

NONLINEAR QUASI-STATIC FINITE ELEMENT MODELING OF
REINFORCED CONCRETE BEAMS WITH THE OPEN SOURCE
SOFTWARE PROGRAMS CODE_ASTER AND SALOME_MECA

A Report Submitted to the
College of Graduate and Postdoctoral Studies
in Partial Fulfillment of the Requirements
for the degree of Master of Engineering
in the Department of Civil, Geological and Environmental Engineering
University of Saskatchewan
Saskatoon

By
Franks Veras Maia

©Franks Veras Maia, August 2019.

PERMISSION TO USE

Attribution-NonCommercial-ShareAlike 4.0 International (CC BY-NC-SA 4.0)

This is a human-readable summary of (and not a substitute for) the license [19].

You are free to:

- **Share** — copy and redistribute the material in any medium or format.
- **Adapt** — remix, transform, and build upon the material.

The licensor cannot revoke these freedoms as long as you follow the license terms.

Under the following terms:

- **Attribution** — You must give appropriate credit, provide a link to the license, and indicate if changes were made. You may do so in any reasonable manner, but not in any way that suggests the licensor endorses you or your use.
- **NonCommercial** — You may not use the material for commercial purposes.
- **ShareAlike** — If you remix, transform, or build upon the material, you must distribute your contributions under the same license as the original.
- **No additional restrictions** — You may not apply legal terms or technological measures that legally restrict others from doing anything the license permits.

Notices:

You do not have to comply with the license for elements of the material in the public domain or where your use is permitted by an applicable exception or limitation.

No warranties are given. The license may not give you all of the permissions necessary for your intended use. For example, other rights such as publicity, privacy, or moral rights may limit how you use the material.

ABSTRACT

VERAS MAIA, F. Nonlinear Quasi-Static Finite Element Modeling of Reinforced Concrete Beams with the Open Source Software Programs `code_aster` and `salome_meca`. techreport, University of Saskatchewan, Saskatoon, SK, Canada, Aug. 2019. doi: [10.5281/zenodo.2529729](https://doi.org/10.5281/zenodo.2529729).

This report investigates the usefulness of the open-source software programs `salome_meca` and `code_aster` in nonlinear quasi-static finite element modeling of reinforced concrete beams. To begin with, the literature review includes the history, principles and advantages of the open-source software, its principles and advantages, the theory of finite element modeling and an introduction to `salome_meca` and `code_aster`. Experimental data from the literature were used for modeling and validation. All steps were explained: from geometrical modeling, through meshing, and generation of the command file. A one-dimensional beam with a multi-fiber section of concrete and reinforcement steel was used for modeling in this report. Mazars damage model was assigned for concrete fiber, and the reinforcement steel fiber followed a von Mises behaviour. Total solutions and calculations were performed in 8.49 seconds in real time (wall clock) resulting in an ultimate load of 334.7 kN and mid-span deflection of 10.1 millimeters. In comparison with the experimental data, the numerical ultimate load was 1.1% greater than the experimental value, and the numerical mid-span deflection was 6.3% greater than the one from the experiment. The curve load versus displacement also had a good fit to the experimental data, with a calculated R squared of 0.999. Therefore, both software programs may be adopted by engineers, granted that the necessary time is invested for a deep understanding of the models, constraints, limitations, and configuration options for each stage.

ACKNOWLEDGEMENTS

First and foremost, I would like to express my gratitude to my supervisor and mentor, Dr. Mohamed Boulfiza, for his guidance and encouragement. Thank you for being a good mentor and for guiding me on the right path.

I am also grateful for the unconditional support of my partner, Dr. Ornwipa Thamsuwan. This degree would not be attained without her.

Lastly, I appreciated the company of the lab-mates from the Centre for Advanced Numerical SIMulation (CANSIM), who helped to make the daily routine fun and enjoyable.

"Finite element analysis is an art to predict the future." (Klaus-Jürgen Bathe)

CONTENTS

Permission to Use	i
Abstract	ii
Acknowledgements	iii
Contents	v
List of Tables	vi
List of Figures	vii
List of Codes	viii
1 Introduction	1
1.1 Objectives	1
1.2 Document Structure	1
2 Literature Review	2
2.1 Finite Element Method	3
2.2 Open Source Software	7
2.2.1 GNU/Linux	9
2.2.2 Git and Github	11
2.2.3 Programming Languages	12
2.3 code_aster and salome_meca	14
2.3.1 Modeling Reinforced Concrete with code_aster and salome_meca	18
2.4 Mazars Damage Model for Concrete	21
2.5 Experimental and Analytical Reexamination of Classic Concrete Beam Tests	22
3 Methodology	24
3.1 Software Setup	24
3.2 Geometrical Modeling	25
3.3 Meshing	26
3.4 AsterStudy	27
4 Results and Analysis	36
5 Summary, Conclusions, and Recommendations	40
References	41
Appendix A Geometrical Model	46
Appendix B Command File	49

LIST OF TABLES

2.1	Overview of material characteristics	5
2.2	Classification of nonlinear analysis	6
2.3	Cross-Section details of Vecchio-Shim beams	23
2.4	Reinforcement properties of Vecchio-Shim beams	23
2.5	Concrete properties of Vecchio-Shim beams	23
2.6	Test results for Vecchio-Shim beams	23
4.1	Time required to run the commands	37
4.2	Summation of nodal reactions at the supports	38

LIST OF FIGURES

2.1	Project scope	2
2.2	Physical FEM	4
2.3	Model updating in physical FEM	4
2.4	High-level view of the GNU/Linux operating system	9
2.5	GNU/Linux development in the Unix timeline	10
2.6	Three snapshots used in a typical merge	12
2.7	GitHub users and commits relative to number internet users	13
2.8	General principles of code_aster	15
2.9	SALOME: generic framework for pre and post-processing	16
2.10	Screenshot of salome_meca	16
2.11	Graphical user interface of the AsterStudy module	17
2.12	Modeling the decohesion steel/concrete	19
2.13	1D Multi-fiber modeling of reinforced concrete beams	20
2.14	Test setup of the twelve Vecchio-Shim beams	22
2.15	Cross-sections of the twelve Vecchio-Shim beams	22
3.1	Geometrical modeling of the beam	25
3.2	Meshing of the section	26
3.3	'SECTION' mesh information	27
3.4	'BEAM' mesh information	27
3.5	Case parameters	35
4.1	Load versus mid-span deflections of experimental data from Vecchio-Schin and numerical data from salome_meca	39

LIST OF CODES

3.1	code_aster overridden preferences	24
3.2	Definition of points in Python	25
3.3	Assignment of numerical values to variables	27
3.4	Reading of the mesh	28
3.5	Definition of fibers	29
3.6	Definition of the model	29
3.7	Definition of the model	29
3.8	Definition of materials	30
3.9	Definition of functions	31
3.10	Boundary conditions	32
3.11	Material behaviour	32
3.12	Material behaviour	33
3.13	Post-processing	34
3.14	Exporting results	34
A.1	Geometrical model for beam OA1	46
B.1	Command file for Beam OA1	49

CHAPTER 1

INTRODUCTION

1.1 Objectives

This study sets out to investigate and report the usefulness of the open source software programs `code_aster` and `salome_meca` in implementing nonlinear quasi-static finite element modeling of reinforced concrete beams. Therefore, specific objectives come forth as branches:

- Review of the open source software programs `code_aster` and `salome_meca`; and
- Validation, with available experimental data, of the results from the finite element modeling of reinforced concrete beams with the open source programs `code_aster` and `salome_meca`.

1.2 Document Structure

The report structure follows the Academic Policies and Guidelines of the Handbook for Graduate Students [20] published by the Department of Civil Geological and Environmental Engineering of the University of Saskatchewan. The following chapters are:

- Chapter 2, Literature Review, presents the conceptual framework with an exploratory review of the topics: finite element method; open source software and the programs `code_aster` and `salome_meca`; mazars damage model for concrete; and experimental data of classic concrete beam tests;
- Chapter 3, Methodology, presents the methods for implementing and evaluating finite element procedures using the software programs `code_aster` and `salome_meca`;
- Chapter 4, Results and Analysis, presents the results, analysis and pertinent discussions;
- Chapter 5, Summary, Conclusions, and Recommendations, presents the summary of the project, conclusions, and recommendations for future work.

CHAPTER 2

LITERATURE REVIEW

Mechanics is the science that studies the effect of forces and energy in bodies [11, 27], and it can be divided in the following four branches [27], also shown in the top level of Figure 2.1.

- Theoretical mechanics, which applies fundamental laws and principles considering their intrinsic value, regardless of application;
- Applied mechanics, which is a bridge between the theoretical knowledge and the scientific or engineering applications, focusing on the mathematical modeling of physical phenomena;
- Computational mechanics, which uses numerical methods to translate mathematical models into instructions understood by a digital machine as sequences of operations; and
- Experimental mechanics, which collects data for the mathematical models and verifies the predictions generated by the theoretical, the applied, and the computational mechanics.

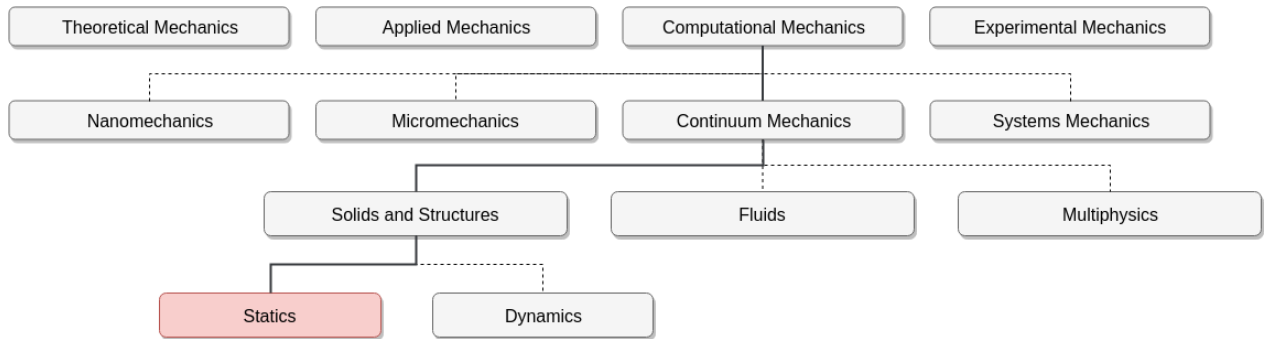


Figure 2.1: Project scope in branches and subbranches of Mechanics (adapted from: [27])

Computational mechanics can be subdivided into branches that consider the physical scale of the focus of attention, shown in the second level of Figure 2.1 and defined as [27]:

- Nanomechanics, which deals with phenomena at the molecular and atomic levels, e.g. in particle physics, chemistry, and quantum mechanics;
- Micromechanics, which studies crystallographic and granular levels of matter;

- Continuum mechanics, which studies bodies in a macroscopic level using continuum mechanics via homogenization by phenomenologically averaging of the microstructure. Traditionally, the areas studied by continuum mechanics are Solids and Structures, Fluids, or a mixture of Solids and Fluids named Multiphysics; and
- Systems, which studies complex aggregation of distinct bodies that perform a specific function, e.g. bridges, airplanes, building systems, etc.

Solids and Structures, i.e. the aggregation of solid elements, could be studied under two categories: Statics and Dynamics.

In Statics, inertia forces are ignored, and the effect of time can be either completely ignored or estimated [27]. On the other hand, in Dynamics, inertia forces and time are explicitly considered because the calculation of inertial and damping forces requires derivatives with respect to the time to be taken [27].

Linear statics deals with problems which have linear stress-strain material response and infinitesimal deformation; therefore, they follow linear equations [15, 27]. In contrast, nonlinear statics considers material and geometrical nonlinearity [15]. The former has a nonlinear stress-strain response, whereas the latter are also subjected to large strains, called finite deformation [15].

2.1 Finite Element Method

The Finite Element Method (FEM) is a numerical technique for solving systems of partial differential equations [15]. Although the theory of FEM cannot be traced to one single individual, the implementation and dissemination into everyday use started with M. J. (Jon) Turner at the Boeing Company between 1950 and 1962 [27].

Turner generalized and optimized the Direct Stiffness Method, where the structure was broken down to its minimum elements, processed, solved and post-processed. In addition, Turner supervised the development of the first continuum-based finite elements [27].

”Stiffness and Deflection Analysis of Complex Structures”, a paper published by Turner et al. in 1956 is considered as the beginning of the current FEM [72].

The first textbook in the subject was written by Olek Zienkiewicz, originally an expert in finite difference methods. Currently, it has been published in three volumes [82, 83, 84].

Mentions should also be made to B. M. Irons for inventing isoparametric models, shape functions, the patch test and frontal solvers; to R. J. Melosh, for recognizing the Rayleigh-Ritz link and systematizing the variational derivation of stiffness elements; and to E. L. Wilson, who developed the first open source FEM software [27].

Figure 2.2 shows the canonical structure of a physical simulation. The physical system is the source of the simulation process. The system is idealized and discretized for obtaining a discrete model, which is subsequently solved, verified and validated. The verification considers of the estimation of errors in the model

itself, whereas validation is a comparison between model solutions and real data. Simulation error from the validation is a combination of the solution error and the error resulted from the way the model is built, called model error.

An ideal mathematical model is rarely necessary, since the simulation can, in most cases, satisfactorily represent the physical system.

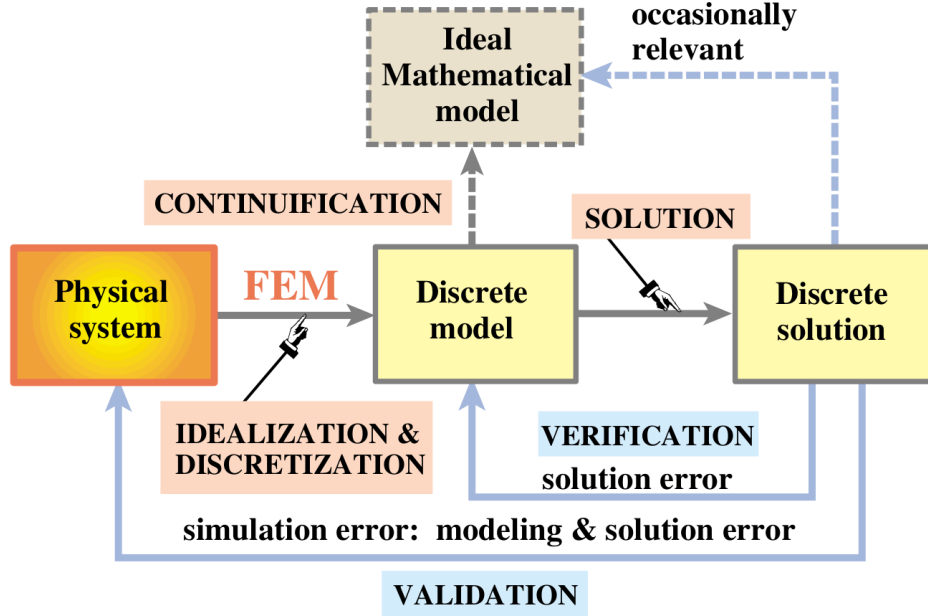


Figure 2.2: Physical FEM (source: [27])

Figure 2.3 shows the model updating strategy of the finite element analysis. An experimental database has relevant parameters from the physical system that will be used for validating results from the discrete model. The process of updating the current model (or choosing a completely new one) finishes when the error is deemed acceptable.

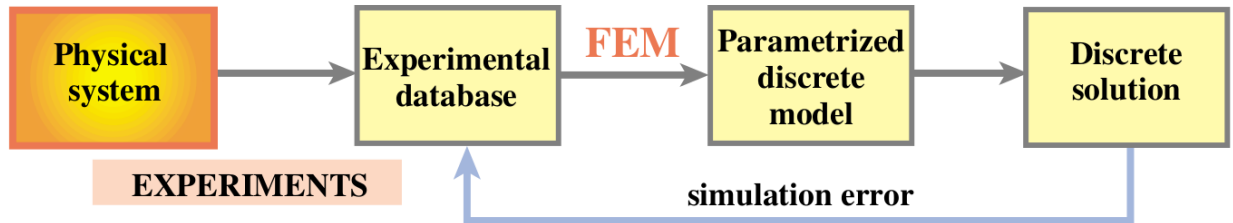


Figure 2.3: Model updating in physical FEM (source: [27])

In an idealized quasi-static structural system, displacements, rotations, and strains are small, and materials behaviour and contact forces are linear. Therefore, the system can be linearly modeled [8]. However, some materials do not have a linear response. This influences the type of analysis, formulations, and measurements, as described in Table 2.1. Other nonlinearities are classified in Table 2.2.

Table 2.1: Overview of material characteristics

Material	Characteristics	Examples
Elastic, linear or nonlinear	<p>Stress is a function of strain only; same stress path on unloading as on loading.</p> ${}^t\sigma_{ij} = {}^tC_{ijrs} {}^te_{rs}$ <p>linear elastic: ${}^tC_{ijrs}$ is constant</p> <p>nonlinear elastic: ${}^tC_{ijrs}$ varies as a function of strain</p>	<p>Almost all materials provided the stresses are small enough: steels, cast iron, glass, rock, wood, etc.; before yielding or fracture</p>
Hyperelastic	<p>Stress is calculated from a strain energy functional W,</p> ${}^t_0S_{ij} = \frac{\partial W}{\partial \delta \varepsilon_{ij}}$	Ruberlike materials
Hypoelastic	<p>Stress increments are calculated from strains increments</p> $d\sigma_{ij} = C_{ijrs} de_{rs}$ <p>The material moduli C_{ijrs} are defined as functions of stress, strain, fracture criteria, loading and unloading parameters, maximum strain reached, and so on</p>	Concrete models
Elastoplastic	<p>Linear elastic behavior until yield, use of yield condition, flow rule, and hardening rule to calculate stress and plastic strain increments; plastic strain increments are instantaneous</p>	Metals, soils, rocks, when subjected to high stresses
Creep	<p>Time effect of increasing strains under constant load, or decreasing stress under constant deformation; creep strain increments are noninstantaneous</p>	Metal at high temperatures
Viscoplasticity	<p>Time-dependent inelastic strain; rate effects are included</p>	Polymers, metals

Source: [8]

Table 2.2: Classification of nonlinear analysis

Type of analysis	Description of classification	Typical formulation used	Stress and strain measures
Materially-nonlinear-only	Infinitesimal displacements and strains; the stress-strain relation is nonlinear	Materially-nonlinear-only (MNO)	Engineering stress and strain
Large displacements; large rotations, but small strains	Displacements and rotations of fibers are large, but fiber extensions and angle changes between fibers are small; the stress-strain relation may be linear or nonlinear	Total Lagrangian (TL) Updated Lagrangian (UL)	Second Piola-Kirchhoff stress, Green-Lagrange strain Cauchy stress, Almansi strain
Large displacements, large rotations, and large strains	Fiber extensions and angle changes between fibers are large, fiber displacements and rotations may also be large; the stress-strain relation may be linear or nonlinear	Total Lagrangian (TL) Updated Lagrangian (UL)	Second Piola-Kirchhoff stress, Green-Lagrange strain Cauchy stress, Almansi strain

Source: [8]

2.2 Open Source Software

"Open Source" is a term coined in the spring of 1997 in California by a group of leaders in the free software community as an attempt to spread the use of free software among the industry [22]. "Source", in the field of software, is defined as "the form in which a computer program is written" [56] and "Open" attempts to approximate the ideal of free as in freedom instead of gratis [42, 65, 81] that drives open collaboration and results in innovation [43].

Although collaborative software writing and sharing started earlier - e.g. Donald Knuth's free publishing-quality TeX typesetting system¹ in 1979 [30] - the idea of free software started in September of 1983 with the conception of the GNU Project by Richard M. Stallman [64]. The GNU Project goal was to "write a complete Unix-compatible software system called GNU (for Gnu's Not Unix), and give it away free to everyone who can use it" [63, 64, 65].

Stallman defines a program as free software if [65]:

- *You have the freedom to run the program, for any purpose;*
- *You have the freedom to modify the program to suit your needs. (To make this freedom effective in practice, you must have access to the source code, since making changes in a program without having the source code is exceedingly difficult;)*
- *You have the freedom to redistribute copies, either gratis or for a fee; and*
- *You have the freedom to distribute modified versions of the program, so that the community can benefit from your improvements.*

The intrinsic requirement to the source code was the genesis of the General Public License (GPL) [78], the most widely recognized free software license in the world today [28]. The license ensures two main restrictions [21, 28, 32, 58, 61, 78]: first, that any derivative work must also be distributed under the GPL; and second, that no additional restrictions should be included neither in the original work nor in the derivative work, written verbatim as "you may not impose any further restrictions on the exercise of the rights granted or affirmed under this License" [66], more restrictive than the open source definition [61].

The internationally recognized [51] open source definition [50] established by the Open Source Initiative has ten criteria for any software license to be labeled as "Open Source Software". The criteria are enumerated and individually paraphrased as follows.

1. Free Redistribution: any party is able to sell or give away the licensed software as a component of an aggregate software distribution. No royalties or other fees are strictly required;

¹The current report is written in the TeX, and the source code is available at <https://github.com/franksmaia/M.Eng>.

2. Source Code: the source code must be provided along with the program unless other available means of obtaining such code are extensively publicized. Reasonable fees to obtain the code can be charged, although free access to download over the internet is preferable. Obfuscation of code or output of a pre-processor or translator [45] is not allowed;
3. Derived Works: any work can be derived from an Open Source Software, and the license should allow the derived work to be distributed in the same terms of the original software;
4. Integrity of The Author's Source Code: licensed software may require that derived works carry a different name or version number from the original software;
5. No Discrimination Against Persons or Groups: discrimination against any person or group of persons is not allowed;
6. No Discrimination Against Fields of Endeavor: restrictions against software use in any specific field of knowledge or venture is not allowed;
7. Distribution of License: license rights should extend to any subsequent redistribution of the program without the need of additional licenses;
8. License Must Not Be Specific to a Product: programs that are components of a software distribution [31] hold its licenses independently. The holding rights extend to any party that uses the components;
9. License Must Not Restrict Other Software: the license should not restrict any other software of having another license; and
10. License Must Be Technology-Neutral: the license should not be technological bound or restricted to a specific style of interface.

There are several additional benefits from the adoption of open source software. Among those, security ranks the top. Two opposite views on software development are the "Security by Obscurity" [1] versus the security of "Many Eyeballs" [71]. The former is adopted by closed source software companies, whereas the latter is advocated by open source adepts.

Open source software developers adopt one of the two main strategies for production [58]: either develop closed code and release it to the public after specific milestones, also known as "Cathedral" model; or transparently develop in a rolling scheme, known as "Bazaar". The latter strategy attracts new developers, stimulates the reporting of bugs, and engages the community of users [28].

"Security by Obscurity" relies on the fact that users do not have access to the source code; thus, are not theoretically able to detect loopholes and mishaps of the programs [1]. The Security of "Many Eyeballs", on the other hand, regards access to the source code as positive: with a larger number of people employing their expertise to examine the code, fewer bugs and malicious code are expected in the software [71].

Security of "Many Eyeballs" was also coined as the Linus's Law [58]: "given enough eyeballs, all bugs are shallow". In a more technical definition, the Linus's Law states that "given a large enough beta-tester and co-developer base, almost every problem will be characterized quickly and the fix obvious to someone" [36]. This concept is one of the underlying reasons leading to the GNU/Linux success [58].

2.2.1 GNU/Linux

The GNU/Linux operating system is a successful example of open source software. All 500 most powerful supercomputers systems run on GNU/Linux [70], and a W3Tech usage statistics and market share research [76] shows that 68.9% of websites run on a GNU/Linux or a Unix compliant system. The Android mobile operational, which uses a Linux kernel, powered 76% of the estimated total number of smartphones worldwide in 2015 [67].

Technically, Linux is a kernel - an abstraction layer closer to the hardware [40], as shown in Figure 2.4. Its development started in 1991 by Linus Torvalds in collaboration with several programmers around the world [28]. Figure 2.5 shows the timeline of the Linux kernel development among other kernels. US Trademark number 1916230 reserves the brand Linux for "computer operating system software to facilitate computer use and operation." [73].

Linux filled a gap provided by the kernel design chosen by GNU project, which turned out to be harder to implement than imagined [40]. Distributed with GNU Applications and a command line interface known as "Shells" (Figure 2.4), GNU/Linux was the first operational system that could be used completely free of any proprietary software [21, 28, 58].

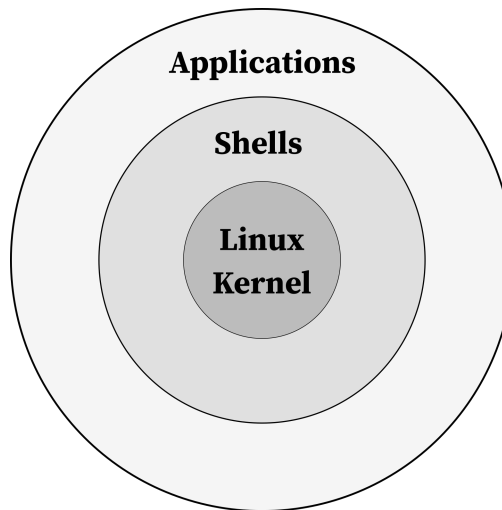


Figure 2.4: High-level view of the GNU/Linux operating system (adapted from: [40])

Initially only a few, hundreds of GNU/Linux distributions are available nowadays [44]. Users can choose from a variety of distribution category, country of origin, desktop interface, architecture, package management, release model, install media size, install method, multi-language support, init software, and status

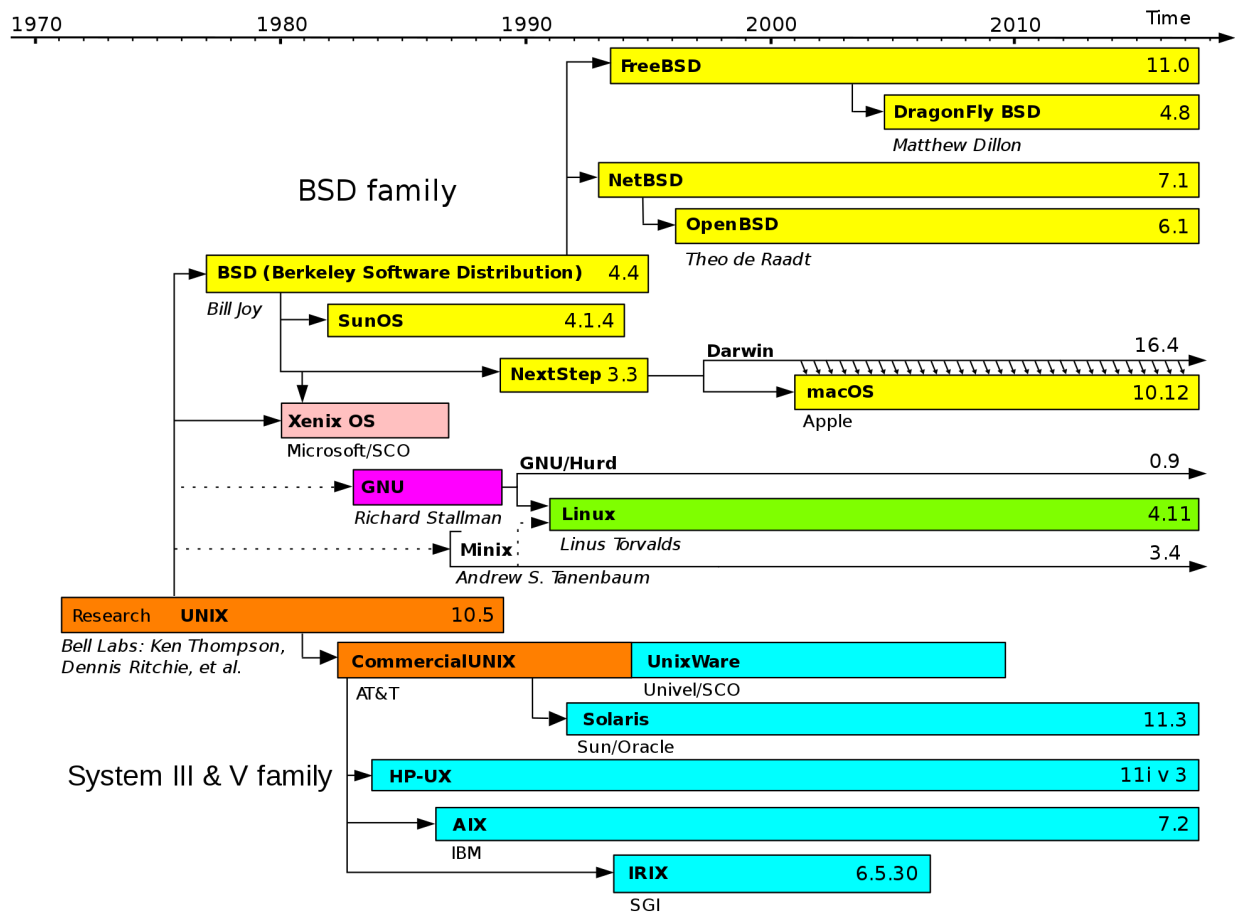


Figure 2.5: GNU/Linux development in the Unix timeline (source: [37])

[24].

Ranking among the top ten Linux distributions [25], Arch Linux was devised in 2002 by the Canadian computer science graduate Judd Vinet [75]. It is widely recognized for its software management infrastructure, customization and tweaking options, and extensive online documentation [25]. In addition to the official software packages provided, the Arch User Repository [3] contains more than 50 thousand packages [2] compiled by users with the script to automate the building of packages "makepkg" [6], generating package descriptions named "PKGBUILD" [4] which are available for installation with the package manager "pacman" [5].

Linus Torvalds's contributions to the open source community went further than the Linux kernel. Software developers interact through several channels during the life-cycle of an open source software [21, 23, 78]: web sites, mailing lists, message forums, version control systems, bug trackers, real-time chat systems, wikis, Q&A forums and other social networks services [28]. Facing the challenge of managing codes from several developers, he started the open source version control system Git [16].

2.2.2 Git and Github

A version control system is a mix of technologies and practices for tracking and controlling changes to files in a project, e.g. source code, documentation, and web pages [28]. Efficiently tracking a large project as the Linux kernel paved the way for the Git project's goals [16]: speed, simple design, strong support for non-linear development, and fully distributed. Common vocabulary adopted in Git should be clearly defined for a better understanding of the system. The following definitions derive from the official Git reference book [16].

commit

Commit is the action of registering previous file changes in a single permanent snapshot into the database. Each commit - i.e. each snapshot of the directory structure and content of files - also contains the author's name, email address, pointers to the commit or commits that came directly before, and a message defined by the user. The commit message contains a synthesis of the work done between the last snapshot and the current one.

branch

A branch is a division of a project where developers can implement new features and solve bugs in isolated lines of development from each other. Figure 2.6, for example, shows two different branches ("master" and "iss53") of a single project. The common ancestor is the snapshot (or commit) "C2", where the branch "iss53" was started. When development ends in the branch "iss53" on commit "C5", it could be merged back into the main branch, now at commit "C4". Conflict might arise if the same code is changed by two different people. In this case, the version control system detects and notifies at least one user to intervene.

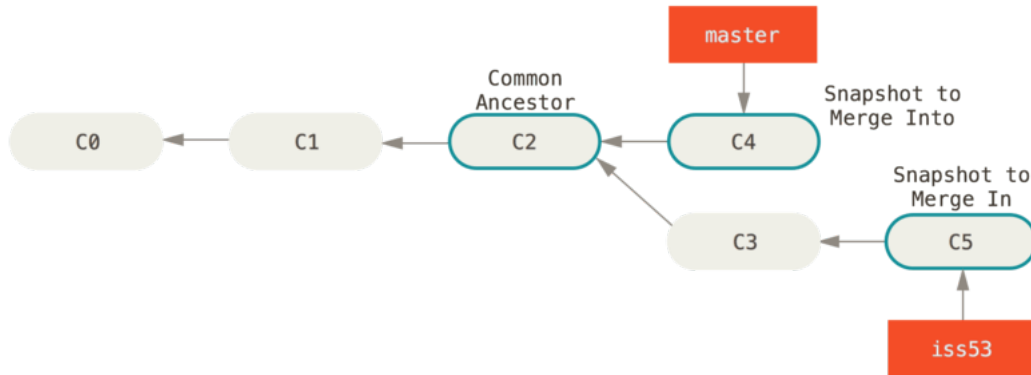


Figure 2.6: Three snapshots used in a typical merge (source: [16])

repository

A repository is a database where the information about a project is stored. A developer can either create a local database or clone it from an existing repository available, for example, at GitHub ². GitHub has more than 31 million registered users across the globe, and over 100 million repositories on the platform [33]. A study of information geographies conducted by the Oxford Internet Institute [35] in 2015 shows the worldwide adoption of the GitHub platform (Figure 2.7), even though there are still large differences in the rate at which people from different countries contribute [35].

checkout

Checking out is the process of switching branches and obtaining the content into the working directory.

2.2.3 Programming Languages

A programming language is a system for writing instructions to be evaluated by a machine as sequences of operations [54]. The Government of Canada’s terminology and linguistic data bank defines code as “a set of rules and conventions according to which the signals representing data should be formed, transmitted, received and processed” [55].

One basic classification of programming languages regards their level of abstraction from the computer architecture: whether a developer code using instructions and data objects at the machine level or through layers of abstraction [38]. In low-level programming languages, the developer deals with the details of opcodes, mnemonics, registers and so forth available on the programming cards issued by the chip manufacturers [13]. High-level programming languages; however, provide the flexibility of specifying the detailed computation

²<https://github.com/>

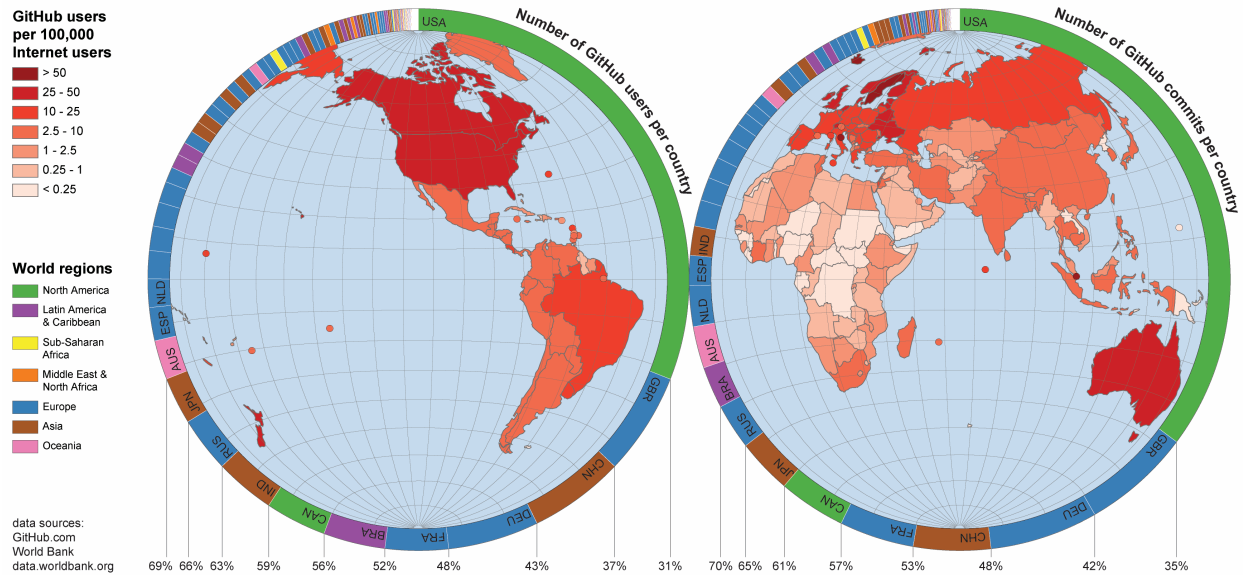


Figure 2.7: Number of GitHub users and commits by these users relative to the number of internet users in 2015 (source: [35])

in a concise, clear and reliable expression [38], quicker and portable [12]. A more flexible level classification considers that "a programming language is low level when its programs require attention to the irrelevant" [53].

Invented by the German engineer Konrad Zuse in 1945, Plankalkül ("Plan Calculus") was the first high-level programming language [59, 60]; nevertheless, the language did not have practical use due mainly to the lack of orientation to numerical computations and the unorthodox elements of Plankalkül in two-dimensional notation [9].

One of the first programming languages with practical use was FORTRAN, developed in the 1950's by an IBM team led by John Backus [12] with the purpose of being a vehicle for expressing scientific and mathematical computation, while allowing for code efficiency [79]. Most programs in Finite Element Method were developed in FORTRAN [27]. Since mid 1990's, FORTRAN has been gradually substituted by C/C++ as a low-level language; and by Python, Matlab, Mathematica as higher-level languages [27].

C was developed in the early 1970's by Dennis Ritchie of Bell Laboratories for implementing the UNIX operating system [12] with many important ideas derived from the language BCPL, developed by Martin Richards [41]. The language became an American National Standards Institute (ANSI) standard in 1989 [12], extensively based on the first edition of the book "C Programming Language" [41] written by Brian W. Kernighan and Dennis Ritchie. C is widely used as a low-level programming language in Finite Element Method programs [27]. Currently, the C programming language is an ISO 9899:2018 standard [39].

C++ is an extension of C developed in the 1980's by Bjarne Stroustrup, also from Bell Laboratories, including support for object-oriented programming [12]. The Annotated C++ Reference Manual by Ellis and Stroustrup [26] was the basis for the first standard by the International Organization for Standardization

(ISO), now in its 2018's version [39].

Python is a general-purpose high-level programming language that can be used to write any kind of program that do not demand direct access to the computer's hardware [38]. It was first released in 1991 by Dutch programmer Guido van Rossum [52] and modernly implemented in 2000 with the Python Enhancement Proposal (PEP) Purpose and Guidelines [77]. Python is the third most used language across all regions in private, public, and open source repositories [34]. Scientific Computing Tools for Python provides the SciPy ecosystem with "general and specialized tools for data management and computation, productive experimentation and high-performance computing" [62], among them:

- NumPy: "fundamental package for scientific computing with Python" [49];
- Matplotlib: "a Python 2D plotting library which produces publication quality figures in a variety of hard copy formats and interactive environments across platforms" [69];
- SymPy: "for symbolic mathematics and computer algebra" [68]; and
- pandas: "providing high-performance, easy to use data structures" [57].

Another common factor of the aforementioned projects is the Numerical Foundation for Open Code and Usable Science (NumFOCUS) as source of financial sponsorship [48]. NumFOCUS mission is "to promote sustainable high-level programming languages, open code development, and reproducible scientific research (...) through educational programs and events as well as through fiscal sponsorship of open source scientific computing projects" [47].

2.3 `code_aster` and `salome_meca`

code_aster is a stand-alone thermo-mechanical open source solver developed and published by Électricité de France (EDF), the French electric utility company [17]. The name ASTER stands for "Analyses des Structures et Thermomécanique pour des Études et des Recherches", the French version of "Structural Analysis and Thermomechanics for Studies and Research" [80].

Several motivations impelled EDF for coding its own finite element package [7]:

- To provide a unique code for mechanics: the need for numerical tools was scattering the EDF Research & Development teams. Instead of using a single general software, every team adopted its own package;
- The high cost associated with procurement, release and maintenance of commercial packages;
- The heavy quality control requirements regarding nuclear safety requirements at EDF plants which demanded a solid platform for aggregation the accumulated experiences throughout different generations of engineers; and

- The consideration of loading history, manufacturing process, and possible repairs on the life span analysis of electricity power plants demanded more calculation hypothesis than the ones used for classical engineering. Nonlinear approaches, thermal effects, dynamic loading stresses, and fluid structure interactions had to be taken into account.

The first version was internally published in 1989 and the code was further licensed under the GNU GPL in October 2001 [80]. EDF uses the software today for modeling the behavior or pathology of its equipment on [7]:

- 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; and
- The civil engineering applications: pre-stressed concrete containment building, cooling towers, hydro-electric dams, nuclear waste storage sites.

Nowadays, more than 3500 verification test cases covering all features are available, which provide an easier starting points for beginners [17].

Isolated, code_aster is a solver configured by a command file (.comm). As shown in Figure 2.8, code_aster takes the **pre-processing** data (e.g. CAD, meshes) combined with the data settings of the mechanical problem (e.g. constitutive models, behaviours, material parameters), creates a finite element model and **solves** it. The results are usually displacement fields, which can be **post-processed** into other fields of interest (e.g. stress, strain).

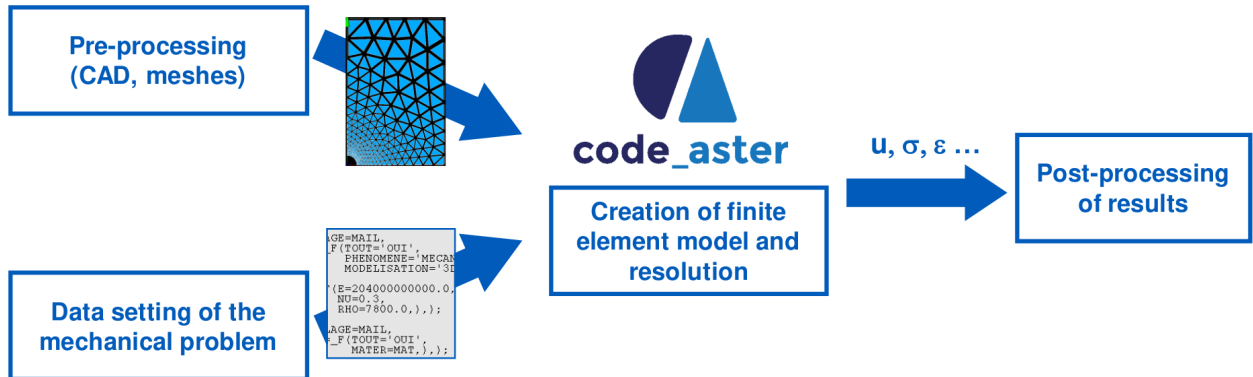


Figure 2.8: General principles of code_aster] (source: [18])

SALOME is another open source software that provides a generic platform for pre- and post-processing for numerical simulation [7]. Figure 2.9 shows the scope of SALOME with a blue background.

The software is based on an open and flexible architecture made of reusable components, and provides a Graphical User Interface (GUI) for the user [7].

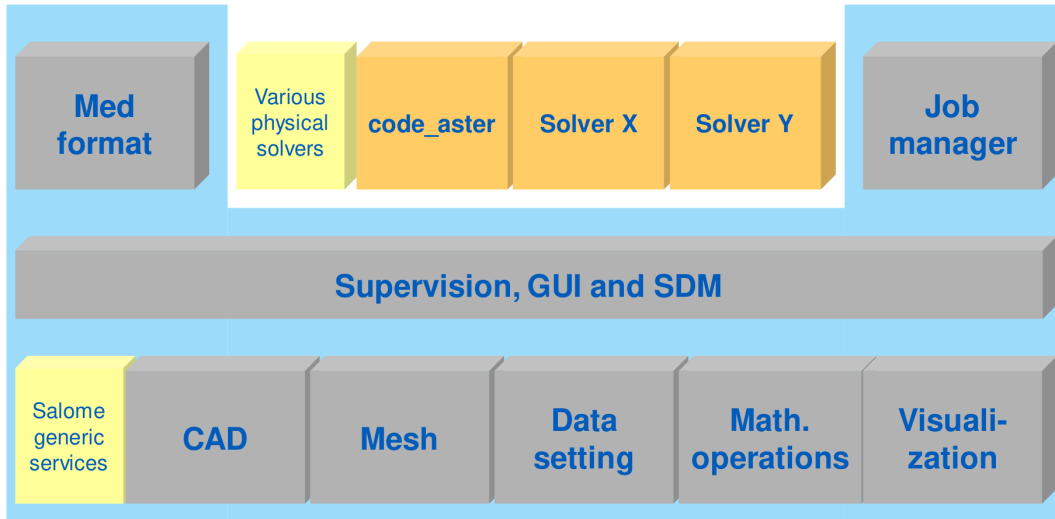


Figure 2.9: SALOME: generic framework for pre and post-processing] (source: [18])

salome_meca is an integrated and complete GUI that allows a complete study to be performed in a single user interface. One of the modules, AsterStudy, can automate syntax checking; thus, it facilitates the building of the command file [7].

Figure 2.10 shows a cropped screenshot of **salome_meca**, once started. The modules can be started either by the drop-down menu or by clicking on the items identified in the photo.

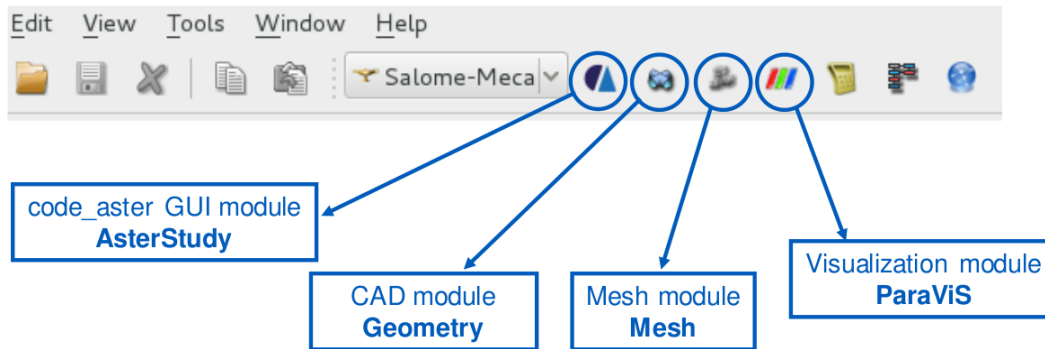


Figure 2.10: Screenshot of **salome_meca** (source: [18])

The overall steps for a finite element analysis follow the standard of common packages:

- Define geometries of the bodies under study using the CAD module;
- Define the meshes including type of elements and refinements using the Mesh module;
- Set all data related to the problem (i.e. model and material definitions, boundary conditions and loads, analysis, etc.) using the AsterStudy module;
- Launch the computation and survey using the AsterStudy module; and

- Visualize the results using the ParaViS module,

Figure 2.11 shows the graphical user interface of the AsterStudy module. The main features of the module are identified. In the "Data Settings" management tab, a case was defined with two stages: st1_thermal.analysis and st2_mechanical.analysis. The separation of stages are used to divide the analysis types, with the output of the thermal analysis being used in the mechanical one.

Each stage has its own commands separated in categories, which are organized in the sorted order of use on a series of drop-down menus indicated in Figure 2.11 as "Main commands".

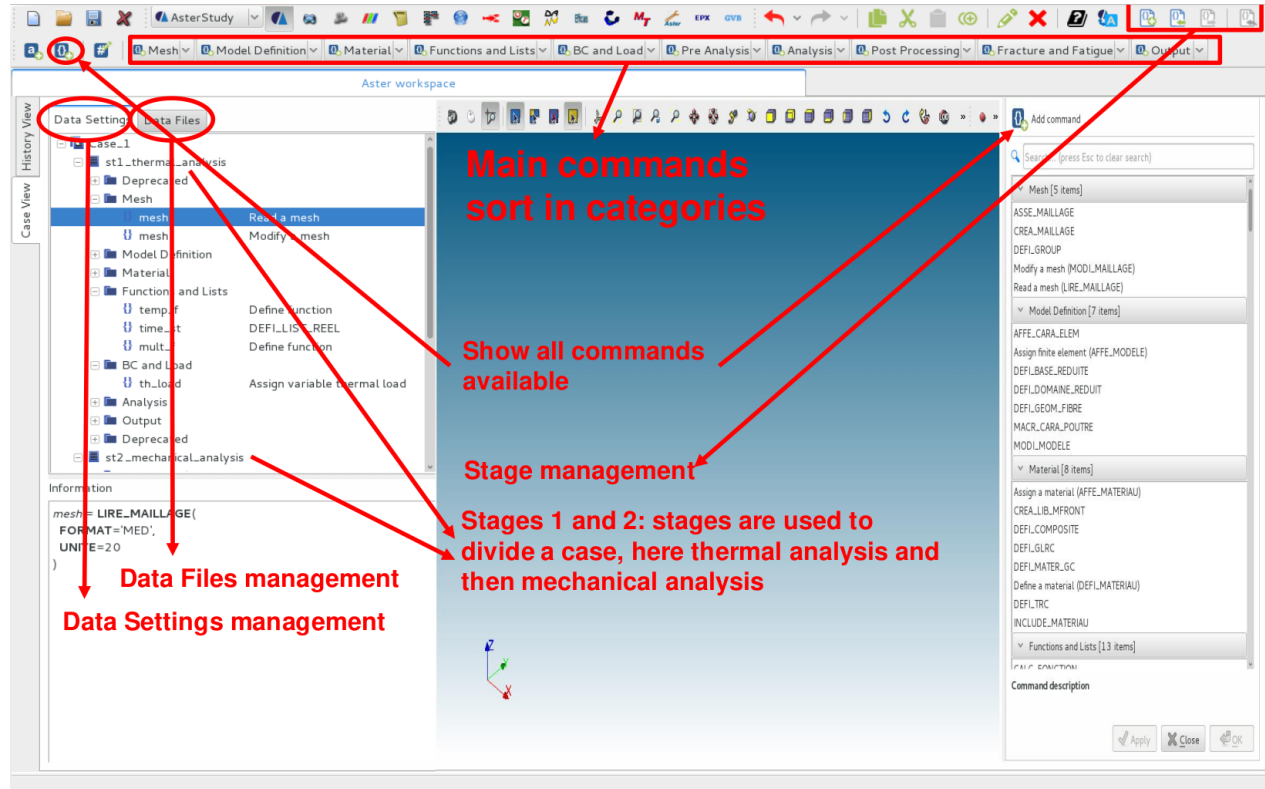


Figure 2.11: Graphical user interface of the AsterStudy module (source: [18])

The following items indicate the necessary steps for a 3D analysis of a reinforced concrete beam under 4-point bending test, considering previously defined 3D geometries and meshes of beam and reinforcement steel.

- Load the meshes using on the Mesh drop-down menu;
- Define the model for the case analysis, e.g. 3D;
- Choose the materials and their properties, e.g. Young's modulus, Poisson's ratio, etc;
- Insert loading functions and time-step lists, e.g. tell the solver when and with which intensity the loads should be applied;

- Define the boundary conditions (BC) and loads, e.g. defined the fixed and pinned supports and the load region;
- Insert any applicable pre-analysis, e.g. a fluid structure interaction;
- Choose the type of analysis and fine-tune solver parameters, e.g. static linear with Newton-Raphson method and automatic re-partitioning of time-steps;
- Define any post-processing calculations, e.g. von Mises;
- Insert fracture and fatigue analysis, if applicable; and
- Define the output data, e.g. tables, spreadsheets, and deformed meshes.

2.3.1 Modeling Reinforced Concrete with `code_aster` and `salome_meca`

Module 4 of the official `code_aster` and `salome_meca` training materials regards civil engineering models, separating the modeling of reinforced concrete in four categories [46]:

- in a 3D model;
- in a 2D model;
- in a 1D model; and
- with a global model.

3D Model

For a 3D model, four options could be chosen:

1. `BARRE` (or if needed `POU_D.T`);
 - Steel meshed with segments of two nodes (`SEG2`);
 - Behavior of the steel is 1D (e.g. `GRILLE_ISOT_LINE`);
 - Steel and concrete nodes match perfectly; and
 - Assume perfect bond between steel and concrete.
2. `GRILLE MEMBRANE`;
 - Steel meshed with plane elements (`QUAD4`, `TRIA3`, `QUAD8`, `TRIA6`);
 - Steel and concrete nodes match perfectly;
 - Assume perfect bond between steel and concrete;
 - Behavior of the steel is 1D (e.g. `GRILLE_ISOT_LINE`); and

- Meshes for different directions of reinforcement are overlaid.

3. MEMBRANE;

- Steel meshed with plane elements (QUAD4, TRIA3, QUAD8, TRIA6);
- Steel and concrete nodes match perfectly;
- Assume perfect bond between steel and concrete; and
- Limits the behaviour law to linear elastic.

4. 3D.

- Steel meshed with 3D elements;
- Assume perfect bond between steel and concrete; and
- No behaviour law restrictions.

Slippage could be introduced by including 3D INTERFACE elements between the 3D concrete and the 3D or MEMBRANE steel using the behaviour law CZM_LAB_MIX, as shown in Figure 2.12.

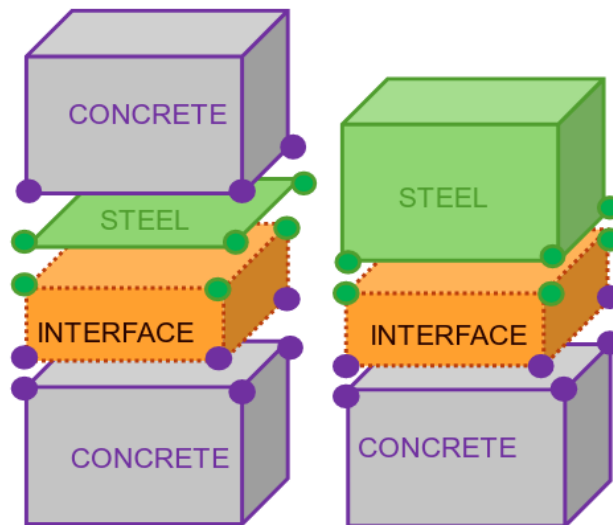


Figure 2.12: Modeling the decohesion steel/concrete (source: [46])

2D Model

For a 2D model, three options could be chosen:

1. 2D_BARRE;

- Steel meshed with segments of two nodes (SEG2);
- Behavior of the steel is 1D (e.g. GRILLE_ISOT_LINE); and

- Assume perfect bond between steel and concrete.

2. 2D;

- Steel meshed with plane elements (QUAD4, TRIA3, QUAD8, TRIA6);
- No behaviour law restrictions; and
- Assume perfect bond between steel and concrete (decohesion can be introduced by X_JOINT elements with JOINT_BA law).

3. GRILLE_EXCENTREE.

- Steel meshed with the plane elements QUAD4 or TRIA3;
- Meshes for different directions of reinforcement are overlayed;
- Assume perfect bond between steel and concrete; and
- Limits the behaviour law to 1D.

1D Model

Modeling the steel in reinforced concrete for a 1D model requires the use of the multi-fiber beam `POU_D_EM` or `POU_D_TGM`. The beam is meshed in a 1D dimension sections of the beam are defined point by point, as shown in Figure 2.13.

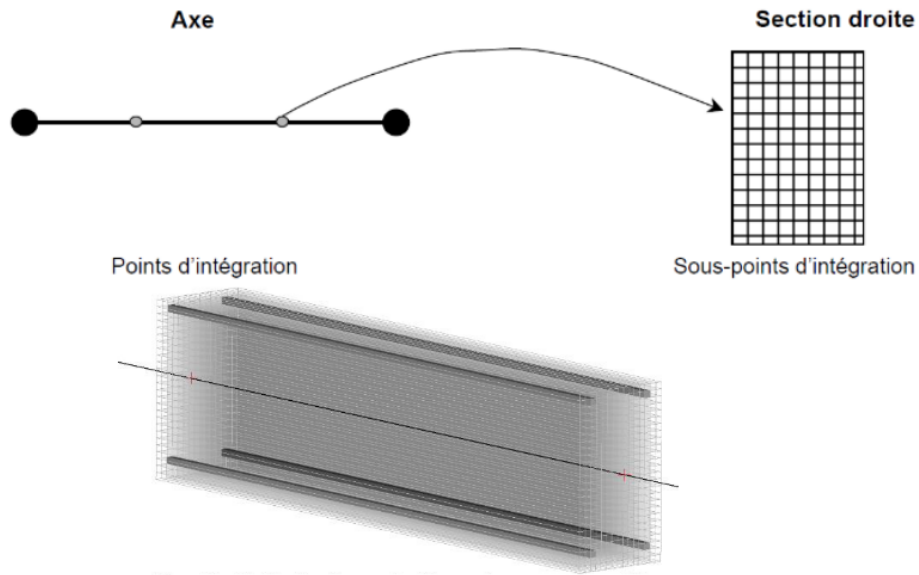


Figure 2.13: 1D Multi-fiber modeling of reinforced concrete beams (source: [46])

Global Model

A global model requires a homogenization of the materials as one single material on which a global constitutive law will be applied, with the advantage of more robustness (without softening) [46]. Elements should be of the type DKTG with one of the following constitutive laws:

- GLRC_DM: moderate damage and symmetrical reinforcements;
- DHRC: moderate damage plus cracking; and
- GLRC_DAMA: impact damage.

2.4 Mazars Damage Model for Concrete

The model of Mazars [29] was elaborated within a framework of the mechanics of damage.

The stress is given by:

$$\boldsymbol{\sigma} = (1 - D)\mathbf{E}\boldsymbol{\varepsilon}^e \quad (2.1)$$

Where:

- \mathbf{E} is the elasticity matrix;
- D is the scalar damage variable; and
- $\boldsymbol{\varepsilon}^e$ is the elastic strain vector.

The continuous damage variable is 0 for a healthy material, and 1 for a damaged one. The elastic strain controls the state of tension in the material; therefore, the concrete damage is activated after a certain deformation threshold ε_{d0} . Equation 2.2 shows equation defining the damage variable.

$$D = \alpha_t^\beta D_t + \alpha_c^\beta D_c \quad (2.2)$$

Coefficient β improves the behaviour in shearing, and it is usually fixed at 1.06. Coefficients α_t and α_c allows for the damage in tension D_t and compression D_c , respectively.

The laws of evolution of damage D_t and D_c are shown at *Equation 2.3* and *Equation 2.4*

$$D_t = 1 - \frac{(1 - A_t)_{d0}}{\varepsilon_{eq}} - A_t e^{-B_t(\varepsilon_{eq} - \varepsilon_{d0})} \quad (2.3)$$

$$D_c = 1 - \frac{(1 - A_c)_{d0}}{\varepsilon_{eq}} - A_c e^{-B_c(\varepsilon_{eq} - \varepsilon_{d0})} \quad (2.4)$$

A_t , A_c , B_t , and B_c are parameters of the material and have to be identified by fitting of the one-dimensional Mazars model into the experimental data.

It is worth noticing that ε_{eq} is defined from the positive eigenvalues of the deformation tensor, and it represents the equivalent deformation derived from the principal directions.

2.5 Experimental and Analytical Reexamination of Classic Concrete Beam Tests

In 1963, Boris Bresler and A. C. Scordelis published the research paper "Shear Strength of Reinforced Concrete Beams" with a primary goal to understating shear-critical behaviour in reinforced concrete [14]. This work has been considered a classic test series used as benchmark data for calibrating or verifying finite element models for reinforced concrete [10, 74], and was reproduced by F. J. Vecchio and W. Shim in 2004 with a nominally identical set of beams [74].

Figure 2.14 shows the test setup for the Vecchio-Shim Beams. Figure 2.15 shows the cross sections of the one tested beams, with details shown in Table 2.3.

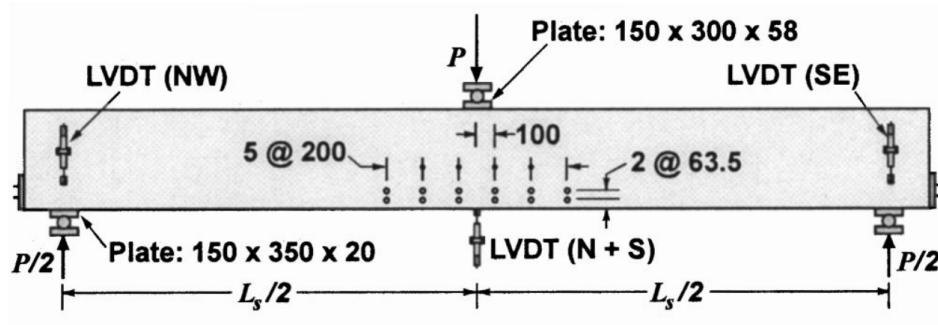


Figure 2.14: Test setup of the twelve Vecchio-Shim beams; source: [74])

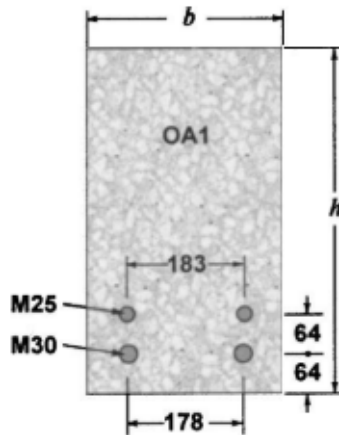


Figure 2.15: Cross-sections of the twelve Vecchio-Shim beams; source: [74])

Table 2.4 shows the reinforcement properties of the OA1 beams and Table 2.5 shows its concrete properties. Vecchio-Shim OA1 test results are shown in Table 2.6.

Table 2.3: Cross-Section details of Vecchio-Shim beams

Beam	b	h	d	L	Span			
(id)	(mm)	(mm)	(mm)	(mm)	(mm)	Bottom steel	Top steel	Stirrups
OA1	305	552	457	4,100	3,660	2 M30, 2M25	-	-

adapted from [74]

Table 2.4: Reinforcement properties of Vecchio-Shim beams

	Diameter	Area	f_y	f_u	E_s
Bar size	(mm)	(mm ²)	(MPa)	(MPa)	(MPa)
M25	25.2	500	445	680	220,000
M30	29.9	700	436	700	200,000

where:

f_y is the steel yield stress

f_u is the steel ultimate strength

E_s is the modulus of elasticity of steel

adapted from [74]

Table 2.5: Concrete properties of Vecchio-Shim beams]

Beam	f'_c	ϵ_0	E_c	f_{sp}
(id)	(MPa)	(mm/mm)	(MPa)	(MPa)
OA1	22.6	0.0016	36,500	2.37

where:

f'_c is the concrete compressive strength at 28 days cylinder test

ϵ_0 is the concrete strain at peak cylinder stress

E_c is the modulus of elasticity of concrete

f_{sp} is the concrete split cylinder strength

adapted from [74]

Table 2.6: Test results for Vecchio-Shim beams

Beam	P test	P calc	P test / P calc	Δ test	Δu calc	Δu test / Δu calc
(id)	(kN)	(kN)		(mm)	(mm)	
OA1	331	311	1.06	9.1	9.5	0.96

Adapted from [74]

CHAPTER 3

METHODOLOGY

3.1 Software Setup

A x86_64 Arch Linux operational system, kernel release 5.1.11-arch1-1-ARCH, was used for the simulation. The processor type was an Intel(R) Core(TM) i7-8650U CPU @ 1.90GHz, with 15781 MB of RAM.

The binary `salome_meca` package 2018-1¹ was installed from the Arch Linux User Repository (AUR), keeping all the configurations intact except for the overridden preferences listed on Code 3.1.

The most important was the alteration of the working directory at line 12. During calculation, `code_aster` writes several files by default on a temporary RAM file system, which may be rapidly filled, halting the process.

Code 3.1: `code_aster` overridden preferences at `./astkrc/prefs`

```
1  nom_user : _VIDE
2  email : _VIDE
3  def_vers : stable
4  xterm : /usr/bin/xterm
5  editeur : /usr/bin/vi
6  nb_reman : 6
7  langue : ENG
8  dbglevel : 3
9  freq_actu : 3
10 nb_ligne : 20
11 nom_domaine : _VIDE
12 rep_trav: /home/franks/tmp/aster
```

It is not recommended changing the working directory for systems without SSD, as the calculation performance might be severely impacted.

¹<https://aur.archlinux.org/packages/salome-meca-bin/>

It is important to note that `salome_meca` does not enforce or allow for any configuration of units. The choice of units depends on the user. For this project, the beam was modeled in SI units.

3.2 Geometrical Modeling

The first step was the geometrical modeling of the structure. It was in this step that majors points of interest were defined, e.g. supports, loads and measurable points.

Figure 3.1 shows the geometrical model of the beam in 1D.

Point 'O' was defined at the origin of the Euclidean space. At this point, load was applied and deflections were measured. Although it might look contradictory, only a line representing the beam needed to be drawn; therefore, a 1D model along the beam length.

Along the 'X' axis, points 'LS' and 'RS' were the left and right supports, respectively. They were 2.05 meters distant from the origin: 'LS' at $(-1.83, 0.00, 0.00)$ and 'RS' at $(1.83, 0.00, 0.00)$.

The beam extremities were also defined along the 'X' axis, 'LE' at $(-2.05, 0.00, 0.00)$ and 'RE' at $(2.05, 0.00, 0.00)$.

Finally, lines were drawn connecting the five points. Line 'R' connected the right extreme point 'RE' to the right support point 'RS'. Reciprocally, line 'L' connected the left extreme point 'LE' to the left support point 'LS'.

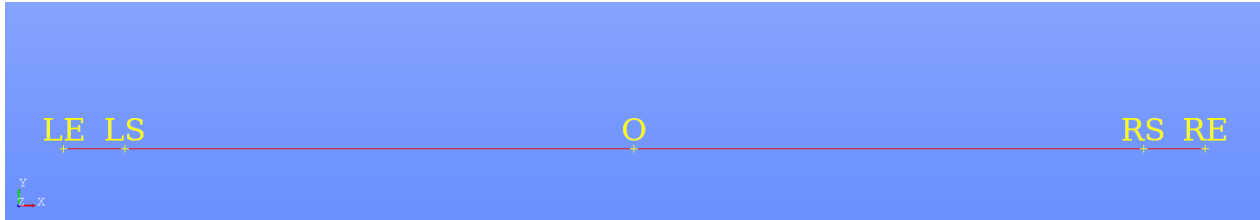


Figure 3.1: Geometrical modeling of the beam

In addition, it was necessary to create a cross section that would later be replicated throughout the beam. The rectangular face is drawn with center at origin, height along the 'Y' axis and width along the 'X' axis. A unitary thickness is draw, which would later be ignored by the solver. The final dimensions were $(0.305, 0.552, 1.000)$.

Using the dump Python functionality of `salome_meca`, the geometry can be exported to a Python file, which is easily readable. The full code is available at Appendix A, with an excerpt at Code 3.2.

Code 3.2: Definition of points in Python

```
1 O = geompy.MakeVertex(0, 0, 0)
2 OX = geompy.MakeVectorDXDYDZ(1, 0, 0)
```

```

3 OY = geompy.MakeVectorDXDYDZ(0, 1, 0)
4 OZ = geompy.MakeVectorDXDYDZ(0, 0, 1)
5 LS = geompy.MakeVertex(-1.83, 0, 0)
6 RS = geompy.MakeVertex(1.83, 0, 0)
7 LE = geompy.MakeVertex(-2.05, 0, 0)
8 RE = geompy.MakeVertex(2.05, 0, 0)

```

The origin was defined at line 1, followed by three vectors defining the Euclidean space. Lines 5 and 6 defined the supports, and lines 7 and 8 defined the extremities of the beam.

From the geometrical entities, two different compounds were defined: 'Section' and 'Beam'. They would hold together the entities and groups for later reference.

3.3 Meshing

Two different meshes were created.

The first, named 'SECTION', as shown in isometric view at Figure 3.2, had a simple 2D quadrangular mapping with standard parameters. Submeshes 'V' (vertical) and 'H' (horizontal) were both 1D wire discretisation with predefined number of segments of 32 and 16, respectively.

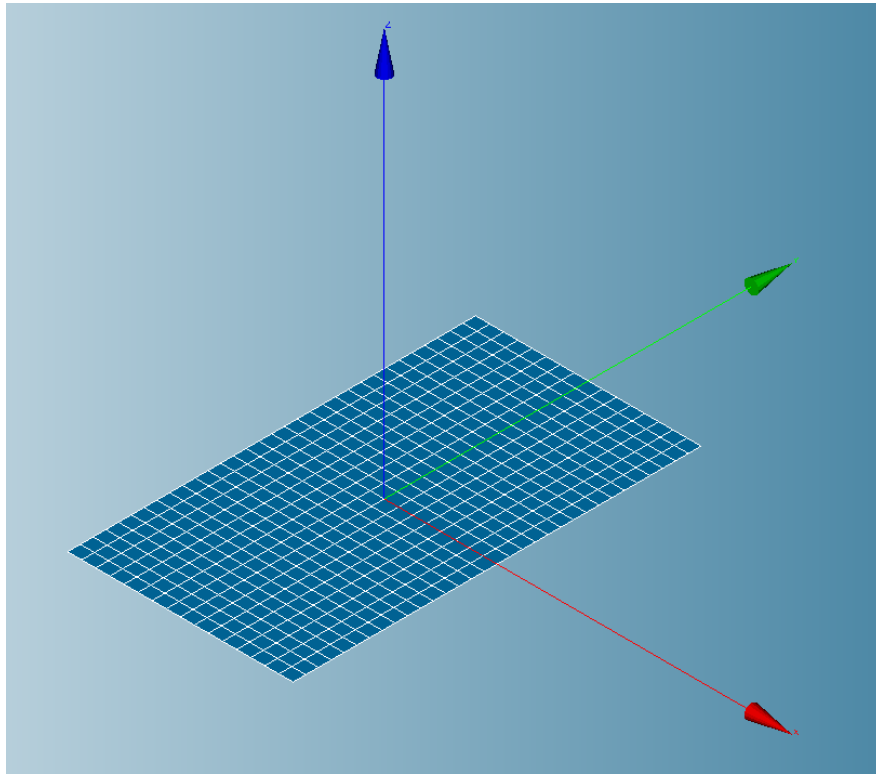


Figure 3.2: Meshing of the section

The second mesh, name 'BEAM', had two submeshes: 'Extreme' for the lines connecting the extremities to the support points; and 'Internal' for the lines connecting the support points to the origin. The external submesh had a total of 6 elements, whereas the internal submesh had a total of 2 elements. Therefore, 'BEAM' mesh had 8 elements, as shown at Figure 3.4.

On each of the 9 nodes of the 'BEAM' mesh, 608 elements (Figure 3.3) were created.

Name:	SECTION			
Object:	Mesh			
Nodes:	561			
Elements:	<i>Total</i>	<i>Linear</i>	<i>Quadratic</i>	<i>Bi-Quadratic</i>
	608	608	0	0
0D:	0			
Balls:	0			
1D (edges):	96	96	0	
2D (faces):	512	512	0	0
Triangles:	0	0	0	0
Quadrangles:	512	512	0	0
Polygons:	0	0	0	
3D (volumes):	0	0	0	0
Tetrahedrons:	0	0	0	
Hexahedrons:	0	0	0	0
Pyramids:	0	0	0	
Prisms:	0	0	0	0
Hexagonal Prisms:	0			
Polyhedrons:	0			

Figure 3.3: 'SECTION' mesh information

Name:	BEAM			
Object:	Mesh			
Nodes:	9			
Elements:	<i>Total</i>	<i>Linear</i>	<i>Quadratic</i>	<i>Bi-Quadratic</i>
	8	8	0	0
0D:	0			
Balls:	0			
1D (edges):	8	8	0	
2D (faces):	0	0	0	0
Triangles:	0	0	0	0
Quadrangles:	0	0	0	0
Polygons:	0	0	0	
3D (volumes):	0	0	0	0
Tetrahedrons:	0	0	0	
Hexahedrons:	0	0	0	0
Pyramids:	0	0	0	
Prisms:	0	0	0	0
Hexagonal Prisms:	0			
Polyhedrons:	0			

Figure 3.4: 'BEAM' mesh informations

Four groups were used in the simulation: three nodes ('LS', 'O', and 'RS'); and one group of edges for the whole beam.

3.4 AsterStudy

The AsterStudy module allows for the generation of command files with a graphic user interface (GUI); however, this project adopts the Python command file directly, which is available at Appendix B. With practice in salome_meca, it is easier and faster to write the file directly than using the GUI.

Code 3.3 shows the beginning of the command file. It started with DEBUT(LANG='EN'), defining the language. Several variables were then assigned. Note that the concrete tension resistance was defined using the formula $0.33\sqrt{f'_c}$, used at the definition of the deformation threshold. Mazars' parameters were then defined and, at last, the reinforcement steel parameters.

Code 3.3: Assignment of numerical values to variables

```
1  DEBUT(LANG='EN')
2
3  FCJ_c = 22.60E+06
4  EIJ_c = 36500.0E+06
5  NU_c = 0.22
6  FTJ_c = 0.33 * sqrt(FCJ_c / 1e+06) * 1e+06
7  EPSI_C_c = 1.6E-03
8  EPSD0_c = FTJ_c / EIJ_c
9  BT_c = 21000.0
10 AT_c = 1.0
11 BC_c = 2000.0
12 AC_c = 0.33
13 K_c = 0.7
14 Signl_c = 13560000
15 EPSI1_c = 0.0035
16 RHO_c = 2400.0,
17 NU_s = 0.33,
18 RHO_s = 7800.0,
19 Yng_25 = 2.2E+11
20 Sy_25 = 445.0E+06
21 DSiEp_25 = Yng_25 / 1.0E+04
22 Yng_30 = 2.0E+11
23 Sy_30 = 436.0E+06
24 DSiEp_30 = Yng_30 / 1.0E+04
```

Next, the meshes were read with the the command 'LIRE_MALLAGE' at Code 3.4. 'MAPOU' refers to the 'BEAM' mesh, and 'MASEC' refers to the 'SECTION' mesh. Every input and output in salome_meca has a number associated to the file. Line 2 associated the 'UNITE' 5 to the 'BEAM' mesh, and line 5 associated the 'UNITE' 20 to the 'SECTION' mesh.

Code 3.4: Reading of the mesh

```
1  MAPOU = LIRE_MALLAGE(FORMAT='MED',
2                          UNITE=5)
3
```

```

4 MASEC = LIRE_MALLAGE(FORMAT='MED',
5
        UNITE=20)

```

Geometries of the fibers were defined with the command `DEFI_GEOM_FIBRE` at Code 3.5. All fibers were defined according to the section centre in the (x, y) plane. The pattern of reinforcement steel assignment was (coordinate x, coordinate y, diameter). Concrete fibers were defined between lines 10 and 13.

Code 3.5: Definition of fibers

```

1 GF = DEFI_GEOM_FIBRE(FIBRE=( _F(CARA='DIAMETRE',
2
        COOR_AXE_POUTRE=(0.0, 0.0),
3
        GROUP_FIBRE='SACI_25',
4
        VALE=(0.0915, -0.148, 0.0252, -0.0915, -0.148,
        ↪ 0.0252)),
5
        _F(CARA='DIAMETRE',
6
        COOR_AXE_POUTRE=(0.0, 0.0),
7
        GROUP_FIBRE='SACI_30',
8
        VALE=(0.089, -0.212, 0.0299, -0.089, -0.212,
        ↪ 0.0299))),
9
        INFO=2,
10
        SECTION=_F(COOR_AXE_POUTRE=(0.0, 0.0),
11
        GROUP_FIBRE='SBET',
12
        MAILLAGE_SECT=MASEC,
13
        TOUT_SECT='OUI'))

```

Code 3.6 shows the definition of the finite element model: mechanical analysis using `POU_D_EM` affecting all meshes.

Code 3.6: Definition of the model

```

1 MOPOU = AFFE_MODELE(AFFE=_F(MODELISATION='POU_D_EM',
2
        PHENOMENE='MECANIQUE',
3
        TOUT='OUI'),
4
        MAILLAGE=MAPOU)

```

Code 3.7 integrated the concrete and reinforcement steel fibers to the model 'MOPOU' defined at Code 3.6. Geometry of fibers 'GF' was affected by the command 'AFFE_CARA_ELEM', assigning the model 'MOPOU' and specifying the multiple fibers previously defined. It is important to note an overlap between the concrete and reinforcement steel fibers; therefore, there is a higher tolerance for the difference in the moment of inertia.

Code 3.7: Definition of the model

```
1  POUCA = AFFE_CARA_ELEM(GEOM_FIBRE=GF,
2
3      MODELE=MOPOU,
4
5      MULTIFIBRE=_F(GROUP_FIBRE=('SBET', 'SACI_25', 'SACI_30'),
6
7          GROUP_MA='POUTRE',
8
9          PREC_AIRE=0.02,
10
11          PREC_INERTIE=0.25),
12
13      ORIENTATION=_F(CARA='ANGL_VRIL',
14
15          GROUP_MA='POUTRE',
16
17          VALE=-90.0),
18
19      POUTRE=_F(CARA=('HY', 'HZ'),
20
21          GROUP_MA='POUTRE',
22
23          SECTION='RECTANGLE',
24
25          VALE=(0.305, 0.552)))
```

Code 3.8 defined the materials to be assigned to each fiber: 'BETON' with the Mazars model assigned to the sections; 'AC_25' and 'AC_30' to the reinforcement steel fibers.

Code 3.8: Definition of materials

```
1  BETON = DEFI_MATER_GC(INFO=2,
2
3      MAZARS=_F(AC=AC_c,
4
5          AT=AT_c,
6
7          BC=BC_c,
8
9          BT=BT_c,
10
11          CODIFICATION='ESSAI',
12
13          EIJ=EIJ_c,
14
15          EPSD0=EPSD0_c,
16
17          EPSI_C=EPSI_C_c,
18
19          EPSI_LIM=EPSI1_c,
20
21          FCJ=FCJ_c,
22
23          FTJ=FTJ_c,
24
25          K=K_c,
26
27          NU=NU_c,
28
29          SIGM_LIM=Signl_c),
30
31      RHO=RHO_c)
```

```

17
18 AC_25 = DEFI_MATER_GC(ACIER=_F(D_SIGM_EPSI=DSiEp_25,
19                               E=Yng_25,
20                               SY=Sy_25),
21                               RHO=RHO_s)
22
23 AC_30 = DEFI_MATER_GC(ACIER=_F(D_SIGM_EPSI=DSiEp_30,
24                               E=Yng_30,
25                               SY=Sy_30),
26                               RHO=RHO_s)
27
28 MATOR = DEFI_MATERIAU(ELAS=_F(E=2e+11,
29                               NU=0.0,
30                               RHO=RHO_s)

```

Code 3.9 defined functions and lists for use during calculation: 'FOFO', 'LINSTD', and 'LINST'.

The first, 'FOFO', was a straight line between the origin and (13,13). The abscissa would be taken as calculation time, and the ordinate would be taken as a multiplication factor for the vertical displacement.

The second, 'LINSTD', defined the calculation steps: 2 intervals until the instant 0.1; and a 0.25 separation time between calculations until the instant 13, i.e. 4 intervals per second. Instant 0.1 was introduced to evaluate if the beam dead load is correctly applied.

The third, 'LINST', allowed for dynamic subdivision of steps in case of lengthy convergence.

Code 3.9: Definition of functions

```

1 FOFO = DEFI_FONCTION(NOM_PARA='INST',
2                       PROL_DROITE='EXCLU',
3                       PROL_GAUCHE='EXCLU',
4                       VALE=(0.0, 0.0, 13.0, 13.0))
5
6 LINSTD = DEFI_LIST_REEL(DEBUT=0.0,
7                         INTERVALLE=(_F(JUSQU_A=0.1,
8                                         NOMBRE=2),
9                                         _F(JUSQU_A=13.0,
10                                             PAS=0.25)))
11
12 LINST = DEFI_LIST_INST(DEFI_LIST=_F(LIST_INST=LINSTD),

```



```

13      ECHEC=_F(ACTION='DECOUPE',
14                EVENEMENT='ERREUR',
15                SUBD_METHODE='MANUEL',
16                SUBD_NIVEAU=5,
17                SUBD_PAS=4,
18                SUBD_PAS_MINI=1e-10),
19      METHODE='MANUEL')

```

Code 3.10 defined the boundary conditions of the simulation. The support at the left 'LS' was a pin, and the support at the right 'RS' was a roller. A displacement of 1 millimeter was imposed at the origin, along the negative direction of the 'Y' axis. Paired with the multiplication factor previously defined, a displacement of 1 millimeter per second can be achieved. Nonlinear calculations have been historically more stable using displacements than loads in salome_meca.

Code 3.10: Boundary conditions

```

1  BLOCAGE = AFFE_CHAR_MECA(DDL_IMPO=_F(DRX=0.0,
2                                     DRY=0.0,
3                                     DX=0.0,
4                                     DY=0.0,
5                                     DZ=0.0,
6                                     GROUP_NO='LS'),
7    _F(DY=0.0,
8        GROUP_NO='RS')),
9    MODELE=MOPOU)
10
11  DEPIMP = AFFE_CHAR_MECA(DDL_IMPO=_F(DY=-0.001,
12                                     GROUP_NO='O'),

```

Code 3.11 defined the material behaviour: von Mises for the reinforcement steel, and Mazars for the concrete. The behaviour was then assigned to the beam at line 13.

Code 3.11: Material behaviour

```

1  PMFMAZAR = DEFI_COMPOR(GEOM_FIBRE=GF,
2                          MATER_SECT=MATOR,
3                          MULTIFIBRE=_F(GROUP_FIBRE='SACI_25',
4                                          MATER=AC_25,

```

```

5          RELATION='VMIS_CINE_GC'),
6          _F(GROUP_FIBRE='SACI_30',
7            MATER=AC_30,
8            RELATION='VMIS_CINE_GC'),
9          _F(GROUP_FIBRE=('SBET', ),
10           MATER=BETON,
11           RELATION='MAZARS_GC'))))
12
13 MATMAZAR = AFFE_MATERIAU(AFFE=_F(GROUP_MA='POUTRE',
14                                MATER=(AC_25, AC_30, BETON, MATOR)),
15                          AFFE_COMPOR=_F(COMPOR=PMFMAZAR,
16                                         GROUP_MA='POUTRE'),
17                          MAILLAGE=MAPOU)

```

Code 3.12 defined the analysis type, convergence limits for avoiding infinite loops, the loads and supports, instant increments, and solver type.

The analysis type was static nonlinear, applied to the model previously defined with multi-fiber relationship. A convergence criterion of maximum iterations was imposed to avoid locked loops. In addition, a global precision of 10 micrometers was defined. Although it might seem excessively precise, divergences might occur in case of larger precision.

Loads and support were defined between lines 6 and 8. Note the multiplication factor acting on the load at line 8.

The increment list of instants was defined at line 9.

Finally, solver configurations were indicated at lines 11 and 12. In this specific case, a tangent matrix would be used on every iteration of the Newton-Raphson method.

Code 3.12: Material behaviour

```

1 U1MAZAR = STAT_NON_LINE(CARA_ELEM=POUCA,
2                          CHAM_MATER=MATMAZAR,
3                          COMPOTEMENT=_F(RELATION='MULTIFIBRE'),
4                          CONVERGENCE=_F(ITER_GLOB_MAXI=10,
5                                         RESI_GLOB_RELA=1e-05),
6                          EXCIT=( _F(CHARGE=BLOCAGE),
7                                _F(CHARGE=DEPIMP,
8                                  FONC_MULT=FOFO)),
9                          INCREMENT=_F(LIST_INST=LINST),

```

```

10         MODELE=MOPOU,
11         NEWTON=_F(MATRICE='TANGENTE',
12                   REAC_ITER=1))

```

Code 3.13 shows the post-processing tasks, applied after the displacements have been solved.

The first task was the calculation of reactions on all nodes, followed by elastic and inelastic deformations. At last, the nodal reactions at the supports 'LS' and 'RS' were calculated.

Code 3.13: Post-processing

```

1  U1MAZAR = CALC_CHAMP(reuse=U1MAZAR,
2                        FORCE=('REAC_NODA', ),
3                        RESULTAT=U1MAZAR)
4
5  U1MAZAR = CALC_CHAMP(reuse=U1MAZAR,
6                        DEFORMATION=('EPSI_ELGA', 'EPSP_ELGA'),
7                        RESULTAT=U1MAZAR)
8
9  SUM_REAC = POST_RELEVE_T(ACTION=_F(GROUP_NO=('LS', 'RS'),
10                                     INTITULE='sum reactions',
11                                     MOYE_NOEUD='OUI',
12                                     NOM_CHAM='REAC_NODA',
13                                     OPERATION=('EXTRACTION', ),
14                                     REPERE='GLOBAL',
15                                     RESULTANTE=('DX', 'DY', 'DZ'),
16                                     RESULTAT=U1MAZAR,
17                                     TOUT_ORDRE='OUI'))

```

Code 3.14 was the last part of the command file, where the results were exported.

Code 3.14: Exporting results

```

1  IMPR_RESU(FORMAT='RESULTAT',
2            MODELE=MOPOU,
3            RESU=_F(GROUP_NO=('O', ),
4                    MAILLAGE=MAPOU,
5                    NOM_CAS=('DY', ),
6                    NOM_CHAM=('DEPL', ),

```

```

7          RESULTAT=U1MAZAR),
8      UNITE=3)
9
10 IMPR_RESU(FORMAT='RESULTAT',
11          MODELE=MOPOU,
12          RESU=_F(GROUP_NO=('LS', 'RS'),
13                NOM_CHAM=('REAC_NODA', ),
14                NOM_CMP=('DY', ),
15                RESULTAT=U1MAZAR,
16                VALE_MAX='OUI',
17                VALE_MIN='OUI'),
18          UNITE=8)
19
20 IMPR_TABLE(TABLE=SUM_REAC,
21           UNITE=2)
22
23 FIN()

```

Switching to the 'History View' tab at AsterStudy, the case can be run with the parameters shown at Figure 3.5.


Basic	Advanced
Case name	RunCase_1
Memory	2048
Time	0 : 15 : 0
Run servers	localhost 
Version of code_aster	stable
Number of MPI CPU	1
User description	

Figure 3.5: Case parameters

CHAPTER 4

RESULTS AND ANALYSIS

Total solution and calculations were performed in 8.49 seconds in real time (wall clock), with a peak memory usage of 693.13 Mb, considerably lower than the allotted 2 Gb. Table 4.1 shows the performance indicators for various stages of the solution process, e.g. elementary calculation, assembly, digital, and factorization.

USER time refers to the computational time spent on user commands, i.e. those specified by the command file, e.g. assembling and solving matrix and calculation of displacements. SYSTEM refers to the computational time the system spent on the calculations, such as saving and accessing the memory, dumping data into hard drives, and printing information on the screen. The third column sums the previous ones, and the last column of the table gives the time spent as measured in real time (wall clock). Since the system has multiple CPUs, the elapsed time is shorter than the summation of the computational times.

Ideally, for each stage of the calculation, the best performance is obtained by a low SYSTEM time and a USER+SYS time higher than the ELAPSED one. The higher the USER+SYS with regards to the ELAPSED time, the greater the parallelism.

It is worth noting that DEBUT, which, in principle is only a start command, takes a considerable amount of time. The reason might be the setup of several files and the allocation of memory for calculation. Indeed, the ELAPSED time is almost the same as the USER+SYS time, indicating no CPU parallelism. In addition, both LIRE_MALLAGE, for example, which deal with mesh reading, were delayed by disk accessing.

STAT_NON_LINEAR, which solves the nonlinear problem, is by far the most intensive calculation stage, taking 68% of the total USER time and 25% of the total SYSTEM time; however, the high parallelism allows for a ELAPSED time 74% inferior to the USER+SYS time: 3.16 seconds.

The second most intensive calculation stage is the elastic and inelastic deformations (CALC_CHAMP), taking 3.18 seconds of the USER time and 0.32 seconds of the SYSTEM time, respectively 18% and 25% of USER and SYSTEM total times.

All other calculation stages were executed in less than 1 second each.

As code_aster is built in Fortran with supervision in Python, it is worth comparing the two performances. Although they are solving different problems, might be a good reference to help to choose FEM approaches: the ratio between the ELAPSED and USER+SYS times is 41% for Fortran and 86% for Python, which indicates that Fortran has a better performance than Python in demanding calculations.

Table 4.1: Time required to run the commands (in seconds)

COMMAND	USER	SYSTEM	USER+SYS	ELAPSED
init (jdc)	1.38	0.06	1.44	1.45
. compile	0.00	0.00	0.00	0.00
. exec_compile	0.18	0.00	0.18	0.18
. report	0.02	0.00	0.02	0.02
. build	0.00	0.00	0.00	0.00
DEBUT	0.04	0.04	0.08	0.07
LIRE_MALLAGE	0.01	0.00	0.01	0.01
LIRE_MALLAGE	0.01	0.01	0.02	0.02
DEFL_GEOM_FIBRE	0.00	0.00	0.00	0.00
AFFE_MODELE	0.01	0.02	0.03	0.01
AFFE_CARA_ELEM	0.05	0.07	0.12	0.03
DEFL_MATER_GC	0.06	0.03	0.09	0.02
DEFL_MATER_GC	0.02	0.05	0.07	0.02
DEFL_MATER_GC	0.03	0.01	0.04	0.01
DEFL_MATERIAU	0.00	0.00	0.00	0.00
DEFL_FONCTION	0.00	0.00	0.00	0.00
DEFL_LIST_REEL	0.00	0.00	0.00	0.00
DEFL_LIST_INST	0.01	0.00	0.01	0.00
AFFE_CHAR_MECA	0.00	0.00	0.00	0.01
AFFE_CHAR_MECA	0.01	0.00	0.01	0.01
DEFL_COMPOR	0.00	0.00	0.00	0.00
AFFE_MATERIAU	0.02	0.00	0.02	0.01
STAT_NON_LINE	11.70	0.33	12.03	3.16
CALC_CHAMP	0.37	0.23	0.60	0.22
CALC_CHAMP	3.18	0.32	3.50	3.11
POST_RELEVE_T	0.08	0.00	0.08	0.09
IMPR_RESU	0.03	0.00	0.03	0.03
IMPR_RESU	0.02	0.00	0.02	0.02
IMPR_TABLE	0.04	0.00	0.04	0.04
FIN	0.02	0.05	0.07	0.07
. part Superviseur	1.72	0.24	1.96	1.69
. part Fortran	15.49	1.06	16.55	6.80
TOTAL_JOB	17.21	1.30	18.51	8.49

Table 4.2 shows an excerpt of the summation of nodal reactions at the supports in the three directions at calculated instants. Recall that our model applies one millimeter per instant unit; therefore, at the instant 8.35, for example, the mid-span deflection was 8.35 millimeter, and the summation of the vertical reaction at the supports was 316.3 kN. As expected, reactions in the directions 'X' and 'Z' are negligible.

This displacement control allows for the direct inference of the pair force versus displacement. By inspection of Table 4.2, one can observe the maximum vertical force of 334.7 kN at the instant 10.10, i.e. an ultimate load of 334.7 kN at a mid-span deflection of 10.10 millimeters.

These results reasonably agreed with the Vecchio-Shin's ultimate loads and deflections. In case of the ultimate load, salome_meca result was 1.1% greater than the experimental value, and 7.6% greater than the Vecchio-Shin's prediction. For the deflections, salome_meca result was 11.0% greater than Vecchio-Shin's calculated mid-span deflection, and 6.3% greater than the experimental one.

The salome_meca model so far is adequate to overcome the challenge of calibrating nonlinear finite element formulations.

Table 4.2: Summation of nodal reactions at the supports (in Newtons)

INST	DX	DY	DZ
8.35	-2.864E-10	3.163E+05	-7.585E-12
8.60	-1.177E-10	3.204E+05	5.760E-13
8.85	3.027E-10	3.242E+05	1.000E-11
9.10	-1.040E-09	3.275E+05	9.561E-12
9.35	-9.713E-10	3.300E+05	-6.105E-12
9.60	-9.907E-10	3.323E+05	-7.992E-13
9.85	-1.213E-09	3.340E+05	-1.176E-12
10.10	7.942E-10	3.347E+05	-5.007E-12
10.35	6.859E-10	3.345E+05	-4.050E-12
10.60	1.971E-09	3.341E+05	2.383E-12
10.85	-8.335E-10	3.326E+05	2.937E-12
11.10	6.219E-10	3.292E+05	-1.312E-12
11.35	-1.141E-09	3.258E+05	5.587E-12
11.60	2.987E-10	3.218E+05	7.758E-12
11.85	-6.698E-10	3.158E+05	1.101E-12
12.10	-5.204E-10	3.099E+05	4.773E-12
12.35	1.717E-10	3.048E+05	-6.192E-12
12.60	-4.943E-10	2.994E+05	1.708E-12
12.85	-2.093E-10	2.919E+05	-1.005E-11
13.00	-1.722E-10	2.876E+05	1.633E-12

A final comparison with the experimental results can be made by plotting load versus mid-span deflections. Figure 4.1 shows all curves drawn up to the ultimate load. It can be visually noted an agreement between the sets of data. However, for a better comparison, the experimental data were fitted (R squared of 0.999) with the Equation 4.1, plotted in red color in Figure 4.1. Due to over-fitting, the equation is not generalizable, and it was used only for calculating correlations with the numerical data.

$$Y = 0.000282428X^9 - 0.0114606X^8 + 0.19664X^7 - 1.86746X^6 + 10.8128X^5 - 39.7294X^4 + 93.5588X^3 - 139.343X^2 + 159.694X - 2.35567 \quad (4.1)$$

Statistically comparing the salome_meca results (plotted in green color in Figure 4.1) with the experimental data leads to an R squared of 0.999, indicating a good fit of the numerical model with the real data. Vecchio-Shin's numerical calculations, shown in blue color in Figure 4.1, has an R squared of 0.980. Therefore, both methods seem to reasonably agree with the experimental data.

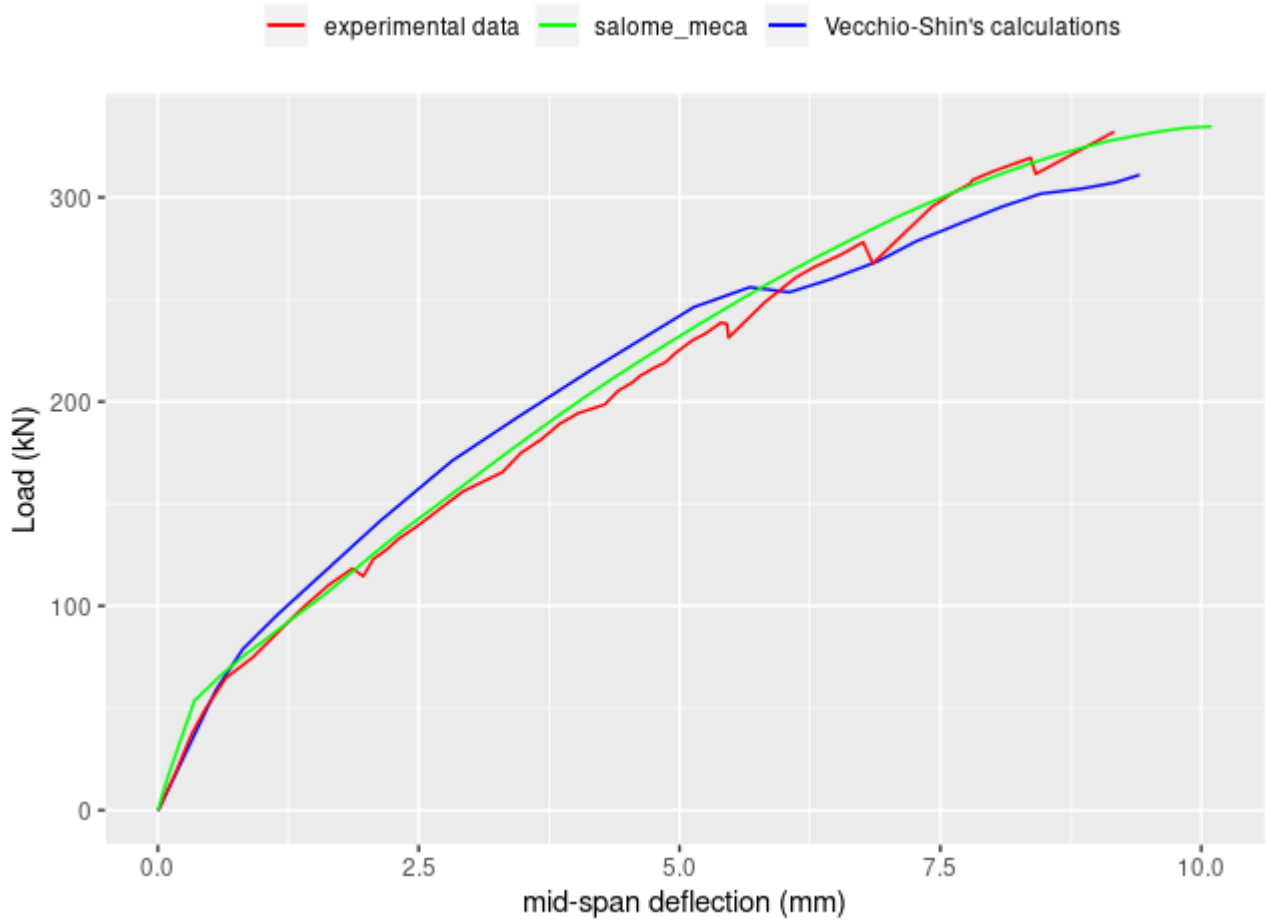


Figure 4.1: Load versus mid-span deflections of experimental data from Vecchio-Schin and numerical data from salome_meca

CHAPTER 5

SUMMARY, CONCLUSIONS, AND RECOMMENDATIONS

This study was performed to investigate and report the usefulness of the open source software programs `code_aster` and `salome_meca` in implementing nonlinear quasi-static finite element modeling of reinforced concrete beams. An extensive literature review about open source software was written, as well as a review of the software programs `code_aster` and `salome_meca`.

Experimental data from the literature were used for modeling and validating the nonlinear finite element model of a reinforced concrete beam. The geometry was drawn in the `salome_meca` Geometry module, followed by meshing in the Mesh module. AsterStudy was started for the creation of a stage; however, the modeling was written using a text editor rather than the graphic user interface. A one-dimensional beam model, with a multi-fiber section of concrete and reinforcement steel was used. Mazars damage model was assigned for concrete fiber, and the reinforcement steel fiber followed a von Mises behaviour.

The open source software programs `salome_meca` and `code_aster` had good performance solving the nonlinear model, calculating all stages in less than nine seconds in real time. Ultimate load and mid-span deflection were compared to the experimental data, obtaining satisfactory results; thus, FEM was validated.

Even though the development and use of `salome_meca` and `code_aster` by the French electric utility company speaks for itself as it had been applied in nuclear facilities, results indicate that both software programs may be adopted. Although the learning curve is steep, there is extensive documentation available online, and support is also provided by the community on official forums. One major advantage of adopting `salome_meca` and `code_aster` is that it requires a deep understanding of the model, constraints, limitations, and configuration options for each stage.

A large database of validating models test-cases¹ paired with documentations are available as templates for new simulations. The use of this database is encouraged for future research, including the simulation of different models of reinforced concrete beams and validation with the Vecchio-Shin (or Bresler-Scordelis) data.

The simulation of small strains and small displacements models can also be studied, allowing for easier comparisons with hand calculations. However, the challenge of nonlinear finite element analysis of reinforced concrete should not be a deterrent. Buckling, thermal, creep, and corrosion models are also available for use, requiring only dedication for understanding the concepts, given that the programs are free and open source.

¹<https://www.code-aster.org/V2/doc/default/en/index.php?man=cas-test>

REFERENCES

- [1] ANDERSON, R. J. *Security Engineering*, 2nd ed. Wiley. ISBN: 978-0470068526, Indianapolis, IN, USA, 2008. Available <https://www.cl.cam.ac.uk/~rja14/book.html>. Accessed on 02 Jan 2019.
- [2] ARCH LINUX. Arch User Repository (AUR). <https://aur.archlinux.org/>, Dec. 2018. Accessed on 29 Dec 2018.
- [3] ARCHWIKI. Arch User Repository. https://wiki.archlinux.org/index.php/Arch_User_Repository, Dec. 2018. Accessed on 29 Dec 2018.
- [4] ARCHWIKI. makepkg. <https://wiki.archlinux.org/index.php/Makepkg>, Oct. 2018. Accessed on 29 Dec 2018.
- [5] ARCHWIKI. pacman. <https://wiki.archlinux.org/index.php/Pacman>, Dec. 2018. Accessed on 29 Dec 2018.
- [6] ARCHWIKI. PKGBUILD. <https://wiki.archlinux.org/index.php/PKGBUILD>, Dec. 2018. Accessed on 29 Dec 2018.
- [7] AUBRY, J.-P. *Beginning with Code_Aster: A Practical Introduction to Finite Element Method using Code_Aster, Gmsh and Salome*, 2nd ed. Framabook. ISBN: 979-1092674033, Paris, FR, Jan. 2019. Available at https://framabook.org/beginning-with-code_aster/. Accessed on 15 Mar 2019.
- [8] BATHE, K.-J. *Finite Element Procedures*, 2nd ed. K.J. Bathe. ISBN: 978-0979004957, Watertown, MA, USA, 2014.
- [9] BAUER, F. L. The Plankalkül of Konrad Zuse — Revisited. In *The First Computers—History and Architectures (History of Computing)*, R. Rojas and U. Hashagen, Eds., 1st ed. The MIT Press. ISBN: 978-0262181976, Cambridge, MA, USA, 2000, ch. Part III, chapter 3, pp. 204–214.
- [10] BAZANT, Z. P. *State-of-the-art Report on Finite Element Analysis of Reinforced Concrete*, 1st ed. American Society of Civil Engineers. ISBN: 0-87262-307-6, New York, N.Y., 1982.
- [11] BEER, F. P., RUSSELL, E., JOHNSTON, J., MAZUREK, D. F., CORNWELL, P. J., AND SELF, B. *Vector Mechanics for Engineers: Statics and Dynamics*, 11th ed. McGraw-Hill Education - Europe. ISBN: 978-0073398242, New York, NY, USA, 2015.
- [12] BEN-ARI, M. *Understanding Programming Languages*, 1st ed. Wiley. ISBN: 978-0471958468, New York, NY, USA, 1996.
- [13] BOWEN, J. Standard Microprocessor Programming Cards. *Microprocessors and Microsystems* 9, 6 (jul 1985), 274–289. doi: [10.1016/0141-9331\(85\)90116-4](https://doi.org/10.1016/0141-9331(85)90116-4).
- [14] BRESLER, B., AND SCORDELIS, A. C. Shear Strength of Reinforced Concrete Beams. *ACI Journal Proceedings* 60, 1 (1963), 86–91. doi: [10.14359/7842](https://doi.org/10.14359/7842).
- [15] CAPALDI, F. M. *Continuum Mechanics: Constitutive Modeling of Structural and Biological Materials*, 1st ed. Cambridge University Press. ISBN: 978-1107011816, New York, NY, USA, 2012. doi: [10.1017/CBO9780511996528](https://doi.org/10.1017/CBO9780511996528).
- [16] CHACON, S., AND STRAUB, B. *Pro Git*, 2nd ed. Apress L.P. ISBN: 978-1484200773, 2014. Available at <https://git-scm.com/book/en/v2>. Accessed on 02 Jan 2019.

- [17] CODE_ASTER. Structural and Thermomechanics for Studies and Research. <https://www.code-aster.org/>, Mar. 2019. Accessed on 15 Mar 2019.
- [18] CODE_ASTER. Training - Code_Aster. <https://www.code-aster.org/spip.php?rubrique68>, Mar. 2019. Accessed on 15 Mar 2019.
- [19] CREATIVE COMMONS. Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International Public License. <https://creativecommons.org/licenses/by-nc-sa/4.0/legalcode>, Nov. 2013. Accessed on 28 Dec 2018.
- [20] DEPARTMENT OF CIVIL GEOLOGICAL AND ENVIRONMENTAL ENGINEERING OF THE UNIVERSITY OF SASKATCHEWAN. *Handbook for Graduate Students. Part 1: Academic Policies and Guidelines*. <https://engineering.usask.ca/documents/depts/cge/CGE-Grad-Handbook-Part-1-Sept-2017.pdf>, Sept. 2017. Accessed on 26 Dec 2018.
- [21] DIBONA, C., AND OCKMAN, S., Eds. *Open Sources: Voices from the Open Source Revolution*, 1st ed. O'Reilly Media. ISBN: 978-1565925823, Sebastopol, CA, USA, 1999. Available at <https://www.oreilly.com/openbook/opensources/book>. Accessed on 02 Jan 2019.
- [22] DiBONA, C., OCKMAN, S., AND STONE, M. Introduction. In *Open Sources: Voices from the Open Source Revolution*, C. Dibona and S. Ockman, Eds., 1st ed. O'Reilly Media. ISBN: 978-1565925823, Sebastopol, CA, USA, 1999, ch. 1, pp. 1–18. Available at <https://www.oreilly.com/openbook/opensources/book/intro.html>. Accessed on 02 Jan 2019.
- [23] DIBONA, C., STONE, M., AND COOPER, D. *Open Sources 2.0: The Continuing Evolution*, 1st ed. O'Reilly Media. ISBN: 978-0596008024, Sebastopol, CA, USA, 2005. Available at <https://www.oreilly.com/library/view/open-sources-20/0596008023>. Accessed on 02 Jan 2019.
- [24] DISTROWATCH.COM: PUT THE FUN BACK INTO COMPUTING. USE LINUX, BSD. Distribution Search. <https://distrowatch.com/search.php>, Dec. 2018. Accessed on 29 Dec 2018.
- [25] DISTROWATCH.COM: PUT THE FUN BACK INTO COMPUTING. USE LINUX, BSD. Top Ten Distributions - an overview of today's top distributions. <https://distrowatch.com/dwres.php?resource=major>, Dec. 2018. Accessed on 29 Dec 2018.
- [26] ELLIS, M., AND STROUSTRUP, B. *The Annotated C++ Reference Manual*, 1st ed. Addison-Wesley Professional. ISBN: 978-0201514599, Reading, MA, USA, 1990.
- [27] FELIPPA, C. A. *Introduction to Finite Element Methods*. Department of Aerospace Engineering Sciences and Center for Aerospace Structures, University of Colorado, Bolder, CO, USA, 2013.
- [28] FOGEL, K. *Producing Open Source Software - How To Run A Successful Free Software Project*. O'Reilly Media. ISBN: 978-0596007591, Beijing, CN, 2009. Available at <https://producingoss.com>. Accessed on 02 Jan 2019.
- [29] FRANCOIS, H. R7.01.08 - Model of damage of Mazars. https://www.code-aster.org/V2/doc/default/en/man_r/r7/r7.01.08.pdf, Feb. 2018. Accessed on 23 Jun 2019.
- [30] GAUDEUL, A. Do Open Source Developers Respond to Competition? The LATEX Case Study. *Review of Network Economics* 6, 2 (jan 2007). doi: [10.2202/1446-9022.1119](https://doi.org/10.2202/1446-9022.1119).
- [31] GEORGE T. HEINEMAN, W. T. C. *Component-Based Software Engineering: Putting the Pieces Together*, 1st ed. Addison Wesley Pub. Co. Inc. ISBN: 978-0768682076, Boston, MA, USA, 2001.
- [32] GHOSH, R. A., Ed. *CODE: Collaborative Ownership and the Digital Economy (Leonardo Books)*, 1st ed. The MIT Press. ISBN: 978-0262572361, Cambridge, MA, USA, 2005.
- [33] GITHUB INC. GitHub Facts. <https://github.com/about/facts>, Nov. 2018. Accessed on 29 Dec 2018.

- [34] GITHUB INC. Projects - The State of the Octoverse. <https://octoverse.github.com/projects#languages>, Dec. 2018. Accessed on 29 Dec 2018.
- [35] GRAHAM, M., SABBATA, S. D., AND ZOOK, M. A. Towards a Study of Information Geographies: (Im)mutable Augmentations and a Mapping of the Geographies of Information. *Geo: Geography and Environment* 2, 1 (jun 2015), 88–105. doi: [10.1002/geo2.8](https://doi.org/10.1002/geo2.8).
- [36] GREENSTEIN, S. The Range of Linus’ Law. *IEEE Micro* 32, 1 (jan 2012), 72–72. doi: [10.1109/mm.2012.10](https://doi.org/10.1109/mm.2012.10).
- [37] GUILLEM, WEREON, AND HOTMOCHA. Unix Timeline. https://commons.wikimedia.org/wiki/File:Unix_timeline.en.svg, July 2009. Accessed on 29 Dec 2018.
- [38] GUTTAG, J. V. *Introduction to Computation and Programming Using Python*, 2nd ed. The MIT Press. ISBN: 978-0262529624, Cambridge, MA, USA, 2016.
- [39] ISO/IEC 9899:2018. Information Technology – Programming Languages – C. Standard, International Organization for Standardization, Geneva, CH, June 2018.
- [40] JONES, M. T. *GNU/Linux Application Programming (Programming Series)*, 2nd ed. Charles River Media. ISBN: 978-1584505686, Boston, MA, USA, 2008.
- [41] KERNIGHAN, B. W., AND RITCHIE, D. *The C Programming Language*, 2nd ed. Microsoft Press. ISBN: 978-0131103627, Englewood Cliffs, NJ, USA, 1988.
- [42] LESSIG, L. Open Code and Open Societies: Values of Internet Governance. *Chicago-Kent Law Review* 74, 3 (June 1999), 1405–1420. Available at <https://scholarship.kentlaw.iit.edu/cklawreview/vol74/iss3/17>.
- [43] LEVINE, S. S., AND PRIETULA, M. J. Open Collaboration for Innovation: Principles and Performance. *Organization Science* 25, 5 (oct 2014), 1414–1433. doi: [10.1287/orsc.2013.0872](https://doi.org/10.1287/orsc.2013.0872).
- [44] LOLI, F., KONIMEX, LUNDQVIST, A., RODIC, D., AND MUSTAFA, M. A. Linux Distribution Timeline. <https://github.com/FabioLolix/LinuxTimeline>, Sept. 2018. Accessed on 29 Dec 2018.
- [45] MATEAS, M., AND MONTFORT, N. A Box, Darkly: Obfuscation, Weird Languages, and Code Aesthetics. In *Proceedings of the 6th Digital Arts and Culture Conference, IT* (http://nickm.com/cis/a_box_darkly.pdf, Dec. 2005), vol. 1, University of Copenhagen, pp. 144–153. Accessed on 07 Dec 2018.
- [46] MICHEL-PONNELLE2018. FORMATION ITechCode_Asteret Salomé-Méca-module 4 : Génie Civil(ARN3960). <https://www.code-aster.org/UPLOAD/DOC/Formations/02-concretece.pdf>, May 2018. Accessed on 08 May 2019.
- [47] NUMFOCUS. FEniCS Project - NumFOCUS. <https://numfocus.org/project/fenics-project>, 2018. Accessed on 30 Dec 2018.
- [48] NUMFOCUS. Mission of NumFOCUS - NumFOCUS. <https://numfocus.org/community/mission>, 2018. Accessed on 30 Dec 2018.
- [49] NUMPY DEVELOPERS. NumPy. <http://www.numpy.org>, 2018. Accessed on 30 Dec 2018.
- [50] OPEN SOURCE INITIATIVE. The Open Source Definition. <https://opensource.org/osd>, Mar. 2007. Accessed on 26 Dec 2018.
- [51] OPEN SOURCE INITIATIVE. International Authority & Recognition. <https://opensource.org/authority>, Sept. 2018. Accessed on 26 Dec 2018.
- [52] PERKEL, J. M. Programming: Pick up Python. *Nature* 518, 7537 (feb 2015), 125–126. doi: [10.1038/518125a](https://doi.org/10.1038/518125a).

- [53] PERLIS, A. J. Special Feature: Epigrams on Programming. *ACM SIGPLAN Notices* 17, 9 (sep 1982), 7–13. doi: [10.1145/947955.1083808](https://doi.org/10.1145/947955.1083808).
- [54] PETZOLD, C. *Code: The Hidden Language of Computer Hardware and Software*, 1st ed. Microsoft Press. ISBN: 978-0735611313, Sebastopol, CA, USA, 2000.
- [55] PUBLIC WORKS AND GOVERNMENT SERVICES CANADA. Code - TERMIUM Plus®, the Government of Canada’s terminology and linguistic data bank. A product of the Translation Bureau. http://www.btb.termiumplus.gc.ca/tpv2alpha/alpha-eng.html?lang=eng&i=1&srchtxt=CODE&index=alt&codom2nd_wet=1#resultreccs, Nov. 2011. Accessed on 28 Dec 2018.
- [56] PUBLIC WORKS AND GOVERNMENT SERVICES CANADA. Source - TERMIUM Plus®, the Government of Canada’s terminology and linguistic data bank. A product of the Translation Bureau. https://www.btb.termiumplus.gc.ca/tpv2alpha/alpha-eng.html?lang=eng&i=1&srchtxt=source&index=alt&codom2nd_wet=1#resultreccs, Nov. 2011. Accessed on 28 Dec 2018.
- [57] PYDATA DEVELOPMENT TEAM. Python Data Analysis Library – pandas: Python Data Analysis Library. <https://www.scipy.org/about.html>, 2018. Accessed on 30 Dec 2018.
- [58] RAYMOND, E. S. *The Cathedral & the Bazaar: Musings on Linux and Open Source by an Accidental Revolutionary*, rev. ed. O’Reilly Media. ISBN: 978-0596001087, Beijing, CN, 2001.
- [59] ROJAS, R., GÖKTEKIN, C., FRIEDLAND, G., AND KRÜGER, M. Plankalkül: The First High-Level Programming Language and its Implementation. techreport B-3/2000, Freie Universität Berlin, Takustr. 9, 14195 Berlin, Germany, Feb. 2000.
- [60] ROJAS, R., AND HASHAGEN, U., Eds. *The First Computers—History and Architectures (History of Computing)*, 1st ed. The MIT Press. ISBN: 978-0262181976, Cambridge, MA, USA, 2000.
- [61] ROSEN, L. *Open Source Licensing: Software Freedom and Intellectual Property Law*, 1st ed. Prentice Hall Computer. ISBN: 978-0131487871, Upper Saddle River, NJ, USA, 2004.
- [62] SCIPY DEVELOPERS. Scientific Computing Tools for Python - SciPy.org. <https://www.scipy.org/about.html>, 2018. Accessed on 30 Dec 2018.
- [63] STALLMAN, R. The GNU Manifesto. *Dr. Dobbs’s Journal* 10, 3 (Mar. 1985), 30. <https://www.gnu.org/gnu/manifesto.html>. Accessed on 26 Dec 2018.
- [64] STALLMAN, R. M. new UNIX implementation. <https://groups.google.com/forum/#!msg/net.unix-wizards/8twfRPM79u0/1xlg1zrWrU0J>, Sept. 1983. Accessed on 28 Dec 2018.
- [65] STALLMAN, R. M. The GNU Operating System and the Free Software Movement. In *Open Sources: Voices from the Open Source Revolution*, C. Dibona and S. Ockman, Eds., 1st ed. O’Reilly Media. ISBN: 978-1565925823, Sebastopol, CA, USA, 1999, ch. 5, pp. 53–70. Available at <https://www.oreilly.com/openbook/opensources/book/intro.html>. Accessed on 02 Jan 2019.
- [66] STALLMAN, R. M. GNU General Public License, v3. <https://www.gnu.org/licenses/gpl.html>, June 2007. Accessed on 29 Dec 2018.
- [67] STATISTICA. Installed base of smartphones by operating system in 2015 (in million units). <https://web.archive.org/web/20161012045013/https://www.statista.com/statistics/385001/smartphone-worldwide-installed-base-operating-systems>, Nov. 2016. Archived from the original on 12 Oct 2016. Accessed on 26 Dec 2018.
- [68] SYMPY DEVELOPMENT TEAM. SymPy. <https://www.sympy.org/en/index.html>, 2018. Accessed on 30 Dec 2018.
- [69] THE MATPLOTLIB DEVELOPMENT TEAM. Matplotlib: Python plotting – Matplotlib 3.0.2 documentation. <https://matplotlib.org/>, 2018. Accessed on Dec 2018.

- [70] TOP500 SUPERCOMPUTER SITES. Operating Systems Family / Linux. <https://www.top500.org/statistics/details/osfam/1>, Nov. 2018. Accessed on 26 Dec 2018.
- [71] TRUSECURE AND RED HAT. Open Source Security: A Look at the Security Benefits of Source Code Access. Tech. rep., TruSecure and Red Hat, http://www.redhat.com/whitepapers/services/Open_Source_Security5.pdf, Aug. 2001. Accessed on 26 Dec 2018.
- [72] TURNER, M. J., CLOUGH, R. W., MARTIN, H. C., AND TOPP, L. J. Stiffness and Deflection Analysis of Complex Structures. *Journal of the Aeronautical Sciences* 23, 9 (sep 1956), 805–823. doi:[10.2514/8.3664](https://doi.org/10.2514/8.3664).
- [73] UNITED STATES PATENT AND TRADEMARK OFFICE. U.S. Trademark number 1916230. Boston, Massachusetts, Aug. 1994.
- [74] VECCHIO, F. J., AND SHIM, W. Experimental and Analytical Reexamination of Classic Concrete Beam Tests. *Journal of Structural Engineering* 130, 3 (2004), 460–469. doi: [10.1061/\(ASCE\)0733-9445\(2004\)130:3\(460\)](https://doi.org/10.1061/(ASCE)0733-9445(2004)130:3(460)).
- [75] VINET, J., AND GRIFFIN, A. Arch Linux. <https://www.archlinux.org/>, 2018. Accessed on 02 Dec 2018.
- [76] W3TECHS.COM. Historical Yearly Trends in the Usage of Operating Systems for Websites. https://w3techs.com/technologies/history_overview/operating_system/ms/y, Dec. 2018. Accessed on 26 Dec 2018.
- [77] WARSAW, B., HYLTON, J., GOODGER, D., AND COGHLAN, N. PEP 1 – PEP Purpose and Guidelines. <https://www.python.org/dev/peps/pep-0001>, June 2000. Accessed on 30 Dec 2018.
- [78] WEBER, S. *The Success of Open Source*, 1st ed. Harvard University Press. ISBN: 978-0674018587, Cambridge, MA, USA, 2004.
- [79] WEXELBLAT, R. L., Ed. *History of Programming Languages—(ACM monograph series)*, 1st ed. Academic Press. ISBN: 978-0127450407, Los Angeles, CA, USA, 1981.
- [80] WIKIPEDIA. Code_Aster — Wikipedia. https://fr.wikipedia.org/wiki/Code_Aster, Mar. 2019. Accessed on 15 Mar 2019.
- [81] WILLIAMS, S., AND STALLMAN, R. *Free as in Freedom (2.0)*, 2nd ed. Free Software Foundation, Inc. ISBN: 978-0983159216, Boston, MA, USA, 2010.
- [82] ZIENKIEWICZ, O. C., TAYLOR, R. L., AND FOX, D. *The Finite Element Method for Fluid Dynamics*, 7th ed. Butterworth-Heinemann. ISBN: 978-1856176354, 2013. doi: [10.1016/C2009-0-26328-8](https://doi.org/10.1016/C2009-0-26328-8).
- [83] ZIENKIEWICZ, O. C., TAYLOR, R. L., AND FOX, D. *The Finite Element Method: its Basis and Fundamentals*, 7th ed. Butterworth-Heinemann. ISBN: 978-1856176330, 2013. doi: [10.1016/C2009-0-24909-9](https://doi.org/10.1016/C2009-0-24909-9).
- [84] ZIENKIEWICZ, O. C., TAYLOR, R. L., AND FOX, D. *The Finite Element Method for Solid and Structural Mechanics*, 7th ed. Butterworth-Heinemann. ISBN: 978-1856176347, 2014. doi: [10.1016/C2009-0-26332-X](https://doi.org/10.1016/C2009-0-26332-X).

APPENDIX A

GEOMETRICAL MODEL

Code A.1: Geometrical model for beam OA1

```
1  # -*- coding: utf-8 -*-
2
3  ###
4  ### This file is generated automatically by SALOME v8.5.0 with dump python
   ↳ functionality
5  ###
6
7  import sys
8  import salome
9
10 salome.salome_init()
11 theStudy = salome.myStudy
12
13 import salome_notebook
14 notebook = salome_notebook.NoteBook(theStudy)
15 sys.path.insert( 0, r'/home/franks/M.Eng./model/POU_')
16
17 ###
18 ### GEOM component
19 ###
20
21 import GEOM
22 from salome.geom import geomBuilder
23 import math
24 import SALOMEDS
25
26
27 geompy = geomBuilder.New(theStudy)
28
29 O = geompy.MakeVertex(0, 0, 0)
30 OX = geompy.MakeVectorDXDYDZ(1, 0, 0)
31 OY = geompy.MakeVectorDXDYDZ(0, 1, 0)
32 OZ = geompy.MakeVectorDXDYDZ(0, 0, 1)
33 O_1 = geompy.MakeVertex(0, 0, 0)
34 LS = geompy.MakeVertex(-1.83, 0, 0)
35 RS = geompy.MakeVertex(1.83, 0, 0)
36 Section = geompy.MakeFaceHW(0.305, 0.552, 1)
37 [Wire_1] = geompy.ExtractShapes(Section, geompy.ShapeType["WIRE"], True)
38 [Vertex_1,Vertex_2,Vertex_3,Vertex_4] = geompy.ExtractShapes(Section,
   ↳ geompy.ShapeType["VERTEX"], True)
39 [Edge_1,Edge_2,Edge_3,Edge_4] = geompy.ExtractShapes(Wire_1,
   ↳ geompy.ShapeType["EDGE"], True)
40 Auto_group_for_V = geompy.CreateGroup(Section, geompy.ShapeType["EDGE"])
41 geompy.UnionList(Auto_group_for_V, [Edge_1, Edge_4])
```

```

42 Auto_group_for_Sub_mesh_1 = geompy.CreateGroup(Section, geompy.ShapeType["EDGE"])
43 geompy.UnionList(Auto_group_for_Sub_mesh_1, [Edge_2, Edge_3])
44 LE = geompy.MakeVertex(-2.05, 0, 0)
45 RE = geompy.MakeVertex(2.05, 0, 0)
46 L = geompy.MakeLineTwoPnt(LE, LS)
47 LB = geompy.MakeLineTwoPnt(LS, O_1)
48 RB = geompy.MakeLineTwoPnt(O_1, RS)
49 R = geompy.MakeLineTwoPnt(RS, RE)
50 Beam = geompy.MakeCompound([LE, LS, O_1, RS, RE, L, LB, RB, R])
51 [L_1, LB_1, RB_1, R_1] = geompy.ExtractShapes(Beam, geompy.ShapeType["EDGE"], True)
52 [LE_1, LS_1, O_2, RS_1, RE_1] = geompy.ExtractShapes(Beam, geompy.ShapeType["VERTEX"],
    ↪ True)
53 Auto_group_for_Extreme = geompy.CreateGroup(Beam, geompy.ShapeType["EDGE"])
54 geompy.UnionList(Auto_group_for_Extreme, [L_1, R_1])
55 Auto_group_for_Internal = geompy.CreateGroup(Beam, geompy.ShapeType["EDGE"])
56 geompy.UnionList(Auto_group_for_Internal, [LB_1, RB_1])
57 geompy.addToStudy( O, 'O' )
58 geompy.addToStudy( OX, 'OX' )
59 geompy.addToStudy( OY, 'OY' )
60 geompy.addToStudy( OZ, 'OZ' )
61 geompy.addToStudy( O_1, 'O' )
62 geompy.addToStudy( LS, 'LS' )
63 geompy.addToStudy( RS, 'RS' )
64 geompy.addToStudyInFather( Beam, RB_1, 'RB' )
65 geompy.addToStudy( RE, 'RE' )
66 geompy.addToStudy( R, 'R' )
67 geompy.addToStudy( LE, 'LE' )
68 geompy.addToStudy( L, 'L' )
69 geompy.addToStudy( LB, 'LB' )
70 geompy.addToStudy( RB, 'RB' )
71 geompy.addToStudy( Beam, 'Beam' )
72 geompy.addToStudy( Section, 'Section' )
73 geompy.addToStudyInFather( Section, Auto_group_for_Sub_mesh_1,
    ↪ 'Auto_group_for_Sub-mesh_1' )
74 geompy.addToStudyInFather( Beam, LB_1, 'LB' )
75 geompy.addToStudyInFather( Section, Auto_group_for_V, 'Auto_group_for_V' )
76 geompy.addToStudyInFather( Beam, R_1, 'R' )
77 geompy.addToStudyInFather( Section, Wire_1, 'Wire_1' )
78 geompy.addToStudyInFather( Section, Vertex_1, 'Vertex_1' )
79 geompy.addToStudyInFather( Section, Vertex_2, 'Vertex_2' )
80 geompy.addToStudyInFather( Section, Vertex_3, 'Vertex_3' )
81 geompy.addToStudyInFather( Section, Vertex_4, 'Vertex_4' )
82 geompy.addToStudyInFather( Wire_1, Edge_1, 'Edge_1' )
83 geompy.addToStudyInFather( Wire_1, Edge_2, 'Edge_2' )
84 geompy.addToStudyInFather( Wire_1, Edge_3, 'Edge_3' )
85 geompy.addToStudyInFather( Wire_1, Edge_4, 'Edge_4' )
86 geompy.addToStudyInFather( Beam, L_1, 'L' )
87 geompy.addToStudyInFather( Beam, LE_1, 'LE' )
88 geompy.addToStudyInFather( Beam, LS_1, 'LS' )
89 geompy.addToStudyInFather( Beam, O_2, 'O' )
90 geompy.addToStudyInFather( Beam, RS_1, 'RS' )
91 geompy.addToStudyInFather( Beam, RE_1, 'RE' )
92 geompy.addToStudyInFather( Beam, Auto_group_for_Extreme, 'Auto_group_for_Extreme' )

```



```
93 | geompy.addToStudyInFather( Beam, Auto_group_for_Internal, 'Auto_group_for_Internal' )
```

APPENDIX B

COMMAND FILE

Code B.1: Command file for Beam OA1

```
1  DEBUT(LANG='EN')
2
3  FCJ_c = 22.60E+06
4
5  EIJ_c = 36500.0E+06
6
7  NU_c = 0.22
8
9  FTJ_c = 0.33 * sqrt(FCJ_c / 1e+06) * 1e+06
10
11 EPSI_C_c = 1.6E-03
12
13 EPSD0_c = FTJ_c / EIJ_c
14
15 BT_c = 21000.0
16
17 AT_c = 1.0
18
19 BC_c = 2000.0
20
21 AC_c = 0.33
22
23 K_c = 0.7
24
25 Signl_c = 13560000
26
27 EPSI1_c = 0.0035
28
29 RHO_c = 2400.0,
30
31 NU_s = 0.33,
32
33 RHO_s = 7800.0,
34
35 Yng_25 = 2.2E+11
36
37 Sy_25 = 445.0E+06
38
39 DSiEp_25 = Yng_25 / 1.0E+04
40
41 Yng_30 = 2.0E+11
42
43 Sy_30 = 436.0E+06
44
```

```

45 DSIp_30 = Yng_30 / 1.0E+04
46
47 MAPOU = LIRE_MALLAGE(FORMAT='MED',
48                       UNITE=5)
49
50 MASEC = LIRE_MALLAGE(FORMAT='MED',
51                       UNITE=20)
52
53 GF = DEFI_GEOM_FIBRE(FIBRE=(_F(CARA='DIAMETRE',
54                                COOR_AXE_POUTRE=(0.0, 0.0),
55                                GROUP_FIBRE='SACI_25',
56                                VALE=(0.0915, -0.148, 0.0252, -0.0915, -0.148,
57                                       ↪ 0.0252)),
58                                _F(CARA='DIAMETRE',
59                                    COOR_AXE_POUTRE=(0.0, 0.0),
60                                    GROUP_FIBRE='SACI_30',
61                                    VALE=(0.089, -0.212, 0.0299, -0.089, -0.212,
62                                           ↪ 0.0299))),
63                                INFO=2,
64                                SECTION=_F(COOR_AXE_POUTRE=(0.0, 0.0),
65                                            GROUP_FIBRE='SBET',
66                                            MAILLAGE_SECT=MASEC,
67                                            TOUT_SECT='OUI'))
68
69 MOPOU = AFFE_MODELE(AFFE=_F(MODELISATION='POU_D_EM',
70                               PHENOMENE='MECANIQUE',
71                               TOUT='OUI'),
72                               MAILLAGE=MAPOU)
73
74 POUCA = AFFE_CARA_ELEM(GEOM_FIBRE=GF,
75                        MODELE=MOPOU,
76                        MULTIFIBRE=_F(GROUP_FIBRE=('SBET', 'SACI_25', 'SACI_30'),
77                                       GROUP_MA='POUTRE',
78                                       PREC_AIRE=0.02,
79                                       PREC_INERTIE=0.25),
80                        ORIENTATION=_F(CARA='ANGL_VRIL',
81                                       GROUP_MA='POUTRE',
82                                       VALE=-90.0),
83                        POUTRE=_F(CARA=('HY', 'HZ'),
84                                   GROUP_MA='POUTRE',
85                                   SECTION='RECTANGLE',
86                                   VALE=(0.305, 0.552)))
87
88 BETON = DEFI_MATER_GC(INFO=2,
89                       MAZARS=_F(AC=AC_c,
90                                  AT=AT_c,
91                                  BC=BC_c,
92                                  BT=BT_c,
93                                  CODIFICATION='ESSAI',
94                                  EIJ=EIJ_c,
95                                  EPSD0=EPSD0_c,
96                                  EPSI_C=EPSI_C_c,
97                                  EPSI_LIM=EPSI_L_c,

```

```

96             FCJ=FCJ_c,
97             FTJ=FTJ_c,
98             K=K_c,
99             NU=NU_c,
100            SIGM_LIM=Signl_c),
101            RHO=RHO_c)
102
103 AC_25 = DEFI_MATER_GC(ACIER=_F(D_SIGM_EPSI=DSiEp_25,
104                               E=Yng_25,
105                               SY=Sy_25),
106                      RHO=RHO_s)
107
108 AC_30 = DEFI_MATER_GC(ACIER=_F(D_SIGM_EPSI=DSiEp_30,
109                               E=Yng_30,
110                               SY=Sy_30),
111                      RHO=RHO_s)
112
113 MOTOR = DEFI_MATERIAU(ELAS=_F(E=2e+11,
114                               NU=0.0,
115                               RHO=RHO_s))
116
117 FOFO = DEFI_FONCTION(NOM_PARA='INST',
118                     PROL_DROITE='EXCLU',
119                     PROL_GAUCHE='EXCLU',
120                     VALE=(0.0, 0.0, 13.0, 13.0))
121
122 LINSTD = DEFI_LIST_REEL(DEBUT=0.0,
123                       INTERVALLE=(_F(JUSQU_A=0.1,
124                                       NOMBRE=2),
125                                   _F(JUSQU_A=13.0,
126                                       PAS=0.25)))
127
128 LINST = DEFI_LIST_INST(DEFI_LIST=_F(LIST_INST=LINSTD),
129                       ECHEC=_F(ACTION='DECOUPE',
130                               EVENEMENT='ERREUR',
131                               SUBD_METHODE='MANUEL',
132                               SUBD_NIVEAU=5,
133                               SUBD_PAS=4,
134                               SUBD_PAS_MINI=1e-10),
135                       METHODE='MANUEL')
136
137 BLOCAGE = AFFE_CHAR_MECA(DDL_IMPO=(_F(DRX=0.0,
138                                         DRY=0.0,
139                                         DX=0.0,
140                                         DY=0.0,
141                                         DZ=0.0,
142                                         GROUP_NO='LS'),
143                           _F(DY=0.0,
144                               GROUP_NO='RS')),
145                          MODELE=MOPOU)
146
147 DEPIMP = AFFE_CHAR_MECA(DDL_IMPO=_F(DY=-0.001,
148                                       GROUP_NO='O'),

```

```

149             MODELE=MOPOU)
150
151 PMFMAZAR = DEFI_COMPOR(GEOM_FIBRE=GF,
152             MATER_SECT=MATOR,
153             MULTIFIBRE=(_F(GROUP_FIBRE='SACI_25',
154                             MATER=AC_25,
155                             RELATION='VMIS_CINE_GC'),
156             _F(GROUP_FIBRE='SACI_30',
157                 MATER=AC_30,
158                 RELATION='VMIS_CINE_GC'),
159             _F(GROUP_FIBRE=('SBET', ),
160                 MATER=BETON,
161                 RELATION='MAZARS_GC'))))
162
163 MATMAZAR = AFFE_MATERIAU(AFFE=_F(GROUP_MA='POUTRE',
164                                   MATER=(AC_25, AC_30, BETON, MATOR)),
165                           AFFE_COMPOR=_F(COMPOR=PMFMAZAR,
166                                           GROUP_MA='POUTRE'),
167                           MAILLAGE=MAPOU)
168
169 U1MAZAR = STAT_NON_LINE(CARA_ELEM=POUCA,
170                         CHAM_MATER=MATMAZAR,
171                         COMPORTEMENT=_F(RELATION='MULTIFIBRE'),
172                         CONVERGENCE=_F(ITER_GLOB_MAXI=10,
173                                         RESI_GLOB_RELA=1e-05),
174                         EXCIT=(_F(CHARGE=BLOCAGE),
175                                _F(CHARGE=DEPIMP,
176                                    FONC_MULT=FOFO)),
177                         INCREMENT=_F(LIST_INST=LINST),
178                         MODELE=MOPOU,
179                         NEWTON=_F(MATRICE='TANGENTE',
180                                   REAC_ITER=1))
181
182 U1MAZAR = CALC_CHAMP(reuse=U1MAZAR,
183                     FORCE=('REAC_NODA', ),
184                     RESULTAT=U1MAZAR)
185
186 U1MAZAR = CALC_CHAMP(reuse=U1MAZAR,
187                     DEFORMATION=('EPSI_ELGA', 'EPSP_ELGA'),
188                     RESULTAT=U1MAZAR)
189
190 SUM_REAC = POST_RELEVE_T(ACTION=_F(GROUP_NO=('LS', 'RS'),
191                                     INITITULE='sum reactions',
192                                     MOYE_NOEUD='OUI',
193                                     NOM_CHAM='REAC_NODA',
194                                     OPERATION=('EXTRACTION', ),
195                                     REPERE='GLOBAL',
196                                     RESULTANTE=('DX', 'DY', 'DZ'),
197                                     RESULTAT=U1MAZAR,
198                                     TOUT_ORDRE='OUI'))
199
200 IMPR_RESU(FORMAT='RESULTAT',
201           MODELE=MOPOU,

```

```

202         RESU=_F(GROUP_NO=('O', ),
203                 MAILLAGE=MAPOU,
204                 NOM_CAS=('DY', ),
205                 NOM_CHAM=('DEPL', ),
206                 RESULTAT=U1MAZAR),
207         UNITE=3)
208
209 IMPR_RESU(FORMAT='RESULTAT',
210           MODELE=MOPOU,
211           RESU=_F(GROUP_NO=('LS', 'RS'),
212                 NOM_CHAM=('REAC_NODA', ),
213                 NOM_CMP=('DY', ),
214                 RESULTAT=U1MAZAR,
215                 VALE_MAX='OUI',
216                 VALE_MIN='OUI'),
217           UNITE=8)
218
219 IMPR_TABLE(TABLE=SUM_REAC,
220            UNITE=2)
221
222 FIN()

```