(E=210000,NU=0.3,),
    TRACTION=_F(SIGM=s235elpl,),
);

mate=AFFE_MATERIAU(
    MAILLAGE=mesh1,
    AFFE=_F(TOUT='OUI',MATER=steel,),
);

fix1=AFFE_CHAR_MECA(
    MODELE=mod1,
    DDL_IMPO=(
        _F(GROUP_MA=('sym1s','load1s',),DY=0.0,),
        _F(GROUP_MA='fix1s',DX=0.0,DY=0.0,DZ=0.0,),
    ),
);

load=AFFE_CHAR_MECA(
    MODELE=mod1,
    FORCE_FACE=_F(GROUP_MA='load1s',FZ=-25.0,),
);

load_m=DEFI_FONCTION(
    NOM_PARA='INST',VALE=(0,0, 5,1),
    PROL_GAUCHE='CONSTANT',PROL_DROITE='CONSTANT',
);

lreel=DEFI_LIST_REEL(
    DEBUT=0.0,
    INTERVALLE=(
        _F(JUSQU_A=1.2,PAS=0.1,),
        _F(JUSQU_A=1.5,PAS=0.1,),
        _F(JUSQU_A=5.0,PAS=0.1,),
    ),
);

#U4.34.03 for DEFI_LIST_INST
#particularly keyword ECHEC
linst=DEFI_LIST_INST(
```

[1] This may not be absolutely necessary in *Code_Aster*, but, from previous experiences, I like to do it like that!

```
    DEFI_LIST=_F(METHODE='MANUEL',LIST_INST=lreel,),
    #ECHEC=_F(
        #EVENEMENT='ERREUR',
        #ACTION='DECOUPE',
        #SUBD_METHODE='MANUEL',
        #SUBD_PAS=5,
        #SUBD_NIVEAU=4,
    #),
);
```

14.1.4 Solving

We perform two calculations:

- 'resulin' is a static linear;
- 'resunl' is a non linear calculation.

This, just to be able to look at the two results side by side in the post-processor viewer.

```
resulin=MECA_STATIQUE(
    MODELE=mod1,
    CHAM_MATER=mate,
    EXCIT=(
        _F(CHARGE=fix1,),
        _F(CHARGE=load,),
    ),
);

resunl=STAT_NON_LINE(
    MODELE=mod1,
    CHAM_MATER=mate,
    EXCIT=(
        _F(CHARGE=fix1,),
        _F(CHARGE=load,TYPE_CHARGE='FIXE_CSTE',FONC_MULT=load_m,),
    ),
    COMP_INCR=_F(
        RELATION='VMIS_ISOT_TRAC', #U4.51.11 para 4.3.1.3
        DEFORMATION='SIMO_MIEHE',  #U4.51.11 para 4.5.3
    ),
    INCREMENT=_F(LIST_INST=linst,),
    NEWTON=_F(
        PREDICTION='TANGENTE',
        MATRICE='TANGENTE',
        REAC_ITER=1,
    ),
    CONVERGENCE=_F(RESI_GLOB_RELA=1e-4,ITER_GLOB_MAXI=300,),
```

```
);
```

Here, we specify the non-linear material behavior as well as the geometric non-linear behavior under the key word COMP_INCR, VMIS_ISOT_TRAC stands for non-linear isotropic von Mises law.

```
resulin=CALC_CHAMP(
  reuse =resulin,
  RESULTAT=resulin,
  CONTRAINTE='SIEF_ELNO',
  FORCE='REAC_NODA',
  CRITERES=(
    'SIEQ_ELNO',
    'SIEQ_NOEU',
  ),
);

resunl=CALC_CHAMP(
  reuse =resunl,
  RESULTAT=resunl,
  CONTRAINTE='SIEF_ELNO',
  FORCE='REAC_NODA',
  CRITERES=(
    'SIEQ_ELNO',
    'SIEQ_NOEU',
  ),
);
```

Print the key values in *.resu* file.

```
#printing a .resu file to hold some minimum and maximum values
#of some fields and components
IMPR_RESU(
  FORMAT='RESULTAT',
  RESU=(
    _F(
      RESULTAT=resulin,NOM_CHAM='DEPL',
      NOM_CMP=('DZ',),VALE_MAX='OUI',VALE_MIN='OUI',
    ),
    _F(
      RESULTAT=resunl,NOM_CHAM='DEPL',
      NOM_CMP=('DZ',),VALE_MAX='OUI',VALE_MIN='OUI',
    ),
    _F(
      RESULTAT=resulin,
      NOM_CHAM='SIEQ_NOEU',
      NOM_CMP=('VMIS',),
      VALE_MAX='OUI',
    ),
    _F(
      RESULTAT=resunl,
      NOM_CHAM='SIEQ_NOEU',
```

```
      NOM_CMP=('VMIS',),
      VALE_MAX='OUI',
    ),
  ),
);
```

And printing the *.med* file.

```
IMPR_RESU(
  FORMAT='MED',
  RESU=(
    _F(
      RESULTAT=resulin,
      MAILLAGE=mesh1,
      NOM_CHAM=('DEPL','SIEF_ELNO','SIEQ_NOEU',),
    ),
    _F(
      RESULTAT=resulin,
      MAILLAGE=mesh1,
      NOM_CHAM='SIEQ_NOEU',NOM_CMP='VMIS',
      NOM_CHAM_MED='vmislin'
    ),
    _F(
      RESULTAT=resunl,
      MAILLAGE=mesh1,
      NOM_CHAM=('DEPL','SIEF_ELNO','SIEQ_NOEU',),
    ),
    _F(
      RESULTAT=resunl,
      MAILLAGE=mesh1,
      NOM_CHAM='SIEQ_NOEU',NOM_CMP='VMIS',
      NOM_CHAM_MED='vmisnl'
    ),
  ),
);

FIN();
```

We print the maximum values and minimum values of some fields and components as a useful check of the values displayed in the post-processing viewers.

We should always do that to ensure that we are displaying the values calculated by *Code_Aster*.

14.1.5 Looking at the results

After the calculation we can look at the results.

Although the maximum vertical displacement, from $1.6mm$ to $60.5mm$, is multiplied by almost 40; the maximum stress, from 1408 to $416N/mm^2$, is actually divided by around 3.4 and the distribution is rather different!

Just a few Gmsh hints about how ro produce the screen caps in figures 14.3 and 14.2

- the view are moved around with: in the **Options - View[x]** window, Transfo » Coordinate transformation:, the values are in model units, in the model global coordinate system;
- the scalar bars are moved around with: in the **Options - View[x]** window, Axis » 2D axes/value scale position, et to manual, fore example with the proper values in the four fields, the values are in pixel, in the screen coordinate system;
- elements outlines are obtained with: in the **Options - View[x]** window, Visibility » Draw element outlines;
- we may also shrink the elements with: in the **Options - View[x]** window, Aspect » Element shrinking factor, for example 0.8 in the top view of figure 14.2.

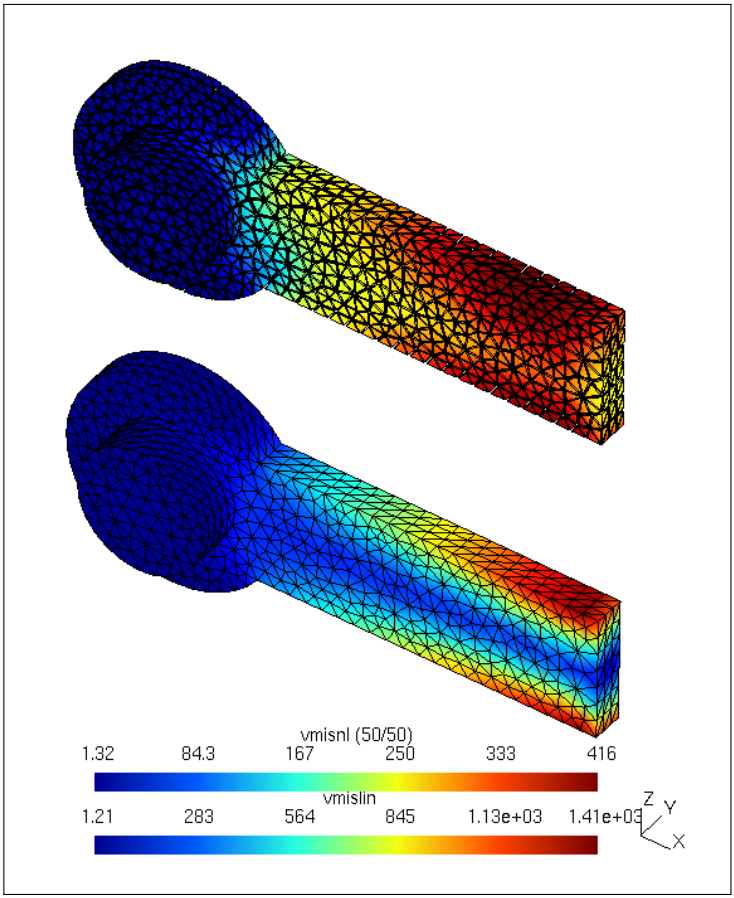


FIGURE 14.2: Compared von Mises stress

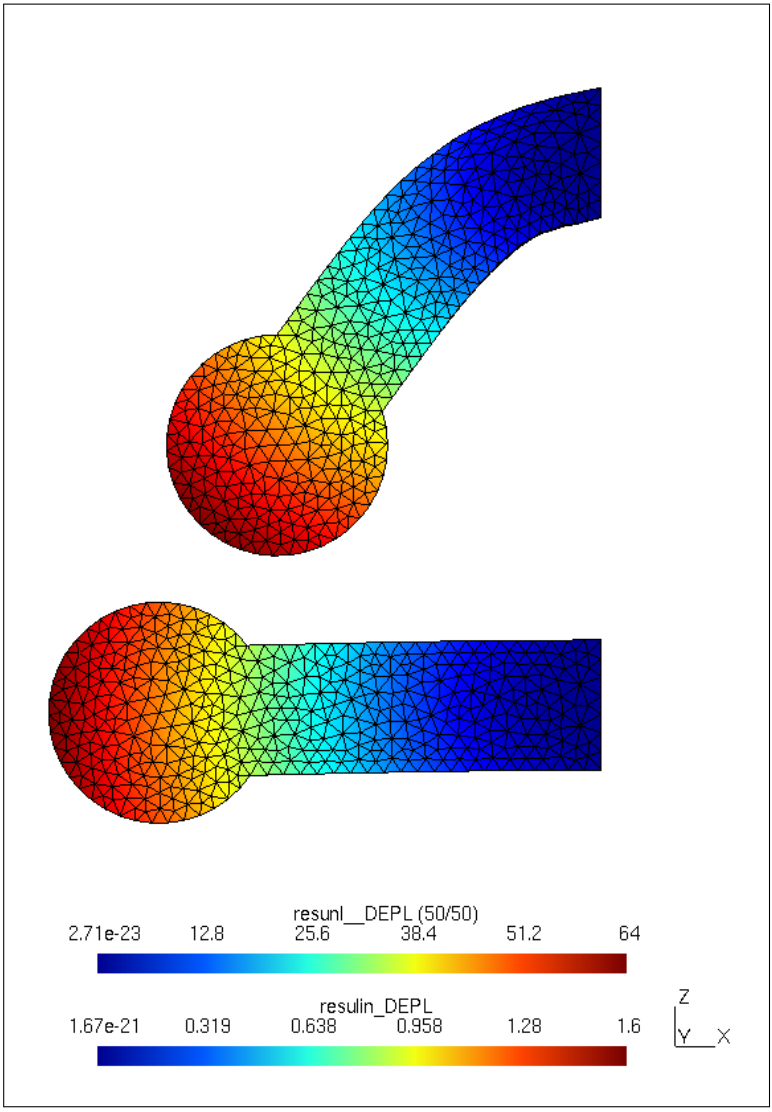


FIGURE 14.3: Compared displacement

14.2 Replacing volumes by beams

We may simplify meshing and reduce calculation time by substituting some areas of the 3D mesh by an equivalent beam section. This can be done in the previous example for some length of the part in between the grounding and the eye.

For this, we use the LIAISON_ELEM command described in U4.44.01 in paragraph 4.19, with the first basic points:

- with this we cannot use the trick of the symmetry anymore;
- the cross section of the 3D elements across the joint and the cross section of the beam must be geometrically strictly identical;
- because of the beam we carry an elastic analysis[1].

14.2.1 Meshing

Here is the modified script for creating the geometry and mesh, with the complete model and the beam elements.

```
cl1=2.5; //characteristic length, for meshing
t1=5; //thickness of half part
r1=10.0; //radius of pin
r2=17.0; //outside radius of eye
Point(1) = {0, -t1, 0, cl1}; //base point
Point(11) = {0, -t1, -r1, cl1};
Point(12) = {r1*Sin(2*Pi/3), -t1, r1+r1*Cos(2*Pi/3), cl1};
Point(13) = {r1*Sin(4*Pi/3), -t1, r1+r1*Cos(4*Pi/3), cl1};
Point(21) = {Sqrt(r2*r2-r1*r1), -t1, r1, cl1};
Point(22) = {-r2, -t1, 0, cl1};
Point(23) = {Sqrt(r2*r2-r1*r1), -t1, -r1, cl1};
Point(24) = {(4-2)*r2, -t1, r1, cl1};
Point(25) = {(4-2)*r2, -t1, -r1, cl1};

Circle(1) = {11, 1, 12};
Circle(2) = {12, 1, 13};
Circle(3) = {13, 1, 11};
Circle(4) = {21, 1, 22};
Circle(5) = {22, 1, 23};
Line(6) = {21, 24};
Line(7) = {23, 25};
```

[1] If we switch to a beam model a plastic analysis is still possible but require a more complex input. Moreover the LIAISON_ELEM with OPTION='3D_POU' is not valid in large displacements.

```
Line(8) = {25, 24};
fix1s[]  = { Extrude {0, t1, 0} {Line{8};} };
pin1s[]  = { Extrude {0, -1.5*t1, 0} {Line{-1, -2, -3};} };
Line Loop(33) = {2, 3, 1};
Plane Surface(34) = {33};
Line Loop(35) = {6, -8, -7, -5, -4};
Plane Surface(36) = {33, 35};
part1v[]  = { Extrude  {0, t1, 0} {Surface{34, 36}; } };
pin1v[]  = { Extrude  {0, -1.5*t1, 0} {Surface{34}; } };

//line to replace the volume on the right
//beam end point slightly offset from the joining surface
Point(201) = {(4-2)*r2+0.1, 0, 0, cl1};
Point(203) = {4*r2, 0, 0, cl1};
Line(201) = {201,203};
Physical Point("fixp") = {203};
Physical Point("linkp") = {201};
Physical Line("beam1") = {201};

//symmetry of 3D part
newfix1s[] = {
  Symmetry {0, 1, 0, 0}
  {Duplicata { Surface{fix1s[]}; } }
};
new1v[] = {
  Symmetry {0, 1, 0, 0}
  {Duplicata { Volume{part1v[], pin1v[]}; } }
};

Physical Point("load1p") = {33};
Physical Surface("bear1s") = {36};
Physical Surface("sym1s") = {95};
Physical Surface("pin1s") = {pin1s[]};
Physical Surface("fix1s") = {fix1s[], newfix1s[] };
Physical Surface("load1s") = {112,295};
Physical Volume("part1v") = {part1v[], pin1v[], new1v[]};
//all surfaces in Cyan
Color Cyan{
  Surface{12, 16, 20, 24, 34, 36, 44, 48, 52, 53,
  78, 82, 86, 94, 95, 102, 112};
}
//then
Color Orange{ Surface{36}; }
Color Black{ Surface{53, 102}; }
Color Green{ Surface{95}; }
Color Red{ Surface{pin1s[]}; }
Color Blue{ Surface{fix1s[]}; }
Color Magenta{ Volume {part1v[], pin1v[] };}
```

The beam end is slightly offset, in the X direction, from the joining surface, this is not required at all, it just makes the meshing view more explicit. However, we must realize that this short length behaves as a rigid body, so it should not be too long! *Code_Aster* raises a warning

about this at run time. Also the geometry is built in such a way that there is not necessarily a node on the face just facing the end of the beam, it is not necessary.

We make a quadratic mesh out of this geometry, but this creates SEG3 on the lines, which is not suitable for the beam elements, this is to be solved in the command file.

14.2.2 Commanding

Here is the command file.

```
DEBUT();

mesh0=LIRE_MAILLAGE(UNITE=20,FORMAT='MED',);

mesh0=DEFI_GROUP(
    reuse =mesh0,
    MAILLAGE=mesh0,
    CREA_GROUP_MA=_F(NOM='TOUT',TOUT='OUI',),
);

mesh0=MODI_MAILLAGE(
    reuse =mesh0,
    MAILLAGE=mesh0,
    ORIE_PEAU_3D=(_F(GROUP_MA='fix1s',),),
);

qmesh=CREA_MAILLAGE(
    MAILLAGE=mesh1,
    QUAD_LINE=_F(GROUP_MA=('beam1',),),
);
```

We revert to a linear mesh on the beam group as beams are only supported by linear elements.

```
qmesh=DEFI_GROUP(
    reuse =qmesh,MAILLAGE=qmesh,
    CREA_GROUP_NO=(
        _F(GROUP_MA='linkp',),
        _F(GROUP_MA='fix1s',),
        _F(GROUP_MA='fixp',),
  ),
);

lmesh=CREA_MAILLAGE(
    MAILLAGE=mesh0,
    LINE_QUAD=_F(GROUP_MA=('TOUT',),),
);
```

```
IMPR_RESU(FORMAT='MED', UNITE=71,  RESU=_F(MAILLAGE=lmesh,),);
IMPR_RESU(FORMAT='MED', UNITE=72,  RESU=_F(MAILLAGE=qmesh,),);
```

We create a model 'qmod' on the quadratic mesh 'qmesh' and another one 'lmod' on the linear mesh 'lmesh'.

```
qmod=AFFE_MODELE(
    MAILLAGE=qmesh,
    AFFE=(
        _F(
            GROUP_MA=('part1v','fix1s','load1s',),
            PHENOMENE='MECANIQUE',MODELISATION='3D',
        ),
        _F(
            GROUP_MA='beam1',
            PHENOMENE='MECANIQUE',MODELISATION='POU_D_T',
        ),
    ),
);

lmod=AFFE_MODELE(
    MAILLAGE=lmesh,
    AFFE=(
        _F(
            GROUP_MA=('part1v','fix1s','load1s',),
            PHENOMENE='MECANIQUE',MODELISATION='3D',
        ),
        _F(
            GROUP_MA='beam1',
            PHENOMENE='MECANIQUE',MODELISATION='POU_D_T',
        ),
    ),
);

steel=DEFI_MATERIAU(
    ELAS=_F(E=210000,NU=0.3,),
);

mate=AFFE_MATERIAU(
    MAILLAGE=qmesh,
    AFFE=_F(TOUT='OUI',MATER=steel,),
);

elemcar=AFFE_CARA_ELEM(
    MODELE=qmod,
    POUTRE=_F(
        GROUP_MA=('beam1',),SECTION='RECTANGLE',
        CARA=('HY','HZ'),VALE=(10, 20,),
    ),
);

#joining the beam to the solid part
#U4.44.01 par 4.19
```

```
link=AFFE_CHAR_MECA(
    MODELE=qmod,
    LIAISON_ELEM=_F(
        OPTION='3D_POU',
        GROUP_MA_1='fix1s',
        GROUP_NO_2='linkp',
    ),
);
```

The beam is a rectangle of exactly the same dimension as the solid part.

The LIAISON_ELEM:

- uses the OPTION='3D_POU', as we are joining a beam to a solid part;
- with GROUP_MA_1, we give the name of the element group of the joint on the 3D side;
- with GROUP_NO_2, the name of the group of nodes, in fact only one single node, at the end of the beam on the joint.

```
fix1=AFFE_CHAR_MECA(
    MODELE=qmod,
    DDL_IMPO=(
        #symmetry is not used for this calculation,
        #the beam does not allow it
        #_F(GROUP_MA=('sym1s','load1s',),DY=0.0,),
        _F(
            GROUP_MA='fixp',
            DX=0.0,DY=0.0,DZ=0.0,
            DRX=0.0,DRY=0.0,DRZ=0.0,
        ),
    ),
);

load=AFFE_CHAR_MECA(
    MODELE=qmod,
    FORCE_FACE=_F(GROUP_MA='load1s',FZ=-25.0,),
);
```

Here, we perform the analysis with the linear behavior.

```
#only one single step linear calculation
resulin=MECA_STATIQUE(
    MODELE=qmod,
    CHAM_MATER=mate,
    CARA_ELEM=elemcar,
    EXCIT=(
```

```
        _F(CHARGE=fix1,),
        _F(CHARGE=link,),
        _F(CHARGE=load,),
    ),
);

resulin=CALC_CHAMP(
  reuse =resulin,
  RESULTAT=resulin,
  CONTRAINTE=('SIEF_ELNO','SIPM_ELNO',),
  FORCE='REAC_NODA',
  CRITERES=('SIEQ_NOEU',),
);

STANLEY();
```

A call to STANLEY here would process and display results on the quadratic mesh. And we do some post-processing to be printed in the *.resu* file.

```
sr1=POST_RELEVE_T(
    ACTION=_F(
        INTITULE='reac1',
        GROUP_NO=('fixp',),RESULTAT=resulin,
        NOM_CHAM='REAC_NODA',TOUT_ORDRE='OUI',
        RESULTANTE=('DX','DY','DZ'),
        OPERATION='EXTRACTION',
    ),
);
IMPR_TABLE (TABLE=sr1,)

IMPR_RESU(
    FORMAT='RESULTAT',
    #a print ou in .resu file of Min and Max values
    #so as to check the viewer's displayed values
    #against the calculated ones
    RESU=(
            _F(RESULTAT=resulin,NOM_CHAM='DEPL',
            NOM_CMP=('DZ',),VALE_MAX='OUI',VALE_MIN='OUI',
        ),
        _F(
            RESULTAT=resulin,
            NOM_CHAM='SIEQ_NOEU',
            NOM_CMP=('VMIS_SG',),
            VALE_MAX='OUI',
        ),
        _F(
            RESULTAT=resulin,
            NOM_CHAM='SIPM_ELNO',
            GROUP_MA='beam1',
            NOM_CMP=('SIXX',),
```

```
            VALE_MAX='OUI',VALE_MIN='OUI',
        ),
    ),
);
```

To finish by projecting the results on the linear mesh and printing them in the *.med* file.

```
resulinp=PROJ_CHAMP(
    RESULTAT=resulin,
    MODELE_1=qmod,MODELE_2=lmod,
    PROL_ZERO='OUI'
);


IMPR_RESU(
    FORMAT='MED',
    RESU=(
      #the result for 3D is from the-projected results
        _F(
              RESULTAT=resulinp,
              NOM_CHAM=('DEPL','SIEQ_NOEU',),
        ),
        _F(
              RESULTAT=resulinp,
              NOM_CHAM='SIEQ_NOEU',NOM_CMP='VMIS_SG',
              NOM_CHAM_MED='vmislin'
        ),
        #the results for beams are from
        #the non-projected results
        _F(
              RESULTAT=resulin,
              GROUP_MA='beaml',
              NOM_CHAM='SIPM_ELNO',
              NOM_CMP='SIXX',NOM_CHAM_MED='sipmsixx'
        ),
        _F(
              RESULTAT=resulin,
              GROUP_MA='beaml',
              NOM_CHAM='DEPL',
              NOM_CHAM_MED='beamdepl'
        ),
    ),
);

FIN();
```

In the med file, the result for the beam are from the non-projected results as the beam has not been converted to quadratic.

14.2.3 Viewing results

After quite a bit of mixing, `DEPL` from the 'resulinp' and 'resulin' for the beam, and homogenizing the scales, we can display a deformed shape looking like figure 14.4 in Gmsh.

And the same for von Mises criteria in the 3D elements, and maximum normal stress, 'sipmsixx', for beam elements in figure 14.5. In which we use `VON_MISES_SG`, which the value of the von Mises criteria signed by the trace of the stress tensor, that is the sum of the principal stress to give a better picture of the tension and compression areas[1].

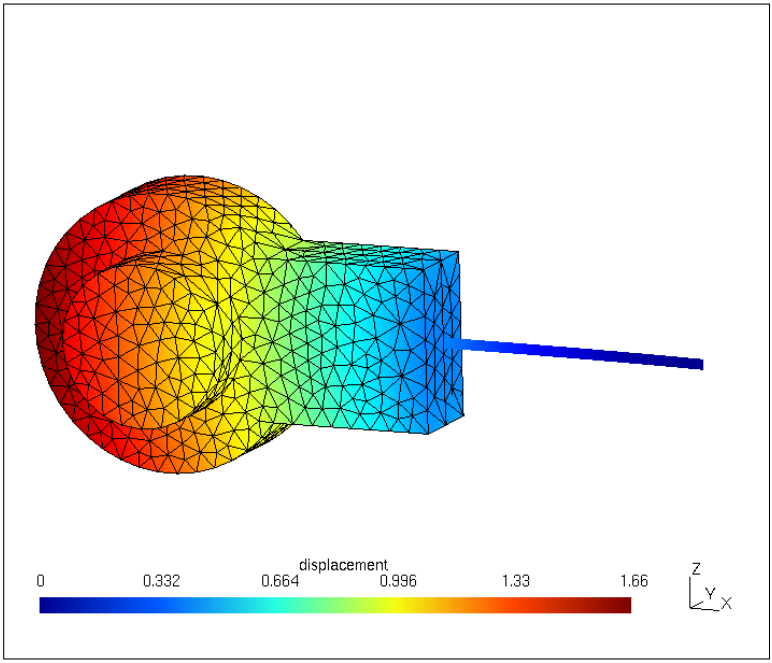


FIGURE 14.4: 3D + beam, displacement

[1] In simpler words: it is positive if there is traction and negative if there is compression.

If we compare these values with the results of the linear analysis of the solid model just above:

- the maximum displacement is $1.66mm$ in the beam+3D model while it is $1.6mm$ in the solid model;
- and the maximum stress, at the ground fixation, `SIPM_ELNO...SIXX` is $1582N/mm^2$ in the beam+3D model which compares favorably with the `SIEQ_ELNO...VMIS` of $1408N/mm^2$ in the solid model;
- also the `SIPM_ELNO...SIXX` is $747N/mm^2$ in the beam at the junction with the solid part which shows a very similar `SIEQ_ELNO...VMIS` of $792N/mm^2$ in the solid part.

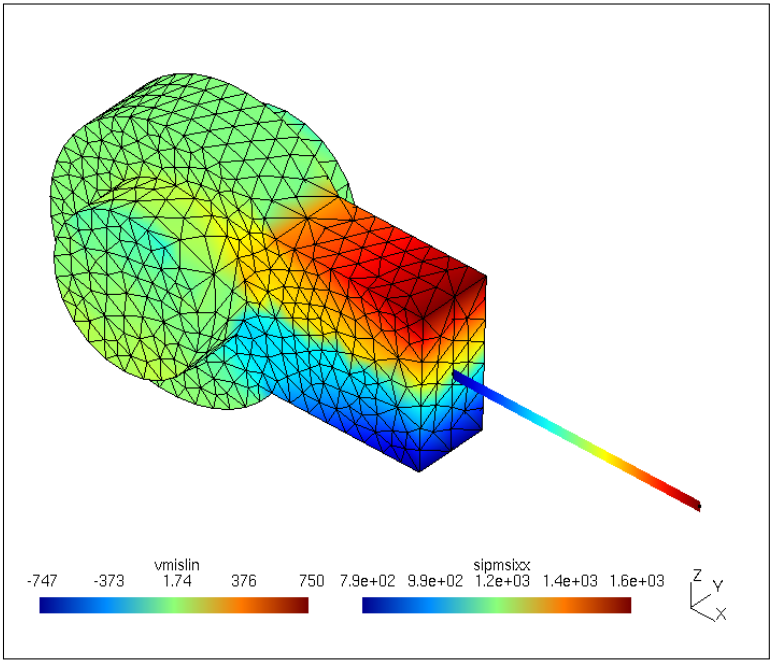


FIGURE 14.5: 3D + beam, stresses

CHAPTER 15

Buckling and modal analysis

In the first part of this chapter, we introduce the modal analysis of a very simple structure.

We compare the results with analytical methods and see how some information like generalized mass or effective mass can be extracted

In the second part, we perform the Eulerian elastic critical buckling load analysis on a simple column example.

We finally go into some hints to calculate the buckling load of a structure made of beams and plates.

15.1 Modal analysis

In this section, we analyze the modal behavior of a very simple structure, very like an inverted pendulum, 1 meter long with a mass of one kilogram at the free end, the circular object in figure 15.1. The other end is fixed in

all three directions, translations and rotations, and carries a mass of one kilogram which cannot move at all, shown as the black rectangle[1]. The section is rectangular, $a \times 2a$, such the mass of the stem is also 1 kg, which gives $a = 7.9mm$, calculated by *Code_Aster*.

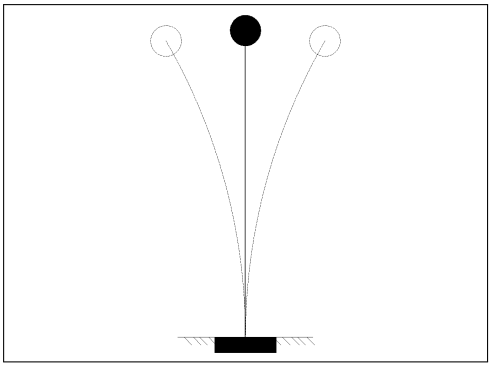


FIGURE 15.1: Inverted pendulum for modal analysis

The first natural frequency can be calculated with the following formula[2]:

$$f = \frac{1}{2\pi}\sqrt{\frac{EI}{L^3(M + 0.24Mb)}}$$

Where L is the length of the beam, Mb its mass and M the end mass, all this in a consistent set of unit (kg, m, s). The end mass is supposed to be a point mass with no rotational inertia.

In addition the mode shape is:

$$y(\frac{x}{L}) = (\frac{x}{L})^3 - 3\frac{x}{L} + 2$$

With the above figures[3] the first two frequencies, respectively in the Y and X directions come out at 2.79 and $5.59Hz$.

[1] In true life this mass could be the structure holding the stem and resting on the ground.
[2] Which can be found in [Blevins].
[3] I is calculated from a, $I = \frac{a^4}{12}$.

15.1.1 Gmsh geometry and mesh

Here is the Gmsh file for geometry.

```
cl1=100;
Point(1) = {0, 0, 0, cl1};
Point(2) = {0, 0, 1000, cl1};
Line(1) = {2, 1};
Physical Line("stem") = {1};
Physical Point("endmass") = {2};
Physical Point("fix") = {1};
```

One remark here: the mode shape calculation will only be true for a number of antinode lower than the number of element along the length, this governs the chosen meshing density, this remark holds true for the buckling analysis as well.

15.1.2 Command file, preliminaries

The first part of the command file is straightforward and is given here without any comment.

```
DEBUT(PAR_LOT='OUI',);

mesh=LIRE_MAILLAGE(INFO=1,UNITE=20,FORMAT='MED',);

mesh=DEFI_GROUP(
    reuse =mesh,MAILLAGE=mesh,
    CREA_GROUP_NO=(_F(TOUT_GROUP_MA='OUI',),),
);

model=AFFE_MODELE(
    MAILLAGE=mesh,
    AFFE=(
        _F(
            GROUP_MA=('stem',),
            PHENOMENE='MECANIQUE',MODELISATION='POU_D_T',
        ),
        _F(
            GROUP_MA=('endmass','fix',),
            PHENOMENE='MECANIQUE',MODELISATION='DIS_T',
        ),
    ),
);

steel=DEFI_MATERIAU(ELAS=_F(E=210000.,NU=0.3,RHO=8.0e-9),);

material=AFFE_MATERIAU(
```

```
    MAILLAGE=mesh,
    AFFE=_F(GROUP_MA=('stem',),MATER=steel,),
);

#dimension for mass 1 kg
a=(1.0/8/10/2)**0.5*100;
elemcar=AFFE_CARA_ELEM(
    MODELE=model,
    POUTRE =_F(
        GROUP_MA=('stem',),
        SECTION='RECTANGLE',CARA=('HY','HZ',),VALE=(a, 2*a),
    ),
    DISCRET=_F(
        GROUP_MA=('endmass','fix',),
        CARA='M_T_D_N',VALE=1.0/1000,
    ),
);

ground=AFFE_CHAR_MECA(
    MODELE=model,
    DDL_IMPO=_F(
        GROUP_NO=('fix',),
        DX=0,DY=0,DZ=0,DRX=0,DRY=0,DRZ=0,
    ),
);

massin=POST_ELEM(
    MODELE=model,CHAM_MATER=material,CARA_ELEM=elemcar,
    MASS_INER=_F(TOUT='OUI',),
    TITRE= 'massin',
);
IMPR_TABLE (TABLE=massin,)
```

The line `a=(1.0/8/10/2)**0.5*100;` calculates the dimension of the rectangular section.

One should write 1.0 -a floating number- and not just simply 1 -an integer- as the result 'a' would not then be a floating number but an integer equal to 0 [1].

Note also how we calculate the total mass model without having performed a statical analysis beforehand. If we look in the *.resu* file we can see that the computed mass , 'massin', is $3.0E-03$ (metric) tons, including the 1 kilogram mass on the top and the other 1 kilogram mass on the ground.

[1] In Gmsh any number is handled by default as a floationg point number.

```
#massin
 LIEU      ENTITE    MASSE
 mesh      TOUT       3.00000E-03
```

15.1.3 Command file, analysis

Then follows the modal analysis itself.

```
#modal analysis
#MACRO_MATR_ASSE( #in version 10.8
ASSEMBLAGE( # in version 11.2 and later
    MODELE=model,
    CHAM_MATER=material,
    CARA_ELEM=elemcar,
    CHARGE=ground,
    NUME_DDL=CO('NUMEDDL'),
    MATR_ASSE=(
        _F(MATRICE=CO('rigidity'),OPTION='RIGI_MECA',),
        _F(MATRICE=CO('masse'),OPTION='MASS_MECA',),
    ),
);

modes=MODE_ITER_SIMULT(
    #MATR_A=rigidity,  #in version 10.8
    #MATR_B=masse,  #in version 10.8
    MATR_RIGI=rigidity, #version 11.3
    MATR_MASS=masse, #version 11.3
    CALC_FREQ=_F(
        OPTION='PLUS_PETITE',
        #DIM_SOUS_ESPACE=12,
        NMAX_FREQ=8,
    ),
    #some of the lines in the following VERI_MODE
    #may have to be uncommented
    VERI_MODE=_F(
        #PREC_SHIFT=5.0000000000000001E-3,
        #STOP_ERREUR='NON',
        #STURM='OUI',
        #SEUIL=9.9999999999999995E-07,
    ),
);
```

One remark, for the modal analysis only boundary conditions are taken into account, the analysis is performed with the mass as assigned with

RHO for the materials and the masses values assigned to discrete elements, like M_T_D_N[1].

Another remark, once *Code_Aster* has calculated the modes it performs by default a check of the modes. This check may fail even if the modes are properly calculated. A workaround is to specify in the *.comm* the key word VERI_MODE with some parameters, a choice of which is proposed and commented in the block of code just above[2]. The error messages in the *.mess* file gives here some hints about the strategy to follow. Once again one has to be very careful before taking for granted the calculated values.

```
#printing of the mode shapes in a med file
IMPR_RESU(
    MODELE=model,FORMAT='MED',UNITE=80,
    RESU=_F(RESULTAT=modes,NOM_CHAM='DEPL',),
);

#printing the values in the .resu file
#for the RESU "modes"
IMPR_RESU(
    MODELE=model, #raises an error in 11.2
    FORMAT='RESULTAT',
    RESU=_F(
        RESULTAT=modes,
        #this prints all the info relative to modal analysis
        TOUT_CHAM='NON', #with this we do not print DEPL
        TOUT_PARA='OUI',#prints all the parameter
        #next lines print only the specified parameter
        #NOM_PARA=(
        #'FREQ','MASS_GENE',
        #'MASS_EFFE_DX','MASS_EFFE_DY',
        #),
        #FORM_TABL='OUI', #optional
    ),
);
```

Note: the MODE_ITER_SIMULT produces a 'data structure' which holds a collection of DEPL fields, these are the Eigen modes and they are indexed by ascending FREQ which is the Eigen frequency.

[1] This is not strictly true as a modal analysis can be performed on a pre-loaded structure, for this general instructions are given at chapter 15.1.9 .

[2] Temporarily uncommenting STOP_ERREUR='NON', it is set by default to STOP_ERREUR='OUI', helped me to identify some problems in a few cases, changing SEUIL value with STURM='OUI' is somewhat more refined.

15.1.4 First results

Here is the print out, of all the parameters, restricted to the first three modes.

```
 ASTER 11.04.00 CONCEPT modes
 CALCULE LE 05/08/2013 A 07:44:04 DE TYPE MODE_MECA



 NUMERO_ORDRE NUME_MODE        FREQ             AMOR_GENE        AMOR_REDUIT
              FACT_PARTICI_DX  FACT_PARTICI_DY  FACT_PARTICI_DZ  MASS_EFFE_DX
              MASS_EFFE_DY     MASS_EFFE_DZ     MASS_GENE        OMEGA2
              RIGI_GENE        CARAELEM         CHAMPMAT         MODELE
              NOEUD_CMP        TYPE_DEFO        TYPE_MODE
              EXCIT                    NORME


         1            1       2.89797D+00      0.00000D+00      0.00000D+00
             5.72277D-17      1.11285D+00     -1.90428D-25      4.05564D-36
             1.53363D-03      4.49062D-53      1.23836D-03      3.31548D+02
             4.10575D-01     elemcar           material         model
                                               MODE_DYN
                                    SANS_CMP: LAGR
         2            2       5.79544D+00      0.00000D+00      0.00000D+00
             1.11284D+00     -9.82865D-17     -6.81417D-17      1.53364D-03
             1.19631D-35      5.75021D-36      1.23839D-03      1.32596D+03
             1.64206D+00     elemcar           material         model
                                               MODE_DYN
                                    SANS_CMP: LAGR
         3            3       3.02328D+01      0.00000D+00      0.00000D+00
            -6.29580D-17      7.25491D-01      9.25319D-22      1.83407D-36
             2.43544D-04      3.96184D-46      4.62716D-04      3.60841D+04
             1.66967D+01     elemcar           material         model
                                               MODE_DYN
```

We can see the first 2 frequencies, at $2.898Hz$ and $5.795Hz$, in agreement with the analytical calculation. However there is here a very large amount of information and we may restrict even more the printed parameters as it is suggested in the commented code lines, we then get the following output, a bit easier to read.

```
 NUMERO_ORDRE FREQ             MASS_GENE        MASS_EFFE_DX     MASS_EFFE_DY



         1  2.89797D+00      1.23836D-03      4.05564D-36      1.53363D-03
         2  5.79544D+00      1.23839D-03      1.53364D-03      1.19631D-35
         3  3.02328D+01      4.62716D-04      1.83407D-36      2.43544D-04
         4  6.04188D+01      4.62871D-04      2.43615D-04      4.55821D-33
         5  9.46502D+01      4.48592D-04      3.15069D-32      7.79235D-05
         6  1.88910D+02      4.48749D-04      7.79878D-05      1.81477D-29
         7  1.95518D+02      4.59906D-04      3.58949D-29      3.78823D-05
         8  3.32870D+02      4.67511D-04      1.70046D-30      2.22815D-05
```

The MASS_GENE at $1.238kg$ for the first two modes seems in good agreement with the theory. So are the MASS_EFFE_DX or MASS_EFFE_DY at $1.533kg$[1].

15.1.5 More results

Here is a bit of code normalizing the modes for the MASS_GENE parameter, extract the MASS_EFFE_UN parameter and print on the fly.

```
#here we normalize the mode for 'MASS_GENE', generalized mass
normode2=NORM_MODE(
    MODE=modes,
    MASSE=masse,
    NORME='MASS_GENE',
);

extrnor2=EXTR_MODE(
    FILTRE_MODE=_F(
        MODE=normode2,
        CRIT_EXTR='MASS_EFFE_UN',
        SEUIL=1.E-3,
    ),
    IMPRESSION=_F(
        CUMUL='OUI',
        CRIT_EXTR='MASS_EFFE_UN',
    ),
);
```

Giving the following print out:

```
 ASTER 10.06.00 CONCEPT normode2
 CALCULE LE 26/02/2012 A 15:11:19 DE TYPE
 MODE_MECA


 NUMERO_ORDRE FREQ              MASS_GENE         RIGI_GENE


           1   2.89797D+00      1.00000D+00       3.31548D+02
           2   5.79544D+00      1.00000D+00       1.32596D+03
           3   3.02328D+01      1.0:0000D+00      3.60841D+04
           4   6.04188D+01      1.00000D+00       1.44113D+05
           5   9.46502D+01      1.00000D+00       3.53673D+05
           6   1.88910D+02      1.00000D+00       1.40887D+06
```

[1] A rule of thumb here would be 1/3 of the stem mass + the end mass.

7	1.95518D+02	1.00000D+00	1.50916D+06
8	3.32870D+02	1.00000D+00	4.37432D+06

One remark here: frequencies may happen to be negative, but with a very low value, this is usually due a rigid body mode, it is the same with positive very low values.

Another remark: with a model made of many beam elements, like a large frame, we may find many values corresponding to the local mode of a single element, before finding global modes, whether these local modes should be considered or not is the choice of the engineer[1]!

And one last remark: if the model is showing symmetry or cyclic symmetry there may be several modes with almost the same Eigen frequencies and shapes due to symmetry, and this is normal, for example an hexahedral section tube submitted to a compressive end load shows 6 local modes with exactly the same shape repeating on each of the six faces.

15.1.6 Estimating (roughly) the natural frequency

Before starting a modal analysis we should always have a rough idea of the value of the first fundamental, e.g. is it 0.1 Hz, 1 Hz, 10 Hz or 100 Hz!

Here comes quite useful a formula relating the value of the first, natural, frequency and the deflection produced by the action of gravity[2] acting in the same direction:

$$f = \frac{1}{2\pi}\sqrt{\frac{g}{\delta_s}}$$

where

- g is the gravity acceleration, $9.81 m/s^2$;
- δ_s is the deflection of the structure due to the sole action of g[3].

[1] And here looking at the values of MASS_GENE or MASS_EFFE_UN may help.
[2] Real or dummy!
[3] More details about the proof of this formula in [Blevins].

15.1.7 Viewing mode shapes

Figure 15.2 shows four modes on a single screen, with a scale of 200, from left to right:

- the far left, is the first mode in the YOZ plane at $2.898Hz$;
- the middle left, is the second mode in the XOZ plane at $5.795Hz$;
- the middle right, is the third "quarter wave" in the YOZ plane at $30.23Hz$;
- the far right, is the high order mode number 8 in the YOZ plane at $322.87Hz$;

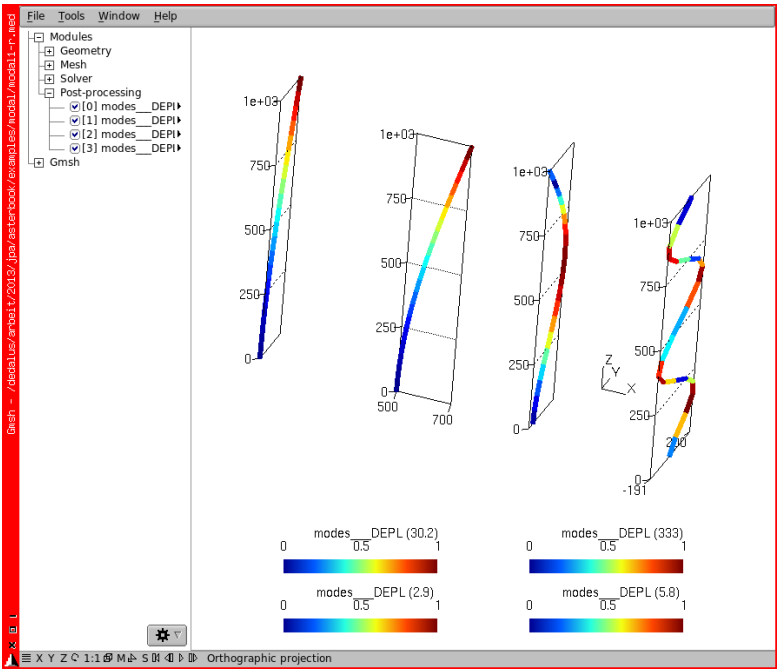


FIGURE 15.2: Four modes in Gmsh

15.1.8 What to read in the documentation

U4.52.03 concerns the operator MODE_ITER_SIMULT.

U4.52.11 concerns the operator NORM_MODE.

U4.52.12 concerns the operator EXTR_MODE.

While R5.01.03 gives a good background to the theory and explains the way *Code_Aster* names the concepts.

15.1.9 Modal analysis on an pre-loaded model

The rough guide lines to preform aModal analysis on an pre-loaded model are as such:

- make a linear elastic analysis with the pre-load, for example here, a tension in the stem;
- extract the geometrical stiffness matrix 'Kg' associated to this deformed shape;
- add this geometrical stiffness matrix to the mechanical stiffness matrix, 'K + Kg';
- perform the modal analysis.

15.2 Checking buckling

Here, we use the same mesh as for the modal analysis to calculate its critical buckling load, precisely its Eulerian linear buckling load. Full description may be found in U2.08.04.

The first critical buckling load can be calculated with the well known Euler's formula

$$P_{cr} = \frac{k\pi^2 EI}{L^2}$$

Where L is the length of the beam, I the quadratic moment in the buckling plane and L the length of the beam all this in a consistent set of unit (kg, m, s).

k is a coefficient whose value is one when both ends are pinned, free to rotate as it is the case here.

In the following example the beam section is a rectangle of $7.91mm$ by $2 \times 7.91mm$, the buckling critical values coming out of the preceding formula are:

- $P_{cr} = 1349N$ in the plane of lowest moment of inertia;
- $P_{cr} = 5397N$ in the plane of highest moment of inertia;

This formula gives only the value of the first mode, with a single antinode, higher modes are calculated by *Code_Aster*.

As far as *Code_Aster* is concerned what is calculated is a coefficient which is the ratio of the critical load, as seen above, by the applied *variable* load. *Variable* because *Code_Aster* allows to make the calculation with a fixed load and a variable load and the coefficient is applied to this second variable load. In the following example the fixed load is only the boundary conditions and no forces, except a commented line which can be un-commented in a second run to see what it does.

15.2.1 Buckling solving

In the *.comm* file we may eliminate what is related to the discrete elements as we do not use them here.

We change the boundary conditions so the bottom is fully pinned and the top fixed along X and Z, but able to move in the Z direction, to avoid instability, we prevent rotation around the beam axis by setting `DRZ=0`.

```
DEBUT();

mesh=LIRE_MAILLAGE(INFO=2,UNITE=20,FORMAT='MED',);

mesh=DEFI_GROUP(
    reuse =mesh,MAILLAGE=mesh,
    CREA_GROUP_NO=(_F(TOUT_GROUP_MA='OUI',),),
);

model=AFFE_MODELE(
    MAILLAGE=mesh,
    AFFE=_F(
        GROUP_MA=('stem',),
        PHENOMENE='MECANIQUE',MODELISATION='POU_D_T',
    ),
);

steel=DEFI_MATERIAU(ELAS=_F(E=210000.,NU=0.3,RHO=8.0e-9),);

material=AFFE_MATERIAU(
    MAILLAGE=mesh,
    AFFE=_F(GROUP_MA=('stem',),MATER=steel,),
);

a=(1.0/8/10/2)**0.5*100;
elemcar=AFFE_CARA_ELEM(
    MODELE=model,
    POUTRE =_F(
        GROUP_MA=('stem',),
        SECTION='RECTANGLE',CARA=('HY','HZ',),VALE=(a, 2*a),
    ),
);

ground=AFFE_CHAR_MECA(
    MODELE=model,
    DDL_IMPO=(
        _F(GROUP_NO=('fix',),DX=0,DY=0,DZ=0,DRZ=0,),
        _F(GROUP_NO=('endmass',),DX=0,DY=0,DRZ=0,),
    ),
    #uncomment in a second run
    #to see the behavior with a fixed force
    #FORCE_NODALE=_F(GROUP_NO=('endmass',),FZ=-1000,),
);
```

```
load=AFFE_CHAR_MECA(
    MODELE=model,
    FORCE_NODALE=_F(GROUP_NO=('endmass',),FZ=-1000.,),
);
```

At first, we make a linear static analysis, for the fixed and variable loads.

```
#for variable load
resc11p1=MECA_STATIQUE(
    MODELE=model,CHAM_MATER=material,CARA_ELEM=elemcar,
    EXCIT=(
        _F(CHARGE=ground,),
        _F(CHARGE=load,),
    ),
    OPTION ='SIEF_ELGA',
);

#for fixed load
resc12p1=MECA_STATIQUE(
    MODELE=model,
    CHAM_MATER=material,
    CARA_ELEM=elemcar,
    EXCIT=_F(CHARGE=ground,),
    OPTION ='SIEF_ELGA',
);
```

Then, we compute the internal forces related to the two previous calculations, using this field to compute the associated geometrical stiffness matrix.

```
#stress fields for variable load
sigc11p1=CREA_CHAMP(
    TYPE_CHAM = 'ELGA_SIEF_R',
    OPERATION = 'EXTR',
    RESULTAT =resc11p1,
    NOM_CHAM ='SIEF_ELGA',
    TYPE_MAXI = 'MINI',
    TYPE_RESU='VALE',
);

#geometrical stiffness matrix for variable load
regc11p1=CALC_MATR_ELEM(
    OPTION = 'RIGI_GEOM',
    MODELE=model,
    CARA_ELEM=elemcar,
    SIEF_ELGA=sigc11p1,
);

#stress fields for fixed load
sigc12p1=CREA_CHAMP(
    TYPE_CHAM = 'ELGA_SIEF_R',
```

```
    OPERATION = 'EXTR',
    RESULTAT =resc12p1,
    NOM_CHAM ='SIEF_ELGA',
    TYPE_MAXI = 'MINI',
    TYPE_RESU='VALE',
);

#geometrical stiffness matrix for fixed load
regc12p1=CALC_MATR_ELEM(
    OPTION = 'RIGI_GEOM',
    MODELE=model,
    CARA_ELEM=elemcar,
    SIEF_ELGA=sigc12p1,
);
```

And, we compute the material stiffness matrix for the total load, and proceed at the assembly of all the matrix.

```
#material stiffness matrix for both loads
remep1=CALC_MATR_ELEM(
    OPTION = 'RIGI_MECA',
    MODELE=model,
    CHAM_MATER=material,
    CARA_ELEM=elemcar,
    CHARGE = (ground, load,),
);

#matrix assemblies
nup1=NUME_DDL(MATR_RIGI=remep1,);

ramc1p1=ASSE_MATRICE(MATR_ELEM=remep1,NUME_DDL=nup1,);

ragep1=ASSE_MATRICE(MATR_ELEM=regc11p1,NUME_DDL=nup1,);

ragc12p1=ASSE_MATRICE(MATR_ELEM=regc12p1,NUME_DDL=nup1,);

#addition
ramep1=COMB_MATR_ASSE(
  COMB_R=(
    _F(MATR_ASSE=ramc1p1,COEF_R=1.0,),
    _F(MATR_ASSE=ragc12p1,COEF_R=1.0,),
  ),
);

#here we set the mini and maxi value of the modes with STURM
#the calculation fails if there are no modes in this range
#it is then necessary to adjust
#sturm u45201
#then we calculate in the range
mini=-100;
maxi=100;

INFO_MODE(
    MATR_RIGI=ramep1,
```

```
    MATR_RIGI_GEOM=ragep1,
    TYPE_MODE='MODE_FLAMB',
    CHAR_CRIT=(mini, maxi,),
);

flamb=MODE_ITER_SIMULT(
    MATR_RIGI=ramep1,
    MATR_RIGI_GEOM=ragep1,
    TYPE_RESU='MODE_FLAMB',
    CALC_CHAR_CRIT=_F(
        OPTION='BANDE',
        CHAR_CRIT=(mini, maxi),
        #alternatice option
        #OPTION='PLUS_PETITE',
        #NMAX_CHAR_CRIT=12,
    ),
);

flamb=NORM_MODE(
    reuse=flamb,
    MODE=flamb,
    NORME='TRAN',
);
```

We may have to make several runs changing the values 'mini' for CHAR_CRIT_MIN and 'maxi' for CHAR_CRIT_MAX, until finding values, without forgetting some in the lower range, which really are the critical ones! It may be a better idea to use the alternative option with OPTION='PLUS_PETITE' and NMAX_CHAR_CRIT=12.

In version 10.8 INFO_MODE is replaced by IMPR_STURM with the same syntax and use, with TYPE_MODE being replaced by TYPE_RESU.

```
IMPR_RESU(
    FORMAT='MED',
    UNITE=80,
    RESU=_F(
        RESULTAT=flamb,
        NOM_CHAM='DEPL',
    ),
);

IMPR_RESU(
    MODELE=model,FORMAT='RESULTAT',
    RESU=_F(
        RESULTAT=flamb,
        #INFO_RESU='OUI',
        TOUT_PARA='OUI',
        FORM_TABL='OUI',
    ),
);
```

```
FIN();
```

15.2.2 Calculating in version 10.8

In these previous versions the syntax was quite different:

```
#the following block is for version 10.8 only
#instead of INFO_MODE
IMPR_STURM(
    MATR_A=ramep1,
    MATR_B=ragep1,
    TYPE_RESU='MODE_FLAMB',
    CHAR_CRIT_MIN=mini,
    CHAR_CRIT_MAX=maxi,
);

flamb=MODE_ITER_SIMULT(
    MATR_A=ramep1,
    MATR_B=ragep1,
    TYPE_RESU='MODE_FLAMB',
    CALC_FREQ=_F(
        OPTION='BANDE',
        CHAR_CRIT=(mini,maxi,),
        DIM_SOUS_ESPACE=80,
        #NMAX_ITER_SOREN=80,
    ),
);
```

15.2.3 Looking at results

This buckling analysis can be carried out for only one load case. For the results in ASCII format the quickest place where to look is the *.mess* file, after the summary of the flamb=MODE_ITER_SIMULT command we can find a table looking like that:

```
------------------------------------------------------------

   LES CHARGES CRITIQUES CALCULEES INF. ET SUP. SONT:
      CHARGE_CRITIQUE_INF : -8.58123E+01
      CHARGE_CRITIQUE_SUP : -1.34914E+00


------------------------------------------------------------
       CALCUL MODAL:  METHODE D'ITERATION SIMULTANEE
                      METHODE DE SORENSEN

    NUMERO     CHARGE CRITIQUE     NORME D'ERREUR
```

```
        1      -8.58123E+01      5.56318E-14
        2      -8.55143E+01      5.14016E-13
        3      -6.57550E+01      6.63348E-14
        4      -4.83596E+01      1.21279E-13
        5      -4.83040E+01      2.09636E-12
        6      -3.36203E+01      3.14336E-13
        7      -2.15401E+01      7.77622E-13
        8      -2.15350E+01      6.39568E-12
        9      -1.21277E+01      1.78573E-12
       10      -5.39407E+00      9.63044E-12
       11      -5.39399E+00      1.82194E-10
       12      -1.34914E+00      1.28157E-10
 NORME D'ERREUR MOYENNE:   0.27676E-10

------------------------------------------------------------

         VERIFICATION A POSTERIORI DES MODES

   DANS L'INTERVALLE (-8.62414E+01,-1.34240E+00)
   IL Y A BIEN   12 CHARGE(S) CRITIQUE(S)
------------------------------------------------------------
```

Note: the table must be read from the bottom up the most critical load being number 12 in the list, at -1.349, in agreement with the hand calculated value.

The `CHARGE CRITIQUE` is the opposite factor by which the load case has to be multiplied to obtain buckling.

As stated in U2.08.04 we have

$$\mu = -\lambda$$

with λ =Eigen value and μ =multiplying coefficient of the variable load.

- The value of -1.349 means than the structure buckles for 1.394 times the applied variable load, the structure may be considered as safe[1];
- A value of -0.5 would mean than the structure would buckle for half the applied variable load, the structure may be considered as ruined;
- A value of +0.5 would mean than the structure would buckle for half the reversed applied variable load, e.g. changed of sign. The structure may be considered as ruined if the load can change sign,

[1] The value of the safe coefficient is the engineer's, or code of practice choice.

for example a wind action, or safe if the load cannot change sign, for example the gravity;

- A value of 1.349 would mean than the structure would buckle for 1.394 times the reversed applied variable load[1].

Looking at the results with Gmsh we can see in figure 15.3[2]:

- with NUMERO 12, we have the lowest mode in the plane YOZ, the plane of lowest moment of inertia with one anti-node for a value of $P_{cr} = 1349N$, it is shown in the figure with the thicker line;
- with NUMERO 11, we have the next one in the plane XOZ, the plane of highest moment of inertia with one anti-node for a value of $P_{cr} = 5393.99N$, it is shown in the figure with the intermediate line thickness;
- and with NUMERO 10, we have the third one in the plane YOZ, the plane of lowest moment of inertia with two anti-nodes for a value of $P_{cr} = 5394.09N$, it is shown in the figure with the thinner line.

We can see that NUMERO 11 and 10 having the same CHARGE_CRITIQUE of 5.394 are quite different: 1 antinode, in XOZ plane for NUMERO 11; and 2 antinodes, in YOZ plane for NUMERO 10. This is exactly what could be expected with 1/2 ratio rectangular section!

[1] We would get this result if we applied FZ=1000, a traction load, in AFFE_CHAR_MECA.
[2] How to superimpose views is explained in chapter 11.3 .

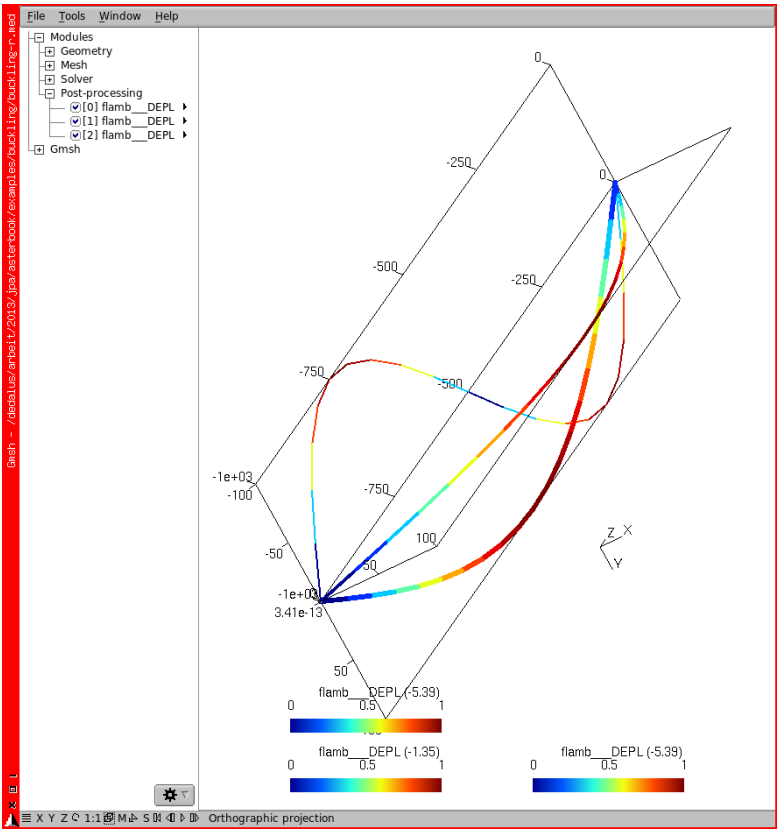


FIGURE 15.3: Third critical load

15.3 Buckling analysis with plates and beams, or rods

Note: for a buckling analysis with plates, like the example *frame3*, elements DKT must be replaced by COQUE_3D not to raise an error.

This can be done in several steps:

1. In Gmsh, we do the meshing in the normal manner then Mesh > Set order > 2.

2. First, in the command file:

```
meshini=LIRE_MAILLAGE(UNITE=20,FORMAT='MED' ,);

mesh1=CREA_MAILLAGE(
    MAILLAGE=meshini,
    MODI_MAILLE=_F(
      GROUP_MA=('panelN','panelS' ,),
      OPTION='TRIA6_7',
      #next line may have to be used
      #PREF_NOEUD='NS',
    ),
);
```

   to transform the TRIA6 element into TRIA7[1].

3. Second, in the command file:

```
mesh=CREA_MAILLAGE(
    MAILLAGE=mesh1,
    QUAD_LINE=_F(
        GROUP_MA=('topbeam','vertb','mast','hinge'),
        #next line may have to be used
      #PREF_NOEUD='m',
    ),
);
```

   to transform the SEG3 line element back into SEG2 so as to support beam elements, and hinge elements.

4. And finally, in the command file:

[1] Triangle with 6 nodes, to triangle with seven nodes, the seventh one in the center, or QUAD8_9 for quadrangular elements.

```
model=AFFE_MODELE(
    MAILLAGE=mesh,
    AFFE=(
         ..............
        _F(
          GROUP_MA=('panel',),
          PHENOMENE='MECANIQUE',MODELISATION='COQUE_3D',
        ),
         ........................
);
```

to apply `MODELISATION='COQUE_3D'` to the shell elements.

All this because second order is set to the whole mesh in Gmsh and a modal or buckling analysis in *Code_Aster* requires `COQUE_3D` for plates.

We should note also the cascading naming of meshes from the initial `meshini` to the intermediate `mesh1` and to the final `mesh`. Also the use of `PREF_NOEUD` may be required, it can take any argument, for its use reading U4.23.02 is required.

Finally, as some post processing viewer seem to be in trouble when displaying quadratic mesh, it is a wise idea to project the med results on a linear mesh, just as we did in 13.1.1 .

In a similar way a model with rod, `BARRE`, elements has to be modified to analyzed in buckling:

- each rod must be replaced by a fair number of beam elements to allow for the buckling mode shape;
- these beam elements must be given a mechanical properties in agreement with the element section;
- one `K_TR_D_L` element, with relaxed rotational DOF, must be provided at each end of the rod.

Just as with modal analysis, if a model with plates is showing symmetry or cyclic symmetry there may be several modes with almost the same buckling loads.

15.4 Some remarks about buckling

In true life a structure should not survive the application of the first global critical load. The reason for seeking more values is to detect some local buckling which could endanger the structure well before the first global one.

An Eulerian buckling analysis in a structure holding beam elements is a rather simple assumption of its behavior[1]. At least too simple for most construction codes calling for more sophisticated check-up needing in turn an extensive use of Python: to extract the desired values and build up some results with the required formulas[2].

Code_Aster allows also to make non linear buckling analysis which comes particularly useful for space frames, this is described in the test case *ssnl135*.

[1] As it does not take into account: flexural-torsional buckling, web buckling, local flange buckling and a few other oddities.

[2] It may sometimes be easier to justify by hand, with a spreadsheet, the few “dangerous” members.

CHAPTER 16

Pre-processing topics

In this chapter, we introduce some pre-processing topics:

- what are the different beam elements available in *Code_Aster*;
- how a call to MACR_CARA_POUTRE can calculate the properties of a beam section if we provide a mesh;
- what are the various plate and shell elements available in *Code_Aster*;
- why, or why not, using a quadratic mesh;
- how to create groups from scratch on a mesh.

16.1 Various type of beams, from Euler-Bernoulli to, multifiber....

Code_Aster provides various types of elements for beam, it is wise to use the type best suited to a given problem.

In the naming the first three characters POU stand for 'poutre', beam in french. The fifth character determines the support. D stands for 'droite', lying along a straight line, C stands for 'courbe', lying along a circular arc.

POU_D_E is a beam in agreement with Euler-Bernoulli hypothesis, the action of the shear forces on the rotation of cross sections are neglected.

POU_D_T is a beam in agreement with Timoshenko hypothesis, the action of the shear forces may change the cross-section orientation, this is the general type for most cases.

POU_D_TG is the same as POU_D_T. In addition the warping of the section is taken into account, this can be useful in non-linear calculation when an important warping is expected. This is described in U3.11.04. However, if the warping influence is taken into account, i.e. the deformed shape of the structure is correct and the SIEF_* are properly calculated, the calculated SIPO and SIPO do not take into account the warping restrain influence[1]. This has to be hand calculated from the DEPL...GRX and SIPO_ELNO...BX components, more information in R3.08.04 and [Roark], precisely in TABLE 21 & 22.

POU_D_T_GD is a beam element based on Timoshenko theory able to model large displacements and rotations in linear elasticity.

Finally the last character M, in the name of the elements (e.g. POU_D_EM) stands for a Multifiber beam section which is quite complicated an affair, outside the scope of this book, some details can be found in R3.08.08.

For all these types of elements, we have to feed *Code_Aster* with a large set of properties for the beam section. The built-in presets are limited to circular and rectangular sections, hollowed or not. We have to calculate these properties for any other type of section.

- The first way is to hard code these values in the *.comm* file with SECTION='GENERALE', the values being calculated beforehand with the formulas available in numerous text books[2].

[1] There is just simply not enough information in AFFE_CARA_ELEM to do that!

[2] Not all of them provide the formula to calculate, for example the warping constant of an unusual section, as far as I am concerned I have a few spreadsheets for this task, derived from [Roark].

- Another way is to have the *Code_Aster* macro command MACR_CARA_POUTRE calculate these properties from a given mesh of the section, this is explained in the next section.

16.2 Using MACR_CARA_POUTRE to calculate section properties

The macro command MACR_CARA_POUTRE[1] allows to calculate within *Code_Aster* the geometrical and mechanical properties of any section for which a mesh is provided. However this requires some care.

Here is the example of a parametric Gmsh script providing a mesh for any H section.

```
d1 = 3; //element size
h = 96; //overall height
b = 100; //overall width
tw = 5; //web thickness
tf = 8; //flange thickness
r = 12; //web to flange radius

Point(1) = { 0, 0, 0., d1};
Point(101) = {tw/2, 0, 0., d1};
Point(102) = { tw/2, h/2-tf-r, 0., d1};
Point(103) = { tw/2+r, h/2-tf, 0., d1};
Point(110) = { tw/2+r, h/2-tf-r, 0., d1};
Point(104) = { b/2, h/2-tf, 0., d1};
Point(105) = { b/2, h/2, 0., d1};
Point(106) = { 0, h/2, 0., d1};

Line (101) = {101, 102};
Circle(102) = {102, 110, 103};
Line (103) = {103, 104};
Line (104) = {104, 105};
Line (105) = {105, 106};
Symmetry {-1, 0, 0, 0} {
  Duplicata { Line{-101, -102, -103, -104, -105}; }
}
Symmetry {0, -1, 0, 0} {
  Duplicata { Line{
    -110, -109, -108, -107, -106, -105,
    -104, -103, -102, -101};
  }
}
Line Loop(121) = {
  101, 102, 103, 104, 105, 110, 109, 108, 107, 106,
```

[1] In U4.42.02.

```
  115, 114, 113, 112, 111, 116, 117, 118, 119, 120
};
Plane Surface(122) = {121};

Physical Line("myborder") = {101:120};
//here there should be some close lines bordering
//the inside for an hollow section
//Physical Line("int") = {};
Physical Surface("mysect") = {122};
Physical Point("noderef") = {1};
```

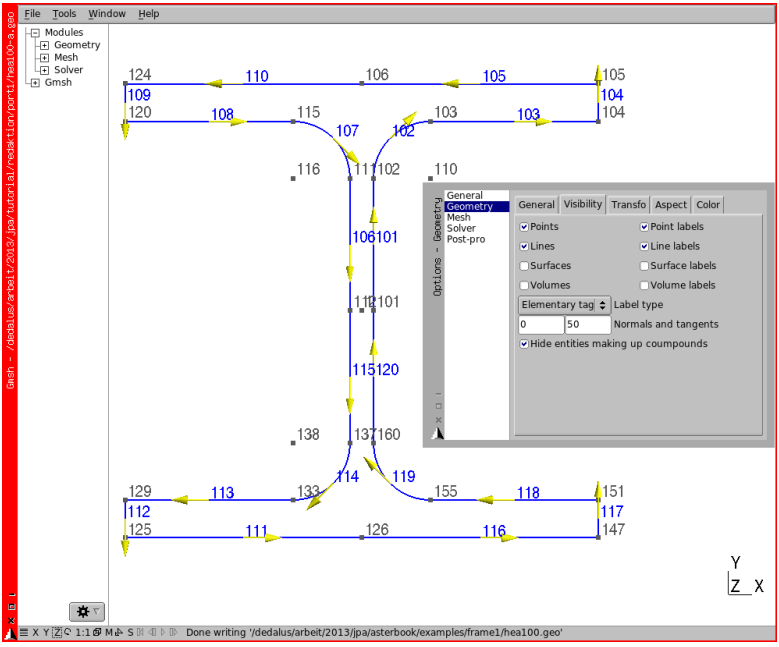


FIGURE 16.1: Orientation of line elements in yellow

- There should be one group of lines following the outside of the contour, here 'myborder', this contour should be closed and all the lines part of it should be in the same direction: if one walks along the border, one is walking on lines always in the same direction shown by the yellow arrows in figure 16.1. There is a more detailed explanation about how to achieve this in appendix B 2.3 . This

group name 'myborder' is at the user's choice, and has to be use used again in the *.comm* file.

- Likewise there should be one[1] group of lines following the border of the hollow for an hollowed section.
- There should be a Physical holding one single Point to enable the calculation of shear coefficients, here it is named 'noderef'.
- It is not mandatory to name the surface group.

Once properly meshed and saved in a *.med* file the following bit of code allows to compute and print the section properties.

```
#PAR_LOT='NON' if we use the Python bits
DEBUT(PAR_LOT='NON',);
.....................
mhea100=LIRE_MAILLAGE(
    UNITE=21,
    FORMAT='MED',
    #NOM_MED='hea100',
    #INFO_MED=2,
    #INFO=2,
);

#next command creates the node on 'no'
mhea100=DEFI_GROUP(
    reuse =mhea100,MAILLAGE=mhea100,
    CREA_GROUP_NO=(_F(GROUP_MA='noderef',),),
);

#prefix s stands for section
shea100=MACR_CARA_POUTRE(
    MAILLAGE=mhea100,
    GROUP_MA_BORD='MYBORDER',
    #GROUP_MA_INTE='int', #if there is an hollow
    GROUP_NO='noderef',
    INFO=2,
    ORIG_INER=(0.0,0.0),
    #comment next 2 lines in 10.8 versions
    TABLE_CARA='OUI',
    NOM='hea100',
    # 'hea100' is at the user's choice
);

#this prints out the table
IMPR_TABLE(
    TABLE=shea100,
    FORMAT='TABLEAU',
```

[1] Or several, for multiple hollows sections.

```
    UNITE=8,
    SEPARATEUR=' * ',
    TITRE='hea100',
    INFO=2,
);

#next section is not mandatory
#and is obsolete with version 11.3
#just to look at what we have done
#here we extract some values in Python variables
lieu=shea100['LIEU',1]
a=shea100['AIRE_M',1]
#...................
#and print them in a separate file
file=open('our_fullpath_file_name.txt','w')
file.write ('hea100 section properties \n')
file.write ('LIEU   = %s \n' % lieu)
file.write ('AIRE_M   = %s \n' % a)
#...................
file.close()
#we could then build up the AFFE_CARA_ELEM arguments
```

Unfortunately there was no simple way to introduce the section's calculated values in AFFE_CARA_ELEM, up to version 10.7[1].

For version 11.2 and later, we can proceed as follows, after uncommenting the previous TABLE_CARA='OUI',.

```
#here we extract some values in Python variables
lieu=shea100['LIEU',1] #lieu has to be extracted!
.....................
elemcar=AFFE_CARA_ELEM(
  MODELE=model,
  #next line to print the values used for the
  #section properties
  INFO=2,
    POUTRE=(
       ...................
        _F(
            GROUP_MA=('topbeam',),SECTION='GENERALE',
            TABLE_CARA=shea100,
            #next line with the section name defined above
            NOM_SEC='hea100',
        ),
     .................
  ),
..............
);
```

And the properties of the section 'shea100' are used in the calculation.

[1] However a few Python lines would do it.

In real life problems we may have many sections calculated this way as long as they have different NOM_SEC.

16.3 Various types of plates and shells....

Just like the beam elements, plates and shells come in various flavors. It is useful to know the basics about their differences, more details may be found in U2.02.01.

These elements exist either as triangles or quadrangles as we have seen in the *frame3* example. Middle nodes may be created by *Code_Aster* using commands like

```
MODI_MAILLAGE / MODI_MAILLE ...OPTION='TRIA6_7'.
```

16.3.1 Plates

A plate element, "plaque" in french, lies in a plane, there is no curvature in the element, which means that the 4 nodes of a quadrangular element **must** lie in a plane[1]. The plate elements come in various flavors:

- DKT[2] , which we use in this book, does not support transverse shear;
- DST[3], does support transverse shear;
- Q4G also supports transverse shear, but usually requires a more refined mesh than DKT, see U2.02.01.

16.3.2 Shells

Unlike plates, shells, "coque" in french, admit a curvature, and have a middle node along the edges. Only them are supported for a buckling analysis in *Code _Aster*.

[1] If this is not the case *Code_Aster* emits a warning in the .mess file, telling how much the element is distorted and leaving to the engineer the choice of altering the mesh or not.

[2] Strictly speaking a DKT, is a triangle, DKQ being a quadrangle, but the call of DKT is enough in a command file.

[3] Strictly speaking a DST, is a triangle, DSQ being a quadrangle, but the call of DST is enough in a command file.

The element here is COQUE_3D[1].

16.4 Using quadratic mesh or not

Except when it is a mandatory requirement, like in a buckling analysis for plates, as explained in chapter 15.3 [2], the question arises whether or not to use a quadratic mesh. But first of all what is a quadratic mesh ?

Taking the examples of a 3 node triangular shell element its quadratic extension is a 6 node element where the nodes situated at the middle of the edges sit on the geometrical curve joining the corner nodes.

Now let's take the example of a ring, 100 in diameter, 50 in depth. Figure 16.2 shows the parent geometry and a rather crude linear mesh with 8 faces around the circumference.

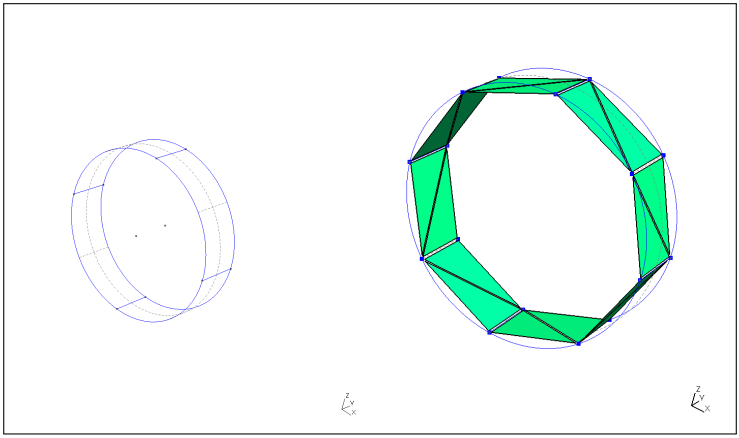


FIGURE 16.2: Parent geometry and 8 faces linear mesh

Figure 16.3 shows the same 8 faces mesh but interpolated, by the mesher, in quadratic[3] and a linear mesh produced when dividing the char-

[1] Elements COQUE_C_PLAN, COQUE_D_PLAN and COQUE_AXIS are outside the scope of this book.

[2] Where we ask *Code_Aster* to create an extra node at the barycenter of the element.

[3] The meshes here are produced by Gmsh, but Salome or any mesher would produce a similar result.

acteristic length by two (50 instead of 100). The curvature of the ring is better reproduced in both cases.

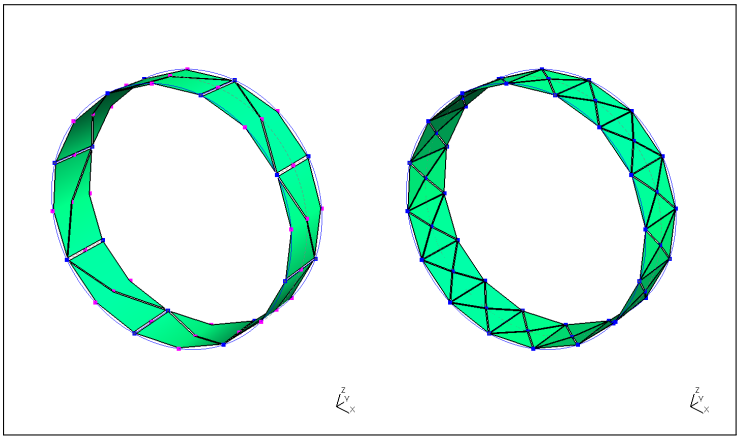


FIGURE 16.3: 8 faces quadratic mesh and 16 faces linear mesh

Figure 16.4 shows the original 8 faces linear mesh and how it is transformed in a quadratic mesh by a simple `CREA_MAILLAGE ... LINE_QUAD` command, the curvature is no better represented[1].

It comes obvious that the `CREA_MAILLAGE ... LINE_QUAD` *Code_Aster* command cannot produce such as good a quadratic mesh as a mesher, or a higher node density linear mesh, for the simple reason that it does know nothing about the geometrical curvature at the refinement time[2].

The calculated results may improve dramatically with a better meshed geometry, from the mesher, versus a command line transformed mesh. Let's think of the behavior of the previous meshes under internal pressure, or buckling under external pressure!

[1] In the pictures the corner nodes, created in the linear command, are depicted in blue, while the intermediate nodes, created either in Gmsh or in *Code_Aster* show in pink, and the mesh 'Element shrinking factor' is set to 0.95.

[2] The mesh file does not hold any such information!

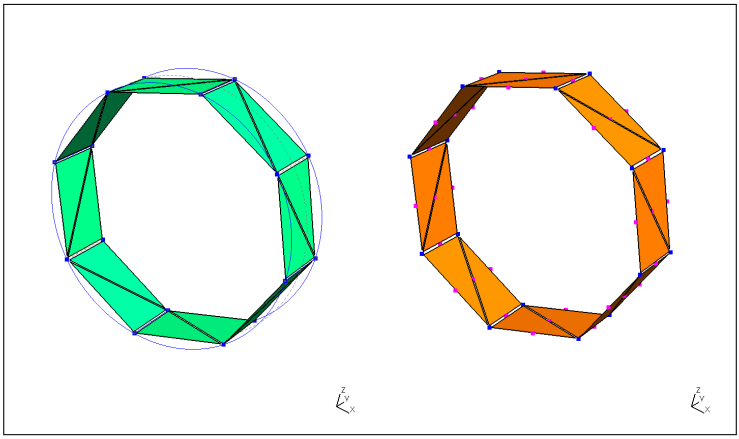


FIGURE 16.4: 8 faces linear mesh and the same transformed by*Code_Aster*

16.5 Creating groups from scratch

It is possible to create groups in a mesh which does not contain any with the CREA_GROUP command. This is fully documented in U4.22.01.

The following excerpt, inserted in the *.comm*, file creates two of the required groups in the 'part2' mesh of the 3D spring.

```
part2=DEFI_GROUP(
  reuse =part2,
  MAILLAGE=part2,
  CREA_GROUP_MA=(
    #group for fixation
    _F(
      NOM='fx2s',
      TYPE_MAILLE='2D',
      OPTION='BANDE',
      POINT=(-68,-5.5,105,),
      VECT_NORMALE=(1,0,0,),
      DIST=0.5,
    ),
    _F(
      NOM='hole2s-1',
      TYPE_MAILLE='2D',
      OPTION='CYLINDRE',
      POINT=(0,0,0,),
      VECT_NORMALE=(0,1,0,),
      RAYON=10.6,
    ),
```

```
    _F(
      NOM='hole2s-2',
      TYPE_MAILLE='2D',
      OPTION='FACE_NORMALE',
      VECT_NORMALE=(0,1,0,),
      VERI_SIGNE='NON',
    ),
    _F(
      NOM='hole2s',
      TYPE_MAILLE='2D',
      DIFFE=('hole2s-1','hole2s-2',),
    ),
  ),
);
```

The first instance creates a group with all the 2D elements:

- lying in a plane, `OPTION='BANDE'`;
- this plane passing through the point situated at one of the corner of 'part2', `POINT=(-68,-5.5,105,)`;
- this plane is normal to the vector `VECT_NORMALE=(1,0,0,)`;
- the group retains all the elements at a distance less than 0.5 of the plane, `DIST=0.5`.

The second instance creates a group, 'hole2s-1', of the element lying in a cylinder, as described.

The third creates a group, 'hole2s-2', with all the elements lying in the XOZ plane.

The last one, a boolean difference between the two previous, creates the group 'hole2s' as required.

These powerful tools allow to use a mesh without any predefined groups. This may be an efficient way to proceed with mesh built from CAD drawings, *.iges*, *.step* even *.stl* format.

And more than ever the next line helps to view the groups on the screen and check that everything is as expected.

```
IMPR_RESU(FORMAT='MED', UNITE=71,  RESU=_F(MAILLAGE=part2,),);
```

CHAPTER 17

Gathering more information before processing

In this chapter, we explain how to gather some information from a study without actually solving the problem:

- how to color a mesh according to various properties;
- how to display vectors along the element's local axis;
- how to display the applied load;
- how to calculate length or area of elements, with Python.

17.1 Coloring mesh and model according to properties

We have seen how to color the mesh by groups in Gmsh, however this feature is limited only to the mesh. When a model is made from this mesh in *Code_Aster* there is many more valuable information we would like to view, like for example material, plate thickness, beam properties, etc. The following lines of code, within the *.comm* file of our *frame3* example, creates a med file in which some concepts are given a color.

```
IMPR_RESU(
    FORMAT='MED',UNITE=82,
    CONCEPT=(
        _F(CHAM_MATER = material,),
        _F(CARA_ELEM= elemcar,REPERE_LOCAL='OUI', MODELE=model,),
        _F(CHARGE= ground),
        _F(CHARGE= selfwght),
        _F(CHARGE= cc,),
        _F(CHARGE= cv,),
    ),
);
```

If we open the *.med* file we can find a field 'selfwgt#CHMEPESAN'[1] whose details can be seen in figure 17.1:

- almost everything is colored in dark red, with a numeric value of 10000, which is the value we specified for gravity acceleration;
- only the tiny discrete elements joining the top structure to the vertical masts are colored in blue, with a numeric value of O, and thus not subject to gravity;

which is exactly what is expected from the load specification in the command file.

The same can be done with concept 'elemcar#CARCOQUE' to display the different thickness like in figure 17.2.

Note: we specified a Logical Unit 82 to save the .med file file used to display the results. This file must be specified in the ASTK setup. Also this `IMPR_RESU` may be done without any call to a solver which allows to ensure the properties are what we expect without launching any lengthy solver step.

[1] Where: selfwgt is the name we gave to the `AFFE_CHAR_MECA`, CHME stands for CHarge-Meca and PESAN is a short cut of PESANteur, the type of load.

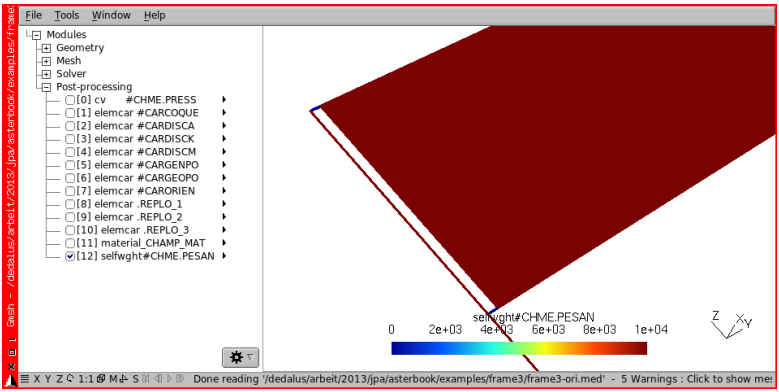


FIGURE 17.1: Coloring concept for gravity load

17.2 Showing element orientation

In addition the line:

```
_F(CARA_ELEM= elemcar,REPERE_LOCAL='OUI', MODELE=model,),
```

allows to display the local axis of the elements.

Figure 17.3 shows what it looks like for:

- local y axis, view[9] elemcar.REPLO_2, as a green arrow;
- line elements drawn in black;
- nodes drawn in dark blue.

This view shows also some of the Gmsh settings, in the dialog boxes, used to obtain it.

With a bit more tweaking we can get the picture just like 17.4 with the CAD coloring, x local axis in red, y in green and z in blue.

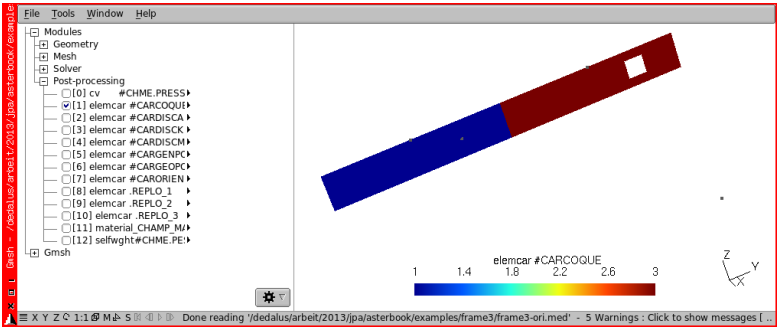


FIGURE 17.2: Coloring concept for plate thickness

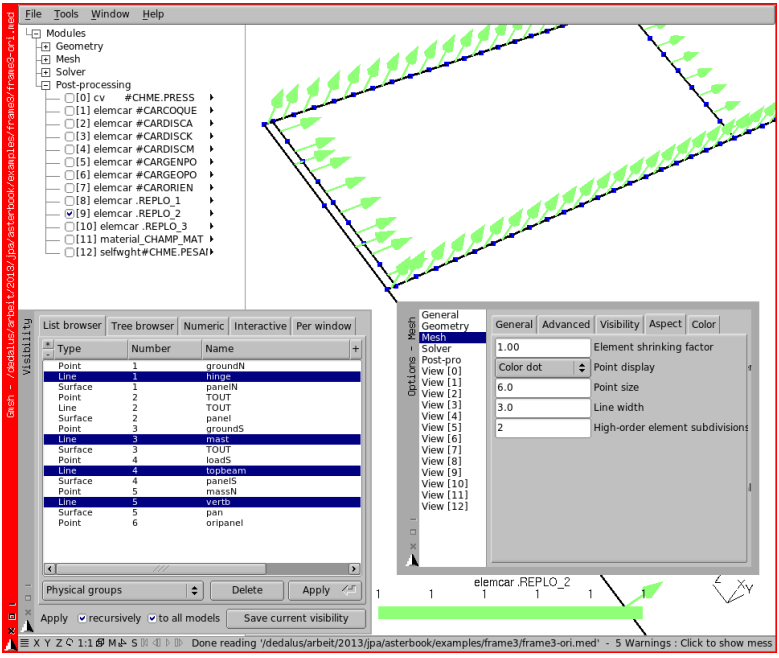


FIGURE 17.3: Local y axis of beams and discret element

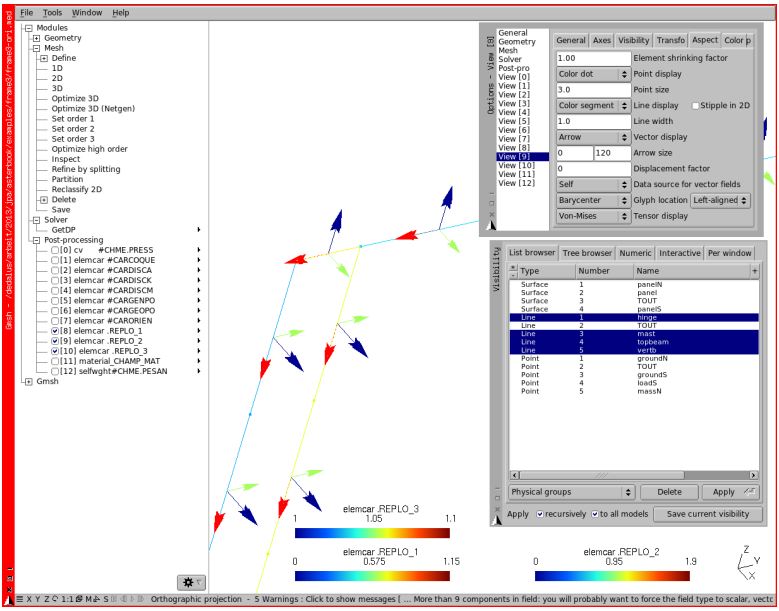


FIGURE 17.4: Local coordinate system of beams and discrete elements

17.3 Showing the applied load

Another way to display the applied load on a model, for a given load case is to run a static calculation under this load case with ALL the nodes of the model fixed in all three directions[1] and to compute and display the reactions, which in this case are exactly the opposite of the applied loads to every node[2]. The following code segment does it for our example *frame3*:

Firstly setting the load multiplier for this particular solving with the prefix "f". Note the minus one multiplier.

```
#code for printing a vector print out
#of applied forces
#fix all nodes
fixall=AFFE_CHAR_MECA(
    MODELE=model,
    DDL_IMPO=_F(GROUP_NO=('TOUT',),DX=0,DY=0,DZ=0,),
);
#multiplier value is -1 for each load case
fsw_m=DEFI_FONCTION(
    NOM_PARA='INST',VALE=(0,-1, 1,0,),
    PROL_DROITE='CONSTANT',
);
fcc_m=DEFI_FONCTION(
    NOM_PARA='INST',VALE=(0,0, 1,-1, 2,0),
);
fcv_m=DEFI_FONCTION(
    NOM_PARA='INST',VALE=(0,0, 1,0, 2,-1,),
);

listf=DEFI_LIST_REEL(
    DEBUT=0.0,INTERVALLE=_F(JUSQU_A=2,PAS=1.0,),
);
```

Secondly make the static analysis with all the individual load cases, and calculate the reactions.

```
#solving for the individual loads
force=MECA_STATIQUE(
    MODELE=model,CHAM_MATER=material,CARA_ELEM=elemcar,
    EXCIT=(
        _F(CHARGE=fixall,),
```

[1] In some case it may be necessary to fix some rotations as well.
[2] In this case a distributed load is shared between the nodes of the elements.

```
        _F(
          CHARGE=selfwght,
          TYPE_CHARGE='FIXE',FONC_MULT=fsw_m,
        ),
        _F(CHARGE=cc,TYPE_CHARGE='FIXE',FONC_MULT=fcc_m,),
        _F(CHARGE=cv,TYPE_CHARGE='FIXE',FONC_MULT=fcv_m,),
    ),
    LIST_INST=listf,
);

force=CALC_CHAMP(
    reuse =force,
    RESULTAT=force,
    CONTRAINTE='SIEF_ELNO',
    FORCE=('REAC_NODA'),
);
```

Thirdly computes the sum of the reactions of each individual load case an print the tables in the *.resu* file.

```
s_reac=POST_RELEVE_T(
    ACTION=_F(
        INTITULE='sum forces',
        TOUT_ORDRE='OUI',
        GROUP_NO=('TOUT'),
        RESULTAT=force,
        NOM_CHAM='REAC_NODA',
        RESULTANTE=('DX','DY','DZ'),
        OPERATION='EXTRACTION',
    ),
);

IMPR_TABLE (TABLE=s_reac,)
```

Fourthly make a *.med* file.

```
IMPR_RESU(
    FORMAT='MED', UNITE=82,
    RESU=(
        _F(
          GROUP_MA=('TOUT',),
          RESULTAT=force,
          NOM_CHAM='DEPL',
        ),
        _F(
      GROUP_NO=('TOUT',),
      RESULTAT=force,
      NOM_CHAM='REAC_NODA',
      NOM_CHAM_MED='applied force',
```

```
          ),
      ),
);
#end of load printout
```

We apply all the individual load cases, actually changed of sign, to view a reaction in the same direction as the force, and in the same time, print out in the *.resu* file the sum of these individual load cases. `NOM_CHAM_MED='applied force'` is used to rename the field in the med file[1].

Opening the file in Gmsh, choosing the dialog box **Options** with:

- 'applied force' as selected, with View[0];
- then in the tab Visibility, Force Vector in the left-hand lower pull down list;
- then in the General tab pull the list Range mode to Custom;
- and push the Min and Max buttons to refresh the display with the proper component values;
- then in the Aspect tab, 3D arrow in the Vector display pull down list;
- and Vertex in the Glyph location one[2].

gives the following view with Lines ticked in the Mesh tab so as to see the background mesh. Which is shown the figure 17.5 for the distributed wind load on the 'panel' group[3].

[1] This method cannot give a print out of externally applied moments, nor can it give a print of `AFFE_CHAR_CINE`.

[2] Choosing Barycenter displays the sum of the distributed load on every element, at the barycenter of the element, instead of the nodal reactions at end nodes.

[3] For such a surface, load the length of the arrow is proportional to the element area, yet not constant!

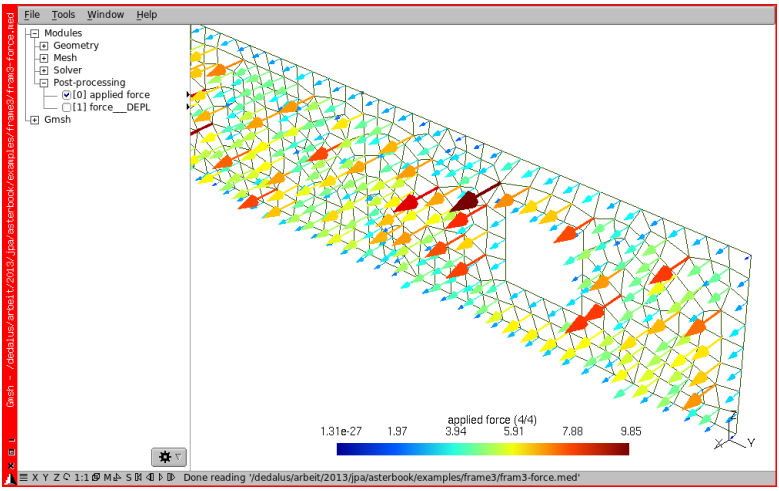


FIGURE 17.5: Applied load vector superimposed on a mesh view, limited to group 'panel'

17.4 Calculating length and area of mesh elements

This could also be called "reading and manipulating mesh data".

It may be very helpful to get some statistics about the mesh such as element's length or area by groups[1], the following Python code can do this, and we use it on our *frame3* example. Modify the beginning of the command file as follows:

```
DEBUT(PAR_LOT='NON',);

#import the necessary libraries
import sys
from Utilitai import partition
import string
import numpy as N

meshf=LIRE_MAILLAGE(
   ............
);
```

[1] Maybe just simply to prepare a bill of material in a beam model, or to retrieve the number of nodes along a beam, together with the beam length so as to exactly compute the nodal forces equivalent to a distributed load.

```
meshf=DEFI_GROUP(
   ..............
);

mesh = partition.MAIL_PY()
mesh.FromAster('meshf')

#two methods to define output file
#first,
#hard code path + file name
#file=open('/dedalus/arbeit/2012/jpa/
#tutorial/redaktion/port3/statistics.txt','w')
#second
#an entry in ASTK with LU=38 for example

DEFI_FICHIER( UNITE=38, FICHIER='./REPE_OUT/stati2.txt')
statis=open('fort.38','w')

listgm1D=['topbeam','vertb','mast',]
listgm2D=['panel',]

#next part to call a comm file that
#calculate length and area of elements
INCLUDE (UNITE=2, INFO=2)
#end of call
```

`DEBUT(PAR_LOT='NON',)` is used since we are going to explore *Code_Aster* data structures[1]. Then the import lines to import the necessary libraries. The two `mesh = partition.MAIL_PY()` and `mesh.FromAster('meshf')` lines import the mesh object [2].

Afterward, we define a file ***statis*** where to write the results, of course there should be a matching entry in the **ASTK** window, with LU=38 and 'R' ticked. We define the groups of element on which we want to perform the calculation.

And finally `INCLUDE` another *.comm* file, which also needs an entry in **ASTK**, with LU=2 and 'D' ticked.

This method with `INCLUDE` uses a rather portable *.comm* file for the intended job and makes the main command file shorter and easier to read.

And now this is the code for this "included" file, doing the job of computing lengths and areas:

[1] U1.03.02 is a must be read when one wants to manipulate *Code_Aster* data objects.

[2] Using the same word for 'meshf' and 'mesh' would have raised a runtine error, since here 'mesh' is an instance of the MAIL.PY class.

```
#length and surface of elements
#intialize the list for coordinates
xcoor=[None]*(4);
ycoor=[None]*(4);
zcoor=[None]*(4);
#first for 1D elements
nbgm =len(listgm1D)
for i in range (0, nbgm):
    beamgm=mesh.gma.get(listgm1D[i])
    totlong=0;
    lgth=len(beamgm)
    for j in range(lgth):
        beamid=beamgm[j]
        for k in range (0, 2):
        nodeid= mesh.co[beamid][k]
        nodecoord= mesh.cn[nodeid]
            xcoor[k]=int(mesh.cn[nodeid,0])
            ycoor[k]=int(mesh.cn[nodeid,1])
            zcoor[k]=int(mesh.cn[nodeid,2])
        lxyz=((xcoor[1]-xcoor[0])**2+(ycoor[1]-ycoor[0])**2
        +(zcoor[1]-zcoor[0])**2))**0.5
        totlong=totlong+lxyz
  avlong=totlong/lgth
  statis.write('group 1D          : %s\n' % listgm1D[i])
  statis.write('number of element : %d\n' % lgth)
  statis.write('total length      : %d\n' % totlong)
  statis.write('average length    : %d\n' % avlong)
  statis.write('\n')
```

In this code abstarct:

- `beamgm[j]` holds the *id* of the *jth* beam in the group beam `gm`;
- `mesh.co[beamid][k]` gives the *id* of the nodes belonging to element *id* `beamid`;
- and `mesh.cn[nodeid,0]` contains the x coordinate of the node, (1 for y and 2 for z)[1].

The same can be done with 2D elements,

```
nbgm =len(listgm2D)
for i in range (0, nbgm):
    plategm=mesh.gma.get(listgm2D[i])
    totsurf=0;
    triacount=0;
    quadcount=0;
    lgth=len(plategm)
```

[1] In Python a loop like `for k in range (0, 2):` is executed for the value 0 and 1 but not 2! The same rule applies to initialization of lists

```
    for j in range(lgth):
    plateid=plategm[j]
    tria=len(mesh.co[plateid])
    if tria==3:
        for k in range (0, 3):
            nodeid= mesh.co[plateid][k]
            nodecoord= mesh.cn[nodeid]
            xcoor[k]=int(mesh.cn[nodeid,0])
            ycoor[k]=int(mesh.cn[nodeid,1])
            zcoor[k]=int(mesh.cn[nodeid,2])
        a=((xcoor[1]-xcoor[0])**2+(ycoor[1]-ycoor[0])**2
        +(zcoor[1]-zcoor[0])**2)**0.5
        b=((xcoor[2]-xcoor[1])**2+(ycoor[2]-ycoor[1])**2
        +(zcoor[2]-zcoor[1])**2)**0.5
        c=((xcoor[2]-xcoor[0])**2+(ycoor[2]-ycoor[0])**2
        +(zcoor[2]-zcoor[0])**2)**0.5
        p=(a+b+c)/2
        r=((p-a)*(p-b)*(p-c)/p)**0.5
        surf=p*r
        totsurf=totsurf+surf
        triacount=triacount+1
```

Calculating the area is a matter of straightforward mathematics with triangles, just below we split the quadrangles into two triangles.

```
    elif tria==4:
        for k in range (0, 4):
            nodeid= mesh.co[plateid][k]
            nodecoord= mesh.cn[nodeid]
            xcoor[k]=int(mesh.cn[nodeid,0])
            ycoor[k]=int(mesh.cn[nodeid,1])
            zcoor[k]=int(mesh.cn[nodeid,2])
        a1=((xcoor[1]-xcoor[0])**2+(ycoor[1]-ycoor[0])**2
        +(zcoor[1]-zcoor[0])**2)**0.5
        b1=((xcoor[2]-xcoor[1])**2+(ycoor[2]-ycoor[1])**2
        +(zcoor[2]-zcoor[1])**2)**0.5
        c1=((xcoor[2]-xcoor[0])**2+(ycoor[2]-ycoor[0])**2
        +(zcoor[2]-zcoor[0])**2)**0.5
        p1=(a1+b1+c1)/2
        r1=((p1-a1)*(p1-b1)*(p1-c1)/p1)**0.5
        surf1=p1*r1
        a2=c1
        b2=((xcoor[3]-xcoor[2])**2+(ycoor[3]-ycoor[2])**2
        +(zcoor[3]-zcoor[2])**2)**0.5
        c2=((xcoor[3]-xcoor[0])**2+(ycoor[3]-ycoor[0])**2
        +(zcoor[3]-zcoor[0])**2)**0.5
        p2=(a2+b2+c2)/2
        r2=((p2-a2)*(p2-b2)*(p2-c2)/p2)**0.5
        surf2=p2*r2
        totsurf=totsurf+surf1+surf2
        quadcount=quadcount+1
avsurf=totsurf/(triacount+quadcount)
statis.write('group 2D             : %s\n' % listgm2D[i])
statis.write('number of TRIA3 elem : %d\n' % triacount)
```

```
    statis.write('number of QUAD4 elem : %d\n' % quadcount)
    statis.write('total area           : %d\n' % totsurf)
    statis.write('average area         : %d\n' % avsurf)
    statis.write('\n')
  #in Python it is not strictly necessary to close a file
  #however
  statis.close()
```

The print out in the file looks like this:

```
group 1D          : topbeam
number of element : 160
total length      : 3960
average length    : 24

group 1D          : vertb
number of element : 32
total length      : 800
average length    : 25

group 1D          : mast
number of element : 80
total length      : 2000
average length    : 25

group 2D             : panel
number of TRIA3 elem : 689
number of QUAD4 elem : 371
total area           : 386000
average area         : 364
```

This example applies only to line elements with 2 nodes, on a `SEG2` or to `TRIA3` and `QUAD4` elements, however it can easily be adapted to another geometry[1].

[1] Any experienced programmer may find this bit of code not very much optimized, and he will be right, I left it this way so the underlying data structure is more easily understood!

CHAPTER 18

Getting more from post-processing

In this chapter, we give some hints about how to to get more from a result:

- how to manipulate tables to get new result values;
- how to rename components fields;
- how to add node coordinates to a result;
- how to print a cleaner ASCII result file
- how to create a mesh from a displacement field;
- how to read a result and enhance it;
- how to calculate forces, stresses and reactions in version 10.

18.1 Manipulating results with TABLE

Using tables allows to extract some values from a result, order them, combine them and more, to finally print them either as ASCII result or in *.med* format. Here we do some examples with the files of *frame3*.

18.1.1 Printing only a few parameters

In the previous examples we requested the print out of the mass of the model. We printed the default set of data which includes: position of barycenter, quadratic moments and many, more or less useful values. Adding a single line, NOM_PARA, in the command file restricts the print out, for example[1]:

```
IMPR_TABLE (
  TABLE=masse,
  NOM_PARA=('LIEU','MASSE',),
  FORMAT_R='1PE12.3',
)
```

gives the following print out, with only the group names and their respective mass:

```
#masse
 LIEU                    MASSE
 topbeam                    2.736E-03
 mast                       2.736E-03
 massN                      1.000E-02
```

18.1.2 Getting the maximum value of a field

In this first example, we extract the minimum value of the force field SIEF_ELNO component N in all the beam groups (individually for each group). In addition, we print all the other SIEF_ELNO components for this element in the *.resu* file:

```
#declare the tuple which contain the group name
#note index 0 is not used
#nvar number of items in vari
nvar=3;
```

[1] We get the actual name of the parameters from a full printout.

```
vari=[None]*(nvar+1);
#specify the group name
vari[1]='topbeam';
vari[2]='vertb';
vari[3]='mast';
#intialize a group name which holds the element
#where the value is minimum
mailncr=[];
```

We first build a table containing the elements where N is extreme, i.e. maximum or minimum.

```
for i in range (1,nvar+1):
  var=vari[i];
  #build a table containing the elements where N is extreme
  #maximum or minimum
  tabminN=POST_RELEVE_T(
    ACTION=_F(
      INTITULE='extreme_N',
      OPERATION='EXTREMA',
      GROUP_MA=(var,),
      RESULTAT=stat,
      NOM_CHAM='SIEF_ELNO',
      NOM_CMP=('N',),
      LIST_INST=(liste,),
    ),
  );
  #eventually print the table to check
  #IMPR_TABLE (TABLE=tabminN,)
```

Then, we order this table in ascending order.

```
  #order this table in increasing order
  #U.4.33.03
  tabminN=CALC_TABLE(
    TABLE=tabminN,
    reuse=tabminN,
    ACTION=(
      _F(
        OPERATION='FILTRE',
        NOM_PARA='EXTREMA',
        VALE_K='MAX',
      ),
      _F(
        OPERATION='TRI',
        NOM_PARA='VALE',
        ORDRE='CROISSANT',
      ),
    ),

  );
  #eventually print the table to check
  #IMPR_TABLE (TABLE=tabminN,)
```

Now, in this table we select the element in which N is minimum, of course it is the first one after the ordering.

```
#select the element where N is minimum
#the first one in the table
thiselem = tabminN['MAILLE',1];
#add it the group of element
mailncr.append (thiselem);
#thisorder is the order at which this value occurs
thisorder = tabminN['NUME_ORDRE',1]
#print all the values of 'SIEF_ELNO' for the element
#where N is minimum
IMPR_RESU(
  MODELE=model,
  FORMAT='RESULTAT',
  UNITE=8,
  RESU=_F(
    RESULTAT=stat,
    NOM_CHAM='SIEF_ELNO',
    MAILLE=thiselem,
    NUME_ORDRE=thisorder,
    SOUS_TITRE=(
      'groupe : ', var, ' -
      numero ordre : ',str(thisorder),'
      N maximum'
    ),
  ),
);
#destroy the concept for the next use of it in the loop
DETRUIRE(CONCEPT=_F(NOM= tabminN),);
#end of the Python loop
```

Finally, we create a group in the mesh holding this element and print the result on this group in the *.med* file.

```
#create the group in the mesh
mesh=DEFI_GROUP(
  reuse =mesh,
  MAILLAGE=mesh,
  CREA_GROUP_MA=(
    _F(
      NOM='mailncr',
      MAILLE=mailncr,
    ),
  ),
);

#print the result in the .med file
IMPR_RESU(,
```

```
    MAILLAGE = mesh,
    FORMAT='MED',
    UNITE=80,
    RESU=_F(
      RESULTAT=stat,
      GROUP_MA='mailncr',
      NOM_CHAM=('SIEF_ELNO',),
      NOM_CMP=('N',),
      NOM_CHAM_MED=('minN',),
    ),
);
```

It is a wise idea to print the intermediate tables when prototyping the problem, this helps in debugging.

With this code, we get the minimum value of N and the INST when this occurs for *each* group.

Putting the group definition outside a loop produces a result for any element in *all* the groups.

18.1.3 Getting values within a range

Now, we extract values lying within a range, a minimum and a maximum. In the previous examples we created the first table with POST_RELEVE_T. In this example we extract the table directly from the result concept. We extract the component SIXX of the field SIPM_ELNO

```
#create the table from the resu 'stat'
#U4.33.02
tab01=CREA_TABLE(
  RESU=_F(
    RESULTAT=stat,
    GROUP_MA=('vpot2'),
    NOM_CHAM='SIPM_ELNO',
    NOM_CMP='SIXX',
  ),
);

#if we print it we can see that it contains many information
IMPR_TABLE (TABLE=tab01,)
```

And, we can see it contains a lot of information so we reduce its content.

```
#we reduce its content
#U.4.33.03
tab01=CALC_TABLE(
```

```
  TABLE=tab01,
  reuse=tab01,
  ACTION=_F(
    OPERATION='EXTR',
    #restricted list of wanted components
    NOM_PARA=('NOM_CHAM','INST','NUME_ORDRE','MAILLE','SIXX',),
  ),
);
```

Finally, we create a new table where the value of SIXX is less than 50 from which we extract the values greater than -50.

```
#create a new table where the value of SIXX is  less than 50
tab02=CALC_TABLE(
  TABLE=tab01,
  ACTION=_F(
    OPERATION='FILTRE',
    NOM_PARA=('SIXX'),
    #LT means Less Than
    CRIT_COMP='LT',
    VALE=50,
  ),
);
#IMPR_TABLE (TABLE=tab02,)

#in this table we extract only the values greater than -50
tab02=CALC_TABLE(
  TABLE=tab02,
  reuse=tab02,
  ACTION=_F(
    OPERATION='FILTRE',
    NOM_PARA=('SIXX'),
    #GT means Greater Than
    CRIT_COMP='GT',
    VALE=-50,
  ),
);
IMPR_TABLE (TABLE=tab02,)
```

18.2 Renaming field's components in a result

Using NOM_CHAM_MED gives a powerful way to rename the fields so as to get nicer and maybe more explicit names, there is also a feature to rename the components in an ASCII file. For example the following bit of code:

1. puts the reaction in a table, in the standard manner;

2. changes the name of the components in the table, it looks nicer to have R for reactions instead of D which reminds of a displacement!

```
rea_sol=POST_RELEVE_T(
    ACTION=_F(
        INTITULE='reactions sol',
        RESULTAT=stat,
        TOUT_ORDRE='OUI',
        GROUP_NO=('sol',),
        NOM_CHAM='REAC_NODA',
        RESULTANTE=('DX','DY','DZ'),
        OPERATION='EXTRACTION',
    ),
);
IMPR_TABLE (TABLE=rea_sol,) #just to see the difference
rea_sol=CALC_TABLE(
    reuse =rea_sol,
    TABLE=rea_sol,
    ACTION=(
        _F(OPERATION='RENOMME',NOM_PARA=('DX','RX',),),
        _F(OPERATION='RENOMME',NOM_PARA=('DY','RY',),),
        _F(OPERATION='RENOMME',NOM_PARA=('DZ','RZ',),),
    ),
);
IMPR_TABLE (TABLE=rea_sol,)
```

18.3 Adding node coordinates in a result

The following lines add the nodes coordinates to a result of reactions, this may help a third party[1] to better understand the results.

```
IMPR_RESU(
    MODELE=model, FORMAT='RESULTAT',
    RESU=_F(
        NOM_CHAM='REAC_NODA',
        GROUP_NO=('sol',),
        RESULTAT=stat,
        NOM_CMP = ("DX","DY","DZ"),
        IMPR_COOR='OUI',
    ),
);
```

[1] For example the concrete engineer in charge of the ground connection for a steel frame building.

18.4 Printing a cleaner ASCII result file

Throughout the examples we use this command:

```
IMPR_RESU(
    MODELE=model, FORMAT='RESULTAT',
    RESU=_F(
        ................
    ),
);
```

to print the *.resu* file directly on LU=8, it is very quick, but in this case the file contains a lot of information, including warnings.

Redirecting to a different file like:

```
IMPR_RESU(
    MODELE=model, FORMAT='RESULTAT',
    UNITE=12,
    RESU=_F(
        ................
    ),
);
```

produces much cleaner a file with just only the results, with no warnings[1].

18.5 Creating a mesh on a deformed shape

The following piece of code allows to write in a med file a deformed mesh resulting from a previous calculation. This new mesh can be used as an entry in any subsequent calculation.

Used this way it would, of course, be considered as a stress free mesh!

```
statnl=STAT_NON_LINE(
   ............
);
#extract the displacements from statnl calculation
defshape=CREA_CHAMP(
    RESULTAT=statnl,
    INST=6,
    OPERATION='EXTR',
    NOM_CHAM='DEPL',
    TYPE_CHAM='NOEU_DEPL_R',
);
```

[1] The same applies to IMPR_TABLE.

```
#make the new mesh
meshdef=MODI_MAILLAGE(
    reuse=mesfdef,
    MAILLAGE=mesh, #the original mesh
    DEFORME=_F(
        OPTION = 'TRAN' ,
        DEPL = DEPL,
  ),
);
#save it
IMPR_RESU(
    FORMAT='MED',
    UNITE=71,
    RESU=_F(MAILLAGE=meshdef,),
);
```

18.6 Reading (and enhancing) a result

There is an alternative to using POURSUITE in order to enhance results. In this section, we see how to read a MED result file created in a previous calculation.

Taking the example of *frame1*, we start by modifying the command file like this: we comment the line containing SIPM_ELNO in CALC_CHAMP and IMPR_RESU ... FORMAT='MED' so the calculated concept 'stat' is saved without this option[1], we can check this by running the calculation and opening the *.med* file.

Then, we create the following command file reading the med result file just created and enhance the results.

```
DEBUT();

#we read the mesh of the study
#this mesh concept is used in the subsequent CALC_CHAMP
mesh=LIRE_MAILLAGE(
    INFO=1,
    UNITE=20,FORMAT='MED',
);

#we need also the MODELE concept
model=AFFE_MODELE(
    MAILLAGE=mesh,
    AFFE=(
        _F(
            GROUP_MA=('topbeam','mast',),PHENOMENE='MECANIQUE',
```

[1] Just like we had forgotten these options at first!

```
            MODELISATION='POU_D_T',
        ),
        _F(
            GROUP_MA=('massN',),PHENOMENE='MECANIQUE',
            MODELISATION='DIS_T',
        ),
    ),
);

#and the field CHAM_MATER  concept
steel=DEFI_MATERIAU(ELAS=_F(E=210000.,NU=0.3,RHO=8e-9),);
material=AFFE_MATERIAU(
    MAILLAGE=mesh,
    AFFE=_F(GROUP_MA=('topbeam','mast',), MATER=steel,),
);

#and the element characteristics
elemcar=AFFE_CARA_ELEM(
    MODELE=model,
    POUTRE=(
        _F(
            GROUP_MA=('mast',),SECTION='RECTANGLE',
            CARA=('HY','HZ','EP',),VALE=(40, 20, 1.5,),
        ),
        _F(
            GROUP_MA=('topbeam',),SECTION='RECTANGLE',
            CARA=('HY','HZ','EP',),VALE=(40, 20, 1.5,),
        ),
    ),
    DISCRET=_F(GROUP_MA='massN', CARA='M_T_D_N',VALE=(.01),),
);
```

And, now the bit of code producing the results.

```
#here we read the EVOL_ELAS result concept
#from the previous calculation
#giving it the name "resur" for result read
resur=LIRE_RESU(
    TYPE_RESU='EVOL_ELAS',
    UNITE=21,
    FORMAT='MED',
    MODELE=model,
    #MAILLAGE=mesh,
    FORMAT_MED=(
        _F(NOM_CHAM='DEPL',NOM_RESU='stat'),
        _F(NOM_CHAM='SIEF_ELNO',NOM_RESU='stat'),
    ),
    TOUT_ORDRE='OUI',
);

#here we create a new result concept name resu
#based on "resur" read above
resu=CALC_CHAMP(
    #with LIRE_RESU next line is not necessary
```

```
    #as we create a new resu
    #reuse =resu,
    #with LIRE_RESU next line is mandatory
    MODELE=model,CHAM_MATER=material, CARA_ELEM=elemcar,
    RESULTAT=resur,
    CONTRAINTE=(
        'SIEF_ELNO',
        'SIPO_ELNO',
        'SIPM_ELNO',
    ),
    FORCE=('REAC_NODA'),
);

#print new med resu
IMPR_RESU(
    MODELE=model, FORMAT='MED', UNITE=80,
    RESU=_F(
        GROUP_MA=('topbeam','mast',),
        RESULTAT=resu,
        NOM_CHAM=(
            'DEPL',
            'SIEF_ELNO',
            'SIPO_ELNO',
            'SIPM_ELNO',
            'REAC_NODA',
        ),
    ),
);

FIN()
```

In LIRE_RESU, we comment the MAILLAGE=mesh line, as the ELNO type field needs only the MODELE concept, some other results may need a MAILLAGE concept.

We need to carefully study what is done in CALC_CHAMP with the name of the concepts:

- we do not use reuse, as there is no *previously calculated result* and we have to use the lines:

  MODELE=model,CHAM_MATER=material,
  CARA_ELEM=elemcar,.

- we add SIPM_ELNO in CALC_CHAMP to calculate this option and in IMPR_RESU ... FORMAT='MED', to print in a med file.

We now create in Astk a study looking like figure 18.1.

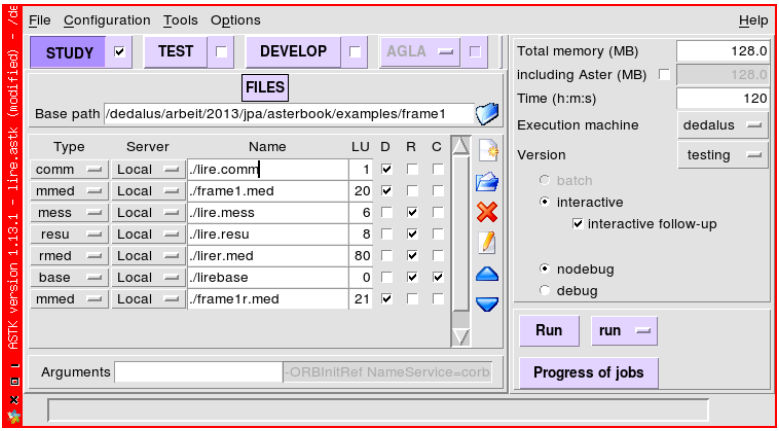


FIGURE 18.1: Astk example for LIRE_RESU study

Note: the file *frame1r.med* is set as data D as we are using it as an input.

After running the study and opening the *lirer.med* file, we find an entry for SIPM_ELNO.

With this method, we have to reload all the data relevant to the study, mesh, model, material and so on, to be able to perform a calculation, the only omitted step is the call to the solver, MECA_STATIQUE, or STAT_NON_LINE, or any other.

This is not as easy, or immediate, as a POURSUITE to handle large studies.

On the other hand it allows to read result concepts produced by IDEAS or ENSIGHT for example. All this is described in the U7.02.01 documentation. Finally, we should notice that it is not not possible to read **ALL** the concepts produced by *Code_Aster*.

18.7 Post-processing in version 10

For the versions older than 11, commands to calculate stresses and forces were a bit different, operators CALC_ELEM and CALC_NO were used.

Here follows how the command file of the example *frame3* in chapter 8.2 should be written to get the same results:

```
stat=CALC_ELEM(
  RESULTAT=stat,
    reuse =stat,
    OPTION=(
        'SIEF_ELNO','SIPO_ELNO','SIPM_ELNO',
        'SIGM_ELNO',
        #'SICO_ELNO', #SICO en non linear
    ),
);

stat=CALC_NO(
  RESULTAT=stat,
    reuse =stat,
    OPTION=('REAC_NODA',),
);
```

Keyword SIGM_ELNO is here to calculate the stress in the plate element, but only in the neutral plane. To calculate them on one of the faces we use a new concept statsup with CALC_ELEM:

```
statsup=CALC_ELEM(
  RESULTAT=stat,
    GROUP_MA=('panel',),
    REPE_COQUE=_F(
        GROUP_MA=('panel',),
        NIVE_COUCHE='SUP',
    ),
    OPTION=('SIGM_ELNO',),
);

statsup=CALC_NO(
  RESULTAT=statsup,
    reuse =statsup,
    OPTION=('SIGM_NOEU',),
);
```

CHAPTER 19

Handling *Code_Aster*, bits and pieces

In this chapter, we review some hints for a better use of *Code_Aster* :

- how to deal with multiple FORCE_POUTRE;
- how to convert a mesh to or from another format;
- how to launch a study from a terminal;
- how to use multiple instances of ASTK;
- why not to alarm when there is an “alarm”;
- how to benefit from the use of the keyword INFO.

19.1 Dealing with multiple FORCE_POUTRE

As we have discussed earlier the following attempt of using two instances of FORCE_POUTRE in two different AFFE_CHAR_MECA

```
selfwght=AFFE_CHAR_MECA(
    MODELE=model,
    PESANTEUR =_F(
        GRAVITE=13500,
        DIRECTION=(0,0,-1),
        GROUP_MA=('topbeam','mast','massN',),
    ),
);
...........
cr=AFFE_CHAR_MECA(
    MODELE=model,
    FORCE_POUTRE=_F(GROUP_MA=('topbeam',),FZ=-0.1,),
);
```

raises the following error:

```
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! <S> Exception utilisateur levee mais pas interceptee.     !
! Les bases sont fermees.                                   !
! Type de l'exception : error                               !
!                                                           !
!  Le chargement contient plus d'une charge repartie.       !
!  Le calcul n'est pas possible pour les modeles de poutre. !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
```

Code_Aster fails despite the fact that the displacements are calculated[1].

However the following gives a result with forces and stresses. The load cases are named `inst_x`, where x takes the same value as the `INST` of example *frame1.comm*, in chapter 4.9

```
inst_3=AFFE_CHAR_MECA(
    MODELE=model,
    PESANTEUR =_F(
        GRAVITE=10000*1.35,DIRECTION=(0,0,-1),
        GROUP_MA=('topbeam','mast','massN',),
    ),
);
...........
inst_5=AFFE_CHAR_MECA(
    MODELE=model,
        PESANTEUR =_F(
        GRAVITE=10000*1.35,DIRECTION=(0,0,-1),
        GROUP_MA=('topbeam','mast','massN',),
    ),
    FORCE_NODALE=_F(GROUP_NO=('loadS',),FZ=-135,),
    FORCE_POUTRE=_F(GROUP_MA=('topbeam',),FZ=-0.1*1.5,),
);
```

[1] In fact writing `OPTION='SANS',` in `MECA_STATIQUE` allows to print a result containing only the displacements.

```
.....
stat_3=MECA_STATIQUE(
    MODELE=model,
    CHAM_MATER=material,
    CARA_ELEM=elemcar,
    EXCIT=(
        _F(CHARGE=ground,),
        _F(CHARGE=inst_3,),
    ),
);
.....
stat_5=MECA_STATIQUE(
    MODELE=model,
    CHAM_MATER=material,
    CARA_ELEM=elemcar,
    EXCIT=(
        _F(CHARGE=ground,),
        _F(CHARGE=inst_5,),
    ),
);
```

This method uses several MECA_STATIQUE[1] with the following drawbacks:

- there are as many solving of the problem as there are MECA_STATIQUE which increases the calculation time;
- FONC_MULT cannot be used at best anymore and the multiplications, if any must be made in the individual AFFE_CHAR_MECA.

This method is quite cumbersome but may help if CPU time is not critical and a true precise FORCE_POUTRE is required.

19.2 Converting mesh

It is very easy to convert a mesh to another format. To do this, in **ASTK**:

- set the Base path to the directory where the mesh to be converted is sitting;
- write a line entry with the mes file to be converted;
- go to menu item Tools > Mesh converter;

[1] Which could easily included in a Python loop, including as well all the post-processing commands.

- choose the directory where yo want the new mesh to be created in Result mesh;
- optionally set a name;
- select the Format;
- push Ok;

and after a few more clicks the new mesh is created, figure 19.1 shows the **ASTK** window.

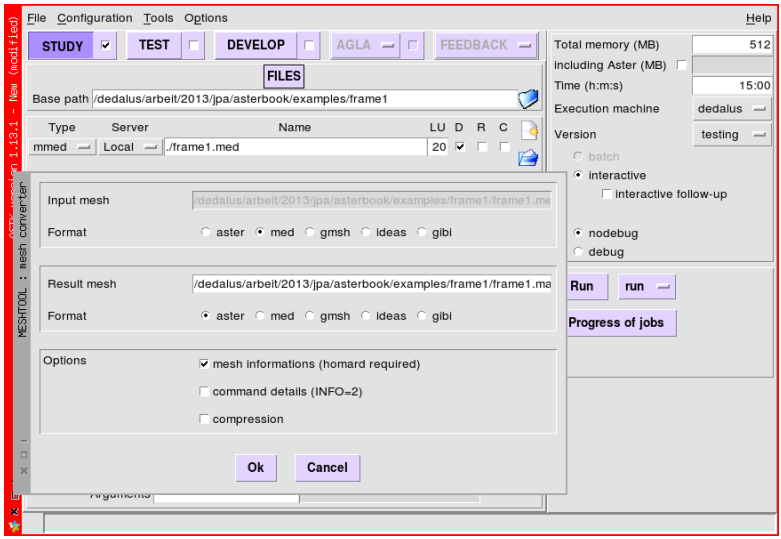


FIGURE 19.1: **ASTK** set to convert a mesh

There is no witchcraft in this tool and the next lines in a *.comm* file will do the same:

```
mesh=LIRE_MAILLAGE(UNITE=20, FORMAT='MED' ,);

IMPR_RESU(
  FORMAT='OUTPUT\_FORMAT',
  UNITE=71,
  RESU=_F(MAILLAGE=mesh,),
);
```

19.3 Launching from terminal

It is possible to launch a study just simply from a terminal.

When we launch a study from ASTK a file named *study_name.export* is created. If we open this file in a text editor we can see that it contains, first a series of parameters, some paths and at the end a list of the files of the study with the LU and R, C or RC state of each file.

The following command typed in a terminal launches this study:

```
/opt/aster/bin/as_run /my_directory/frame3.export
```

- assuming */opt/aster* is the *Code_Aster* installation directory;
- assuming *my_directory* is the study directory name;
- assuming *frame3* is the study name.

Creating a new *.export* file by hand in a text editor with the right parameter for the new study is just enough to run a study. Detailed explanation about this matter can be found in U1.04.00 or more generally in the whole U1.* documentation.

More! Writing a script like this:

```
#!/bin/sh
/opt/aster/bin/as_run /directory_study_1/study_1.export
/opt/aster/bin/as_run /directory_study_2/study_2.export
............
/opt/aster/bin/as_run /directory_study_43/study_43.export
```

saving it somewhere, making it executable and launching it from a terminal would mimic a batch behavior and solve 43 studies one after the other!

19.4 Multiple ASTK configurations

The following command:

```
/opt/aster/bin/astk ––rcdir \$HOME/.astkrc–114
```

launches ASTK with a configuration described in the directory *$HOME/.astkrc-114*.

This allows to have several configurations for each of the ASTK instances in case of multiple *Code_Aster* installations, for example. If the *$HOME/.astkrc-114* does not exist it will created, with default options, on the first launch. The default ASTK configuration file is *$HOME/.astkrc*[1].

19.5 Alarming about 'alarme'?

Very often we can see many items like this one in the *.mess* file:

```
<A> <here a name>
...................
Ceci est une alarme. Si vous ne comprenez pas le sens de cette
alarme, vous pouvez obtenir des resultats inattendus !
```

Whose translation is "This is a warning. If you do not understand the meaning of this warning, you may get unexpected results". This means that we should know what we are doing here.

In fact, in this context, the exact translation of the french word "alarm" is warning, thus the alarm is not so alarming!

Yet we should always tend towards zero warnings by making the necessary changes suggested by these messages.

[1] In the Unix world, he leading "dot" in a file name means a hidden file or directory, it thus may not be visible at fist with the default settings of a graphical window manager.

19.6 Keeping informed with INFO

Almost any *Code_Aster* command supports the keyword INFO, it can take two values:

- with INFO=1, a standard set of information is printed in the *.mess* file[1];
- with INFO=2, a more comprehensive set of information is printed in the *.mess* file.

It is always a wise idea to have a look at what is printed in the *.mess* file for a given command.

For any command, in addition to the keywords stated by the user in the *.comm* file, *Code_Aster* adds a few other keywords considered as default. Looking at what has been added is useful in two ways:

- at the learning stage (for a given command) is helps to understand what *Code_Aster* uses as parameter for the command;
- with more experience it gives a good hint of what could be altered to produce the expected, or a better, result.

For example setting INFO=2 within a LIAISON_* command allows to see the equations and the eliminated DOF actually used!

Another example is with the commands LIRE_MAILLAGE, MODI_MAILLAGE, CREA_MAILLAGE, DEFI_GROUP... With INFO=1 a decent set of information is printed in the *.mess* file which help to check that:

- the read mesh is what we expect;
- the expected groups do exist, with their expected number of elements and their right name;
- and much more information.

[1] INFO=1 is the default value and it is not possible to print no information!

With INFO=2 the level of information goes more into the details:

- for every node, its coordinates;
- for every element its connectivity;
- and so on.

Here is a example of the LIRE_MAILLAGE with INFO=1 for *frame3* example:

```
  # Commande No :  0002
Concept de type : maillage_sdaster
  # ------------------------------------------------------------------
  mesh = LIRE_MAILLAGE(INFO=1,
                       FORMAT='MED',
                       UNITE=20,
                       INFO_MED=1,
                       VERI_MAIL=_F(APLAT=1.E-3,
                                    VERIF='OUI',),
                       )
====== VERIFICATION DU MAILLAGE ======

  MAILLE POI1 M1      INCLUSE DANS UNE AUTRE
  MAILLE POI1 M2      INCLUSE DANS UNE AUTRE
  MAILLE POI1 M3      INCLUSE DANS UNE AUTRE
  MAILLE POI1 M4      INCLUSE DANS UNE AUTRE
  MAILLE POI1 M5      INCLUSE DANS UNE AUTRE

------------ MAILLAGE mesh - IMPRESSIONS NIVEAU  1 -----------

MED file generated by Gmsh

NOMBRE DE NOEUDS                        907

NOMBRE DE MAILLES                      1439
                              POI1                 5
                              SEG2               224
                              TRIA3              842
                              QUAD4              368

NOMBRE DE GROUPES DE MAILLES             12
                          loadS                            1
                          massN                            1
                          groundS                          1
                          groundN                          1
                          oripanel                         1
                          topbeam                        160
                          hinge                            4
                          vertb                           40
                          mast                            20
                          panelS                         842
```

```
                          panelN                                  368
                          pan                                      40

 ------------------------------------------------------------------------


   # Fin commande No : 0002   user+syst:
 0.03s (syst:        0.00s, elaps:        0.03s)
   # ------------------------------------------------------------
```

This valuable information should be read at first if something goes wrong.

In the same manner one may use `DEBUT (IMPR_MACRO='OUI')` to print in details what is happening inside each macro command.

APPENDIX A

Living with good practice

When all else fails, read the directions.

Allen's axiom.

Before moving to some technical appendixes we review now a few advices of good practice:

1. Put all the files related to a given study in a single directory [a].
2. In this directory, do NOT allow file names with special character, like space or blank, which may prevent reading other files in the directory.
3. In the source files, command, Gmsh or Salome script, write more comment than we think necessary, as we will probably have forgotten why we did "this like that" when we re-open the file in a few months. And even more if somebody else has to use them.
4. When working with Gmsh, keep an eye on the 'Message Console', some strange behaviors are probably traced in it.

[a] In which we have read, write, execute permissions.

5. Check the geometry overall dimensions, with CAD imported file there maybe a confusion between meter and millimeter.
6. Check the mesh for double nodes or double elements.
7. Use a coherent system of units throughout the study.
8. As much as possible do some hand calculation to guess [and check] some of the result's key values.
9. Check the *Code_Aster* output results for mass, of the whole model, or group by group, with the expected values.
10. Check the coherence of the sum of the reactions with the supposed applied external loads for the various load cases.
11. Always read carefully the *.mess* file output:
    - if things went wrong, hints for the solution are lying in it;
    - if we got a result, the explanation why it may be meaningless often lies in it;
    - more generally, check all the warning messages lines, they begin with <A>.
12. Do not take for granted the colorful pictures and the associated scalar bars produced by the post-processing tool, always print in the *.resu* file the `VALE_MAX` and `VALE_MIN` for the same field and component, as we expect the post-processing tool to show the *Code_Aster* calculated values[a]. If it does not, at first try to find out how and why the post-processor is misused.
13. Make sure that the *.mess*, *.resu* files we are reading in a text editor, and the *.med* file we are viewing in the Post-pro module really belongs to the same analysis of the same problem.
14. Before blaming the software, try to find out where and why things went wrong.
15. And if a bug is suspected[b] report it on the *Code_Aster* forums.
16. Be patient and obstinate.

[a] And experience shows it is not always the case!
[b] Yes it happens! A bug is commonly described as a "feature" by many developers.

APPENDIX B

Using Gmsh, tips and tricks

This appendix reviews some hints for a better use of Gmsh:

- how to really view what we want;
- how to use powerful Gmsh Plugins, to create a view of a composite result or to animate a mode shape;
- how to properly create and orient surfaces;
- a quick introduction to the legacy *.pos* Post-pro file.

2.1 Viewing the right results

We may get puzzled by the many mouse clicks necessary to view what we want in Gmsh Post-processing module, here are some hints.

First of all it is a good idea to go in Tools > Options > Mesh > Visibility and un-check everything so the post-processing view is not be polluted by

some mesh views[1]. Pushing Alt + M is also useful to hide the mesh (or click the M in the status bar).

In the **Gmsh** window only the checked views, numbered from [0] up are visible, for example in figure B.1 only view [2], named 'stat___DEPL' is checked and visible in the main window.

In the window Tools > Options all the options chosen in the right tab apply to the View[x] highlighted in the left-hand list.

2.1.1 Viewing ELNO type fields

To view one specific component in an ELNO type field (like forces or stresses) the following sequence must be followed:

1. in the Visibility tab pull the lower left list to Force scalar;
2. in the box immediately to the right type in the field Id (for a SIPO_ELNO field Id is 0 for SN, 4 is for SMFY for example);
3. in the General tab pull the list Range mode to Custom;
4. and push the Min and Max buttons to refresh the display with the proper component values.

We should not forget the Min and Max buttons to be sure that the displayed field and the scalar bar are matched.

2.1.2 Viewing vector type fields

To view the deformed shape:

1. in the Visibility tab pull the lower left list to Force vector;
2. in the Aspect tab pull the Vector display list to Displacement;
3. choose a significant value for Displacement factor.

And to view reactions as vector arrows:

[1] Though sometimes it is useful to see the mesh or some groups in it.

1. in the Visibility tab pull the lower left list to Force vector;
2. in the Aspect tab pull the Vector display list to Arrow;
3. pull the Glyph location list to Vertex to display the arrow on the nodes.

2.1.3 Viewing scalar fields on deformed shapes

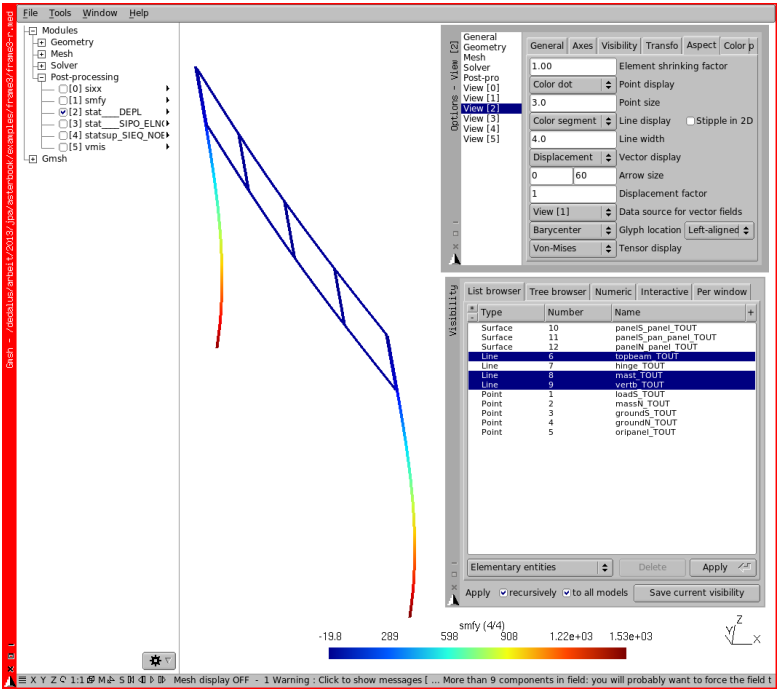


FIGURE B.1: Bending stress on deformed shape

The use of Data source list to View[x] allows to draw on the specified selected view a field coming from the Data source view, for example a stress field on a deformed shape. For example view B.1 shows as active view View[2] which contains a field of DEPL type with a large displacement factor, to show the deformed shape, Data source is View[2] which

is `smfy` bending stress. The Show scalar value is unchecked for View[2] and in addition in Tools » Visibility the view is restricted to some groups of elements.

2.2 Using Plugins

2.2.1 For creating and viewing a composite result

In this section, we see how Gmsh plugins can be used to view new results calculated from the existing ones. We create a result which is the vector of the displacement in the horizontal plane only, for our example *frame4*.

First of all, we add this in the command file so as to create a view for displacement component DX and another one for DY.

```
IMPR_RESU(
  MODELE=model, FORMAT='MED', UNITE=80,
  RESU=(
  _F(
    GROUP_MA=('topbeam','vertb','mast','panel',),
    RESULTAT=stat,NOM_CHAM=('DEPL',),
  ),
  _F(
    NOM_CHAM='DEPL',NOM_CMP=('DX',),NOM_CHAM_MED='dx',
    RESULTAT=stat,
  ),
  _F(
    NOM_CHAM='DEPL',NOM_CMP=('DY',),NOM_CHAM_MED='dy',
    RESULTAT=stat,
  ),
.........................
```

Once the calculation is made, we open the med result file in Gmsh

In the tree of the **Gmsh** window, we RMB click on the view named dx » Plugins,

- then in the pull down list, select MathEval,
- on the right-hand side frame, go down to the View field, we type 0, which is the number of the view where the component DX is displayed[1],

[1] It could well be a different number.

- and for OtherView field we choose 1, where the component DZ is displayed
- in the top field write the following equation `Sqrt(v0^2+w0^2)` which stands for $\sqrt{DX^2 + DY^2}$ where DX in the 'View' v0=View[1], DY in the 'OtherView' w0=View[2].

then Run[1] .

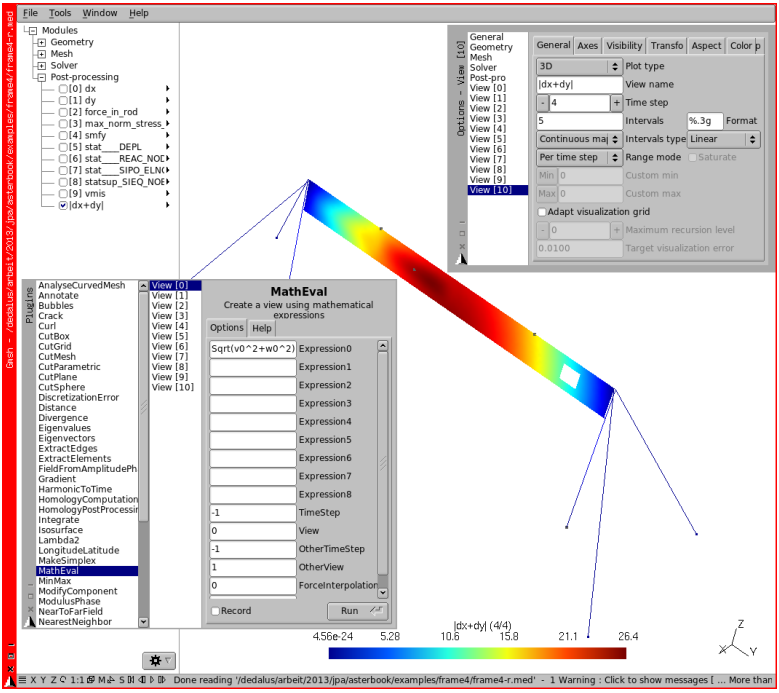


FIGURE B.2: MathEval Gmsh plugin in action

A new View, number [10], is created at the end of the list, it contains a field with the value we have specified, we can change its name in Options > View general > View name.

[1] We left the other fields by default, -1 means all, -1 for TimeStep means making the evaluation for all the existing values of TimeStep.

The same result could be obtained by typing View[0], using DEPL as data source and the formula `Sqrt(v0^2+v1^2)`.

Figure B.2 sums all this up with the result displayed.

If we tick the check box Record in the **Plugins** window a new file is created in the study directory named like the *.med* file with the extra extension *.opt*. Every time we open the med file in Gmsh the Plugin is run so as to enhance the result. This *.opt* file may be edited if we want to get rid of some results, or create new ones[1].

This was just a simple example of Gmsh Plugins the list and the possibilities are large[2].

Maybe just one note: this kind of *.opt* file may be appended to a *.geo* file to pass any valid Gmsh set of instruction to the main file, they will be executed on the fly when the file is read.

2.2.2 For animating a mode shape

A modal analysis result is a displacement result related to time, it would be good to view the deformed shape as an animated view, this is possible in Gmsh. We do it now, for mode 7, at 196 Hz, of the pendulum example of chapter 15.1 .

In Gmsh all the mode shape are saved in a single view, we first extract the right view, due to Gmsh numbering it happens to be at 'Time step 6'. We extract this 'Time step' in a new view using Plugin 'Extract Elements':

- setting TimeStep to 6;
- setting Visible to 1;
- setting Dimension to -1;
- setting View to 0, as we extract from view [0],

and we rename this newly created view [1] 'modes-extract 7'.

[1] If we open this file we can see that it contains the instructions we have just entered in the Gmsh GUI windows, which mean we can tailor the Plugins or create new ones juts by editing this *.opt* file.

[2] There is an Help tab in the Gmsh **Plugins** window briefly describing the Plugin functions and the syntax.

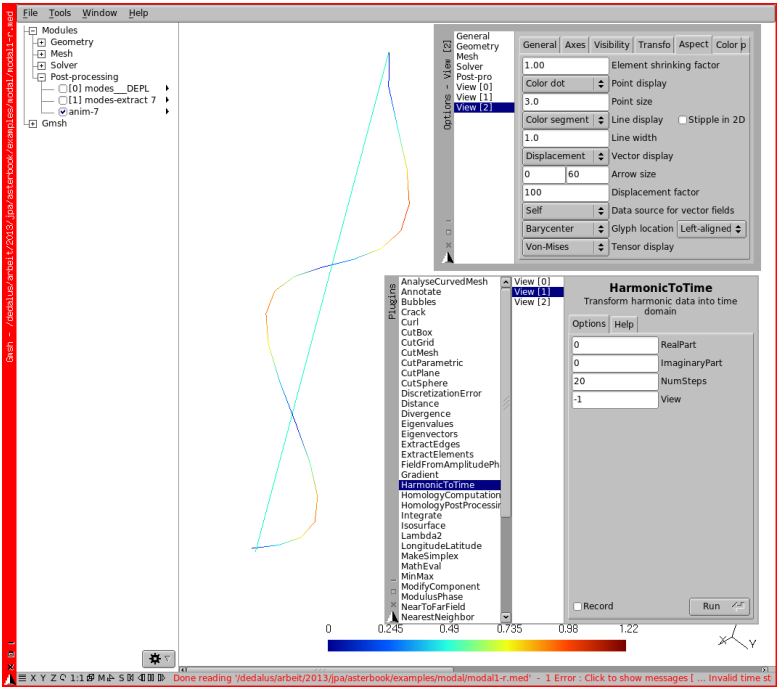


FIGURE B.3: HarmonicToTime setup with a snapshot of animation

And now, we RMB click on the view named [1] modes-extract 7 > Plugins,

- then in the pull down list, select HarmonicToTime;
- on the right-hand side frame, set the values as below:
  - RealPart=0;
  - ImaginaryPart=0[1];
  - NumSteps=20 to set the number of views of the animation;
  - View=-1 to run the plugin on the current view;

[1] We are taking the real part of view 0 and the imaginary part of view 0 as well, mixing up the view numbers would lead to meaningless result display.

- pushing Run creates a new view[1].

Making this view active and pushing the little Play button at the bottom of the window runs the animation as long as Force Vector is selected in Visibility and a large enough Displacement value[2] is set in the Aspect tab.

Figure B.3 shows the plugin's dialog box setup together with a snap shot of the animation.

2.3 Orienting Surfaces

We have stressed earlier the fact that a proper orientation of surface elements is needed, otherwise the loads may not be what expected and/or the stress results may be meaningless. Let's take the example of the *frame3.geo* of example *frame3* to illustrate the way surfaces are oriented in Gmsh as shown on figure B.4[3].

Surface 311 is defined this way:

```
Line Loop(310) = {201, 110, -210, -10};
Plane Surface(311) = {310};
```

The `Line Loop` describes the surface's borders in a sequential way[4] just like we were walking along the border:

- 201 is the first Line;
- 110 is the second one, this couple sets the surface Normals, shown here in red;
- 210 and 10 are the next lines, they are changed of sign because they are walked on the reverse way when describing the border, the orientation of the lines, the Tangents are here shown in yellow.

The next Line `Plane Surface` creates the plane from the loop.

[1] We may explore the Help tag to fully understand the meaning of the parameters.
[2] In this very example a value of 100 provides a sensible view.
[3] On the figure the **Gmsh** tree is teared off to a proper box with Window > Attach/Detach Menu.
[4] It may not be the order in which we clicked on the Lines in the GUI.

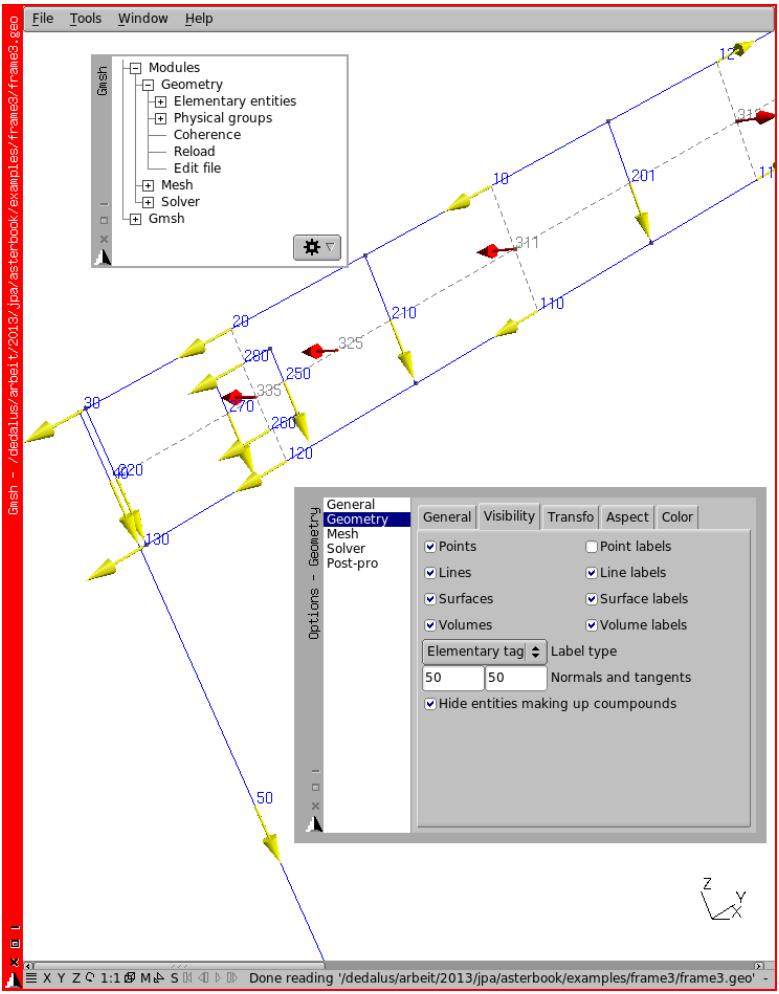


FIGURE B.4: Surface orientation

A loop made with RMB on the lines in the GUI produces a valid surface. On the contrary when we do so in a script, directly in the text editor, it is very easy to create an inverted loop, upon reading by Gmsh this causes an annoying crash!

Hollowed Surface 325 and the little inside surface 335 are created like this:

```
Line Loop(320) = {210, 120, -220, -20};
Line Loop(321) = {280, 270, -260, -250};
Plane Surface(325) = {320, 321};
Plane Surface(335) = {-321};
```

The first loop is just as before, it sets the normal to the surface, the loop 321 sets an inner border for the hollow, and `Plane Surface` sets the surface[1].

The surface 335 is created from the second loop, reverting the loop number, with the minus sign to keep a consistent normal with surface 325, this because the loop 321 is walked along in the direction opposite to loop 320.

We must point here that an extreme care must be taken in orienting the normals to the surfaces, we must choose one policy consistent with the problem and the way the *.comm* file is written.

In fact it is somewhat easier to do in practice than to explain!

And, as we stated in chapter 8 it is a good practice to reorient the surfaces within the `MODI_MAILLAGE` operator.

2.4 Using the legacy Gmsh Post-pro files

When using STANLEY we have seen result windows popping out named **fort33.pos**, this is the legacy Gmsh graphical Post-pro format. The next code abstract would print such a file in LU 37 if appended to the command file of example *frame3*.

```
IMPR_RESU(
    FORMAT='GMSH',
    UNITE=37,
    RESU=(
```

[1] There may be more inner loops than a single one

```
        _F(
            GROUP_MA=('topbeam','vertb','mast','panel',),
            RESULTAT=stat,NOM_CHAM=('DEPL',),
            #next 2 lines to be able to view the deformed shape
            #and or arrow vectors
            #this maybe applied to REAC_NODA,
            #and all vector fields as well
            NOM_CMP=('DX','DY','DZ',),
            TYPE_CHAM='VECT_3D',
        ),
        _F(
            GROUP_MA=('topbeam','vertb','mast','panel',),
            RESULTAT=stat,NOM_CHAM=('DEPL',),
            #to print all individual components
        ),
        _F(
            GROUP_MA=('topbeam','vertb','mast',),
            #if no CMP are specified they are all printed
            #and this can lead to very large files
            RESULTAT=stat,NOM_CHAM=('SIPO_ELNO',),
        ),
        _F(
            GROUP_MA=('topbeam','vertb','mast',),
            RESULTAT=stat,NOM_CHAM=('SIPO_ELNO',),
            NOM_CMP=('SMFY',),
        ),
        _F(
            GROUP_MA=('topbeam','vertb','mast','panel',),
            RESULTAT=stat,NOM_CHAM=('SIPM_ELNO',),
            NOM_CMP=('SIXX',),
        ),
        _F(
            GROUP_MA=('panel',),RESULTAT=statsup,
            NOM_CHAM='SIEQ_NOEU',NOM_CMP='VMIS',
        ),
        _F(
            GROUP_MA=('panel',),RESULTAT=statsup,
            NOM_CHAM='SIEQ_NOEU',
        ),
    ),
);
```

The main advantage of this file format is, it can be scripted at will, “à la Gmsh”.

Its disadvantage: no group selection for viewing can be done within the GUI, it has to be specified in the command file, or scripted.

2.5 Importing Nastran® and other alien files

If we import a Nastran® created data file, *.bdf* or *.dat*, we may find that the groups created do not seem to exist any more. In fact all the properties related to groups are described as “Elementary Entity”.

The code given below allows to restore the groups. If we open it in Gmsh we can see at the end of the translation a *.msh* file appearing on the screen as well as being saved, with all the groups restored.

```
Merge "/my_problem.bdf";

vols[] = Volume "*";
For i In {0 : #vols[] - 1}
  Physical Volume(vols[i]) = vols[i];
EndFor

surfs[] = Surface "*";
For i In {0 : #surfs[] - 1}
  Physical Surface(surfs[i]) = surfs[i];
EndFor

lines[] = Line "*";
For i In {0 : #lines[] - 1}
  Physical Line(lines[i]) = lines[i];
EndFor

Save "/my_problem.msh";
```

Restored groups, but without names, only Ids! Next bit of code added at the beginning of the *.msh* file will name the groups. Then it is just a matter of saving it as a *.med* file

```
\$MeshFormat
2.2 0 8
\$EndMeshFormat
\$PhysicalNames //example for 'frame3'
11              //number of groups
0 7 "groundS"   //0 for point group, 7 is the Id, "group_name"
0 8 "groundN"
0 9 "loadS"
0 10 "massN"
0 11 "oripanel"
1 1 "topbeam" //1 for line group
1 2 "hinge"
1 3 "mast"
1 4 "vertb"
2 5 "panelN"  //2 for surface group
2 6 "panelS"
```

```
\$EndPhysicalNames
\$Nodes
.....................
```

`http://laurent.champaney.free.fr/perso/outils.html` provides a few perl scripts converting fea files. I found the utility *dxf2geo*, converting *.dxf* to *.geo* quite handy. There also are scripts for NASTRAN®, CAST3M®, SAMCEF®, UNV®, ABAQUS®.

2.6 Customizing Gmsh

Menu File > Save Model Options saves a file with the current file name plus the extensions *.opt* containing all the options of the current file. Opening again *current_file_name* loads the options with it.

Menu File > Save Options A default saves the current options as default Gmsh options in *$HOME/.gmsh-options*.

The file *$HOME/.gmshrc* holds the options specific to a session when *$HOME/.gmsh-options* holds wider scope options, in these files, with the help of the manual we can configure Gmsh just as we want.

APPENDIX C

Using discrete elements

This appendix reviews how to use and set the proper values of the discrete stiffness and mass elements, point and line.

3.1 Stiffness matrix

When using discrete element to mimic end release at the end of a beam it is best to use the right[1] values for the stiffness matrix coefficients. This is particularly true if one wants to retrieve the lateral shear force on a lug and pin assembly, for example.

[1] Or as right as possible!

3.1.1 K_TR_D_L element

Here are some formulae to calculate the coefficients for a K_TR_D_L element.

$$K_x = K1_n = \frac{EA_x}{s}$$

$$K_y = K1_z = \frac{12}{\beta s^3} EI_z$$

$$K_z = K1_y = \frac{12}{\alpha s^3} EI_y$$

$$KR_x = K1_x = \frac{GI_x}{s}$$

$$KR_y = K3_y = \frac{3+\alpha}{\alpha s} EI_y$$

$$KR_z = K3_z = \frac{3+\beta}{\beta s} EI_z$$

where

$K_x, K_y, ...KR_x, ...$ are the notations used in #U4.42.01.

$$\alpha = 1 + \frac{12EI_y}{GA_z s^2} \qquad \beta = 1 + \frac{12EI_z}{GA_y s^2}$$

s being the length of the element, E the Young modulus and G the Coulomb modulus[1].

This, applied to a rectangular section, HY = 10, HZ = 100, yields the following value:

$$A_x = HY.HZ = 1000$$

$$A_y = A_z \simeq \frac{2A_x}{3} = 667$$

$$I_x = \frac{HY^3.HZ}{3} = 33333^2$$

[1] The above formulae and the notation are extracted from [Tuma].

[2] HY^3 as $HY << HZ$, this value is more and more false as HY gets closer to HZ to become $2.25 \times H^4$ when $H = HY = HZ$.

$$I_y = \frac{HY.HZ^3}{12} = 833333$$

$$I_z = \frac{HY^3.HZ}{12} = 8333$$

When one wants to relax a rotation around an axis the KR* should be set to zero or a very small value.

We can print the `SIEF_ELNO` for these elements and make the proper dimensioning of lugs and pins by hand.

We should be careful for the choice of the length "s" in calculating the stiffness matrix values, as too high a value may give wrong results, not only in the discrete elements but in the surrounding ones.

This is due to the fact that the stiffness matrix is diagonal as reflected in the `D` of its name. This means that a shear force (or a lateral displacement) at the left end of the element is transmitted as a shear force at the right end but does not produce a bending moment at the right end, neither does it produce a rotation of the left end.

To solve this issue we have to use `K_TR_L` elements.

3.1.2 `K_TR_L element`

With this element we mimic exactly the behavior of of beam element. The drawback is: we have to fill the full matrix with 78 terms!

First of all, we need to compute the terms linking left shear force to right bending moment:

$$K2_y = \frac{6}{\alpha s^2} EI_y$$

$$K2_z = \frac{6}{\beta s^2} EI_y$$

$$K4_y = \frac{3-\alpha}{\alpha s} EI_z$$

$$K4_z = \frac{3-\beta}{\beta s} EI_z$$

Then fill the matrix like this:

```
#numerical value for a rod section, diam 10 mm, length 100 mm
k1n=1.65e5;
k1z=1.23e3;
k1y=1.23e3;
k1x=7.93e5;
k3y=4.06e6;
#k3y=0; #release rotation about local y
k3z=4.06e6;
k2z=6.05e4;
k2y=6.05e4;
#k2y=0; #release rotation about local y
k4y=2.00e6;
#k4y=0; #release rotation about local y
k4z=2.00e6;
elemcar=AFFE_CARA_ELEM(
    MODELE=model,
    DISCRET=(
    _F(
      GROUP_MA=('kt',),
      CARA='K_TR_L',
      VALE=(
        k1n,
        0.0, k1z,
        0.0, 0.0, k1y,
        0.0, 0.0, 0.0, k1x,
        0.0, 0.0,-k2y, 0.0, k3y,
        0.0, k2z, 0.0, 0.0, 0.0, k3z,
       -k1n, 0.0, 0.0, 0.0, 0.0, 0.0, k1n,
        0.0,-k1z, 0.0, 0.0, 0.0,-k2z, 0.0, k1z,
        0.0, 0.0,-k1y, 0.0, k2y, 0.0, 0.0, 0.0, k1y,
        0.0, 0.0, 0.0,-k1x, 0.0, 0.0, 0.0, 0.0, 0.0, k1x,
        0.0, 0.0,-k2y, 0.0, k4y, 0.0, 0.0, 0.0, k2y, 0.0, k3y,
        0.0, k2z, 0.0, 0.0, 0.0, k4z, 0.0,-k2z, 0.0, 0.0, 0.0, k3z,
      ),
      SYME='OUI',
      REPERE='LOCAL',
    ),
    #here for a 'K_TR_D_L' element
    #_F(
      #GROUP_MA=('ktd',),
      #CARA='K_TR_D_L',
      #VALE=(k1n, k1z, k1y, k1x, k3y, k3z,),
      #REPERE='LOCAL',
    #),
  ),
);
```

With this quite lengthy input, the element can be any length and behaves just like a short beam, except on the relaxed DOF, or DOFs. The fancy layout just above tries to display the lower half of the symmetrical ma-

trix in a clean manner. We could even input an unsymmetrical matrix if needed, more about this in U4.21.01[1].

3.2 Mass matrix

In the simpler case we use discrete element, M_T_D_N, to model a point mass, this is quite straightforward and is explained in chapter 4.6 . Furthermore the mass may be given an offset, which is useful in some dynamic calculation, this is fully explained in #U4.42.01.

3.3 Combining both

If one single discrete line element carries at the same time a mass and a stiffness it needs two entries in AFFE_CARA_ELEM. Here is an abstract of code allowing to give a mass to the stiffness line element of *frame2* example:

```
elemcar=AFFE_CARA_ELEM(
    MODELE=model,
    ......................
    DISCRET=(
        _F(
            GROUP_MA=('hinge',),
            CARA='K_TR_D_L',
            #the values used in the previous calculation
            #VALE=(1.00e6,1.00e6,1.00e6,1.00e9,0,1.00e9,),
            #the values calculated with this chapter's formula
            VALE= (3.59e6,6.32e5,3,22e5,2.16e8,0,2.58e8,),
            REPERE='LOCAL',
         ),
        _F(
            GROUP_MA=('hinge',),
            CARA='M_TR_D_L',
            VALE=(M1,M2,M3,M4),
            REPERE='LOCAL',),
        ),
    ),
    ..........................
);
```

Where:

[1] The slightest error in the indexing of the matrix components produces, of course, wrong results.

- M1 is half the mass of the element[1];
- M2 is the rotational moment of inertia of the element about its own axis;
- M3 and M4 are the rotational moments of inertia of the element about the perpendicular axis.

Which gives for a prismatic element:

$$M1 = \frac{m}{2}$$

$$M2 = \frac{m.r^3}{3}$$

$$M3 = M4 = \frac{m.s^2}{12}$$

m being the mass of the element, s its length and r the radius of gyration of the element cross section[2].

In most practical static problems giving the value of M1 is enough.

[1] Yes, half the mass here, don't ask me why!
[2] For more complete computing of the mass matrix we, again refer to [Tuma].

APPENDIX D

Drawing and meshing with Salome

Given the most inappropriate time for something to go wrong, that's when it will occur.

9th Murphy's law.

This appendix gives an initiation to the use of Salome Geometry and Mesh modules.

For this:

- with *frame1* example:
  - we create the geometry and mesh it, in the GUI;
  - we alter the *.comm* file to take into account Salome features;
  - we see how we can "dump" the study to a python script;
- with *frame3* example:
  - we create the geometry from script and in the GUI;
  - we review some hints about group creation;
  - we mesh the geometry;

  - and take a look at how to view beam sections and plate thickness with Eficas;
- we draw and mesh 'part2' of the solid example, using the "NoteBook";
- we review the basic differences between Gmsh and Salome;
- to finish with some hints on how to to mesh an imported CAD file.

4.1 First example with beams

4.1.1 Creating geometry and meshing

At first it is better once Salome is open to go to File > New to create a new study and to save it in a working directory with File > Save As... . Then in the pull down list of Salome we choose Geometry.

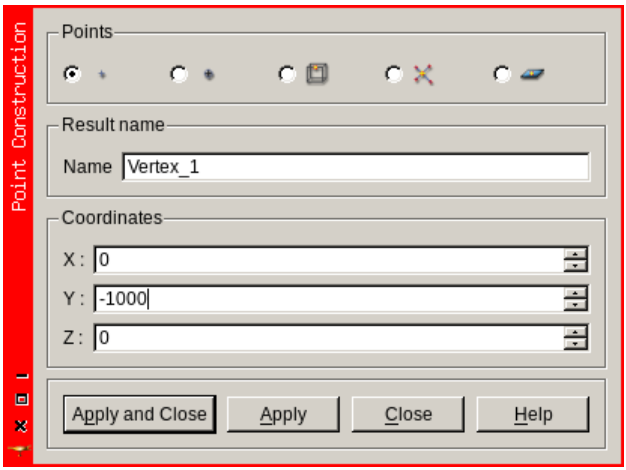


FIGURE D.1: Create a point

In the menu New Entity > Basic > Point, we fill the dialog box like in figure D.1, with (x=0, y=-1000, z=0), click on Apply, repeat for the points (0, -1000, 1000), (0, -500, 1000), (0, 0, 1000) and click Apply and close.

In the menu New Entity > Basic > Line, LMB click on the points to create the three lines, click on Apply after each line is created and Apply and close after the third one.

As we go along we can see the newly created objects in the **Object Browser** window.

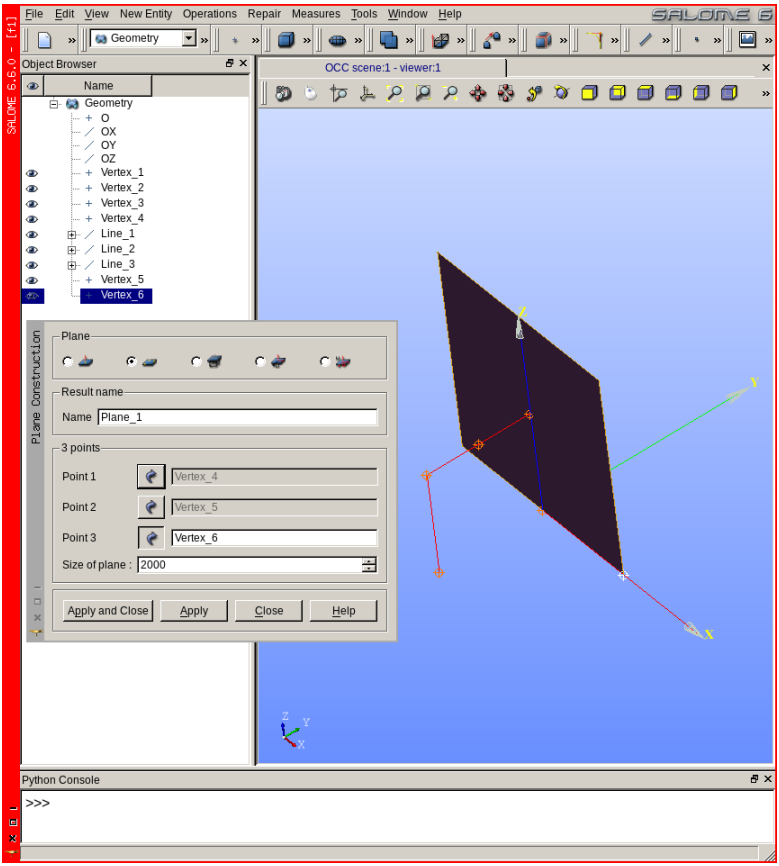


FIGURE D.2: Create a plane

Now, we add two construction points at (0, 0, 0), (1000, 0, 0), create a plane with these two points and the point on top of the frame, with New Entity > Basic > Plane like in figure D.2

And in Operations » Transformation » Mirror Image, select the 3 lines (with ⇧ + LMB), the mirror plane and Apply and close.

In the Object browser we may RMB click on Plane_1 » Hide to get a view restricted to the significant entities. In New Entity » Build » Compound, we select, in the **Object Browser**, or in the graphical window the 6 lines previously created, this creates an object named 'Compound_1'[1]. The creation of this Compound is mandatory as it behaves as a container for the sub entities and the groups, and can be meshed at once[2].

In New Entity » Group » Create, in the dialog box:

- in Shape Type LMB click on the line radio button;
- in Group Name » Name type the name, here 'mast';
- in Main Shape And Sub-shapes:
  - in Main Shape add Compound_1;
  - in Main Shape Selection restriction select No restriction;
  - select one of the mast in the geometry window, push Add, its Id appears in the selection box, like in figure D.3;
  - select the second one, push Add;
  - push Apply and close.

Repeat this for

- a group 'topbeam' for the four top line;
- a group 'groundS' for the point (0, -1000, 0);
- a group 'loadS' for the point (0, -500, 1000).

[1] We could have changed its name.

[2] After building an object it is a wise idea to unroll it in the **Object Browser** to check it is built as we expect.

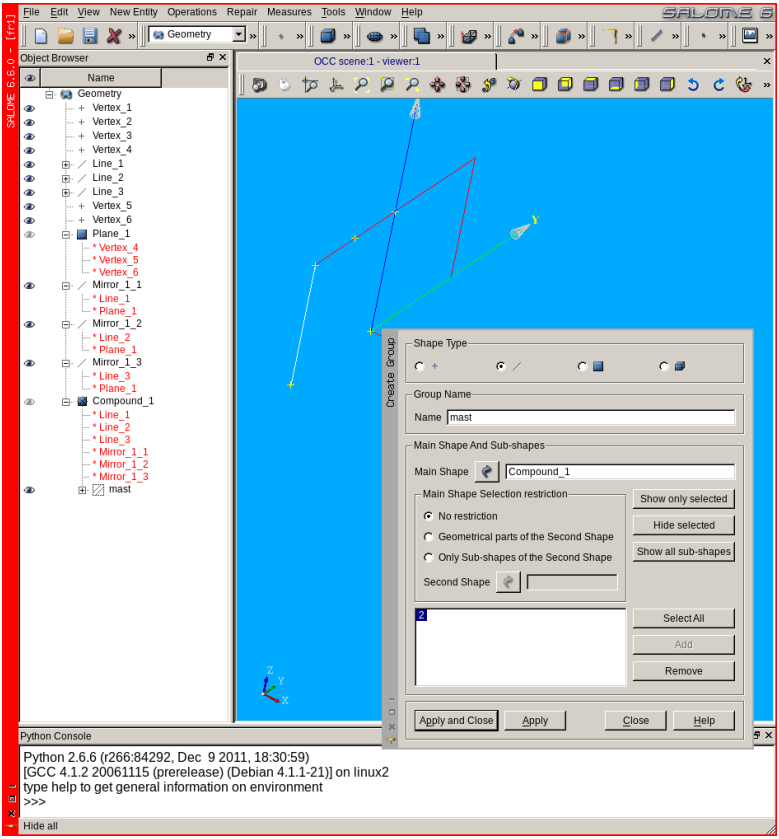


FIGURE D.3: Create group 'mast' in Compound_1

Now is the time to create the mesh, in the **Object Browser**, we go in the module Mesh through the pull down list of Salome, then Mesh > Create Mesh in the dialog box popping up we do as follows:

- in Geometry, we select 'Compound_1' in the Object Browser;
- in the 1D tab:
  - Algorithm to Wire discretisation;
  - we click in the gear wheel like icon right of Hypothesis and choose Max Size which we set at 200 in the following box;

- push Apply and close.

We RMB click in the Mesh_1 just appeared in the Object Browser, with a yellow "!" triangle icon  , by it[1], and we select Compute. A box named Mesh computation succeed should appear with the summary of the mesh and the icon on the left of Mesh_1 turns to something like a little mesh,  [2].

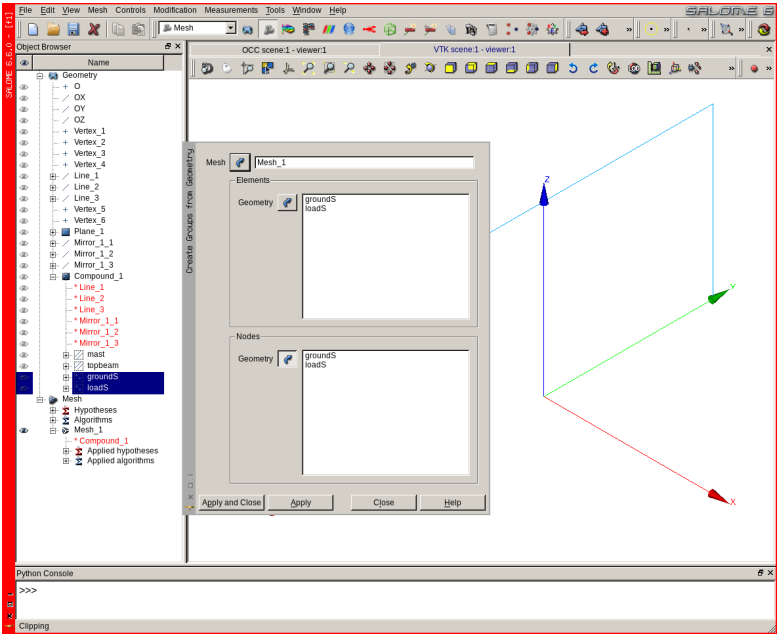


FIGURE D.4: Create mesh groups from geometry groups

And now it is time to create group, for this, we go to Mesh > Create groups from geometry and fill the dialog box like in figure D.4 to create a group from the geometry line groups 'mast' and 'topbeam' and repeat this for the geometry point groups 'groundS' and 'loadS'.

[1] Which means, take care, this object is not meshed yet!
[2] A valid meshed exist.

The job is not really finished as we have no mesh groups for 'massN' and 'groundN'. To create these ones we go to Mesh > Create Group and fill the dialog box like in figure D.5 to create a group for the node at (0, 500, 1000) named 'masseN' and 'groundN' on node (0, 1000, 0).

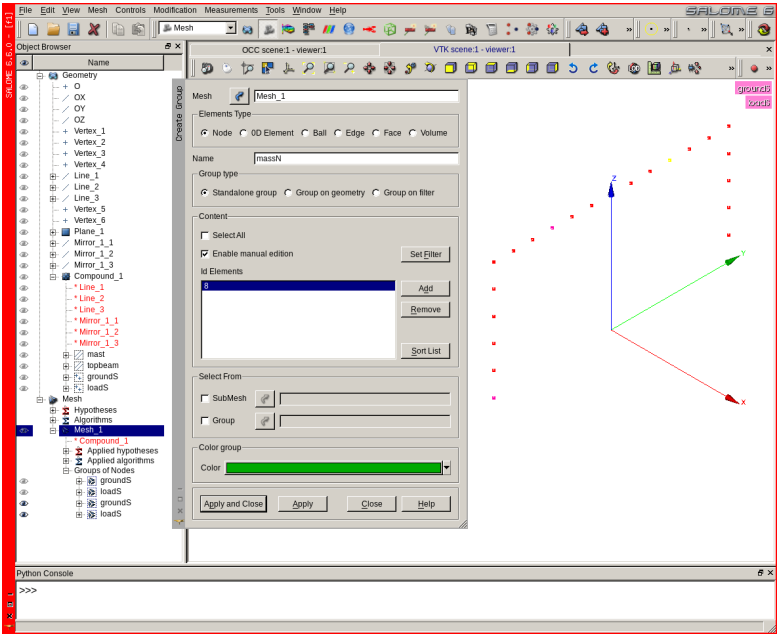


FIGURE D.5: Create mesh groups from mesh entities

At this stage as we just create groups within a given mesh it is not necessary to compute or update the mesh.
We have created groups, in the first place from a geometry entity, on the second place from a mesh entity, obviously the first solution is the more flexible and should always be preferred.
We should note forget to do Modification > Transformation > Merge Nodes to remove the duplicate nodes, figure D.6[1]. In any more complicated mesh a Modification > Transformation > Merge Elements might be useful [2].
RMB click in the browser Mesh_1 > Export > MED File saves a med file of this mesh for later use, for example a *Code_Aster* analysis!

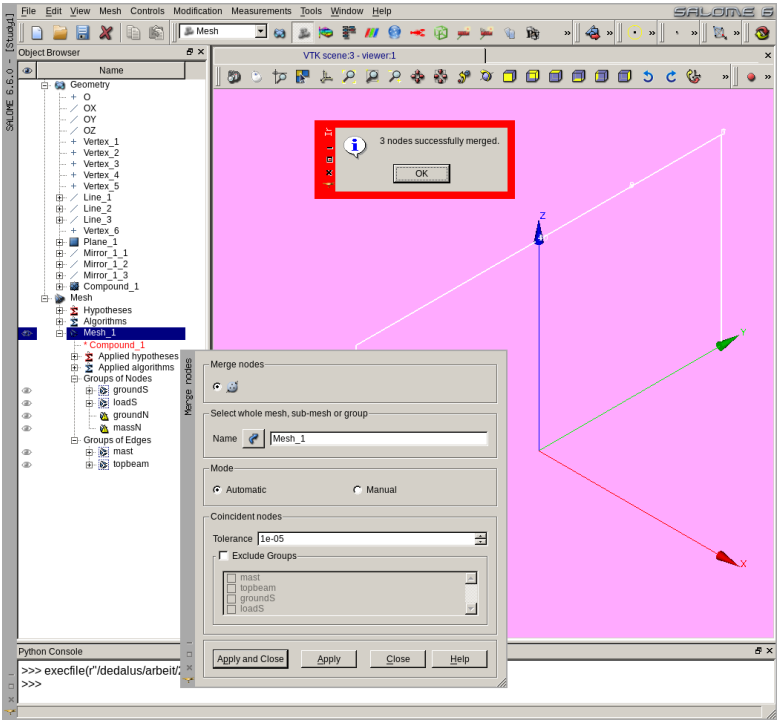


FIGURE D.6: Merge nodes

4.1.2 Modifying the command file

At the time Salome does not seem to be able to create properly 0D element such as the one we need at 'massN', we tell *Code_Aster* to create this 0D element with the following modification at the beginning of the command file.

```
DEBUT();

#read the Salome created mesh in an intermediate concept
meshinit=LIRE_MAILLAGE(
  INFO=1,
  INFO_MED=2,
  UNITE=20,FORMAT='MED',
);

#create a new mesh with all the infos of 'meshinit'
#plus the 'massN' element created on the same node
mesh=CREA_MAILLAGE(
  MAILLAGE=meshinit,
  CREA_POI1=_F(
    NOM_GROUP_MA='massN',
    GROUP_NO='massN',
  ),
);
```

This is self explanatory and we may continue command editing and solving the problem just as explained in chapter 4.2 .

4.1.3 Dumping and replaying the study

With File » Dump Study... we can save a file, which is a Python script of all the commands created in the Salome GUI, Geometry and Mesh. With File » Load Script... this study can be loaded in a new Salome study.

Dumping a study and replaying the file is also, at the time, the only way to modify data, like vertex coordinates, in the geometry. With a bit more Python knowledge, this Python file may be modified to change the geometry, the groups and more...

And with more knowledge of Python, and perseverance, a study can be created from scratch as a Python script in Salome TUI. There is a comprehensive documentation in the online help or on the Salome website.

4.2 Example with beams and plates

4.2.1 Geometry

And this is exactly what we are going to do, to draw the beams of the example *frame3*. With a text editor, we prepare the following Python script. It starts with some generic Salome initialization:

```
# -*- coding: iso-8859-1 -*-
#this very first line sets the coding of the file
#it is generic
#and has nothing to do with Salome
#it is not mandatory either
#it helps for reading in some text editor

import salome
import GEOM
import geompy
```

Then the initialization for our study:

```
a = 500
ly = 2000
dly=10
hz = 2000
hz1=200
#we do not use Point or Line number greater than 300
Point=[None]*(300);
Line=[None]*(300);
```

Followed by the definition of the points and lines:

```
Point[1] = geompy.MakeVertex(0, 0, hz)
Point[101] = geompy.MakeVertex(0, 0, hz-hz1)
for i in (0,2):
  Point[10+i] = geompy.MakeVertex(0,ly/2*(i-1), hz)
  Point[20+i] = geompy.MakeVertex(0,(ly-dly)*(i-1), hz)
  Point[30+i] = geompy.MakeVertex(0,ly*(i-1), hz)
  Point[40+i] = geompy.MakeVertex(0,ly*(i-1), hz-hz1)
  Point[50+i] = geompy.MakeVertex(0,ly*(i-1), 0)
  Point[60+i] = geompy.MakeVertex(0,ly/4*(i-1), hz-hz1/2)
  Point[110+i] = geompy.MakeVertex(0,ly/2*(i-1), hz-hz1)
  Point[120+i] = geompy.MakeVertex(0,(ly-dly)*(i-1), hz-hz1)
  Point[210+i] = geompy.MakeVertex(0,ly/2*(i-1)*1.4, hz-hz1/4)
  Point[220+i] = geompy.MakeVertex(0,ly/2*(i-1)*1.6, hz-hz1/4)
  Point[230+i] = geompy.MakeVertex(0,ly/2*(i-1)*1.4, hz-hz1*3/4)
  Point[240+i] = geompy.MakeVertex(0,ly/2*(i-1)*1.6, hz-hz1*3/4)
for i in (0,2):
  Line[10+i]= geompy.MakeLineTwoPnt(Point[1], Point[10+i])
```

```
  Line[20+i]= geompy.MakeLineTwoPnt(Point[10+i], Point[20+i])
  Line[110+i]= geompy.MakeLineTwoPnt(Point[101], Point[110+i])
  Line[120+i]= geompy.MakeLineTwoPnt(Point[110+i], Point[120+i])
  Line[30+i]= geompy.MakeLineTwoPnt(Point[20+i], Point[30+i])
  Line[130+i]= geompy.MakeLineTwoPnt(Point[120+i], Point[40+i])
  Line[40+i]= geompy.MakeLineTwoPnt(Point[30+i], Point[40+i])
  Line[50+i]= geompy.MakeLineTwoPnt(Point[40+i], Point[50+i])
  Line[210+i]= geompy.MakeLineTwoPnt(Point[10+i], Point[110+i])
  Line[220+i]= geompy.MakeLineTwoPnt(Point[20+i], Point[120+i])
  Line[250+i]= geompy.MakeLineTwoPnt(Point[210+i], Point[230+i])
  Line[260+i]= geompy.MakeLineTwoPnt(Point[230+i], Point[240+i])
  Line[270+i]= geompy.MakeLineTwoPnt(Point[220+i], Point[240+i])
  Line[280+i]= geompy.MakeLineTwoPnt(Point[210+i], Point[220+i])
#next line outside of the loop
Line[201]= geompy.MakeLineTwoPnt(Point[1], Point[101])
```

To finally add these entities to the study:

```
geompy.addToStudy( Point[1],'Point_%02g' %(1))
geompy.addToStudy( Point[101],'Point_%02g' %(101))
for i in (0,2):
  for j in range (1,7):
    geompy.addToStudy( Point[10*j+i],'Point_%02g' %(10*j+i))
  geompy.addToStudy( Point[110+i],'Point_%02g' %(110+i))
  geompy.addToStudy( Point[120+i],'Point_%02g' %(120+i))
  for j in range (1,5):
    geompy.addToStudy( Point[200+10*j+i],'Point_%02g' %(200+10*j+i))

for i in (0,2):
  for j in range (1,6):
    geompy.addToStudy( Line[10*j+i],'Line_%02g' %(10*j+i))
  for j in range (1,4):
    geompy.addToStudy( Line[100+10*j+i],'Line_%02g' %(100+10*j+i))
  for j in range (1,3):
    geompy.addToStudy( Line[200+10*j+i],'Line_%02g' %(200+10*j+i))
  for j in range (5,9):
    geompy.addToStudy( Line[200+10*j+i],'Line_%02g' %(200+10*j+i))
geompy.addToStudy( Line[201], 'Line_%02g' %(201)
```

And display them in the Salome graphic window if it exists. These two lines should be the last ones of the script as they command to display in the **Object Browser** of the **Salome** window all the object which have been created above! Although if they were not there, the 'Geometry' tab would allow to display the geometry as a whole without access to the individual entities.

```
if salome.sg.hasDesktop():
  salome.sg.updateObjBrowser(1)
```

This script is similar to the one used in the Gmsh definition of the same geometry, it uses the same numbering for points and lines. I just find it more verbose than the Gmsh one, it however is undoubtedly less tiresome than using Salome GUI.

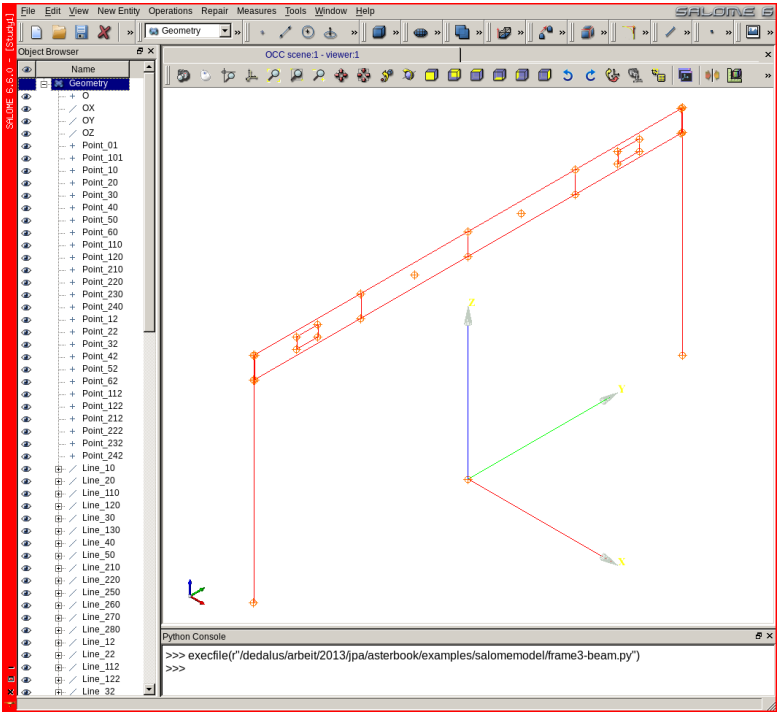


FIGURE D.7: Script loaded in Geometry module

Opening Salome, going in Geometry module and to File › Load Script... a Geometry tab appears in **Object Browser**. RMB Show in this tab displays something like figure D.7 in the **Salome** window[1].

To create the leftmost hollowed surface: go to the menu New Entity › Build › Face, and select the four outer lines of the surface outer border

[1] However it is a wise idea to look in the **Python Console** window at the bottom of the **Salome** window to check there are no error messages, and if there are, put the script right until everything loads fine.

and the four lines of the inner border while keeping the ⇧ pushed and finally click Apply just like in figure D.8.

We keep going on for the other faces. As we want 'Point_60' to be a node in the face of 'Face_3 we' create a 'Partition' on this face with Operations » Partition.

Once all the basic geometric entities are created, we put all of them in a compound with New Entity » Build » Compound.

Within this compound, we create group with New Entity » Group » Create for all the faces, lines and points as required.

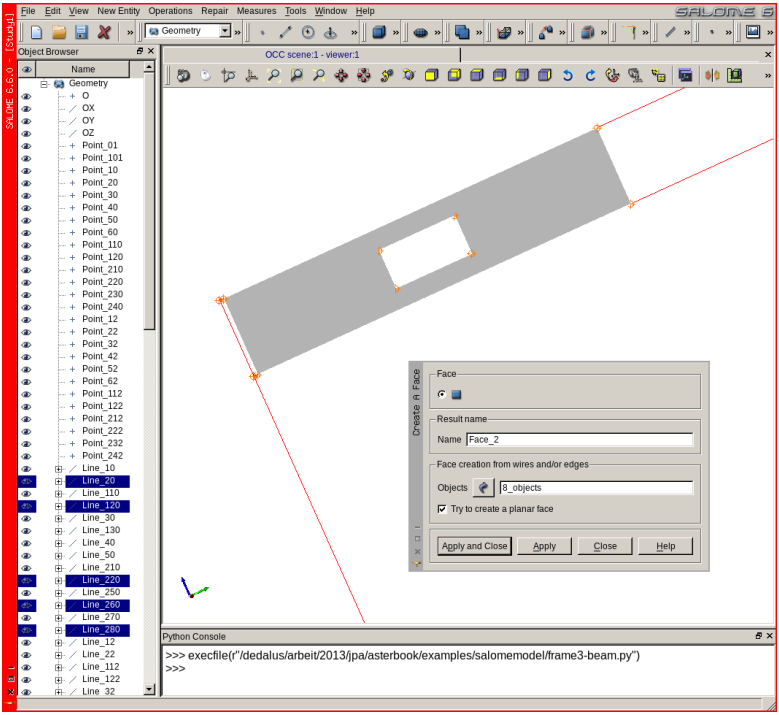


FIGURE D.8: Creating the first Face

All these geometric operations result in the following dumped Python file[1]. I had to include an end of line character \, as some of the lines were longer than this page width.

At first some initialization[2]:

```
# -*- coding: iso-8859-1 -*-

###
### This file is generated automatically by SALOME v6.6.0 \
with dump python functionality
###

import sys
import salome

salome.salome_init()
theStudy = salome.myStudy

import salome_notebook
notebook = salome_notebook.notebook
#here is the directory of the sutdy
sys.path.insert( 0, r'/.........../salomemodel')

###
### GEOM component
###

import GEOM
import geompy
import math
import SALOMEDS


geompy.init_geom(theStudy)

O = geompy.MakeVertex(0, 0, 0)
OX = geompy.MakeVectorDXDYDZ(1, 0, 0)
OY = geompy.MakeVectorDXDYDZ(0, 1, 0)
OZ = geompy.MakeVectorDXDYDZ(0, 0, 1)
O_1 = geompy.MakeVertex(0, 0, 0)
OX_1 = geompy.MakeVectorDXDYDZ(1, 0, 0)
OY_1 = geompy.MakeVectorDXDYDZ(0, 1, 0)
OZ_1 = geompy.MakeVectorDXDYDZ(0, 0, 1)
O_2 = geompy.MakeVertex(0, 0, 0)
OX_2 = geompy.MakeVectorDXDYDZ(1, 0, 0)
OY_2 = geompy.MakeVectorDXDYDZ(0, 1, 0)
OZ_2 = geompy.MakeVectorDXDYDZ(0, 0, 1)
O_3 = geompy.MakeVertex(0, 0, 0)
```

[1] Once dumped, the loops and all the programming features of the geometry input script are lost.

[2] Including the vectors of the coordinates.

```
OX_3 = geompy.MakeVectorDXDYDZ(1, 0, 0)
OY_3 = geompy.MakeVectorDXDYDZ(0, 1, 0)
OZ_3 = geompy.MakeVectorDXDYDZ(0, 0, 1)
O_4 = geompy.MakeVertex(0, 0, 0)
OX_4 = geompy.MakeVectorDXDYDZ(1, 0, 0)
OY_4 = geompy.MakeVectorDXDYDZ(0, 1, 0)
OZ_4 = geompy.MakeVectorDXDYDZ(0, 0, 1)
O_5 = geompy.MakeVertex(0, 0, 0)
OX_5 = geompy.MakeVectorDXDYDZ(1, 0, 0)
OY_5 = geompy.MakeVectorDXDYDZ(0, 1, 0)
OZ_5 = geompy.MakeVectorDXDYDZ(0, 0, 1)
```

Followed by the *frame3* definition points and lines:

```
Point_01 = geompy.MakeVertex(0, 0, 2000)
Point_101 = geompy.MakeVertex(0, 0, 1800)
Point_10 = geompy.MakeVertex(0, -1000, 2000)
Point_20 = geompy.MakeVertex(0, -1990, 2000)
Point_30 = geompy.MakeVertex(0, -2000, 2000)
Point_40 = geompy.MakeVertex(0, -2000, 1800)
Point_50 = geompy.MakeVertex(0, -2000, 0)
Point_60 = geompy.MakeVertex(0, -500, 1900)
Point_110 = geompy.MakeVertex(0, -1000, 1800)
Point_120 = geompy.MakeVertex(0, -1990, 1800)
Point_210 = geompy.MakeVertex(0, -1400, 1950)
Point_220 = geompy.MakeVertex(0, -1600, 1950)
Point_230 = geompy.MakeVertex(0, -1400, 1850)
Point_240 = geompy.MakeVertex(0, -1600, 1850)
Point_12 = geompy.MakeVertex(0, 1000, 2000)
Point_22 = geompy.MakeVertex(0, 1990, 2000)
Point_32 = geompy.MakeVertex(0, 2000, 2000)
Point_42 = geompy.MakeVertex(0, 2000, 1800)
Point_52 = geompy.MakeVertex(0, 2000, 0)
Point_62 = geompy.MakeVertex(0, 500, 1900)
Point_112 = geompy.MakeVertex(0, 1000, 1800)
Point_122 = geompy.MakeVertex(0, 1990, 1800)
Point_212 = geompy.MakeVertex(0, 1400, 1950)
Point_222 = geompy.MakeVertex(0, 1600, 1950)
Point_232 = geompy.MakeVertex(0, 1400, 1850)
Point_242 = geompy.MakeVertex(0, 1600, 1850)
Line_10 = geompy.MakeLineTwoPnt(Point_01, Point_10)
Line_20 = geompy.MakeLineTwoPnt(Point_10, Point_20)
Line_110 = geompy.MakeLineTwoPnt(Point_101, Point_110)
Line_120 = geompy.MakeLineTwoPnt(Point_110, Point_120)
Line_30 = geompy.MakeLineTwoPnt(Point_20, Point_30)
Line_130 = geompy.MakeLineTwoPnt(Point_120, Point_40)
Line_40 = geompy.MakeLineTwoPnt(Point_30, Point_40)
Line_50 = geompy.MakeLineTwoPnt(Point_40, Point_50)
Line_210 = geompy.MakeLineTwoPnt(Point_10, Point_110)
Line_220 = geompy.MakeLineTwoPnt(Point_20, Point_120)
Line_250 = geompy.MakeLineTwoPnt(Point_210, Point_230)
Line_260 = geompy.MakeLineTwoPnt(Point_230, Point_240)
Line_270 = geompy.MakeLineTwoPnt(Point_220, Point_240)
Line_280 = geompy.MakeLineTwoPnt(Point_210, Point_220)
```

```
Line_12 = geompy.MakeLineTwoPnt(Point_01, Point_12)
Line_22 = geompy.MakeLineTwoPnt(Point_12, Point_22)
Line_112 = geompy.MakeLineTwoPnt(Point_101, Point_112)
Line_122 = geompy.MakeLineTwoPnt(Point_112, Point_122)
Line_32 = geompy.MakeLineTwoPnt(Point_22, Point_32)
Line_132 = geompy.MakeLineTwoPnt(Point_122, Point_42)
Line_42 = geompy.MakeLineTwoPnt(Point_32, Point_42)
Line_52 = geompy.MakeLineTwoPnt(Point_42, Point_52)
Line_212 = geompy.MakeLineTwoPnt(Point_12, Point_112)
Line_222 = geompy.MakeLineTwoPnt(Point_22, Point_122)
Line_252 = geompy.MakeLineTwoPnt(Point_212, Point_232)
Line_262 = geompy.MakeLineTwoPnt(Point_232, Point_242)
Line_272 = geompy.MakeLineTwoPnt(Point_222, Point_242)
Line_282 = geompy.MakeLineTwoPnt(Point_212, Point_222)
Line_201 = geompy.MakeLineTwoPnt(Point_01, Point_101)
```

Building the five top surfaces:

```
Face_1 = geompy.MakeFaceWires([Line_20, Line_120, Line_220, \
Line_260, Line_280, Line_210, Line_250, Line_270], 1)
Face_2 = geompy.MakeFaceWires([Line_260, Line_280, Line_250, \
Line_270], 1)
Face_3 = geompy.MakeFaceWires([Line_10, Line_110, Line_210, \
Line_201], 1)
Face_4 = geompy.MakeFaceWires([Line_112, Line_12, Line_212, \
Line_201], 1)
Face_5 = geompy.MakeFaceWires([Line_272, Line_22, Line_122, \
Line_212, Line_222, Line_252, Line_262, Line_282], 1)
```

Making a Partition with 'Face_5' and 'Point_60' so that there is a node at this point:

```
Partition_1 = geompy.MakePartition([Face_3], [Point_60], [], [], \
geompy.ShapeType["FACE"], 0, [], 0)
```

Building a compound with all the geometric entities:

```
Compound_1 = geompy.MakeCompound([O_5, OX_5, OY_5, OZ_5, Point_01,\
Point_101, Point_10, Point_20, Point_30, Point_40, Point_50, \
Point_60, Point_110, Point_120, Point_210, Point_220, Point_230,\
Point_240, Point_12, Point_22, Point_32, Point_42, Point_52, \
Point_62, Point_112, Point_122, Point_212, Point_222, Point_232,\
Point_242, Line_10, Line_20, Line_110, Line_120, Line_30, Line_130,\
Line_40, Line_50, Line_210, Line_220, Line_250, Line_260, Line_270,\
Line_280, Line_12, Line_22, Line_112, Line_122, Line_32, Line_132,\
Line_42, Line_52, Line_212, Line_222, Line_252, Line_262, Line_272,\
Line_282, Line_201, Face_1, Face_2, Face_3, Face_4, Face_5, \
Partition_1])
```

Creating the required geometric groups:

```
mast = geompy.CreateGroup(Compound_1, geompy.ShapeType["EDGE"])
geompy.UnionIDs(mast, [45, 44, 59, 58])
panelS = geompy.CreateGroup(Compound_1, geompy.ShapeType["FACE"])
geompy.UnionIDs(panelS, [67, 70, 79])
panelN = geompy.CreateGroup(Compound_1, geompy.ShapeType["FACE"])
geompy.UnionIDs(panelN, [74, 76])
topbeam = geompy.CreateGroup(Compound_1, geompy.ShapeType["EDGE"])
geompy.UnionIDs(topbeam, [39, 40, 41, 55, 52, 38, 53, 54])
vertb = geompy.CreateGroup(Compound_1, geompy.ShapeType["EDGE"])
geompy.UnionIDs(vertb, [47, 46, 88, 60, 61])
hinge = geompy.CreateGroup(Compound_1, geompy.ShapeType["EDGE"])
geompy.UnionIDs(hinge, [42, 56, 43, 57])
groundS = geompy.CreateGroup(Compound_1, \
geompy.ShapeType["VERTEX"])
geompy.UnionIDs(groundS, [18])
groundN = geompy.CreateGroup(Compound_1, \
geompy.ShapeType["VERTEX"])
geompy.UnionIDs(groundN, [30])
massN = geompy.CreateGroup(Compound_1, \
geompy.ShapeType["VERTEX"])
geompy.UnionIDs(massN, [26])
loadS = geompy.CreateGroup(Compound_1, \
geompy.ShapeType["VERTEX"])
geompy.UnionIDs(loadS, [14])
oripanel = geompy.CreateGroup(Compound_1, \
geompy.ShapeType["VERTEX"])
geompy.UnionIDs(oripanel, [19])
```

Adding the coordinate system vectors to the study:

```
geompy.addToStudy( O, 'O' )
geompy.addToStudy( OX, 'OX' )
geompy.addToStudy( OY, 'OY' )
geompy.addToStudy( OZ, 'OZ' )
geompy.addToStudy( O_1, 'O' )
geompy.addToStudy( OX_1, 'OX' )
geompy.addToStudy( OY_1, 'OY' )
geompy.addToStudy( OZ_1, 'OZ' )
geompy.addToStudy( O_2, 'O' )
geompy.addToStudy( OX_2, 'OX' )
geompy.addToStudy( OY_2, 'OY' )
geompy.addToStudy( OZ_2, 'OZ' )
geompy.addToStudy( O_3, 'O' )
geompy.addToStudy( OX_3, 'OX' )
geompy.addToStudy( OY_3, 'OY' )
geompy.addToStudy( OZ_3, 'OZ' )
geompy.addToStudy( O_4, 'O' )
geompy.addToStudy( OX_4, 'OX' )
geompy.addToStudy( OY_4, 'OY' )
geompy.addToStudy( OZ_4, 'OZ' )
geompy.addToStudy( O_5, 'O' )
geompy.addToStudy( OX_5, 'OX' )
geompy.addToStudy( OY_5, 'OY' )
```

```
geompy.addToStudy( OZ_5, 'OZ' )
```

Adding the points to the study:

```
geompy.addToStudy( Point_01, 'Point_01' )
geompy.addToStudy( Point_101, 'Point_101' )
geompy.addToStudy( Point_10, 'Point_10' )
geompy.addToStudy( Point_20, 'Point_20' )
geompy.addToStudy( Point_30, 'Point_30' )
geompy.addToStudy( Point_40, 'Point_40' )
geompy.addToStudy( Point_50, 'Point_50' )
geompy.addToStudy( Point_60, 'Point_60' )
geompy.addToStudy( Point_110, 'Point_110' )
geompy.addToStudy( Point_120, 'Point_120' )
geompy.addToStudy( Point_210, 'Point_210' )
geompy.addToStudy( Point_220, 'Point_220' )
geompy.addToStudy( Point_230, 'Point_230' )
geompy.addToStudy( Point_240, 'Point_240' )
geompy.addToStudy( Point_12, 'Point_12' )
geompy.addToStudy( Point_22, 'Point_22' )
geompy.addToStudy( Point_32, 'Point_32' )
geompy.addToStudy( Point_42, 'Point_42' )
geompy.addToStudy( Point_52, 'Point_52' )
geompy.addToStudy( Point_62, 'Point_62' )
geompy.addToStudy( Point_112, 'Point_112' )
geompy.addToStudy( Point_122, 'Point_122' )
geompy.addToStudy( Point_212, 'Point_212' )
geompy.addToStudy( Point_222, 'Point_222' )
geompy.addToStudy( Point_232, 'Point_232' )
geompy.addToStudy( Point_242, 'Point_242' )
```

Adding the lines to the study:

```
geompy.addToStudy( Line_10, 'Line_10' )
geompy.addToStudy( Line_20, 'Line_20' )
geompy.addToStudy( Line_110, 'Line_110' )
geompy.addToStudy( Line_120, 'Line_120' )
geompy.addToStudy( Line_30, 'Line_30' )
geompy.addToStudy( Line_130, 'Line_130' )
geompy.addToStudy( Line_40, 'Line_40' )
geompy.addToStudy( Line_50, 'Line_50' )
geompy.addToStudy( Line_210, 'Line_210' )
geompy.addToStudy( Line_220, 'Line_220' )
geompy.addToStudy( Line_250, 'Line_250' )
geompy.addToStudy( Line_260, 'Line_260' )
geompy.addToStudy( Line_270, 'Line_270' )
geompy.addToStudy( Line_280, 'Line_280' )
geompy.addToStudy( Line_12, 'Line_12' )
geompy.addToStudy( Line_22, 'Line_22' )
geompy.addToStudy( Line_112, 'Line_112' )
geompy.addToStudy( Line_122, 'Line_122' )
geompy.addToStudy( Line_32, 'Line_32' )
geompy.addToStudy( Line_132, 'Line_132' )
```

```
geompy.addToStudy( Line_42, 'Line_42' )
geompy.addToStudy( Line_52, 'Line_52' )
geompy.addToStudy( Line_212, 'Line_212' )
geompy.addToStudy( Line_222, 'Line_222' )
geompy.addToStudy( Line_252, 'Line_252' )
geompy.addToStudy( Line_262, 'Line_262' )
geompy.addToStudy( Line_272, 'Line_272' )
geompy.addToStudy( Line_282, 'Line_282' )
geompy.addToStudy( Line_201, 'Line_201' )
```

Adding the faces, partition, compound and groups to the study:

```
geompy.addToStudy( Face_1, 'Face_1' )
geompy.addToStudy( Face_2, 'Face_2' )
geompy.addToStudy( Face_3, 'Face_3' )
geompy.addToStudy( Face_4, 'Face_4' )
geompy.addToStudy( Face_5, 'Face_5' )
geompy.addToStudy( Partition_1, 'Partition_1' )
geompy.addToStudy( Compound_1, 'Compound_1' )
geompy.addToStudyInFather( Compound_1, mast, 'mast' )
geompy.addToStudyInFather( Compound_1, panelS, 'panelS' )
geompy.addToStudyInFather( Compound_1, panelN, 'panelN' )
geompy.addToStudyInFather( Compound_1, topbeam, 'topbeam' )
geompy.addToStudyInFather( Compound_1, vertb, 'vertb' )
geompy.addToStudyInFather( Compound_1, hinge, 'hinge' )
geompy.addToStudyInFather( Compound_1, groundS, 'groundS' )
geompy.addToStudyInFather( Compound_1, groundN, 'groundN' )
geompy.addToStudyInFather( Compound_1, massN, 'massN' )
geompy.addToStudyInFather( Compound_1, loadS, 'loadS' )
geompy.addToStudyInFather( Compound_1, oripanel, 'oripanel' )
```

4.2.2 Hints about creating groups

The dump script for creating the group 'mast' looks like this:

```
mast = geompy.CreateGroup(Compound_1, geompy.ShapeType["EDGE"])
geompy.UnionIDs(mast, [45, 44, 59, 58])
```

Where we can see the list [45, 44, 59, 58], how do we know where these numbers come from?

They are internal, inside 'Compound_1', identification numbers "Id" of the lines, which we know to be lines Line_40, Line_42, Line_50, Line_52[1]. Then the following script would build the group 'mast':

[1] That's how we built the script.

```
mast = geompy.CreateGroup(Compound_1, geompy.ShapeType["EDGE"])
Id1 = geompy.GetSubShapeID(Compound_1, Line_40)
Id2 = geompy.GetSubShapeID(Compound_1, Line_42)
Id3 = geompy.GetSubShapeID(Compound_1, Line_50)
Id4 = geompy.GetSubShapeID(Compound_1, Line_52)
geompy.UnionIDs(mast, [Id1, Td2, Id3, Id4])
```

Of course the list of Id could better be built inside a loop.

'geompy.AddObject' could be used instead of 'geompy.UnionIDs', I do not understand why New entity > Group > Create calls 'UnionIDs'[1].

4.2.3 Meshing

We first create 'Mesh_1' on 'Compound_1', without any hypothesis, it acts just as a container for the following sub meshes, one for each group. We have seen in the previous example *frame1* how to mesh a line, figure D.9 shows how to proceed to mesh the group 'panelN'.

And the meshing section of the dump file:

```
####
#### SMESH component
####

import smesh, SMESH, SALOMEDS

smesh.SetCurrentStudy(theStudy)
import StdMeshers
import NETGENPlugin
Mesh_1 = smesh.Mesh(Compound_1)
```

Last line is the creation of 'Mesh_1' on 'Compound_1'. And now we create the sub mesh 'mesh-mast' on group 'mast' with:

- 'Regular_1D' as internal mesh Id
- 'Mesh_1.Segment' algorithm as we are meshing a line;
- 'geom=mast', on geometry group 'mast';
- 'Local_Length_1' with a value of 100 as size of the element.

[1] This is one of the numerous mysteries of Salome.

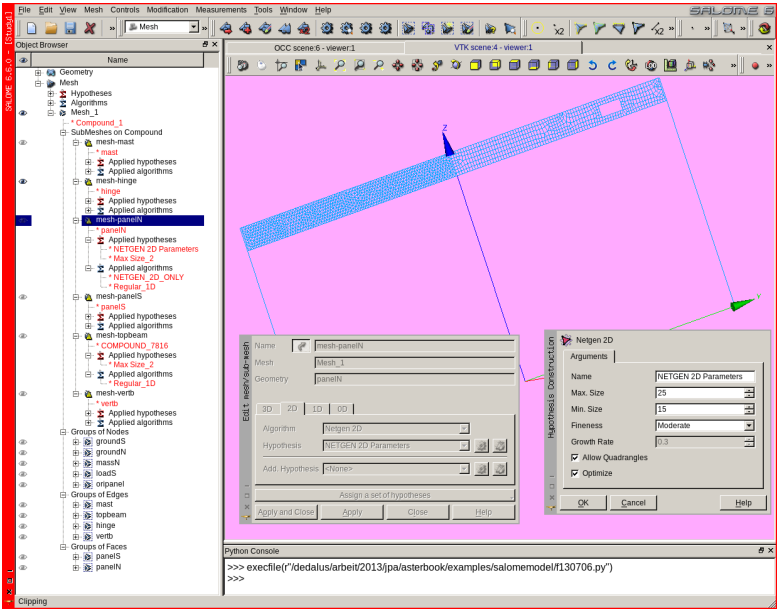


FIGURE D.9: Meshing hypothesis for 'panelN' with Allow quadrangle ticked

```
Regular_1D = Mesh_1.Segment(geom=mast)
Local_Length_1 = Regular_1D.LocalLength(100,[],1e-07)
```

Followed by the sub mesh 'mesh-topbeam" on group 'topbeam' with:

- 'Regular_1D_1' as internal mesh Id
- 'Mesh_1.Segment' algorithm as we are meshing a line;
- 'geom=topbeam', on geometry group 'topbeam';
- 'Local_Length_2' with a value of 25 as size of the element.

```
Regular_1D_1 = Mesh_1.Segment(geom=topbeam)
Local_Length_2 = Regular_1D_1.LocalLength(25,[],1e-07)
```

And we repeat the operation for group 'vertb', reusing 'Local_Length_2'.

```
Regular_1D_2 = Mesh_1.Segment(geom=vertb)
status = Mesh_1.AddHypothesis(Local_Length_2,vertb)
```

Group 'hinge' owes a special treatment as we want only one single element along the geometry line.

```
Regular_1D_3 = Mesh_1.Segment(geom=hinge)
Nb_Segments_1 = Regular_1D_3.NumberOfSegments(1,[],[ ])
Nb_Segments_1.SetDistrType( 0 )
```

Then 'mesh-panelS'.

- first the parameters along the edges
- 'algo=smesh.NETGEN_2D' on the surface;
- 'SetMaxSize(25)' for the size of the element;
- 'SetQuadAllowed(0)' as we want only triangles;
- 'SetSecondOrder(0)' as we want a linear mesh.

```
Regular_1D_4 = Mesh_1.Segment(geom=panelS)
status = Mesh_1.AddHypothesis(Local_Length_2,panelS)
mesh_panelS = Mesh_1.GetSubMesh( panelS, 'NETGEN_2D_ONLY' )
NETGEN_2D_ONLY = Mesh_1.Triangle(algo=smesh.NETGEN_2D,geom=panelS)
NETGEN_2D_Parameters = NETGEN_2D_ONLY.Parameters()
NETGEN_2D_Parameters.SetMaxSize( 25 )
NETGEN_2D_Parameters.SetSecondOrder( 1 )
NETGEN_2D_Parameters.SetOptimize( 1 )
NETGEN_2D_Parameters.SetFineness( 2 )
NETGEN_2D_Parameters.SetMinSize( 15 )
NETGEN_2D_Parameters.SetQuadAllowed( 0 )
NETGEN_2D_Parameters.SetSecondOrder( 0 )
```

We repeat this on group 'panelN' with the same hypothesis but 'SetQuadAllowed(1)' as we want quadrangles in this area[1].

```
Regular_1D_5 = Mesh_1.Segment(geom=panelN)
status = Mesh_1.AddHypothesis(Local_Length_2,panelN)
mesh_panelN = Mesh_1.GetSubMesh( panelN, 'NETGEN_2D_ONLY' )
NETGEN_2D_ONLY_1 = Mesh_1.Triangle(algo=smesh.NETGEN_2D,geom=panelN)
NETGEN_2D_Parameters_1 = NETGEN_2D_ONLY_1.Parameters()
NETGEN_2D_Parameters_1.SetMaxSize( 25 )
```

[1] 'SetSecondOrder(1)' would be used to create a quadratic mesh.

```
NETGEN_2D_Parameters_1.SetSecondOrder( 1 )
NETGEN_2D_Parameters_1.SetOptimize( 1 )
NETGEN_2D_Parameters_1.SetFineness( 2 )
NETGEN_2D_Parameters_1.SetMinSize( 15 )
NETGEN_2D_Parameters_1.SetQuadAllowed( 1 )
NETGEN_2D_Parameters_1.SetSecondOrder( 0 )
```

Next section creates the groups.

```
mast_1 = Mesh_1.GroupOnGeom(mast,'mast',SMESH.EDGE)
panelS_1 = Mesh_1.GroupOnGeom(panelS,'panelS',SMESH.FACE)
panelN_1 = Mesh_1.GroupOnGeom(panelN,'panelN',SMESH.FACE)
topbeam_1 = Mesh_1.GroupOnGeom(topbeam,'topbeam',SMESH.EDGE)
vertb_1 = Mesh_1.GroupOnGeom(vertb,'vertb',SMESH.EDGE)
hinge_1 = Mesh_1.GroupOnGeom(hinge,'hinge',SMESH.EDGE)
groundS_1 = Mesh_1.GroupOnGeom(groundS,'groundS',SMESH.NODE)
groundN_1 = Mesh_1.GroupOnGeom(groundN,'groundN',SMESH.NODE)
massN_1 = Mesh_1.GroupOnGeom(massN,'massN',SMESH.NODE)
loadS_1 = Mesh_1.GroupOnGeom(loadS,'loadS',SMESH.NODE)
oripanel_1 = Mesh_1.GroupOnGeom(oripanel,'oripanel',SMESH.NODE)
```

Next section actually sets the sub-meshes, together with the one on nodes used as BC and loads.

```
mesh_mast = Regular_1D.GetSubMesh()
mast_topbeam = Regular_1D_1.GetSubMesh()
mesh_vertb = Regular_1D_2.GetSubMesh()
mesh_hinge = Regular_1D_3.GetSubMesh()
mesh_panelS = Regular_1D_4.GetSubMesh()
mesh_panelN = Regular_1D_5.GetSubMesh()
mesh_groundS = Mesh_1.GetSubMesh( groundS, 'mesh-groundS' )
mesh_groundN = Mesh_1.GetSubMesh( groundN, 'mesh-groundN' )
mesh_massN = Mesh_1.GetSubMesh( massN, 'mesh-massN' )
mesh_loadS = Mesh_1.GetSubMesh( loadS, 'mesh-loadS' )
mesh_oripanel = Mesh_1.GetSubMesh( oripanel, 'mesh-oripanel' )
```

Next bit names the various objects.

```
## set object names
smesh.SetName(Mesh_1.GetMesh(), 'Mesh_1')
smesh.SetName(Regular_1D.GetAlgorithm(), 'Regular_1D')
smesh.SetName(Local_Length_1, 'Local Length_1')
smesh.SetName(Local_Length_2, 'Local Length_2')
smesh.SetName(Nb_Segments_1, 'Nb. Segments_1')
smesh.SetName(mesh_panelS, 'mesh-panelS')
smesh.SetName(NETGEN_2D_ONLY.GetAlgorithm(), 'NETGEN_2D_ONLY')
smesh.SetName(NETGEN_2D_Parameters, 'NETGEN 2D Parameters')
smesh.SetName(mesh_panelN, 'mesh-panelN')
smesh.SetName(NETGEN_2D_Parameters_1, 'NETGEN 2D Parameters')
smesh.SetName(mesh_mast, 'mesh-mast')
smesh.SetName(mast_topbeam, 'mast-topbeam')
```

```
smesh.SetName(mesh_vertb, 'mesh-vertb')
smesh.SetName(mesh_hinge, 'mesh-hinge')
smesh.SetName(mesh_groundS, 'mesh-groundS')
smesh.SetName(mesh_groundN, 'mesh-groundN')
smesh.SetName(mesh_massN, 'mesh-massN')
smesh.SetName(mesh_loadS, 'mesh-loadS')
smesh.SetName(mesh_oripanel, 'mesh-oripanel')
```

Finally making it visible

```
if salome.sg.hasDesktop():
  salome.sg.updateObjBrowser(1)
```

We should not forget to perform Modification > Transformation > Merge Nodes, not printed in the above dump. And export the mesh in *.med* file to be able to run the study in *Code_Aster*.

This example shows how verbose Salome is, there are a mere 350 lines to describe the geometry and build the mesh while Gmsh do the same in a little above a hundred lines.

4.2.4 View 3D with Eficas in Salome-Meca

If we run this script in a Salome-Meca study and import in this study the *fram3.comm* file created previously we can get a 3D view of the beam section and plate thickness like D.10. For this:

- run the script to load the geometry and mesh;
- run Eficas from the pull down list;
- open *frame3.comm* in the **Eficas** window;
- on the list at the left hand side in the **Eficas** window RMB click on AFFE_CARA_ELEM > View3D.

The ability to produce this kind of view is an interesting advantage of Salome-Meca. However:

- a geometry must exist, with the groups defined in it;
- a command file must exist;

- and the view of the beam section is restricted to the sections available in the catalog, i.e. circle and rectangle, hollow or not, which is short of many sections used in actual buildings.

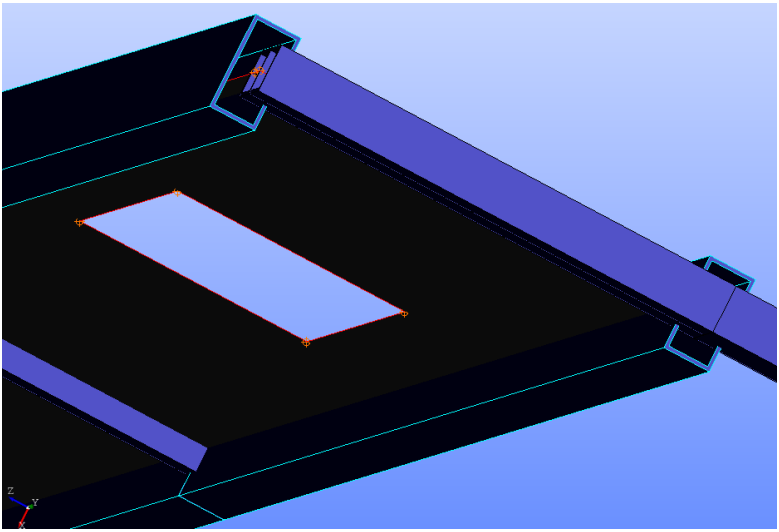


FIGURE D.10: View 3D of the top north end of *frame3* model in Salome with Eficas

4.3 3D Example

Now, we are going to draw and mesh the 'part2' of the 3D example leaf spring. We retain the same groups as in the Gmsh mesh, so the produced *.med* file is fully exchangeable in the *Code_Aster* analysis. We also keep the same parameter names as within the Gmsh script.

To define these parameters, we use the NoteBook tool of Salome, available through the File > NoteBook menu. Once filled the **NoteBook** should look like D.11. The parameters 'r1', 'r2', 't1', 'off1', are the same as in Gmsh while 'b1z', 'b2x', 'b2y', 'b2z' are the results of mathematical expressions.

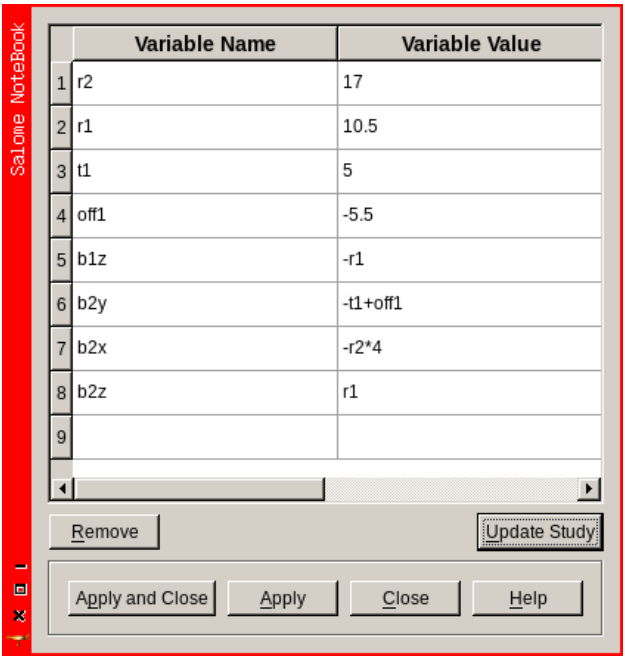


FIGURE D.11: Salome NoteBook for 'part2'

If we, later on, change one or several of the variables it is just a matter of pushing the Update Study to change the geometry and the mesh. This makes a kind of parametric tool.

Then follows the step by step geometry creation:

1. create a point, 'Vertex_1' with coordinates (0, "off1", 0), this is the base point for the cylinder;
2. create a vector, 'Vector_1' with coordinates Dx=0, Dy=-10, Dz=0;
3. create the outer cylinder, 'Cylinder_1' with New Entity > Primitive > Cylinder with:
   - left most button ticked;
   - Base Point = "Vertex"_1';

- Vector = "Vector1_1";
- Radius = "r2"";
- Height= "t1";

4. create the inner cylinder, 'Cylinder_2' in the same way with Radius = "r1", Height = "t1";

5. create a point named 'p1', with coordinates (0, "off1", "b1z"), this is for one corner of the box;

6. create another point named 'p2', with coordinates ("b2x", "b2y", "b2z"), this is the other corner of the box;

7. create a box, 'Box_1' with New Entity > Primitive > Box with the two points 'p1' and 'p2';

8. create a new volume entity, 'Fuse_1', with Operations > Boolean > Fuse with 'Cylinder_1' and 'Box_1';

9. and finally remove the inner cylinder with Operations > Boolean > Cut with 'Fuse_1' and 'Cylinder_2', creating final entity 'Cut_1', looking like figure D.12.

A bit of work is necessary on this CAD like geometry so as to be able to produce a workable mesh.

In the tree select 'Cut_1' then menu New Entity > Explode to create 'Shell_1' containing all the boundary surfaces of 'Cut_1'.

In the tree select the newly created 'Shell_1' then menu New Entity > Explode to create 12 'Face_xx' which are the individual boundary surfaces of 'Cut_1'. This is necessary, as we have to create groups and boundary conditions on some of them.

If we created the part with a script along the previous path it is not trivial at all to identify the faces Ids, and creating groups directly in a *Code_Aster* command file, like explained in chapter 16.5 , may well prove the best way here.

Create mesh on 'Cut_1' with:

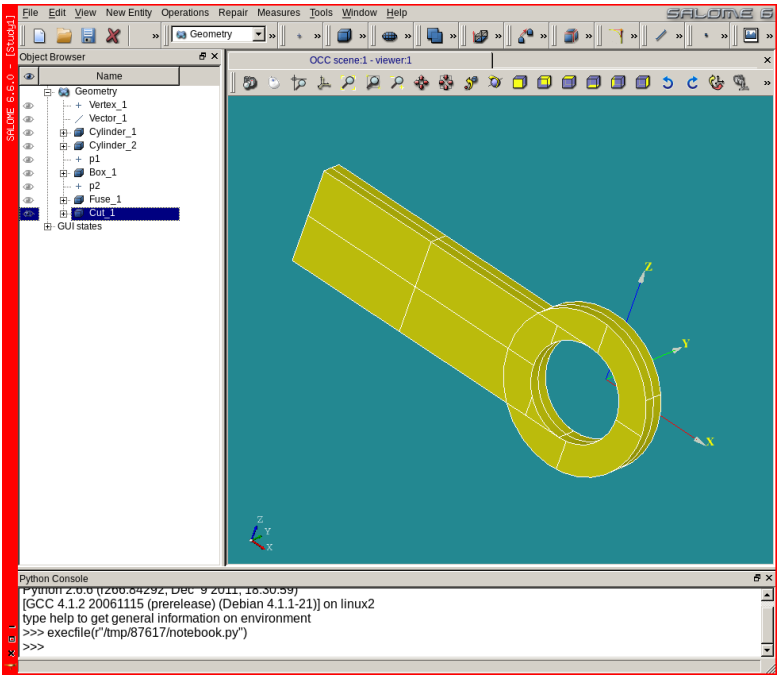


FIGURE D.12: Geometric volume created

- Algo » NetGen 1D-2D-3D[1];
- Hypothesis » NETGEN 3D Simple Parameters with Local length set to 2.

Mesh object, 'Mesh_1' is created, RMB click and Compute actually creates the mesh in it.

Navigating through the surface list we Create group 'fix2s', 'bear2s' and 'hole2s' to hold the necessary boundary conditions and load. After this we have to compute again or Update the mesh.

Meshing is finished!

[1] Or something similar as Salome naming seems to change every new version, and to make things even more complicated to understand the name in the dump script is also different here, NETGEN_2D3D.

Oops? NO! as we forgot to create the group 'move2p' containing the node where to extract displacement.

We have to identify a Face on which this node is, at one corner, and then explode this Face, Explode » Sub Shape Type » Vertex.

Then Create group » Node » Enable Manual Selection pick the node and name the group 'move2p'.

4.4 Further reading

This chapter is a little bit short to cover all the capabilities of Salome and I can only advise the reader to go through the Help » (module) » User's Guide menu in the **Salome** window, to find an instant help on a given topic.

The `http://www.salome-platform.org/user-section/salome-tutorials` section of the web site provides many useful exercises.

4.5 Salome setup and preferences

Salome preferences are numerous and can be accessed in the menu File » Preferences.

Particularly useful is the ability to recover the appearance from one session to the other this is done in the module SALOME, general tab with the Store positions of windows or Store/restore last GUI state check boxes.

4.6 Differences between Gmsh and Salome

In Gmsh, Points must be defined to draw Lines, Lines must be defined to draw Surfaces, Surfaces must be defined to finally draw Volumes, the groups can be created along. GMSH is not a 3D modeler and does not have 3D primitives.

Salome follows a CAD modeler behavior, as just only a few Vertex are necessary to draw volume primitives. Boolean operations can be performed on these primitives, but necessary surfaces must be extracted from the volume shapes[1], the lines or points from the surfaces, if necessary[2].

Salome may have an unpredictable behavior, for example crashing, at the worst time of course[3]! In addition to the auto save mechanism, available through File > Preferences > SALOME > General > Auto-save interval (min) it is a wise idea to, at strategic time:

1. File > Dump Study... as a re-playable Python file;
2. File > Save the study in *.hdf* format.

We should not forget that dumping a study in a Python script destroys all the structure that was in the original script if any[4]. And a wise idea is to split the script in several ones (geometry, group building, meshing...) which can be loaded one after the other.

Summing it all up, as far as professional everyday studies are concerned, I use Gmsh, with imported CAD files in case of complex solid parts.

4.7 Meshing imported CAD file

Here are some hints on how to produce a valid mesh, with groups, from a *.step* file in Salome and Gmsh.

The file must first be exported from the CAD software, I have had the best result by exporting in *.step* with preference "AP214"[5].

In the **Geometry** module of Salome, we import the file with File > Import... > STEP Files. Upon importing the *.step* CAD file Salome asks if one want to convert the units from mm to m, the answer depends upon how the CAD file was created[6]. It is wise to check the dimensions with

[1] Retrieving their Ids, to create groups, in a script is not a trivial operation!
[2] Salome can also be used in a point -> line -> surface -> volume manner.
[3] In this respect Salome follows some Murphy's laws!
[4] Likewise in Gmsh saving as 'Gmsh Unrolled GEO' destroys the structure of the script.
[5] Additionally, for Rhinoceros ®, the volume should be enclosed by a "Closed Polysurface".
[6] Most CAD software propose an option upon exporting the file.

Measures > Point Coordinates on a known point or Measures > Dimensions > Bounding Box.

We can now create groups on this geometry, in Salome:

- RMB click on the geometry name, select Create Group;
- push one of the radio button in agreement with the type of group: node, line, face, volume;
- name the group;
- move the mouse pointer over the geometry until the desired object is highlighted;
- push the left button;
- push the button Create Group to add the object to the group;
- once the selection is finished push the button Apply to create the group.

Once this is done the job is no different of what we have seen before.

In Gmsh the way of proceeding is similar.

We have to save the file in a *.geo* format to save the group creation which has been done. This can be done just after importing the step file.

A few remarks about step format and Gmsh:

- The only line type known, in the *.step* file is BSpline or NURBS, and the controls points are shown as points in Gmsh, thus a straight line shows many control points along its length. Once meshed nodes do not necessarily sit on these control points!
- One BSpline may describe one circle arc equal to or larger than π, then very strange things are going to happen at mesh time. The source CAD file should use only circle arc smaller than π[1].

[1] A full circle needs at least three arcs

And some other general remarks about CAD files.

CAD systems are much more tolerant than any mesher, quite often one fails to mesh a surface because it is not closed, the surface is considered closed by the CAD software because of a rather tolerant precision. This has to be corrected before meshing.

Sometimes CAD systems do not mind to have several similar entities piled one on top of each other, at the same place, describing the very same physical object, the mesher may fail here, or worst, create several layers of the same entity, which, if undetected may produce a very stiff model in this area!

Many detailed features drawn in CAD file, like fillets or chamfers and so on, end up with a mesh larger than necessary, while in some other cases they are the very object of the study, the engineer must make clever choices here!

One should always have a very critical look at the mesh built from a CAD file.

APPENDIX E

Installing and maintaining, tips

A web page is only a page until its printed. Then it can be any number of pages.

Kent's Law.

How to install, update, maintain *Code_Aster*, Gmsh, Salome and Salome-Meca.

In the next sections we, quite often, find the sequence: download an archive, unpack it... It is a wise idea to perform, before anything else, a "md5sum"[1] checkup to ensure the archive integrity, any archive failing this checkup should be immediately discarded!

5.1 *Code_Aster* installation

For a stand alone version of *Code_Aster* go to this link:

[1] If this package is not installed by default in our distribution, we have to install it (in openSUSE it is in coreutils package).

`http://www.code-aster.org`

Look for the download area, once it is found, download the archive that suits.

For the installation, *strictly* follow the instructions given on the page.

I make the installation in /opt/asterxxx, where xxx is the aster version number[1]. Of course I change the ownership of this directory so that I own this file (not root) with read and write access. The line `ASTER_ROOT='/opt/asterxxx'` has also to be changed in the *setup.cfg* file, sitting in the unpacked archive directory.

This installation is somewhat long, about 30 minutes, as it is a true compilation, not installation of binaries.

In the grand old days there was, somewhere in the installation directory a sub-directory with a substantial set of documentation. This is no longer the case and the documentation must be read online, thankfully it is *.pdf* files which can be downloaded for a more comfortable local reading.

The full set can also be downloaded, it's an archive of over 300 Mb.

And finally the English version is an excellent demonstration of what machine translation can do, in good, and in bad[2]. But it is much better than nothing.

In addition *Code_Aster* comes with a bundle of test cases, lying in *$ASTER_ROOT/11.x/share/aster/tests* these tests maybe studied and run to understand unusual commands.

5.2 *Code_Aster* versions

On the *Code_Aster* download page one can see several versions available. Here are some hints about their naming, helping to choose the one that suits[3]:

[1] With this method i keep a working installation of previous versions, just in case.
[2] Not to say "worst"!
[3] This description is abstracted from Pronet website

- ***stable***

  This release is the version of exploitation validated by EDF on the basis a complete documentation and tests of validation put on line. This version is the object independently of a qualification by EDF for its internal needs. During two years, it is put up to date every six months, by integrating only bug fixes, without modification of the user interface or documentation.

- ***stable-updates***

  This refers to the intermediate states between two stable versions. They contain only bug fixes and are updated each month. That corresponds to the increments of versions 10.s.y, which constitute at the end of six months the new "stable" version 10.s+1.

- ***unstable***

  This version is weekly updated with the new features and bug fixes. That refers to the updates numbered 11.x.y.

- ***testing***

  That version is a frozen state, each six months, of the "unstable" version. It is updated every six months.

At the time of this writing, October 2013, we have:

- stable, 11.4.
- stable-updates, none yet.
- unstable, the same as testing.
- testing, 12.0.

5.3 *Code_Aster* setup

The following is a summary of how I solved some issues, with quite a bit of trial and error, but, there may be other ways, or my advices may not work.

Once we get working with *Code_Aster* graphical tools, like STANLEY, the tool may not launch at all with nasty messages in the *.mess* file. Here is the usual workaround:

In the **ASTK** window, menu Configuration » Preferences » Network:

- the Client machine name field should be the machine name, which can be obtained by typing the command 'hostname' in a terminal;
- the Domain name can be left empty;
- Forced DISPLAY variable can be set to :0.0;
- and button ssh and scp pushed on.

And STANLEY should then launch itself on request.

Some other issues may need to apply the following recipes:

In the **ASTK** window, menu Configuration » Servers... the Server name maybe changed from localhost to the machine name as above.

If nothing happens when we push TRACER in **STANLEY** window, despite a green light, lets try the following:

In the **STANLEY** window, menu Parametres » Editer, in the pull down list right off Mode choose Gmsh/Xmgrace, push OK it should then work in this mode.

We may also choose Salome mode, if Stanley is ran from a Salome-Meca study. To make it run from stand alone *Code_Aster* I suppose we need a stand alone version of Salome and a bit of tinkering around with Port de Salome and Chemin de RunSalomeScript, though I have never used STANLEY this way.

When the setup is right the interactive follow-up box in **ASTK** may be checked and when pushing Run a terminal window opens telling us all

what's going on, in fact it is almost a copy of what is written in the *.mess* file, quite useful in case of longish problems, we know where we are.

Some of these advices are worth only for a single machine setup, the same machine acting as client and calculation server.

5.4 *Code_Aster* update

If we choose to install the "testing" version of *Code_Aster* it is possible to create an "unstable" version in the same install directory and update it[1].

As its name implies this version is the last out of the box development version, it may be buggy, and should not be used for production work.

How to build and update this version is fully described on *Code_Aster* web site, particularly on the "Download" tab.

5.5 *Code_Aster* directories maintenance

Every time a study is launched:

- a directory is created in */tmp*, its name maybe something like *user-machine-xxxx-user*[2] where:
  - `user` is user's name;
  - `machine` is machine name;
  - `xxxx` is a randomly created number.

  this directory is created with user permissions, after the study is successfully finished it contains almost only *xxxx-user.export*;
- a few files are created in *$HOME/flasheur*, they are named, for example *frame3-oxxxx-user*[3]. File *frame3-oxxxx-user* contains full information about the calculation run, a bit more than the *.mess* file.

[1] Updates come about every week.
[2] This naming convention changed from time to time.
[3] There are also files with preceding letter o, i, u ,e ,p.

These directories and files are never automatically deleted [1] [2], we have to delete them at times otherwise *Code_Aster* may run out of disk space, in which case the study may stop abruptly without any clear error message.

5.6 Salome-Meca Installation

To get Salome-Meca go to this link:

`http://www.code-aster.org`

Look for the download area, once found download the archive that suits.

For the installation, *strictly* follow the instructions given on the page.

For this, I make the installation in */opt*, owned by regular user with read write access or *$HOME/opt*.

5.7 Salome Installation

To get Salome go to this link:

`http://www.salome-plaform.org`

Look for the download area, once found download the archive that suits[3].

For the installation, *strictly* follow the instructions given on the page.

As openSuSE, which I use, is not listed under the officially supported binaries list, I use the *Universal binaries* and I am happy with this, just like for Gmsh I never made a compilation from sources.

For this, I make the installation in */opt*, owned by regular user with read write access or *$HOME/opt*.

[1] At least on my openSuSE distribution.
[2] However a study */tmp* directory and the files in *$HOME/flasheur* can be deleted in the **ASJOB** window.
[3] You have to register and log to be able to download.

5.8 Salome or Salome-Meca Installation Problems

If Salome or Salome-Meca fails to launch with the following lines printed in the terminal:

```
runSalome running on dedalus
Searching for a free port for naming service: 2810 2811 2812 - OK
Searching Naming Service++++++++++++++++++++++++++++++++++++.....
```

we should have a look at the file */etc/hosts* it should contain, amongst some others, two lines like these ones:

```
127.0.0.1       localhost
.................
127.0.0.2       machine_name.domain_name machine_name
```

The first one is the main loop back `IP-Address`. In the second one:

- `127.0.0.2` is a second loop back `IP-Address`[1];
- `machine_name.domain_name`, given by typing `hostname -f` in a terminal is the `Full-Qualified-Hostname` ;
- `machine_name` is the `Short-Hostname` given by typing `hostname` in a terminal.

If this second line is not present we have to add it[2], without altering the first line.

Finally one should NEVER try to make a first run of Salome or Salome-Meca as root, as this seems to upset many of the configuration files [3], that is why I strongly advise to install as regular user.

[1] It could just as well be 127.0.10.1 or 127.0.10.2.
[2] With su privileges.
[3] I bet this comes from omniORB.

5.9 Gmsh Installation

To get Gmsh go to this link:

`http://geuz.org/gmsh/`

Download the "Current stable release", which is a *.tgz* archive. Unpack it somewhere, I do that in */opt*, again as regular user.

The executable is then */opt/gmsh/bin/gmsh*. Run it any way, command line, launching script or window manager menu entry[1].

There is no real need to compile anything from source for an everyday use, I have never done it[2]!

On the same link under *Documentation* the *Reference manual* is also a very useful mine of information.

5.10 A word about CAELinux

There is a very easy way to try out all these programs, it comes under the shape of a live CD containing all of them, plus much more. This is called CAELinux.

It can be downloaded from: `http://caelinux.com`

I have started this way!

The site contains also a wiki, a forum and many useful information.

In the Wiki among may useful sections I must quote the ones by:

- Claws Andersen: `http://www.caelinux.org/wiki/index.php/Contrib:Claws`;
- Kees Wouters: `http://www.caelinux.org/wiki/index.php/Contrib:KeesWouters`;

which contain many information, hints and examples.

[1] It is always a good idea to try launching any software from a terminal for the first time as there is then a clear warning of what is missing or what is going wrong.

[2] Except for fun!

5.11 About the forums

Code_Aster web site hosts some forums which are very helpful in solving many issues. It's like a potluck. What's available is what people have brought. And the forums must not be confused with a commercial software hot line help.

The same applies to Salome or Gmsh forums or mailing lists.

5.12 Distribution, window manager and more

Since 1998 I have used the openSUSE distribution[1], having started a few years before with a Slackware, I found life to be easy with it. The web site is:

`http://software.opensuse.org`

To make things a little bit more difficult I use "FVWM" as a window manager, it can be heavily customized through hand written configuration files, the look and behavior can be almost anything one likes[2]. The web site is:

`http://www.fvwm.org`

And, of course, this book is made with LaTeX, what else?

[1] At the time it was named S.u.S.E Linux, and that was version 5.2.
[2] Like the left-hand side tittle bar in the screen shot of this book.

Bibliography

[Roark] ROARK, Raymond J. and YOUNG, Warren C. (1976). *Formulas for Stress and Strain*. New York: McGraw-Hill Book Company.

First published in 1938, many times updated, has been, and still is, the reference handbook for a few generations of engineers all over the world!

[Blevins] BLEVINS, Robert D. (2001). *Formulas for Natural Frequency and Mode Shape*. Malabar: Krieger Publishing Company.

A comprehensive set about anything that can vibrate.

[Tuma] TUMA, Jean J. (1988). *Handbook of Structural and Mechanical Matrices*. New York: McGraw-Hill Book Company.

[Donaldson] DONALDSON, Toby (2009).*Visual Quickstart Guide Python*. Berkeley: Peachpit Press.

Very short, 185 pages, yet comprehensive introduction to Python.

Index