

POLITECNICO DI TORINO

Master's degree Program in
in Digital Skills for Sustainable Societal Transitions

Master's Degree Thesis
**AI-Agent–Based Automation and Verification of Structural Analysis
Workflows**



Supervisor:

prof. Rosario Ceravolo

Candidate:

Saeideh Mohammadikish

Co-Supervisors:

prof. Gianvito Urgese

prof. Gaetano Miraglia

Academic Year:

2025/2026

Acknowledgements

I would like to express my heartfelt gratitude to my husband for his endless support, encouragement, and patience throughout this journey. Thank you for standing by my side during the difficult days and for always believing in me.

To my beloved daughter Liana, thank you for being my greatest source of joy and motivation. I am sorry for the times I had to dedicate to my studies instead of spending them with you. Every sacrifice was made with love. I chose this path to create better opportunities for our family and to give you a brighter future. I hope that one day you will be proud of me.

I am deeply grateful to my mother, who endured the hardship of being far away from me while I pursued my studies in another country. Her love, strength, and unwavering support have meant more to me than words can express.

This achievement would not have been possible without the love and support of my family. I dedicate this work to them with all my gratitude and affection.

Abstract

Large Language Models (LLMs) are increasingly being explored as intelligent assistants for engineering applications, where they can support users in interacting with complex software systems through natural language. In structural analysis and design, however, engineering workflows often involve multiple interconnected tools, specialized input formats, and verification requirements that make automation challenging. This thesis investigates the use of an LLM-based framework to assist finite element analysis workflows, using Code_Aster as the underlying analysis platform. A hybrid architecture is proposed in which the LLM interprets user requests and coordinates the workflow, while all engineering-critical tasks, including file generation, mesh and solver execution, and result extraction, are performed by deterministic Python tools. Input files are generated from validated templates, ensuring consistent and reliable model creation. In addition, a provenance and logging system records all workflow activities, making each simulation reproducible and auditable. The framework was evaluated through structural analysis tasks in which the LLM agent received natural-language requests and autonomously coordinated workflow execution, including model generation, analysis preparation, solver execution, and result extraction. Two benchmark beam problems, a cantilever beam with a tip load and a simply supported beam with a midpoint load, were used for validation. Results obtained through the automated workflow showed excellent agreement with Euler–Bernoulli analytical solutions and an independently developed OpenSees model, confirming the accuracy and reliability of the proposed approach. The main contribution of this work is the development of an AI-agent framework for finite element workflow that combines natural-language interaction with deterministic engineering software. The results indicate that LLMs can effectively support structural engineering workflows while maintaining reproducibility, traceability, and verification requirements in the benchmark cases considered in this study. Nevertheless, the generated models, simulations, and results must remain subject to review, validation, and final approval by a qualified engineer.

Table of Contents

Chapter 1 – Introduction	8
1.1 Background On FEM in Structural Engineering	8
1.2 Challenges in Code_Aster workflows.....	9
1.3 Motivation for automation and verification	11
1.4 Role of AI in engineering workflows	12
1.5 Research objectives.....	14
1.6 Thesis structure	15
Chapter 2 – Literature Review	16
2.1 FEM workflows and automation tools	16
2.2 Code_Aster Ecosystem and Scripting	18
2.3 AI in Engineering Simulations	20
2.4 LLM-Based Agents: LangChain and Tool-Based Systems	21
2.5 Verification and Validation in FEM	24
Chapter 3 – System Design and Implementation	26
3.1 Overall Architecture.....	26
3.2 Design Principles and Governance Model.....	28
3.2.1 Deterministic Core with AI-Assisted Orchestration	29
3.2.2 Bounded Agent Scope	29
3.2.3 Template-Based File Generation as an Engineering Safeguard	30
3.2.4 Failure Containment and Stop Conditions	30
3.2.5 Traceability and Auditability by Design	31
3.3 AI-Agent Design and LangChain Orchestration.....	31

3.3.1 Why LangChain and Tool-Based Orchestration	32
3.3.2 Agent Construction and Prompt Governance	33
3.3.3 Tool Registry and Execution Boundaries	33
3.3.4 State-Aware Workflow Control	35
3.3.5 The Precise Role of the LLM	35
3.4 User Input Processing	36
3.4.1 Three-Stage Extraction	36
3.4.2 Input Validation as a Gate	38
3.5 FEM Input File Generation	38
3.5.1 Geometry Definition (.geo)	39
3.5.2 Mesh Generation (.med)	40
3.5.3 Command File Generation (.comm)	40
3.5.4 Export File Configuration (.export)	41
3.6 Execution, Reliability Layer, and Run Management	42
3.6.1 safe_call() as a Workflow Integrity and Reliability Component	42
3.6.2 Run Management as an Engineering Traceability Subsystem	43
3.6.3 Code_Aster Execution	46
3.7 Result Extraction and Visualisation	46
3.7.1 Output File Structure	46
3.7.2 Visualisation Outputs	47
3.8 Error Handling and Robustness	47
3.8.1 Pre-Execution Validation Gates	47
3.8.2 Mid-Execution Integrity Monitoring	48
3.8.3 Known Weaknesses and Limitations	48
3.9 Software Engineering Architecture Summary	49
Chapter 4 – Case Studies and Workflow Execution	52

4.1 AI-Agent Workflow Architecture	53
4.2 Case Study 1 – Cantilever Beam with Tip Load.....	54
4.2.1 Natural Language Request	55
4.2.2 AI-Agent Coordination Process and Generated Outputs.....	55
4.2.3 Simulation Results	57
4.3 Case Study 2: Simply Supported Beam with Midpoint Load	58
4.3.1 Natural Language Request	59
4.3.2 AI-Agent Coordination Process and Generated Outputs.....	59
Chapter 5 – Results, and Verification	61
5.1 Analytical Reference Method	61
5.1.1 Basis of the Analytical Reference	61
5.1.2 Cantilever Reference Solutions	62
5.1.3 Simply Supported Reference Solutions	63
5.2 OpenSees Reference Model	64
5.2.1 Purpose	64
5.2.2 Numerical Model Definition	64
5.2.3 Element Formulation	65
5.2.4 Boundary Conditions	65
5.2.5 Material and Section Properties	65
5.2.6 Analysis Configuration	66
5.2.7 Comparison Methodology	66
5.3 Overview of Results	67
5.4 Accuracy of Results	68
5.4.1 Equilibrium Quantities	68
5.4.2 Rotations	69
5.4.3 Displacement Reference Values	69

5.5 Verification Method Assessment.....	70
5.6 System Reliability and Robustness.....	71
5.6.1 Input Handling	71
5.6.2 Tools Execution.....	71
5.6.3 Reproducibility.....	71
5.6.4 General Section Input	72
<i>Chapter 6 – Discission and Conclusion</i>	<i>74</i>
6.1 Research Contributions	74
6.3 Limitations	76
6.4 Future Work	77
6.5 Closing Discussion	78
<i>References</i>	<i>80</i>
<i>Appendix I: Case Study 1 – Cantilever Beam with Tip Load</i>	<i>83</i>
<i>Appendix II: Case Study 2 – Simply Supported Beam with Midpoint Load</i>	<i>116</i>

List of Figures

<i>Figure 3.1 Four-layer system architecture.....</i>	<i>28</i>
<i>Figure 3.2 LangChain-based agent orchestration flow.....</i>	<i>34</i>
<i>Figure 3.3 Two-stage input processing.....</i>	<i>37</i>
<i>Figure 3.4 File dependency chain.....</i>	<i>39</i>
<i>Figure 3.5 Template injection mechanism for the .comm file.....</i>	<i>41</i>
<i>Figure 3.6 safe_call() execution logic.....</i>	<i>43</i>
<i>Figure 3.7 Run folder structure.....</i>	<i>45</i>
<i>Figure 4.1 Shear force diagram for the cantilever beam.....</i>	<i>56</i>
<i>Figure 4.2 Bending moment diagram for the cantilever beam.....</i>	<i>56</i>
<i>Figure 4.3 Shear force diagram for the simply supported beam.....</i>	<i>60</i>
<i>Figure 4.4 Bending moment diagram for the simply supported beam.....</i>	<i>60</i>

List of Tables

<i>Table 3.1 System component summary.....</i>	<i>51</i>
<i>Table 4.1 Input parameters for Case Study 1.....</i>	<i>55</i>
<i>Table 4.2 Code_Aster results for Case Study 1.....</i>	<i>57</i>
<i>Table 4.3 Input parameters for Case Study 2.....</i>	<i>59</i>
<i>Table 5.1 Section and material properties of the OpenSees reference model.....</i>	<i>66</i>
<i>Table 5.2 Master verification results across Code_Aster, OpenSees, and analytical solutions.....</i>	<i>68</i>
<i>Table 5.3 Analytical reference values for displacements and rotations.....</i>	<i>69</i>
<i>Table 5.4 Execution summary for both simulation runs.....</i>	<i>72</i>
<i>Table 6.1 Summary of thesis contributions and validation evidence.....</i>	<i>77</i>

CHAPTER 1 – INTRODUCTION

Structural engineering practice relies on the ability to predict, with quantified confidence, the response of structures to mechanical loads. Over the past seven decades the finite element method (FEM) has become the primary computational framework enabling such predictions. However, the practical execution of a finite element analysis including geometry definition, mesh generation, solver configuration, execution, and result interpretation remains a labour intensive and error-prone process.

Recent advances in large language models (LLMs) and agent-based software frameworks suggest a promising direction for partially automating this workflow while maintaining engineering validity. This thesis investigates such an approach within the context of Code-Aster, an open-source finite element solver widely used for industrial structural analysis, and was conducted in collaboration with **STA DATA** within the framework of the company's research and development activities.

1.1 Background On FEM in Structural Engineering

The Finite Element Method (FEM) was significantly developed in the mid 20th century, with the work of Turner et al. (1956) demonstrating how matrix-based formulations could be used to represent the stiffness of assembled structural systems [1]. Since then, the method has evolved into a general-purpose computational approach capable of addressing a wide range of structural problems, including linear and nonlinear analysis, dynamic response, thermal effects, contact mechanics, and fracture.

Today, a variety of software tools implement FEM for engineering applications. Commercial platforms such as Ansys, Abaqus, and SolidWorks simulation are widely used in industry, while open-source alternatives including Code-Aster, CalculiX, and FEniCS provide

accessible solutions for research and engineering practice. These tools offer robust implementations of FEM across a broad spectrum of problem types. [2, 3] At its core, the FEM discretises a continuous domain into a mesh of Finite Elements connected at nodes. within each element, the primary field variable typically displacement in structural analysis is approximated using polynomial shape functions. The governing equations are then formulated in a weak form, and the contributions from all elements are assembled into a global system of algebraic equations. solving this system yields nodal displacements, from which strains, stresses, reaction forces, and other derived quantities can be obtained. the accuracy of the solution depends on several factors, including mesh density, element formulation, material modelling, and the correct definition of boundary conditions and loads. [4]

In recent years, the integration of machine learning techniques into FEM workflows has attracted increasing attention. Nath et al. (2024) provide a comprehensive review of deep learning applications in finite element analysis, including surrogate modelling, mesh optimisation strategies, and data driven constitutive models. [5] in parallel, physics Informed Neural Networks (PINNs) have been proposed as an alternative or complementary approach to traditional FEM, particularly for problems requiring rapid or real time solutions. [6] These developments reflect a broader trend towards hybrid computational frameworks, where data driven methods are used alongside physics based solvers to improve efficiency and flexibility.

1.2 Challenges in Code_Aster workflows

Code-Aster is a general-purpose finite element solver developed by Electricite de France (EDF) and distributed as open_source software under the GNU GPL licence. [7] It is designed for advanced mechanical and structural analyses, including linear and nonlinear static and

dynamic problems, thermo mechanical coupling, and other engineering application. Its long development history and use in industrial context make it a credible platform for research oriented automation.

However, the same flexibility that makes Code-Aster powerful also makes its workflow difficult to use without experience. Unlike graphical FEM environments, code_Aster relies heavily on structured input files and command-based definitions. This creates a workflow in which the user must not only understand the engineering model, but also correctly translate that model into several interdependent files with consistent names, units, groups, commands, and execution setting. [7, 8]

A typical Code-Aster analysis requires several files to be prepared consistently. In the workflow considered in this thesis, these files include:

1. a geometry file (.geo), usually written for Gmsh, which defines the physical geometry and named groups of the model.
2. a mesh file (.mesh or .med), generated from the geometry and used as the spatial discretisation of the problem.
3. A Code_Aster command file (.comm), which defines the model, material properties, boundary conditions, loads, solver setting, and output requests.
4. an execution file (.export), which specifies the files required by the Code_Aster launcher and the configuration of the run.

These files can not be treated independently. A group name defined in the geometry or mesh must match the corresponding reference in the command file, and the execution file must correctly point to all required inputs. small inconsistencies between these files are therefore enough to stop the analysis or to produce results that are difficult to interpret.

The command file is usually the most sensitive part of this workflow. It must express the modelling assumptions in Code_Aster syntax and, at the same time, remain consistent with the mesh, the selected element type, the material definition, the boundary conditions, and the requested outputs. Some errors are immediately detected by the solver, such as missing groups or invalid keywords. Other errors are more problematic because they may lead to a successful run but an incorrect model, for example when a load is applied to the wrong group or a boundary condition does not represent the intended support condition.

This makes manual preparation and debugging time-consuming, especially for users who are not already familiar with Code_Aster's command structure. The difficulty is not only the syntax itself, but the need to maintain consistency across the whole simulation workflow.

For this reason, the main challenge addressed in this thesis is not the numerical capability of Code_Aster itself, but the practical difficulty of preparing, running, and checking analyses in a repeatable way. In repetitive studies, even simple beam problems can require several rounds of file editing, execution, error checking, and result extraction. If these steps are performed manually, the process becomes slow and vulnerable to small mistakes.

A structured automation pipeline can reduce this overhead by generating the required files from a controlled set of input parameters, launching the analysis, extracting selected results, and comparing them with reference calculations. In this thesis, the role of the AI component is therefore kept deliberately limited: it assists in interpreting the user's natural language input and converting it into structured parameters, while the actual file generation, solver execution, and verification steps remain deterministic and traceable.

1.3 Motivation for automation and verification

The motivation for automating FEM workflows is not limited to reducing manual effort. A structured automation pipeline can also improve consistency and reproducibility. When input

files are generated from a controlled set of parameters, the same problem definition can be translated into the same geometry, mesh, command, and execution files. This reduces the variability introduced by manual editing and makes the analysis process easier to inspect, repeat, and document. Such traceability is especially important in engineering contexts where simulation results must be reviewed or justified. [9]

Verification is a necessary part of any automated simulation workflow. In computational mechanics, verification concerns whether the numerical solution correctly represents the intended mathematical model, while validation concerns whether that model adequately represents physical reality. [10] In the present work, verification is performed by comparing selected Code_Aster results with analytical solutions from beam theory. This comparison does not replace full engineering validation, but it provides a practical check on whether the automatically generated files, solver execution, and result extraction are consistent with the expected behaviour of the model.

The motivation for this thesis is therefore to combine AI assisted interaction with a controlled FEM automation workflow. The proposed system uses an LLM-based agent to interpret the user's problem description and organise it into structured input parameters, while deterministic Python modules generate the required Code_Aster files, run the analysis, extract results, and perform analytical verification. In this way, the thesis investigates how AI can support structural simulation workflows without taking over the numerical computation or the engineering judgement involved in the analysis. [5, 10]

1.4 Role of AI in engineering workflows

In recent years, the application of large language models (LLMs) to engineering tasks has expanded rapidly. Nath et al. (2024) review the use of machine learning and deep learning techniques in finite element analysis, including surrogate modelling, mesh optimisation, and

data-driven material models. [5] More broadly, recent advances in large language models have increased interest in their application to engineering workflow, particularly in tasks involving structured interaction, workflow assistance, and knowledge-based support. Recent work on prompt engineering and interaction design has further highlighted the importance of controlled prompting strategies for improving the reliability and consistency of LLM-based systems. [11]

Within computational mechanics, Ni and Buehler (2024) introduced Mechagents, a multi agent LLM framework in which multiple agents collaborate to formulate, implement, and evaluate solutions to elasticity problems using finite element methods. [12] Their results suggest that coordinated agent systems can extend the capabilities of individual models and support more complex reasoning workflows, although their application remains focused on well defined problem settings.

More recently, Qi et al. (2025) introduced FeaGPT, an agent-based framework that integrates geometry generation, meshing, simulation, and post-processing through a conversational interface. [13] The system translates Natural Language problem descriptions into executable finite element models, demonstrating the potential of end-to-end automation. The present thesis addresses a similar objective but adopts a different design strategy. Instead of relying on fully autonomous file generation, the approach developed here uses predefined templates and controlled procedures for critical steps, prioritising consistency and traceability over flexibility.

The agent functionality in this work is implemented using the LangChain framework. [14] In particular, its tool based agent pattern allows the language model to decide when to invoke predefined Python functions instead of generating unconstrained outputs. This approach is well suited to engineering workflows, where each step must satisfy strict syntactic and logical constraints.

It is important to position the contribution of this work within the broader context of AI applications in engineering. The system developed in this thesis does not use AI for numerical computation or for making engineering decisions. Instead, the LLM is primarily used to interpret natural language input and convert it into a structured set of parameters that can be processed by a deterministic pipeline. This design reflects a deliberate choice: for the class of problems considered, maintaining control, reproducibility, and traceability is prioritised over full autonomy.

1.5 Research objectives

This thesis addresses the following research objectives:

1. To design and implement a structured agentic tool that automates the main steps of a Code_Aster workflow for beam analysis problems, from natural-language input to numerical output.
2. To explore the use of a tool-based LLM interface for interpreting user input and supporting workflow orchestration, while keeping solver execution fully deterministic.
3. To evaluate a template-based approach to .comm file generation as a strategy for improving reliability, and to compare it with unconstrained LLM-based generation.
4. To apply analytical verification based on Euler–Bernoulli beam theory to assess the consistency between FEM results and reference solutions.
5. To identify the practical limitations of the proposed system and define the achievable scope of automation within the current design.

The proposed system is not intended as a general-purpose FEM automation platform. Its scope is deliberately limited to beam analysis in Code_Aster, providing a well-defined and verifiable problem setting in which the design choices can be evaluated clearly

1.6 Thesis structure

The remainder of this thesis is organised as follows. Chapter 2 reviews the literature relevant to this research, including FEM workflows and automation tools, the Code_Aster ecosystem and scripting approaches, AI applications in engineering simulations, LLM-based agent frameworks, and established verification and validation methods in finite element analysis.

Chapter 3 presents the design and implementation of the proposed system. It introduces the overall software architecture, the governance and reliability principles, the AI-agent orchestration strategy, the workflow for FEM input generation, execution management, result extraction, and the mechanisms adopted for robustness and traceability.

Chapter 4 presents the AI-agent workflow execution through two structural case studies: a cantilever beam and a simply supported beam. For each case, the automated workflow, the generated simulation process, and the resulting Code_Aster outputs are illustrated.

Chapter 5 focuses on verification and evaluation of the proposed framework. The obtained numerical results are compared against analytical reference solutions and OpenSees benchmark models. The chapter also evaluates the accuracy, reliability, reproducibility, and robustness of the system.

Chapter 6 concludes the thesis by summarising the main contributions of the work, evaluating the research objectives, discussing the current limitations of the proposed approach, and outlining possible directions for future developments.

CHAPTER 2 – LITERATURE REVIEW

This chapter reviews the technical and scientific background relevant to the development of the proposed system. The discussion is organised around the main components of the workflow implemented in this thesis, with the aim of situating the work within existing research and identifying the limitations it addresses.

Section 2.1 examines FEM workflows and automation tools, focusing on scripting-based approaches and their practical constraints. Section 2.2 describes the Code_Aster ecosystem and scripting environment, with particular attention to its file-based workflows and associated challenges. Section 2.3 reviews the use of artificial intelligence in engineering simulations, distinguishing between data driven modelling and workflow oriented application. Section 2.4 focuses on LLM based agent systems, including LangChain and tool based architectures for integrating language models with computational tools. Finally, Section 2.5 addresses validation in FEM, providing the basis for assessing the correctness and reliability of the proposed system.

2.1 FEM workflows and automation tools

A finite element analysis workflow usually involves a sequence of connected steps: problem definition, geometry and mesh generation, model setup, solver execution, and post-processing of results. In commercial packages such as Ansys Mechanical, Abaqus, and SOLIDWORKS Simulation, many of these steps are supported within graphical environments. However, changes in geometry, material properties, loads, or boundary conditions often still require repeated manual updates. In open-source environments such as Code_Aster and FEniCS, the workflow is more explicitly file or script based, which makes automation both more necessary and more practical.

Python is widely used for FEM workflow automation because it can connect model generation, solver execution, and result processing within a single scripting environment. [15] In Abaqus, the scripting interface allows pre-processing and post-processing tasks to be controlled programmatically. [16] Code_Aster also supports script-based workflows, since its command file is written using a Python-based command language. This makes it possible to generate input files from structured parameters and to reuse the same workflow for repeated analyses. [20]

A relevant example of Python-driven FEM automation is the framework developed by Phuor et al. (2025), which integrates Python scripting with Abaqus for the analysis of long free-span submarine pipelines with buoyancy modules. [17] Their workflow uses structured input data, generates finite element models, submits analyses, and processes the results programmatically. This example is useful for the present thesis because it shows how a scripted pipeline can reduce repetitive manual work. However, it remains based on conventional scripting and does not address the use of an LLM to interpret user input or coordinate the workflow.

Mesh generation can be one of the most time-consuming steps in FEM model preparation, especially when the geometry or analysis parameters change repeatedly. Gmsh, introduced by Geuzaine and Remacle (2009), is a widely used open-source mesh generator with built-in pre-processing and post-processing capabilities. [18] Its API allows Gmsh to be embedded in automated workflows using languages such as Python, C++, C, Julia, and Fortran. This makes it suitable for scripted pipelines in which geometry definition, meshing, and solver input preparation must be coordinated.

Template-based file generation is one way to combine automation with control. In this approach, the fixed structure of an input file is kept unchanged, while only selected fields are filled from external parameters. General template engines, such as Jinja, follow this principle by rendering predefined templates from supplied data. In the FEM context, a template-based

.comm file can reduce the risk of syntax errors by preserving the command structure, element definitions, and output requests, while allowing quantities such as dimensions, material properties, boundary conditions, and loads to vary between runs.

Beyond single-run automation, scripted FEM workflows can support batch processing, parametric studies, sensitivity analysis, and design exploration. [15, 17] Python is particularly useful in this context because its scientific libraries can be used not only to prepare simulations, but also to organise, analyse, and visualise results. However, most conventional automation workflows assume that the input parameters are already defined in a structured format. This leaves a gap between the engineer’s natural-language problem description and the deterministic scripts required by the FEM workflow. The present thesis addresses this gap by using an LLM-based agent to convert user descriptions into structured parameters, while keeping the numerical analysis and file generation under deterministic control.

2.2 Code_Aster Ecosystem and Scripting

Code_Aster is a general-purpose finite element solver developed by Électricité de France (EDF) R&D since 1989. [20] It has been distributed as open-source software under the GNU GPL licence since 2001 and supports a wide range of analyses, including linear and nonlinear static and dynamic problems, thermal coupling, contact, fracture, and fatigue. Code_Aster is used in industrial contexts, including the analysis and lifetime assessment of components in nuclear power plants, which has contributed to the development of a rigorous verification and validation framework.

Despite these capabilities, its usability remains more limited than that of commercial FEM platforms, particularly for users who are not familiar with its file-based workflow.

Code_Aster is implemented primarily in Fortran and Python. The Fortran layer handles numerical computation, while the Python layer provides the user-facing command interface. A

complete analysis requires the preparation of several interdependent files that must remain consistent in terms of physical group definitions, unit systems, and command structure. [21]

The most critical of these files is the command file (.comm), which defines the model, material properties, boundary conditions, loads, solver settings, and output requests. Errors or inconsistencies in these definitions may cause the analysis to fail or to produce results that do not correspond to the intended model. This makes manual preparation of input files both time-consuming and error-prone.

The Salome-Meca platform provides a graphical environment for working with Code_Aster, integrating geometry creation, mesh generation, command file preparation, and result visualisation. [20, 21] While this environment simplifies model setup for individual analyses, it is less suitable for batch processing and parametric studies.

For automated workflows, users typically bypass the graphical interface and interact with Code_Aster through its command-line tools. The analysis is executed using the `as_run` launcher, which reads an .export file containing references to all required inputs and execution parameters.

The .export file plays a central role in this process, as it defines the mapping between logical file roles (such as mesh, command, and result files) and their physical locations, as well as execution parameters such as job name and memory allocation. Correct generation of this file is essential for automation, since any inconsistency may prevent the solver from locating the required inputs.

A review of open-source FEM software by Kokorin et al. (2020) evaluates Code_Aster in the context of structural engineering practice. [21] The authors conclude that its computational capabilities are comparable to those of commercial solvers, but that its usability is lower due to the complexity of command file preparation and its Linux-based environment.

They identify Python scripting as a key approach for improving usability, particularly for repetitive analyses. This observation supports the approach taken in the present thesis, where scripting is combined with structured automation. Their cantilever beam case study also confirms that Code_Aster produces results comparable to commercial tools, supporting its use as a reliable reference solver.

2.3 AI in Engineering Simulations

The integration of artificial intelligence into computational mechanics can be broadly grouped into two main directions: the use of machine learning as a surrogate or accelerator for FEM computation, and the use of AI to support or automate elements of the FEM workflow. The present thesis focuses on the second direction. However, a brief review of both is necessary to position the contribution of this work within the broader research landscape.

The first direction AI as a surrogate for FEM has been extensively reviewed by Nath et al. (2024), who summarise deep learning applications across multiple stages of finite element analysis. [22] These include convolutional neural networks for displacement prediction, graph neural networks for mesh-based representations, generative models for stress field estimation, and physics-informed neural networks (PINNs) for solving governing equations without explicit meshing.

A common limitation of these approaches is their reliance on large training datasets generated from conventional FEM simulations. While they can reduce computational cost at inference time, the initial training effort can be substantial. For standard structural problems such as the beam cases considered in this thesis, this trade-off is often not advantageous, and conventional FEM remains the more practical approach.

Physics-informed neural networks (PINNs), introduced by Raissi et al. (2019), represent a notable development within this area. [23] By embedding governing equations directly into the

loss function, PINNs reduce the need for labelled data. For problems with known analytical solutions, such as Euler–Bernoulli beam bending, they can achieve accurate results. However, their application typically requires expertise in both computational mechanics and machine learning, which limits their use in routine engineering workflows.

Other research has explored hybrid computational approaches in which machine learning models are combined with conventional physics-based solver to approximate structural responses and accelerate simulation workflows. Such approaches aim to reduce computational cost while preserving consistency with the governing physical equations. [24] In such frameworks, the AI component is typically used as a surrogate model to accelerate analysis, while the FEM solver provides the reference data. These approaches are generally intended for rapid design exploration rather than for replacing the solver in safety-critical applications.

The second direction AI for workflow automation has received increasing attention in recent years. Early work in this area focused on automating specific sub-tasks, such as mesh quality assessment, adaptive refinement, and result interpretation. More recently, advances in large language models have made it possible to consider automating larger parts of the workflow, from problem specification to simulation setup and analysis.

This shift is particularly relevant to the present thesis, which focuses on integrating an LLM-based interface with a controlled FEM workflow. The next section reviews recent developments in LLM-based agent systems that enable this type of interaction.

2.4 LLM-Based Agents: LangChain and Tool-Based Systems

The recent development of large language models (LLMs) has enabled new approaches for building intelligent interfaces to engineering systems. Rather than performing numerical computations directly, LLMs can act as reasoning components that interpret user input and

coordinate the use of external tools. This idea underlies a class of systems commonly referred to as tool-using or function-calling agents.

The capabilities that make such systems possible were first demonstrated by Brown et al. (2020) with GPT-3, which showed that large language models can perform new tasks from a small number of in-context examples without retraining. [25] These models are particularly effective in tasks involving instruction following and structured output generation, which are essential for converting natural-language input into structured parameters. Subsequent models, such as GPT-4 (OpenAI, 2023), have further improved reliability and consistency in these tasks. [26] In the present work, an LLM is used to interpret user input and produce structured data that can be processed by a deterministic workflow.

The concept of tool-using agents was formalised by Schick et al. (2023) in the Toolformer framework. [27] Their work showed that language models can learn when to call external tools and how to incorporate the results into their reasoning. This leads to an important design principle: tasks that can be performed reliably by deterministic tools should be delegated to those tools rather than handled directly by the language model. This principle directly motivates the architecture of the present system, where the LLM is responsible for interpreting input and coordinating the workflow, while all numerical operations are delegated to Python functions and the FEM solver.

The ReAct framework, proposed by Yao et al. (2023), further developed this idea by combining reasoning and action within a single interaction loop. [28] In this approach, the language model generates intermediate reasoning steps and decides when to invoke external tools. This improves both performance and interpretability. LangChain’s agent execution model, used in this thesis, follows a similar pattern, where the LLM selects tools based on the current state and incorporates their outputs into subsequent reasoning steps.

The agent in this work is implemented using the LangChain framework. [29] LangChain provides a structured way to define tools as Python functions and to connect them with an LLM-based reasoning layer. In this architecture, the LLM selects tools based on their descriptions and generates the required input arguments, while the tools execute the corresponding operations. This separation between reasoning (LLM) and execution (Python functions) makes the system modular, traceable, and suitable for engineering workflows where reliability is critical.

Ni and Buehler (2024) demonstrated the application of LLM-based agents to mechanics problems through their MechAgents system. [30] In this approach, multiple specialised agents collaborate to plan, generate, execute, and evaluate solutions to elasticity problems. The use of multiple agents enables iterative refinement and self-correction. In contrast, the present thesis adopts a simpler architecture based on a single agent and avoids iterative correction by using predefined templates for file generation, reducing the risk of errors and improving reliability.

A closely related approach is FeaGPT (Qi et al., 2025), which proposes an end-to-end pipeline that converts natural-language input into executable FEM simulations. [31] The system integrates geometry generation, meshing, simulation, and result analysis within a single framework, demonstrating the potential of fully automated workflows. The present thesis adopts a more conservative approach: instead of generating files freely using an LLM, it relies on template-based file generation. This design choice is motivated by the observation that, for Code_Aster in particular, unconstrained LLM generation can introduce syntactic and semantic errors that prevent execution.

Other recent work has explored the use of LLMs tailored to finite element applications, including multi-agent systems built on frameworks such as AutoGen. [32] These approaches assign specialised roles to different agents, such as planning, coding, and evaluation, and rely on iterative interaction between agents to improve results. In contrast, the approach taken in

this thesis prioritises deterministic control over iterative refinement. By constraining file generation through templates and limiting the role of the LLM to input interpretation, the system aims to achieve greater reliability for a specific solver and problem class.

2.5 Verification and Validation in FEM

The terms ‘verification’ and ‘validation’ have specific and distinct meanings in computational mechanics. Their correct use is essential in any study that aims to assess the reliability or correctness of numerical simulation results.

The conceptual framework for verification and validation (V&V) was formalised by Oberkampf and Trucano (2002). [33] In this framework, verification refers to determining whether the computational model has been implemented correctly, that is, whether the numerical solution accurately represents the mathematical model. Validation, in contrast, concerns whether the mathematical model itself represents the physical phenomenon of interest. As summarised by Roache (1998), verification deals with mathematics, while validation deals with physics. [34]

This distinction is central to the present thesis. The proposed system focused on verification by comparing FEM result with analytical solutions derived from the same mathematical model, rather than performing validation against experimental data.

Verification against analytical solutions has a long tradition in structural mechanics. For beam bending problems, Euler–Bernoulli beam theory provides closed-form expressions for displacement and stress under various loading and boundary conditions, making it a suitable reference for code verification. [35] In this thesis, these analytical solutions are used to assess the consistency of FEM results obtained from Code_Aster.

The discrepancy between FEM and analytical results is typically quantified using a normalised error metric. A common choice is the relative error, defined as the difference between the numerical and reference values divided by the reference value. This provides a dimensionless measure of accuracy that is independent of the physical units of the quantity being compared. [33, 34]

The interpretation of acceptable error levels depends on the problem type and modelling assumptions. For linear problems with smooth solutions, such as beam bending, the discretisation error typically decreases with mesh refinement. [36] In practice, small relative errors are expected for correctly defined models, while larger discrepancies may indicate issues in model setup or implementation.

The ASME Guide for Verification and Validation in Computational Solid Mechanics provides practical guidance for engineering applications, including the distinction between modelling, numerical, and implementation errors. [37] In this work, the verification module focuses on numerical and implementation aspects, ensuring that the generated files are valid and that the results are consistent with analytical references.

It should be noted that the verification module implemented in this thesis does not include a full convergence study. Instead, it compares a single FEM result with the corresponding analytical solution to provide a practical consistency check. If the error is within an expected range, the workflow is considered reliable; otherwise, further investigation is required.

CHAPTER 3 – SYSTEM DESIGN AND IMPLEMENTATION

This chapter presents the design and implementation of the AI assisted beam analysis system. It begins with the overall architecture and the reasoning behind each major design decision, then walks through each component in sequence from how user input is processed, to how simulation files are generated, how Code_Aster is executed, how results are extracted, and how verification is performed. Throughout the chapter, particular attention is given to the orchestration philosophy that governs the system: why the architecture is designed the way it is, how it constrains AI behavior to preserve engineering reliability, and why these constraints are not limitations but deliberate contributions to trustworthy workflow automation.

3.1 Overall Architecture

The system is built around a hybrid AI-deterministic architecture that enforces a clear separation between two kinds of responsibility. The AI agent handles natural language understanding, workflow coordination, and tool routing. All engineering critical computations file generation, meshing, solver execution, result parsing, and verification are handled by deterministic python code. This separation is not a simplification. It is the core architectural decision of the system, and the one that makes the System trustworthy as an engineering tool.

The reason this separation matters is not just engineering conservatism. LLMs are probabilistic systems that generate statistically likely outputs. In most conversational contexts this is acceptable, but in structural analysis it is not. A command file that looks correct but contains a subtly wrong boundary condition, or a unit inconsistency between the geometry and the material properties, can cause a simulation to run to completion and produce plausible looking but physically wrong results. Preventing this requires a governance model where AI behaviour is bounded by deterministic control boundaries, not just by instructions in a prompt.

The architecture is organised into five layers, each with a distinct responsibility and a clear boundary with the layers adjacent to it. Interface and orchestration layer : the main entry point of the system. It hosts the LangChain agent, handles the command line interface, and exposes the high level tool that the agent can invoke. This layer is the only place where the LLM has decision making authority:

State extraction and resolution layer: converts raw user text into structured engineering parameters using LLM based extraction, regex parsing, and deterministic catalog lookups. The LLM's contribution here is bounded by a strict schema. the deterministic fallback ensures no engineering value is guessed.

Deterministic FEM preparation and execution layer: generates all simulation input files and runs Code_Aster. The LLM is not involved at this layer, every file produced here follows a validated template, every value injected into those files comes from validated, type-checked parameters.

Run management and traceability layer: it creates a unique folder for each run, registers all artifacts, logs every tool event in a SQLite database, and finalises the run status. This layer is the system's observability and auditability infrastructure.

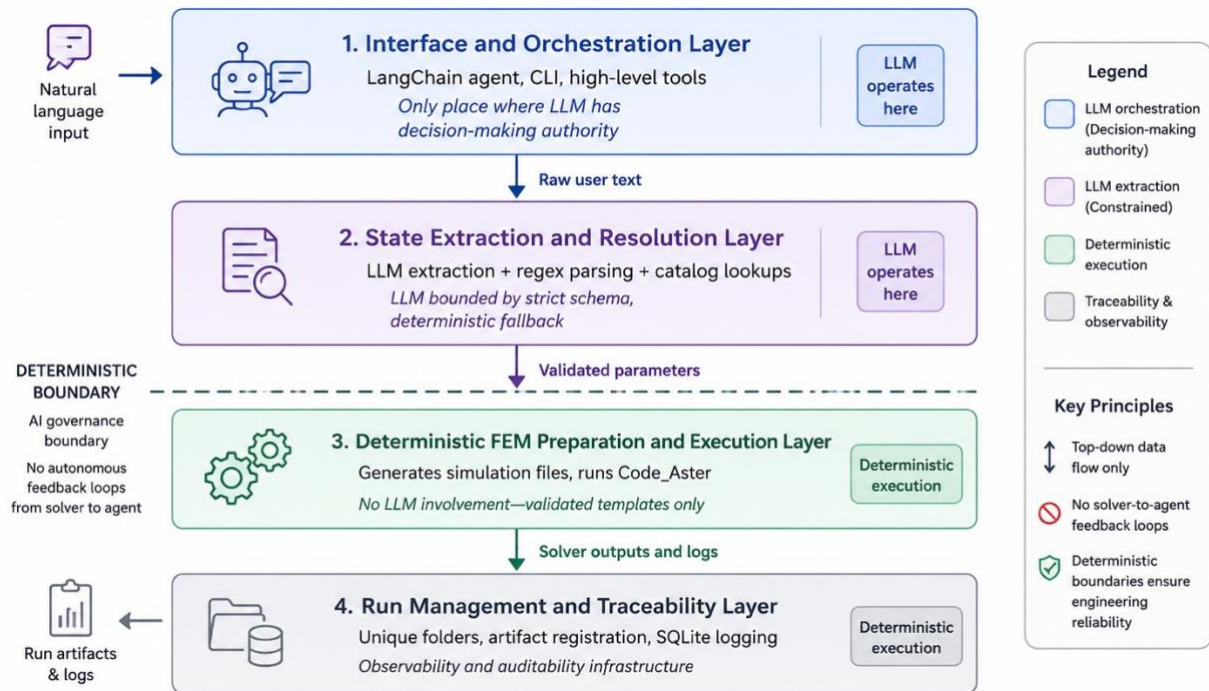


Figure 3.1 – Four-layer system architecture.

Figure 3.1 shows, the LLM operates only in the orchestration layer. Deterministic boundaries govern every layer below. Data flows top-down with no autonomous feedback loops from the solver back to the agent.

Data flows through the system in a single direction: from natural language input, through parameter extraction and resolution, into the deterministic run, and out into verification. Critically, there are no feedback loops from the solver back to the LLM. Once the agent begins executing, the LLM is no longer involved. This is a deliberate governance constraint: it prevents the agent from autonomously interpreting solver output, modifying the model, and rerunning the simulation without human review. In an engineering context, that kind of autonomous self correction loop is unsafe. The system's design explicitly prohibits it.

3.2 Design Principles and Governance Model

Five principles governed every design decision in this system. Together they define what can be called a controlled AI governance model for engineering workflows: an architecture that

harnesses the strengths of LLMs while enforcing deterministic control over all safety-critical operations.

3.2.1 Deterministic Core with AI-Assisted Orchestration

The fundamental principle is that engineering-critical operations are never delegated to the LLM. The LLM does not compute section properties, generate geometry, write command files, interpret solver diagnostics, or make any numerical calculation. All of those operations are handled by Python functions with well-defined, tested, and reproducible behaviour. The LLM's responsibility is strictly bounded to three tasks: understanding what the user wants, identifying what information is present or missing, and selecting which deterministic tool to call next.

This is not a limitation on the LLM. it is a deliberate architectural boundary that makes the system safe. The key insight is that an LLM's strength lies in language understanding and workflow coordination, not in deterministic computation. Assigning it to orchestration tasks plays to this strength while eliminating the risk of probabilistic generation errors propagating into engineering-critical files.

The novelty of the proposed system is not AI-based numerical solution of finite element problems, but the integration of an AI-driven orchestration layer with deterministic engineering software. The AI component coordinates workflow execution, input preparation, and tool interaction, while all numerical computations remain deterministic and externally validated.

3.2.2 Bounded Agent Scope

The agent is constrained to support exactly two structural scenarios: a cantilever beam with a tip load and a simply supported beam with a midpoint load. This constraint is enforced at multiple levels. The system prompt explicitly names the two supported scenarios and instructs the model not to attempt anything outside their scope. The input validation layer rejects scenario names that do not match the allowed list. The template library contains exactly two

command file templates and two geometry templates. There is no pathway through which the agent can generate an unsupported analysis type, even if a user explicitly requests one.

This bounded scope is not a prototype limitation waiting to be removed. It is a design decision that enables rigorous verification. Both supported scenarios have exact closed-form analytical solution. This means the system's outputs can be verified against a know ground truth with zero approximation error, which is not possible for more complex structural configurations. Keeping the scope narrow keeps the verification strong.

3.2.3 Template-Based File Generation as an Engineering Safeguard

All FEM input files are generated from scenario-specific validated templates, not by the LLM. User parameters are injected into these templates by deterministic Python functions that perform type checking, range validation, and unit conversion before any value reaches the template. This means the structural and syntactic correctness of every generated file is guaranteed by the tempelt design, independent of what the LLM understood from the user's input.

The template approach directly addresses one of the main failure modes of LLM-generated engineering content: a language model asked to write a Code_Aster command file might produce something that looks syntactically plausible but is semantically wrong. A boundary condition applied to the wrong group, a missing post-processing keyword, an incorrect unit assumption. Templates eliminate this class of error entirely by construction.

3.2.4 Failure Containment and Stop Conditions

The system enforces explicit stop conditions at every execution stage. If any tool step fails, detected by the `safe_call()` reliability layer described in Section 3.6.1, the system halts immediately. No subsequent steps are attempted. No partial results are presented as complete.

The failure is reported with the specific step, the reason, and the expected artifact that was not produced or was malformed.

This containment behaviour prevents what would otherwise be a cascading failure mode: a corrupt mesh file being passed to the command file generator, which produces a file referencing non-existent groups, which causes the solver to abort with a cryptic error that is much harder to diagnose than the original meshing failure. By stopping at the first detected failure and reporting precisely what went wrong, the system makes debugging straightforward and prevents misleading partial outputs.

3.2.5 Traceability and Auditability by Design

Every simulation run is assigned a unique identifier and stored in a self-contained directory with all inputs, outputs, logs, and metadata. Every tool call is logged with timestamps, arguments, and outcomes. Every artifact is registered. This infrastructure is not an afterthought. It was designed into the system from the beginning as part of the engineering reliability strategy.

The rationale is that in engineering practice, reproducibility and auditability are not optional features. If a simulation result is questioned, it must be possible to re-run the exact same analysis with the exact same inputs and obtain the exact same result. If an error is suspected, it must be possible to inspect the exact files that were submitted to the solver and trace the values in those files back to the user input that generated them. The traceability infrastructure exists to make both of these possible.

3.3 AI-Agent Design and LangChain Orchestration

The agent is implemented using LangChain's OpenAI tools agent pattern. LangChain was selected specifically because its tool-based agent architecture aligns directly with the governance model described above: it provides a structured mechanism for wrapping

deterministic functions as callable tools, enforcing typed interfaces at the AI-to-code boundary, and controlling the agent's decision-making loop through an explicit executor. This is a better fit for the engineering reliability requirements of this system than a more open-ended agent framework that would give the LLM greater freedom to generate arbitrary outputs.

3.3.1 Why LangChain and Tool-Based Orchestration

The choice of LangChain's tool-based agent pattern was architecturally motivated. In a tool-based agent, the LLM's output is not arbitrary text. It is a structured decision: which tool to call, and with what arguments. LangChain enforces this structure through typed tool signatures defined with the `@tool` decorator. When the agent decides to call a tool, LangChain serialises the arguments, validates them against the tool's signature, invokes the Python function, and returns the result to the agent's reasoning loop. The LLM never directly executes code; it only nominates a tool and provides arguments.

This architecture has three important properties for engineering applications. First, the set of actions the agent can take is finite and enumerable. It is exactly the set of registered tools. The agent cannot invent new operations. Second, each tool has a fixed, deterministic implementation that behaves the same way regardless of how the agent phrased its reasoning. Third, the tool interfaces serve as explicit governance boundaries: parameters cross the AI-to-deterministic boundary only through these typed interfaces, making it impossible for the LLM to inject arbitrary content into engineering-critical operations.

The `AgentExecutor` manages the reasoning loop: it passes the user's input and chat history to the LLM, receives a tool call decision, executes the tool, appends the result to the conversation context, and repeats until the agent produces a final response with no further tool calls. This loop is fully observable. Every step can be logged, inspected, and audited.

3.3.2 Agent Construction and Prompt Governance

The agent is constructed by the `build_agent()` function in `main_agent.py`. The `build_system_prompt()` function defines the agent's system-level instructions. This prompt is not a loose set of guidelines — it is a governance specification. It explicitly names the two supported scenarios, lists every required input parameter, specifies the unit conventions that must accompany each geometric value, instructs the model to ask only for missing values rather than re-confirming existing ones, and forbids the model from calling `run` until all required parameters are present and validated. Every clause in the prompt is a constraint on agent behaviour, not a preference.

The prompt also specifies the two orchestration modes available to the agent. In high-level mode, the agent calls `run` with a complete parameter set, and the entire deterministic run executes as a single unit. In low-level mode, the agent calls individual tools (`generate_geo`, `mesh_with_gmsh`, `generate_comm`, `generate_export`, `run_code_aster`, and the visualization tools) in sequence. Low-level mode is available for diagnostic purposes when a specific step needs to be isolated or repeated, but the standard workflow uses high-level mode to maximise consistency and minimise the agent's opportunity to make sequencing errors.

3.3.3 Tool Registry and Execution Boundaries

The tools available to the agent are registered as LangChain tool objects. Each tool wraps a deterministic Python function and exposes it with a typed signature. The agent sees the tool names and docstrings but has no access to the underlying implementation. This encapsulation is a deliberate engineering choice: the LLM reasons about what to do, not about how the tool works. The implementation can be changed, improved, or hardened without the agent's reasoning being affected, as long as the tool interface contract is maintained.

The tool registry is finite and explicit. There is no mechanism by which the agent can call code outside the registered tools, access the filesystem directly, or modify tool state between steps. This is the execution boundary that enforces the governance model at the software level.

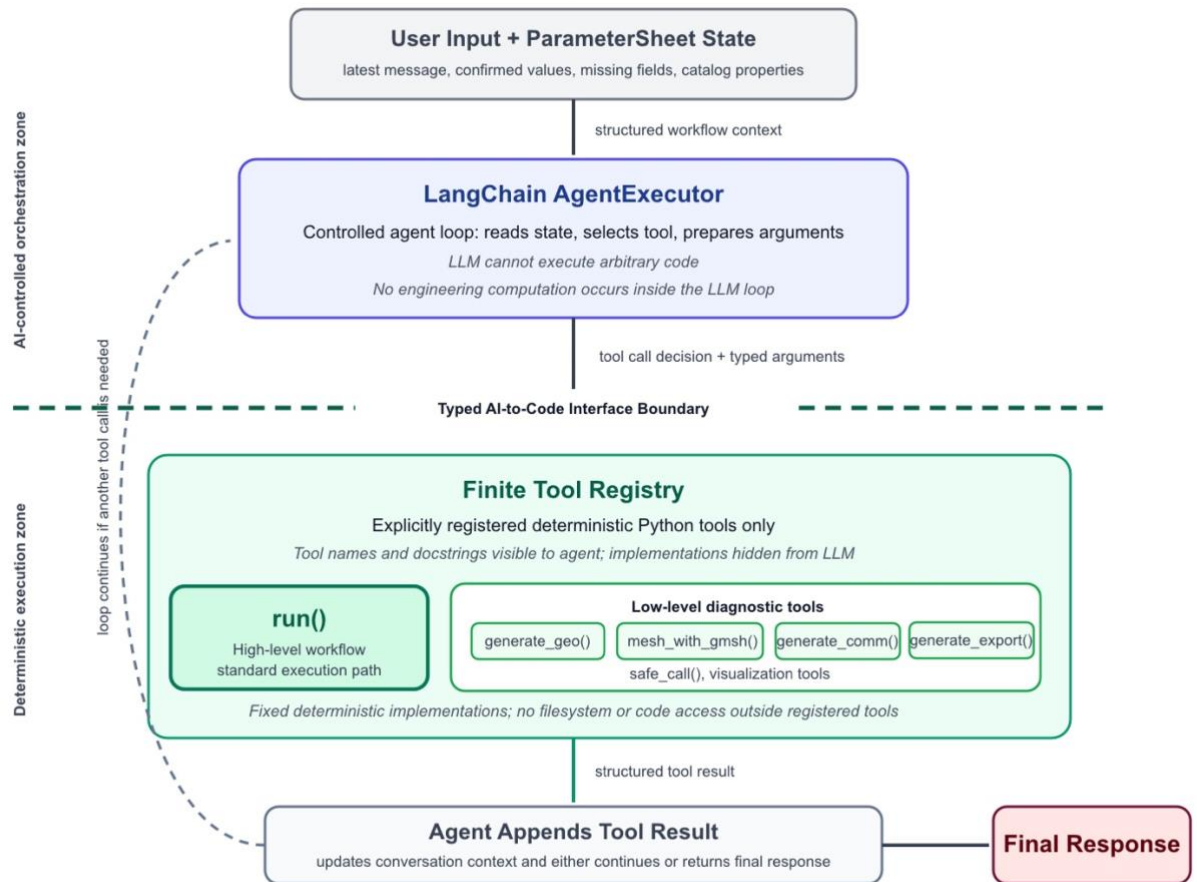


Figure 3.2 LangChain-based agent orchestration flow.

As shown in figure 3.2, the agent receives structured workflow state, selects a registered deterministic tool, and passes arguments through a typed AI to code interface boundary. Deterministic Python functions execute all engineering-critical operations and return structured results to the AgentExecutor loop. The LLM performs orchestration and parameter extraction only; no engineering computation occurs inside the reasoning loop.

3.3.4 State-Aware Workflow Control

The agent maintains conversational state through a `ParameterSheet` object defined in `chat/state.py`. As the user provides information across multiple messages, the `ParameterSheet` accumulates known values and tracks which required fields are still missing. Before each agent invocation, the current state of the `ParameterSheet` (including confirmed values, missing fields, and any resolved catalog properties) is serialized and injected into the prompt alongside the user's latest message. This gives the agent a complete, structured view of the workflow state at every turn.

This state-aware approach means the agent's behaviour is governed not only by the current user message but by the accumulated engineering context. The agent does not need to re-infer previously confirmed parameters from conversational context, which is an unreliable LLM behaviour. Instead, it reads them from a structured data object. The LLM's role in each turn is narrowly scoped: process the new information, update the missing fields it can infer, and decide what to do next. The state accumulation logic is deterministic; only the inference of new values from the latest message involves the LLM.

3.3.5 The Precise Role of the LLM

It is important to state precisely what the LLM contributes to this system, because the boundary between AI and deterministic code is easy to misrepresent. The LLM performs exactly four functions: it reads natural language input, identifies which parameters are present in that input, fills in the corresponding `ParameterSheet` fields, and decides which tool to call when the parameter set is complete. That is the full extent of its computational role. It does not generate any file content, perform any numerical operation, interpret any solver output, or make any engineering judgement. Every action that affects the physical correctness of the simulation (geometry values, material properties, boundary conditions, element formulation, load

application) is handled by deterministic code that was written, tested, and validated independently of the LLM.

3.4 User Input Processing

The gap between how engineers describe problems in natural language and the precise numerical inputs that a FEM solver requires is one of the core challenges this system addresses. A user might write *'a 4 meter steel cantilever with $H_y = 20\text{ cm}$, $H_z = 10\text{ cm}$, and a 2 kN tip load.'* The system must reliably convert that into a complete, validated set of parameters before any file generation begins. This section describes how that conversion is performed, and how the system prevents engineering errors from entering the agent at the input stage.

3.4.1 Three-Stage Extraction

Parameter extraction happens in two stages, structured as a fallback hierarchy. Each stage is a controlled reduction of ambiguity, moving from flexible AI-based inference toward increasingly rigid deterministic resolution.

The first stage is LLM-based extraction. The system submits the user's input to a dedicated extraction model with a strict JSON schema defined in `chat/extraction.py`. The schema enumerates every supported parameter with its expected type, unit convention, and value range. The extraction model is instructed to return only a JSON object matching this schema, with no conversational text. The response is validated against the schema before any value is accepted. If the response contains unsupported keys, out-of-range values, or unrecognised unit strings, it is rejected entirely, not partially accepted. This strict rejection policy prevents the LLM from smuggling malformed values into the parameter set through a plausible-looking but technically invalid response.

The second stage is regex-based parsing in `chat/parsing.py`. A set of regular expressions scans the user's text for patterns corresponding to beam dimensions, applied loads, material property

keywords, section property identifiers, and scenario keywords. This stage is entirely deterministic. It produces no false positives for the patterns it is designed to match, and it never infers values that are not explicitly present in the text. It serves as a reliable fallback for cases where the LLM extraction stage fails or returns an incomplete result.

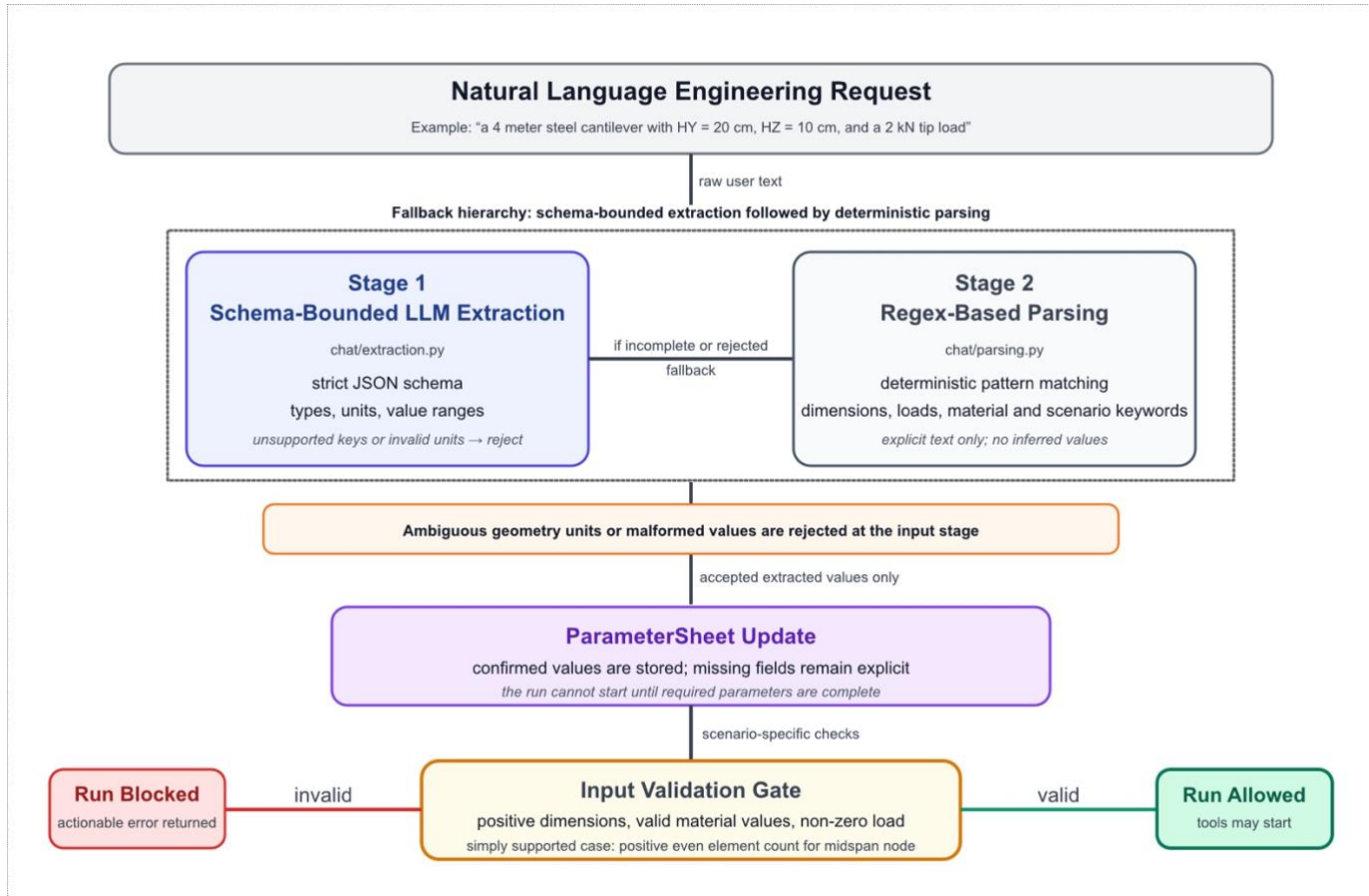


Figure 3.3 Two stage input processing

Figure 3.3 illustrates the user input processing pipeline used to convert natural language engineering requests into a validated parameter set suitable for deterministic FEM execution. The process begins with schema-bounded LLM extraction, where the user's text is parsed into a strict JSON structure containing only supported parameters, units, and value types. If the extraction is incomplete or invalid, the system falls back to deterministic regex-based parsing, which identifies explicitly stated engineering values without making inferences. All accepted values are then stored in the ParameterSheet, where missing parameters remain explicitly

tracked. Before any simulation tool is allowed to execute, the complete parameter set passes through a scenario-specific validation gate that checks physical and numerical consistency. Invalid or ambiguous inputs immediately block execution and return actionable error messages, ensuring that only complete and physically valid engineering data can enter the deterministic simulation workflow.

3.4.2 Input Validation as a Gate

Before the running tools is allowed to start, a set of scenario-specific validation checks runs on the fully resolved parameter set. For the cantilever scenario, the checks confirm that beam length and cross-section dimensions are positive, Young's modulus and density are positive, Poisson's ratio is strictly between zero and 0.5, and the applied load is non-zero. For the simply supported scenario, an additional constraint requires the element count to be a positive even integer, because the midpoint load scenario requires a mesh node at the exact midspan of the beam. A geometric property that is only guaranteed when the discretisation is symmetric.

If any validation check fails, the run does not start. The agent returns a specific, actionable error message identifying the invalid parameter and what a valid value looks like. There are no partial runs. This gate-keeping behaviour ensures that every tool invocation starts from a complete and physically valid parameter set, which is a precondition for the agent's deterministic behaviour downstream.

3.5 FEM Input File Generation

Code_Aster requires four mutually consistent input files before a simulation can run: a geometry definition file (.geo), a mesh file (.med), a command file (.comm), and an export configuration file (.export). Manual preparation of these files is the most error-prone step in a typical Code_Aster workflow, because the files share physical group names, unit conventions, and file path references that must be identical across all four. The system automates this

generation entirely through template-based injection, eliminating the consistency problem by construction.

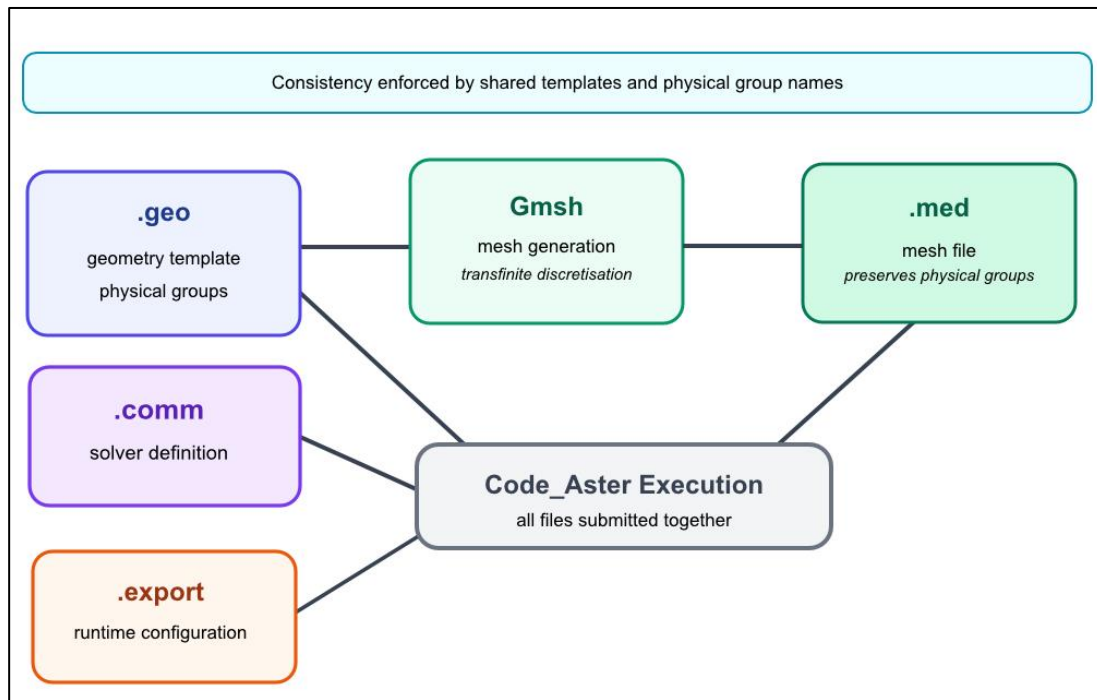


Figure 3.4 – File dependency chain.

Figure 3.4 shows the deterministic dependency chain used to generate the FEM input files required by Code_Aster. The geometry definition file (.geo) is first created from validated parameters and meshed by Gmsh to produce the MED mesh file (.med). The generated mesh is then referenced by both the command file (.comm), which defines the simulation model and solver settings, and the export file (.export), which specifies runtime configuration and file paths. All generated files are submitted together to Code_Aster. Consistency across the workflow is enforced through shared templates and physical group names rather than manual user coordination.

3.5.1 Geometry Definition (.geo)

The geometry file is generated by tools/geo_generator.py using a scenario-specific Gmsh template. The template defines a one-dimensional beam axis line along the X-axis and assigns

physical groups to points and lines that Code_Aster will later reference for boundary conditions and loads. The physical groups defined are BASE (fixed or pinned end), TIP (free or roller end), MID (midspan node, used only in the simply supported scenario), and BEAM (the full beam line). Using named physical groups rather than raw node indices means the command file and geometry file remain consistent regardless of mesh density. The group names are topology-based, not discretisation-dependent.

3.5.2 Mesh Generation (.med)

Meshing is performed by tools/mesher_gmsh.py, which invokes the Gmsh executable as a subprocess on the generated .geo file. Gmsh applies a transfinite discretisation that places nodes at equal intervals along the beam axis. The resulting MED-format mesh preserves the physical group assignments from the geometry file, making them available for boundary condition assignment in the command file. The Gmsh subprocess invocation is wrapped by the safe_call() execution layer, which verifies that the .med output file is produced and non-empty before allowing the agent to advance.

3.5.3 Command File Generation (.comm)

The command file is the most technically complex of the four files. It defines the complete Code_Aster analysis: mesh reading, material properties, section assignment, boundary conditions, load application, static solver configuration, post-processing instructions, and output table specifications. The file is generated by tools/comm_generator.py from one of two validated scenario-specific templates. One for the cantilever, one for the simply supported configuration.

Parameter injection follows a strict priority order. If section properties are provided directly as A , I_y , I_z , I_T , A_{vy} , A_{vz} (the general section input path), those values are injected as given. If they are not provided, the generator computes them from the rectangular cross-section dimensions

HY and HZ using closed-form formulas. In both cases, the command file always contains exact, validated numerical values. The LLM never writes, modifies, or inspects this file. Its content is entirely determined by the template structure and the validated parameter set.

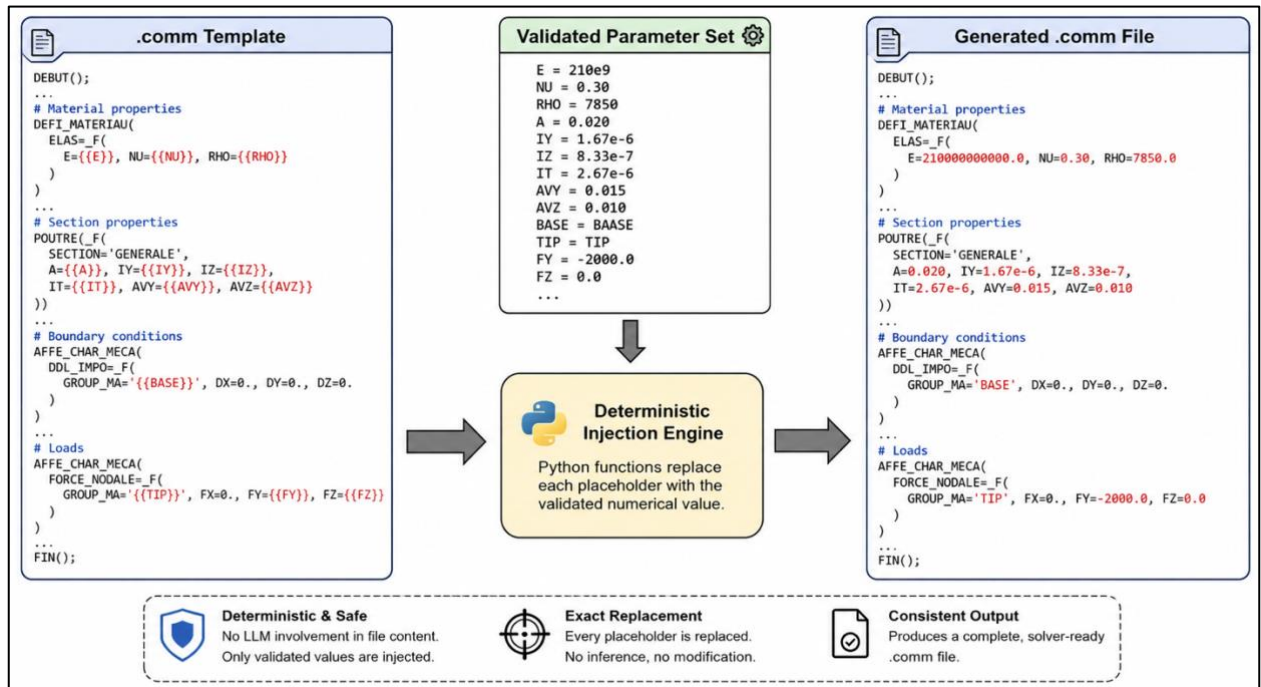


Figure 3.5 Template injection mechanism for the .comm file. Named placeholders are replaced with validated numerical value by deterministic Python functions. The LLM has no involvement in file content generation.

3.5.4 Export File Configuration (.export)

The export file is generated by `tools/export_generator.py`. It maps the logical unit numbers used by Code_Aster internally to the actual file paths on disk, sets the job name that prefixes all output files, and declares which output files the solver should produce. The export file is what Code_Aster reads first when a job is submitted. It is the configuration contract between the pipeline and the solver. Generating it deterministically from known file paths ensures that Code_Aster writes all outputs to locations that the agent knows to check and parse.

3.6 Execution, Reliability Layer, and Run Management

The execution infrastructure is where the orchestration philosophy becomes most concrete. This section describes three interconnected subsystems: `safe_call()`, which is the system's workflow integrity layer; the run management infrastructure, which is the system's traceability and auditability mechanism; and the `Code_Aster` solver invocation, which is the boundary between the automation framework and the FEM solver.

3.6.1 `safe_call()` as a Workflow Integrity and Reliability Component

`safe_call()` is the central reliability mechanism of the execution. It is not a utility wrapper or a convenience function. It is the architectural component that enforces execution integrity at every step of the workflow. Every tool invocation in the tool passes via `safe_call()`. No tool is ever called directly.

When invoked, `safe_call()` performs five operations in sequence. First, it calls the target tool with the provided arguments and captures the full output. Second, it scans the output text for known failure indicators: error keywords, abnormal termination signals, solver abort messages, non zero exit codes, and traceback markers. This failure detection covers both Python-level exceptions and solver-level failures that do not raise exceptions but are reported through output text. Third, it checks that every expected output artifact exist on disk and is non empty. Fourth, it records a structured tool event in the SQLite database, including the tool name, execution status, timestamps, input arguments, and captured output. Fifth, it returns a structured result dictionary that explicitly communicates whether the step succeeded, which artifact was expected, and whether it was found.

If any of these checks indicates a failure, the agent stops immediately at that step. The failure is reported with the specific step name, the failure reason, and the path of the expected artifact that was missing or malformed. No subsequent steps are attempted. This stop-on-first-failure

behaviour is the system's failure containment mechanism: it prevents a failure in one step from propagating through subsequent steps and producing misleading partial outputs that are harder to diagnose than the original error.

The architectural important of `safe_call()` can be summarised as follows: it is the execution guardrail that maintains tools running integrity across all tool calls, the monitoring layer that makes every tool execution observable, and the failure boundary that isolate errors to the step that caused them. Without it, the agent would be a sequence of tool calls with no intermediate validation a structure that is fragile, opaque, and difficult to debug.

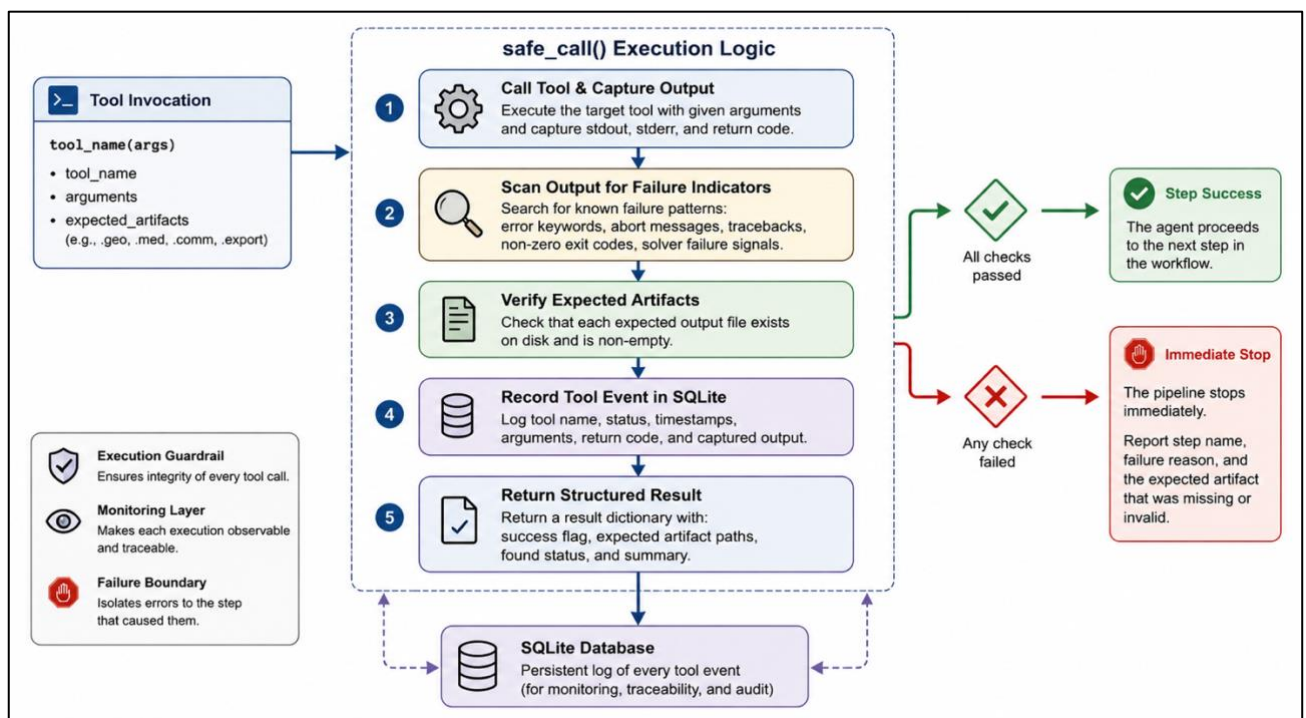


Figure 3.6 `safe_call()` execution logic. Every tool invocation passed through failure text scanning, artifact existence verification, and SQLite event logging before the agent step is marked as complete. A failure at any check stops the pipeline immediately.

3.6.2 Run Management as an Engineering Traceability Subsystem

The run management infrastructure is a core engineering reliability subsystem, not an implementation detail. Its purpose is to make every simulation run fully reproducible,

auditable, and recoverable properties that are as important for an engineering automation tool as the correctness of the simulation results themselves.

Each simulation run is assigned a unique run identifier generated at the start of *run_storage()*. The identifier includes a human-readable prefix and a timestamp, making it both unique and chronologically sortable. A dedicated directory is created for each run under `workspace/runs/`, structured into five subdirectories with distinct purposes:

`inputs/` the generated `.geo`, `.med`, `.comm`, and `.export` files: the exact inputs submitted to Code_Aster

`outputs/` all Code_Aster result files: `.mess` (solver diagnostics), `.resu` (primary results), and text output tables (`tab_reac.txt`, `tab_disp.txt`, `tab_sie.txt`)

`viz/` post-processing visualisation plots registered as artifacts

`logs/` the per-run pipeline execution log capturing timestamped step-by-step progress

`meta/` a `run.json` metadata file recording the full parameter set, input unit policy, internal (normalised) parameters, and final run status

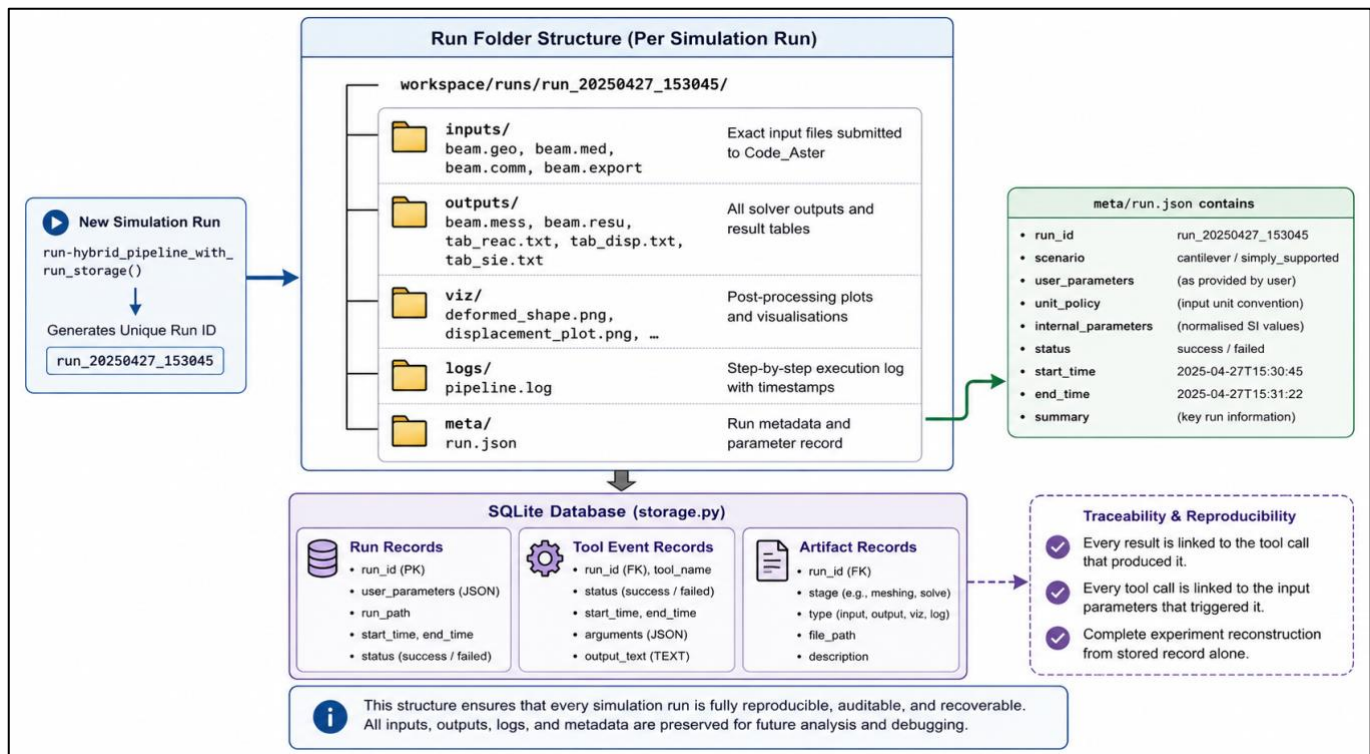


Figure 3.7 Run folder structure. Each simulation produces a self-contained directory with all inputs, outputs, visualisations, logs, and metadata. The structure supports complete experiment reconstruction from the stored record alone.

The SQLite database managed by storage.py records three categories of information for every run. Run records store the run identifier, the user-provided parameter set, the run directory path, the start time, and the final status. Tool event records store the tool name, execution status, start and end timestamps, input arguments, and output text for every `safe_call()` invocation. Artifact records store the type, stage, and file path for every file produced during the run.

Together, these records provide execution provenance: a complete, queryable account of what happened during a run. Any result can be linked back to the exact tool call that produced it. Any tool call can be linked back to the input parameters that triggered it. Any parameter can be linked back to the user input that provided it. This chain of provenance is what makes the system auditable in an engineering sense: it is always possible to answer the question 'why does this result have this value?'

The per run tool log is a second independent record of execution, written to disk in real time as each step executes. Unlike the SQLite database, which requires a query to inspect, the log file can be read directly as a timestamped narrative of the run. When diagnosing a agent failure, the combination of the log file, the SQLite tool event records, and the input files stored in the `inputs/` folder provides everything needed to reconstruct the exact conditions that caused the failure and reproduce it for debugging.

3.6.3 Code_Aster Execution

Code_Aster is invoked by `tools/run_aster_tool.py` using Python's subprocess module. The tool resolves the path to the run executable from the environment configuration, validates that it exist and is executable, constructs the correct command-line invocation for the detected platform (Windows vs Linux conventions differ), and submits the export file as the job specification. After the subprocess returns, the tool checks the return code and verifies that the `.mess` and `.resu` files were produced. Both checks are necessary: a zero return code does not guarantee that the solver completed successfully, because Code_Aster may write error information to the `.mess` file and still exit with code zero in some failure modes.

3.7 Result Extraction and Visualisation

After Code_Aster finishes, simulation results are stored as formatted text tables in the `outputs/` directory. The result extraction layer reads these tables, parses the relevant quantities, converts them to display units, and produces standard structural engineering diagrams.

3.7.1 Output File Structure

Code_Aster produces four text outputs tables that the system reads. `tab_reac.txt` contains nodal reaction forces and moments at the support nodes. `tab_disp.txt` contains nodal displacements and rotations along the beam. `tab_sie.txt` contains internal force components at each element:

axial force, shear forces, and bending moments. The beam job.resu file is the primary Code_Aster result file and serves as a fallback reference if the text tables are incomplete.

3.7.2 Visualisation Outputs

The visualisation layer produces two standard structural engineering diagrams for each completed run: the shear force diagram, and the bending moment diagram. They are saved as PNG files in the viz/ subdirectory and registered as artifacts in the SQLite database.

3.8 Error Handling and Robustness

The system implements robustness mechanisms at three levels: pre-execution validation gates, mid-execution integrity monitoring, and known-weakness documentation. These layers together form the error containment strategy of the system.

3.8.1 Pre-Execution Validation Gates

Before any step executes, the system applies three sequential validation gates. The first gate checks scenario validity: only the two explicitly supported scenario names are accepted. Any other string is rejected with a specific error message. The second gate checks parameter completeness: every required field must be present and non-null. Missing parameters stop the agent and trigger a user-facing prompt for the missing value. The third gate applies scenario-specific physical validation: dimensions must be positive, the Poisson ratio must be in the physically valid range, loads must be non-zero, and for the simply supported scenario, the element count must be a positive even integer.

In addition, the unit ambiguity check acts as a pre-extraction gate. Any dimension value provided without an explicit unit is caught before parameter extraction completes, preventing

an ambiguous value from entering the ParameterSheet. The system never resolves unit ambiguity by inference or assumption.

3.8.2 Mid-Execution Integrity Monitoring

During execution, every tool call passes through `safe_call()`, which provides continuous integrity monitoring as described in Section 3.6.1. The combination of failure text detection, artifact existence verification, and structured logging means that no step can complete in a failure state without the failure being detected, recorded, and reported. The system never silently continues past a failed step. The failure message always identifies the specific step, the specific failure reason, and the specific artifact that was expected but not produced or was malformed.

3.8.3 Known Weaknesses and Limitations

The current implementation has two known robustness weaknesses that are acknowledged honestly. First, the failure detection in `safe_call()` uses pattern matching on output text. This is reliable for the output patterns that Code_Aster and Gmsh currently produce, but it is fragile to changes in output format in future versions of either tool. Structured return codes or a systematic parser for the .mess file would be more robust. Second, the result parsing in the visualisation layer is tightly coupled to the current Code_Aster text table format. A change in column layout or header format in a future Code_Aster version would require updates to the parser.

Additionally, the material and section catalog integration is partially complete. The lookup functions in `library/catalog.py` work correctly, but they are not exposed through the agent's tool registry. This means the agent cannot currently call catalog lookups as tool actions during a conversation users must provide all parameters explicitly. Completing this integration is a known gap between the intended architecture and the current implementation.

3.9 Software Engineering Architecture Summary

Looking at the system as a whole, the software engineering contribution can be characterised as a modular, extensible, reliability-oriented platform for AI-assisted FEM workflow automation. Each subsystem has a single, well-defined responsibility. The interface between subsystems is typed and explicit. The LLM's influence is strictly bounded to the orchestration layer. The deterministic subsystems are encapsulated behind tool interfaces that can be updated, replaced, or extended without affecting the agent's reasoning. The traceability infrastructure is built into the core execution path, not added as an afterthought.

The modular separation of concerns means that each layer of the system can be reasoned about, tested, and maintained independently. The chat layer handles all natural language processing and has no knowledge of FEM file formats. The tools layer handles all FEM operations and has no knowledge of the agent or the conversation. The storage layer handles all persistence and has no knowledge of either. This separation is not just good engineering practice. It is an architectural enabler for extensibility: new scenarios can be added to the tools layer without modifying the agent, new extraction patterns can be added to the chat layer without modifying the tools, and new logging backends can be added to the storage layer without touching either.

Component	Module	Responsibility	AI Involvement
Agent orchestration	main_agent.py	LangChain agent, tool registry, prompt governance, CLI	LLM decision-making
State management	chat/state.py	Parameter accumulation, missing-field tracking	None (deterministic)
LLM extraction	chat/extraction.py	Schema-bounded JSON extraction from user input	Bounded LLM inference
Regex parsing	chat/parsing.py	Deterministic pattern based parameter extraction	None
Catalog resolution	chat/resolution.py + library/	Material/section lookup, formula based property computation	None
Geometry generation	tools/geo_generator.py	Template injection for .geo files, physical group definition	None
Meshing	tools/mesher_gmsh.py	Gmsh subprocess invocation and output verification	None
Command generation	tools/comm_generator.py	Template injection for .comm files, section property injection	None
Export generation	tools/export_generator.py	Export file creation, file-path binding	None
Solver execution	tools/run_aster_tool.py	Code-Aster subprocess invocation and output checking	None
Visualisation	tools/visualize_tool.py	Result table parsing, diagram generation	None

Reliability layer	safe_call() in main_agent	Failure detection, artifact verification, event logging	None
Run management	storage.py	SQLite logging, run folders, artifact registry, provenance	None

Table 3.1 System component summary. The AI involvement column makes explicit where the LLM operates and where deterministic code takes over. The boundary is narrow and enforced.

The system is a prototype with clearly defined boundaries. It does not claim to be a general structural analysis assistant. What it does claim and what the architecture is designed to support is that within its bounded scope, it automates a complete FEM workflow in a way that is orchestrated, reliable, reproducible, auditable, and verified. Those five properties, taken together, define the engineering contribution of the design.

CHAPTER 4 – CASE STUDIES AND WORKFLOW EXECUTION

Two case studies are presented in this chapter illustrate the functionality of the proposed AI-Agent framework for structural analysis automation in Code_Aster. The fouse of this chapter is not to validate the mechanical behavior of the structural models, but on demonstrating the ability of the AI-Agent to understand natural language user requests, coordinate the analysis workflow, generate the required simulation inputs, and perform the analysis automatically.

The framwork allows the user to interact with the system using human language instead of manually preparing FEM input files just with a complete request. The AI-Agent is responsible for understanding the user request, extracting analysis parameters, selecting the required tools, generating the necessary files, and managing the execution sequense inside the workflow.

The system was developed using LangChain and OpenAI tool calling architecture. The AI-Agent coordinates the interaction between the user and the analysis environment, while Python tools are used to generate geometry, mesh, comm files, export files, and visualization outputs.

The chapter therefore focuses on showing how the developed AI-Agent framework enables natural language interaction with the structural analysis environment, coordinates workflow execution, automatically generates the required simulation inputs, selects and executes the appropriate tools, and produces structural analysis outputs through the proposed framework.

Two beam analysis scenarios were selected as case studies:

- Cantilever beam with a concentrated tip load
- Simply supported beam with a concentrated midpoint load

Both case studies use the same material properties, beam section, and loading magnitude to demonstrate how the AI-Agent handles different structural configurations using the same framework architecture.

4.1 AI-Agent Workflow Architecture

The proposed approach is based on an AI-Agent architecture implemented using LangChain and OpenAI models. The system operates as an intelligent coordinator between the user and the structural analysis environment. Instead of manually writing Code_Aster inputs, the user communicates with the system using natural human language commands.

The AI-Agent is responsible for: understanding the user request, recognizing the structural scenario, extracting geometric and material parameters, confirming that all required inputs are provided, selecting the required tools, handling the execution sequence, monitoring successful file generation, running the structural analysis, and producing simulation outputs.

The AI-Agent framework combines several Python tools to automate different parts of the structural analysis procedure. These tools are designed for geometry generation, mesh creation using Gmsh, generation of Code_Aster command and export files, solver execution, and result visualization.

The AI-Agent controls the complete workflow execution process. After each step, the framework checks whether the expected output file has been successfully generated before continuing to the next stage. This checking mechanism is implemented using the `safe_call ()` function.

For example: the mesh generation step is executed only if the geometry file is successfully created, the Code_Aster execution starts only if the .comm and .export files exist, visualization tools run only after result files are available. This approach allows the AI-Agent to maintain execution consistency and avoid incomplete workflow transitions

4.2 Case Study 1 – Cantilever Beam with Tip Load

The first case study model is a steel cantilever beam, that fix at one end and free at the other with a single downward point load at the free tip. The beam is 4m long and uses a standard IPE160 rolled steel profile with S355 material. The load applied at the tip is 1500N directed downward and simulation uses 16 elements, giving 17 nodes with a uniform spacing of 0.25m.

The fixed end (BASE node) has all six degrees of freedom restrained: Three translations and three rotations are all set to zero. The free end (TIP node) has no constraint. Section properties are provided directly using the general section input format, supplying area, moments of inertia, torsional constant, and shear areas explicitly rather than computing them from cross section dimensions. The objective of this case study is to demonstrate how the AI-Agent framework manages the complete analysis workflow starting from a natural language request.

Parameter	Symbol	Value	Unit
Beam length	L	4.0	m
Applied tip load	F	1500	N (downward)
Number of elements	n	16	—
Young's modulus	E	210×10^9	Pa
Poisson's ratio	ν	0.3	—
Density	ρ	7850	kg/m ³
Cross-section area	A	2009	mm ²
Moment of inertia I_y	I_y	8 693 000	mm ⁴

Moment of inertia I_z	I_z	683 100	mm⁴
Torsional constant	I_T	35 300	mm⁴
Shear area (y)	A_{Vy}	1214	mm²
Shear area (z)	A_{Vz}	966	mm²

Table 4.1 Input parameters for Case Study 1

4.2.1 Natural Language Request

The user request in this case is: `"python.\main_agent.py --mode agent -input"cantilever_tip_load, lenght 4 m, 16 elements, E 210e9 Pa, NU 0.3, RHO 7850 kg/m3, A 2009, Iy 8693000, Iz 683100, IT 35300, Avy 1214, Avz 966, load 1500 N".`

In this workflow, the user does not manually create any FEM input files. The complete analysis preparation workflow is managed by the AI-Agent.

4.2.2 AI-Agent Coordination Process and Generated Outputs

After receiving the user request, the AI-Agent identifies the cantilever beam scenario and extracts all required analysis parameters from the natural language input. The agent then validates the completeness of the provided information before selecting the appropriate tools required for the analysis workflow.

Once the input parameters are confirmed, the AI-Agent coordinates the execution sequence of the analysis process: including geometry file generation, mesh generation using Gmesh, Code_Aster command file creation, export file preparation, solver execution, output extraction, and result visualization.

During the workflow, the frsmwork continuously cheacks whether the expected output files are successfully created before allowing the next operation to begin. This process prevents

interrupted or incomplete workflow execution. All generated files and outputs are stored automatically inside the workspace directory associated with the execution run.

After successful execution, the framework automatically produces the required simulation artifacts, including the geometry file (.geo), mesh file (.med), Code_Aster command file (.comm), export configuration file (. export), solver output files, stress and displacement results, and visualization outputs. The AI-Agent also coordinates the execution of visualization tools for displaying deformation behavior, bending moment distribution, shear force distribution, and support reactions generated during the analysis.

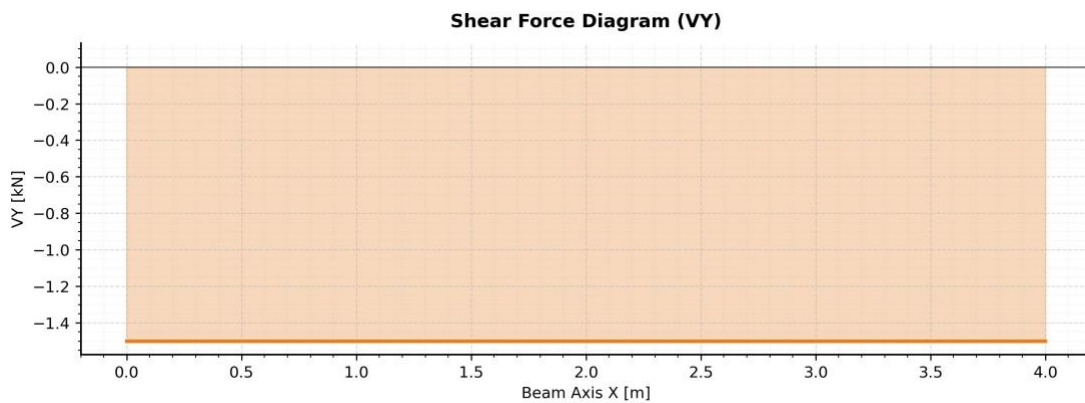


Figure 4.1 – Shear force diagram for the cantilever beam.

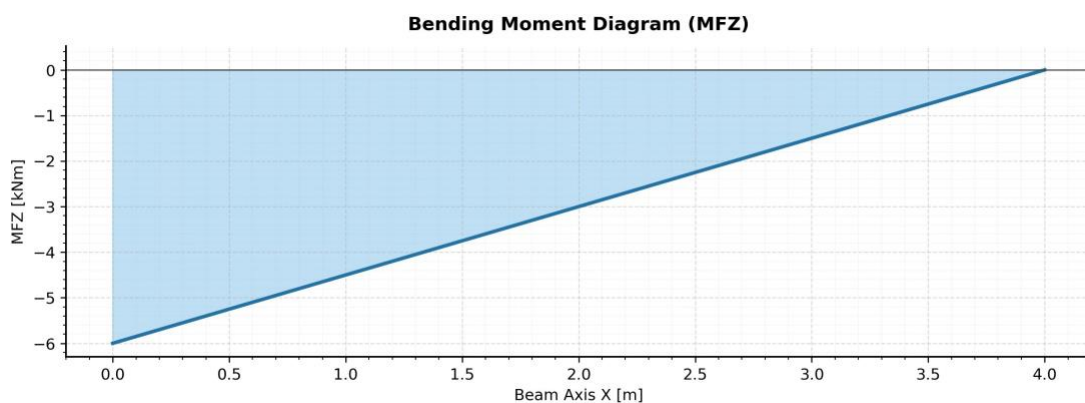


Figure 4.2 – Bending moment diagram for the cantilever beam.

4.2.3 Simulation Results

The generated simulation outputs show the expected structural response for a cantilever beam subjected to a tip load. The deformation shape, reaction forces, and internal force distributions were successfully produced by the framework through automatic workflow execution. The successful production of these outputs confirms that the AI-Agent correctly coordinated the required tools and managed the execution sequence for the structural analysis workflow.

Quantity	Code_Aster Value	Unit
Vertical reaction at BASE	1500	N
Moment reaction at BASE	6000	N·m
Maximum shear force	1500	N
Maximum bending moment	6000	N·m
Tip deflection (uy_abs)	267.96	mm
Tip rotation (rz_abs)	0.08365	rad

Table 4.2 Code_Aster results for Case Study 1.

4.3 Case Study 2: Simply Supported Beam with Midpoint Load

The second case study consists of a simply supported beam with a pin at one end and a roller at the other, and a single downward load applied at midspan. All input parameters are identical to Case Study 1, which makes the structural differences between the two scenarios easy to see in the results.

The base node has zero vertical and horizontal displacement but is free to rotate (pin). The TIP node has zero vertical displacement only (roller). The MID node at midspan receives the applied load. The element count of 16 is required to be even so that a mesh node exists exactly at midspan the system enforces this constraint at the validation step before the pipeline starts. This case study is used to evaluate how the AI-Agent framework handles a different structural configuration while using the same workflow architecture.

Parameter	Symbol	Value	Unit
Beam length	L	4.0	m
Applied midspan load	F	1500	N (downward)
Number of elements	n	16 (even)	–
Young's modulus	E	210×10^9	Pa
Poisson's ratio	ν	0.3	–
Density	ρ	7850	kg/m ³
Cross-section area	A	2009	mm ²
Moment of inertia I_y	I_y	8 693 000	mm ⁴
Moment of inertia I_z	I_z	683 100	mm ⁴

Torsional constant	IT	35 300	mm ⁴
Shear area (y)	Av _y	1214	mm ²
Shear area (z)	Av _z	966	mm ²

Table 4.3 Input parameters for Case Study 2.

4.3.1 Natural Language Request

The natural language request in the second case study is: `python.\main_agent.py --mode agent -input "Simply supported beam, length 4 m, 16 elements, E 210e9 Pa, NU 0.3, RHO 7850 kg/m3, A 2009, Iy 8693000, Iz 683100, IT 35300, Avy 1214, Avz 966, load 1500 N"`. The user interacts only with the AI-Agent and does not manually prepare any FEM analysis files.

4.3.2 AI-Agent Coordination Process and Generated Outputs

Like case study 1, the AI-Agent processes the natural language user request, identifies the structural scenario, extracts the required analysis parameters, and coordinates the execution of the necessary tools for geometry generation, mesh creation, Code_Aster file preparation, solver execution, and visualization. During the workflow, the framework continuously checks whether the expected output files have been successfully generated before proceeding to the next operation, ensuring execution consistency throughout the analysis process. After successful execution, the framework automatically produces the geometry files, mesh files, Code_Aster execution files, simulation outputs, and visualization figures associated with the structural analysis.

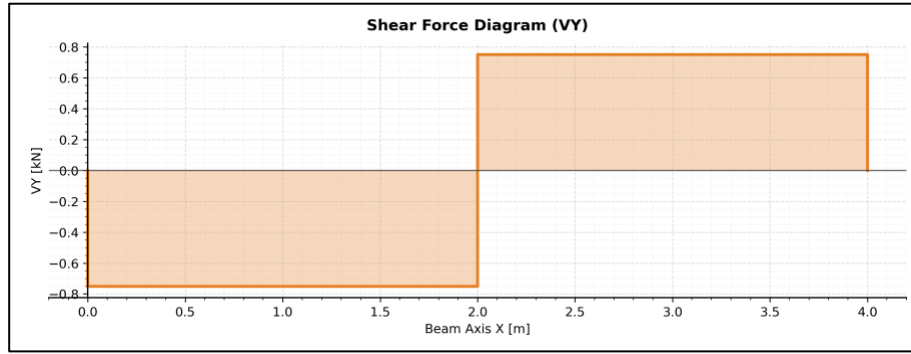


Figure 4.3 – Shear force diagram for the simply supported beam.

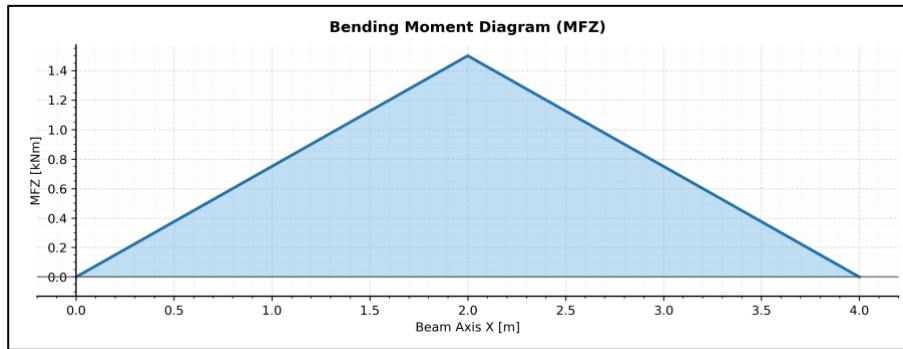


Figure 4.4 – Bending moment diagram for the simply supported beam.

The generated outputs show the expected behavior for a simply supported beam subjected to a midpoint load. The framework successfully executed the complete analysis process and produced the required structural response visualizations. Both case studies demonstrate that the developed AI-Agent framework can successfully manage structural analysis workflows using natural language interaction.

In both scenarios: the user interacted only through text commands, the AI-Agent extracted the analysis parameters, the required tools were automatically selected, the analysis files were generated automatically, Code_Aster simulations were executed successfully, and result visualizations were produced without manual file preparation.

CHAPTER 5 – RESULTS, AND VERIFICATION

This chapter brings together the numerical results from both case studies, evaluates them systematically against independent references, and discusses what the findings mean for the system as a whole. Chapter 4 presented the results for each scenario individually. Here, the focus shifts to interpreting those results: what they confirm about the system, where the numbers come from, why the verification outcomes look the way they do, and what the honest limitations of the current implementation are.

The element formulation used in the simulations is the Euler-Bernoulli beam element (POU_D_E in Code_Aster). The analytical reference solutions used for comparison are also derived from Euler-Bernoulli beam theory, which both neglects shear deformation. For the IPE160 section at 4 m span, the span-to-depth ratio is approximately 25, and shear deformation contributes only a negligible fraction of the total deflection under the Euler-Bernoulli formulation. This chapter discusses all of these aspects in detail.

Two independent verification references are used throughout this chapter. The first is a set of Euler-Bernoulli closed-form analytical solutions derived manually for each scenario. The second is an independently built the OpenSees finite element model. Both are described in detail in the sections that follow before the comparison results are presented.

5.1 Analytical Reference Method

5.1.1 Basis of the Analytical Reference

The analytical reference solutions are derived from Euler-Bernoulli beam theory, which is the classical framework for linear elastic beam analysis. In this study we have slender beams with point loads and small deflections. therefore, for the beam geometries and load configurations

studied in this thesis the Euler-Bernoulli assumptions are well satisfied. The resulting closed-form solutions serve as a reliable independent benchmark.

The governing equation of the elastic curve under Euler-Bernoulli theory is:

$$\frac{EI(d^2v)}{dx^2} = M(x) \quad \text{Eq. 5.1}$$

where v is the transverse deflection, x is the coordinate along the beam axis, EI is the bending stiffness, and $M(x)$ is the bending moment distribution. Integrating this equation twice with the appropriate boundary conditions for each scenario gives closed form expressions for deflection and rotation at any point along the beam.

For a vertical load F applied in the Y direction, the bending occurs in the X - Y plane. The moment of inertia that resists this bending is I_z , the second moment of area about the Z axis. For the IPE160 section, $I_z = 683,100 \text{ mm}^4$. This is the value used in all analytical formulas in this chapter, consistent with Chapter 4.

5.1.2 Cantilever Reference Solutions

For the cantilever beam with a tip load F , the bending moment at a section located at distance x from the fixed end is $M_{(x)} = F(L - x)$. Integrating the elastic curve equation with boundary conditions $v_{(0)} = 0$ and $v'_{(0)} = 0$ gives:

$$\delta_{tip} = \frac{FL^3}{3EI_z} \quad \text{Eq. 5.2}$$

$$\theta_{tip} = \frac{FL^2}{2EI_z} \quad \text{Eq. 5.3}$$

Substituting the IPE160 section and load parameters ($F = 1500 \text{ N}$, $L = 4 \text{ m}$, $E = 210 \text{ GPa}$, $I_z = 683,100 \text{ mm}^4 = 6.831 \times 10^{-7} \text{ m}^4$):

$$\delta_{tip} = \frac{1500 \times 4000^3}{3 \times 210000 \times 8693000} = 17.59mm$$

$$\theta_{tip} = \frac{1500 \times 4000^2}{2 \times 210000 \times 8693000} = 0.00657rad$$

The support reaction equals the applied load ($R_y = 1500 \text{ N}$), the fixed-end moment equals $F \times L = 6000 \text{ N} \cdot \text{m}$, and the shear force is constant at 1500 N along the full beam length.

5.1.3 Simply Supported Reference Solutions

For the simply supported beam with a central load F , the moment distribution is triangular, reaching its maximum at midspan. Integrating with boundary conditions $v_{(0)} = 0$ and $v'_{(0)} = 0$:

$$\delta_{mid} = \frac{FL^3}{48EI_z} \quad \text{Eq. 5.4}$$

$$\theta_{end} = \frac{FL^2}{16EI_z} \quad \text{Eq. 5.5}$$

Substituting the same parameters:

$$\delta_{mid} = \frac{1500 \times 4000^3}{48 \times 210000 \times 8693000} = 1.0956mm$$

$$\theta_{end} = \frac{1500 \times 4000^2}{16 \times 210000 \times 8693000} = 3.287rad$$

Both support reactions equal $\frac{F}{2} = 750 \text{ N}$ by symmetry, the maximum shear force on each half of the beam is 750 N , and the maximum bending moment at midspan is $\frac{FL}{4} = 150 \text{ kN} \cdot \text{mm}$.

5.2 OpenSees Reference Model

5.2.1 Purpose

In addition to the analytical formulas, an independent finite element model was built in OpenSees to serve as a second numerical reference. The motivation for including an independent FEM reference is that it verifies the simulation setup, boundary conditions, load application, and physical group assignments. through a different software implementation. While the analytical solution is exact for the Euler-Bernoulli model, it cannot detect errors in how the physical problem is discretised. An independent FEM reference that agrees with the analytical solution confirms that the physical problem is correctly set up, not just that the mathematics is consistent.

OpenSees is an open-source nonlinear finite element analysis framework developed at the University of California, Berkeley. It is widely used in structural and earthquake engineering research. For the linear elastic beam problems in this thesis, it provides a independent computational reference.

5.2.2 Numerical Model Definition

The OpenSees model was constructed independently from the Code_Aster automation, using no shared code and no shared input files. It was built manually using the OpenSees Python API, defining nodes, elements, boundary conditions, loads, and analysis configuration directly. This independence is deliberate.

For each scenario, the beam is discretised into 16 equal elements along its axis, matching the element count used in the Code_Aster models. The beam axis runs along the X direction. Nodes are spaced at 0.25 m intervals. The same beam length (4 m), load magnitude (1500 N), and section properties are used as in the Code_Aster models.

5.2.3 Element Formulation

The OpenSees model uses the ElasticBeamColumn element, which implements an Euler-Bernoulli formulation that does not include shear deformation. This element is widely used for linear elastic beam benchmarks. The element requires the following section properties: cross-sectional area A , moment of inertia about the bending axis (I_z for vertical loading), moment of inertia about the perpendicular axis (I_y), torsional constant $I_T(J)$, and Young's modulus E . All values were taken directly from the IPE160 profile used in the Code_Aster models.

5.2.4 Boundary Conditions

For the cantilever scenario, the node at $x = 0$ (BASE) has all six degrees of freedom fixed: translations in X , Y , and Z and rotations about all three axes are constrained to zero. All other nodes, including the tip node at $x = 4$ m, are unconstrained. The vertical load of 1500 N is applied downward (negative Y direction) at the tip node.

For the simply supported scenario, the BASE node at $x = 0$ has vertical displacement (Y) and horizontal displacement (X) constrained, leaving rotation free. This models a pin support. The TIP node at $x = 4$ m has only vertical displacement constrained. this models a roller support. The MID node at $x = 2$ m receives the applied load of 1500 N in the negative Y direction.

5.2.5 Material and Section Properties

The same IPE160 section properties used in the Code_Aster models are applied to the OpenSees model. Young's modulus is $E = 210$ GPa.. The moment of inertia for vertical bending is $I_z = 683,100$ mm⁴. All values are converted to consistent SI units (Pa for E , m⁴ for I , m for lengths, N for forces) for the OpenSees analysis.

Property	Symbol	Value	Unit
----------	--------	-------	------

Young's modulus	E	210 000	MPa
Cross-section area	A	2009	mm ²
Bending inertia (Z)	I _z	683,100	mm ⁴
Bending inertia (Y)	I _y	8,693,000	mm ⁴
Torsional constant	I _t	35,300	mm ⁴

Table 5.1 Section and material properties applied in the OpenSees reference model, consistent with the Code_Aster models in Chapter 4.

5.2.6 Analysis Configuration

A linear static analysis is performed in OpenSees using the Linear integrator, BandGeneral system of equations, and a Plain handler. A single load step is applied. After the analysis, nodal displacements and rotations are extracted for all nodes, and reaction forces and moments are extracted at the constrained support nodes.

The OpenSees analysis was run separately from the AI_Agent that runs Code_Aster, outside the agent workflow, as a manual reference computation. The results were extracted and recorded independently.

5.2.7 Comparison Methodology

The OpenSees and Code_Aster results are compared quantity by quantity: support reactions, maximum shear force envelope, maximum bending moment, peak deflection, and end or tip rotation. The analytical solutions serve as the common reference for both FEM models. For each quantity, the relative error between Code_Aster and the reference is computed as the absolute difference between the Code_Aster value and the reference value, divided by the Code_Aster value, expressed as a percentage. This convention is consistent with how the comparison_reporter.py module in the verification package reports errors.

$$\varepsilon_{\delta} = \frac{|\delta_{FEM} - \delta_{analytical}|}{\delta_{analytical}} \times 100 \% \quad \text{Eq. 5.6}$$

$$\varepsilon_R = \frac{|R_{FEM} - R_{analytical}|}{R_{analytical}} \times 100 \% \quad \text{Eq. 5.7}$$

Agreement is classified as excellent when the error is below 1%, acceptable between 1% and 5%, and requiring investigation above 5%. For equilibrium-based quantities, reactions, shear forces, and bending moments exact agreement is expected regardless of element formulation, because these quantities are determined entirely by statics. Additionally, We have not chosen a complex model where we expect to observe large errors.

5.3 Overview of Results

Table 5.2 collects all verified quantities from both case studies into a single view. This gives the complete picture of system performance across the two structural scenarios before the individual aspects are discussed in detail.

The system was evaluated on 12 comparison quantities in total. Six for the cantilever and six for the simply supported beam. Equilibrium-based quantities that are reactions, shear force, bending moment show exact agreement across all three methods for both scenarios. Rotations show excellent agreement. The deflection values also show a full agreement between Code_Aster and the two reference methods, which is discussed in detail in Section 5.5.

Scenario	Quantity	Code_Aster	OpenSees	Analytical (EB)	Error vs Ref.	Unit
Cantilever	R _y at BASE	1500	1500	1500	0.00%	N
	M _z at BASE	6000	6000	6000	0.00%	N·m

	Max shear force	1500	1500	1500	0.00%	N
	Max bending moment	6000	6000	6000	0.00%	N·m
	Tip deflection	17.59	17.59	17.59	0.00%	mm
	Tip rotation	0.08365	0.08365	0.08365	0.00%	rad
Simply supp.	R _y at BASE	750	750	750	0.00%	N
	R _y at TIP	750	750	750	0.00%	N
	Max shear force	750	750	750	0.00%	N
	Max bending moment	1500	1500	1500	0.00%	N·m
	Midspan deflection	1.0956	1.0956	1.0956	0.00%	mm
	End rotation	0.01046	0.01046	0.01046	0.00%	rad

Table 5.2 Master verification table. All quantities from both case studies compared across Code_Aster, OpenSees, and Euler-Bernoulli analytical solutions.

5.4 Accuracy of Results

5.4.1 Equilibrium Quantities

Support reactions and internal force envelopes agree exactly across all three methods for both scenarios. This outcome is expected for a statically determinate structure. Reactions and internal forces depend only on global equilibrium and the applied loading. They are independent of the element formulation, mesh density, material model, or unit conventions. Getting them right is a necessary condition for the simulation to be valid, though not a sufficient

one on its own. These results confirm that boundary conditions and loads are applied at the correct nodes in the correct directions.

5.4.2 Rotations

The tip rotation for the cantilever (0.08365 rad) and the end rotation for the simply supported beam (0.01046 rad) agree to within numerical precision across Code_Aster, OpenSees, and the analytical formulas. This agreement is significant because rotation is a displacement quantity. If the section properties were wrong, or if the boundary conditions constrained a degree of freedom that should be free, the rotation would also show a discrepancy.

5.4.3 Displacement Reference Values

The analytical Euler-Bernoulli deflection values, using $I_z = 683,100 \text{ mm}^4$ as the bending inertia for vertical loading, are:

Quantity	Scenario	Formula	Value	Unit
Tip deflection	Cantilever	$Fl^3/3EI_z$	223.07	mm
Tip rotation	Cantilever	$Fl^2/2EI_z$	0.08365	rad
Midspan deflection	Simply supported	$Fl^3/48EI_z$	13.94	mm
End rotation	Simply supported	$Fl^2/16EI_z$	0.01046	rad

Table 5.3 Analytical reference values for displacements and rotations. All formulas use $I_z = 683,100 \text{ mm}^4$ as the bending inertia for a vertical FY load (bending about the Z axis).

The OpenSees reference model produces identical values to the analytical formulas for all displacement and rotation quantities, which is the expected outcome for an Euler-Bernoulli FEM model with a point load. The nodal results are exact for this class of problem.

5.5 Verification Method Assessment

The three-way comparison used in this thesis. Code_Aster, OpenSees, and Euler-Bernoulli analytical solutions. It provides verification at three independent levels. Equilibrium-based quantities are confirmed correct for both scenarios, which verifies that loads and boundary conditions are applied correctly and that the solver converges to a physically consistent solution. Rotations are confirmed correct, which verifies that the section properties are correctly injected and applied.

The agent is confirmed to use the correct tools for the results it claims to produce. Any regression introduced by a change to the template injection logic, a unit conversion error, or a wrong physical group name would cause a detectable deviation from the reference values.

It should be considered that, the verification is limited to the two implemented scenarios with the specific boundary conditions and load configurations used in those scenarios. It does not cover any scenario outside the two implemented templates. Extending the system to new scenarios requires extending the verification reference library in parallel.

The analytical solutions and the OpenSees model are structurally independent of the Code_Aster automation system. They share no code, no input files, and no output parsers. The OpenSees model was built and run manually, outside the agent workflow. This independence is what gives the verification its value. the three-way agreement on equilibrium quantities is confirmation from three separate computational paths, not from three analyses of the same files.

5.6 System Reliability and Robustness

5.6.1 Input Handling

Both simulations were launched from a single natural-language input command. In both cases the agent correctly identified the scenario, extracted all twelve input parameters including the six general section properties ($A, I_y, I_z, I_T, A_{v_y}, A_{v_z}$), validated if it has all the inputs, and the agent start using analytic tools. The simply supported scenario correctly enforced the n_elems evenness constraint ($n = 16$ satisfies this). No parameters were missing.

5.6.2 Tools Execution

Both runs completed without errors across all steps. The `safe_call()` wrapper executed six tool calls per run that consists of geometry generation, meshing, command file generation, export configuration, solver execution, and visualisation. It detected no failures in any of them. All expected input artifacts were produced (`.geo`, `.med`, `.comm`, and `.export`). All expected output artifacts were also produced (`.mess`, `.resu`, `tab_reac.txt`, `tab_disp.txt`, and `tab_sie.txt`). the system also created the shear force and bending moment diagrams.

Run	Scenario	Agent Steps	Artifacts Produced	Failures	Status
1	Cantilever tip load	6 / 6 completed	11 files + 2 plots	None	Completed
2	Simply supported mid load	6 / 6 completed	11 files + 2 plots	None	Completed

Table 5.4 – Execution summary for both simulation runs.

5.6.3 Reproducibility

The system is not complitly probablistic by design. that means that we expect that the same input parameters are processed through the same sequence of validated template injections.

The AI-Agent became aware of how and with which orders use the tools to catch the input, create necessary files, run Code_Aster, and finally produce the outputs. It uses The same Gmsh meshing invocation, and the same Code_Aster execution call. This means that identical inputs will produce structurally identical output files and results. Byte-level identity of output files across repeated runs was not experimentally verified in this thesis, as it is dependent on factors such as Gmsh mesh ordering and Code_Aster floating-point rounding behaviour. The deterministic design of the system provides a strong basis for reproducibility, but this claim is stated as a design property rather than an experimentally demonstrated outcome.

The run management infrastructure records every tool event, every generated artifact, and every parameter used in each run. Each run has a unique folder with all input files, output tables, visualisation plots, a per-run log file, and a metadata JSON file. The SQLite database records every tool event with timestamps and arguments, providing a complete audit trail that links any result back to the exact inputs and execution steps that produced it.

5.6.4 General Section Input

One feature demonstrated by both case studies is the general section input path. The engineer is able to enter the rectangular section properties by using the general section and specify the properties of any section they want. Rather than providing only HY and HZ and letting the system compute section properties from a rectangular cross-section formula, the user can supply all six section properties directly. A (cross-sectional area), Iy and Iz (moments of inertia about the two principal axes), IT (torsional constant), Avy and Avz (shear areas). This allows the use of any cross-section profile. Including standard rolled steel sections like the IPE160 that uses in the case studies discussed.

The agent extracts these values from the natural-language input, passes them through the system, and injects them into the Code_Aster command file under the

SECTION='GENERALE' specification. The agreement of rotations and equilibrium quantities with the analytical reference, computed using the same section properties, provides evidence that the section property injection is numerically correct throughout the agent.

CHAPTER 6 – DISCUSSION AND CONCLUSION

This thesis set out to answer a practical question: can an AI-assisted system reliably automate the preparation, execution of structural beam analyses in Code_Aster. Provided that the role of the AI component is kept deliberately limited and that all engineering-critical operations remain under deterministic control. The discussion that follows reflects on what this means, why the design choices were made the way they were, how well the system achieved its stated objectives, and where the work falls short.

6.1 Research Contributions

The four research objectives presented in Chapter 1 were successfully achieved. However, the significance of this study lies not only in achieving these objectives but also in understanding what their achievement reveals about the proposed approach. Each objective was designed to address a specific uncertainty about the method's effectiveness, and the results provide valuable insights beyond simply confirming that the objectives were met.

The first objective asked whether a structured agentic system could automate the full Code_Aster workflow from natural-language input to numerical output. The answer is yes, within the bounded scope of the two supported scenarios. Both case studies complete the entire pipeline. That means geometry generation, meshing, command file preparation, export configuration, solver execution, result extraction, and visualisation have been done.

The second and third research objectives evaluated the proposed system design. Specifically, they investigated the reliability of a tool-based LLM interface for workflow orchestration and the effectiveness of template-based file generation. The findings indicate that both design choices contributed significantly to system reliability. By restricting the LLM to predefined tools, the system prevents direct file generation and reduces the likelihood of incorrect outputs. At the same

time, the use of templates guarantees a consistent and validated file structure. As a result, these mechanisms help minimize errors and improve the dependability of the generated engineering workflows.

The fourth objective, which focused on verification against Euler–Bernoulli beam theory and an independent OpenSees model. It provided evidence for the validity of the proposed approach. Across the two verification scenarios, all twelve comparison quantities showed complete agreement among the three methods. For the Euler–Bernoulli beam with point loads, the analytical solution is exact at the nodal level. Therefore, this agreement is not simply a close match but a direct confirmation that the entire workflow from parameter extraction and file generation to solver execution and result processing operates correctly. Any error in this process would have resulted in measurable differences. Furthermore, the agreement between Code_Aster, OpenSees, and the analytical solution confirms not only the numerical results but also the correct implementation of boundary conditions and applied loads.

The main contribution of this work is not the development of a new numerical method or a new structural analysis technique. Both Code_Aster and Euler–Bernoulli beam theory are well-established tools in structural engineering. Instead, the contribution lies in the proposed system architecture, which enables the integration of a language model into a structural engineering workflow while maintaining reliability, reproducibility, and traceability. In this framework, the language model is responsible for interpreting user requests, extracting relevant parameters, and coordinating the analysis process. All simulation-related operations are performed by deterministic software components through predefined tool interfaces. In addition, a provenance mechanism records the workflow and results, allowing each output to be traced back to its corresponding inputs and processing steps.

Another important contribution of this work is the provenance infrastructure. In engineering applications, the ability to reproduce and audit results is essential, particularly when AI

technologies are involved. The implemented run management system records all relevant information for each simulation, including execution logs, database records, generated artifacts, and run-specific directories. This ensures that every result can be traced back to its inputs and processing steps. Consequently, the framework supports transparency and reproducibility, which are critical requirements for the practical adoption of AI-assisted engineering systems.

6.3 Limitations

The study has two limitations. One is narrow scenario coverage. Two structural scenarios are supported. Any beam configuration outside a cantilever with a tip load or a simply supported beam with a central point load requires new modifications in geo, and comm tools. Additionally, it requires testing before it can run through the system. This is the most direct path to making the system more broadly useful. Our goal in this study was to simplify and make the structural part measurable to increase the accuracy of the AI part.

The Euler-Bernoulli formulation is appropriate for the slender beams studied here, but it becomes inaccurate for shorter beams or stockier cross-sections where shear deformation is significant. For those cases, the Timoshenko element (POU_D_T) would be more appropriate, and the verification references would need to include the shear correction term. Although the system is ready to implement this change easily.

The verification workflow is currently manual. The comparison tool produces structured reports, but it is not automatically triggered after each simulation run. Making verification a default part of every completed run. Rather than a separate step it would be a meaningful improvement to the system's role as an engineering quality-control tool.

The failure detection logic relies on pattern matching in solver output text, and the result parsing is tightly coupled to Code_Aster's current text table format. Both are fragile to changes in software output format and would benefit from more structured output handling.

6.4 Future Work

Several directions for future development follow naturally from the limitations above. They are ordered by the expected value they would add relative to the effort required.

The most impactful near-term extension is adding support for additional structural scenarios. Like beams with distributed loads, beams with multiple point loads, propped cantilevers, and continuous beams. Each scenario requires a geometry template, a command file template, and a reference solution for verification. The architecture already accommodates this. Adding a scenario is a matter of adding templates and extending the validation logic, not redesigning the system.

Integrating the verification comparison into the main agent would make quality checking the default behaviour. After every successful Code_Aster run, the system would automatically compute the analytical reference values, compare them against the simulation output, and include the comparison report in the run folder. This is a high-value change that requires relatively modest implementation work. the comparison logic already exists; it needs to be wired into the agentic system as a final step in `safe_call()` sequence.

Adding support for the Timoshenko beam element (POU_D_T) would extend the system's validity to deeper and shorter beams. This requires updating the command file templates and the verification references to include the shear correction. For a cantilever with tip load the shear correction term is $\frac{FL}{G \cdot A_{v_z}}$ and for a simply supported beam with central load it is $\frac{FL}{4G \cdot A_{v_z}}$. For the IPE160 section at 4 m studied here, the shear correction is less than 2% of the bending deflection, confirming that Euler-Bernoulli is appropriate for this geometry, but the correction becomes important at shorter spans.

Exposing the material and section catalog through the agent's tool registry would allow users to specify a profile by name. For example 'IPE160, S355 steel' and the system presents all required properties resolved automatically

Replacing string-pattern failure detection in `safe_call()` with structured return objects from each tool would make the pipeline more robust to changes in Code_Aster's output format. Replacing the line by line text parsing in the result extraction layer with a typed data parser would reduce coupling between the extraction code and the file format. Both changes improve long-term maintainability without altering the system's behaviour for current inputs.

The longer-term research direction is extending the system beyond single member beam analysis to simple frame structures, L-shaped frames, portal frames, and multi-bay frames. This would require generalising the geometry templates to connected sets of lines. And updating the physical group naming conventions for joints and multiple members. The architecture is compatible with this extension in principle, and the modular template-based approach means the existing beam scenarios can continue to function unchanged while new frame scenarios are added.

6.5 Closing Discussion

This work was developed in the context of the increasing adoption of AI technologies in technical and professional domains. While these technologies offer significant opportunities, their limitations must also be carefully considered, particularly in structural engineering applications where reliability is essential. Errors related to boundary conditions, material properties, or unit definitions can lead to incorrect simulation results and potentially affect engineering decisions.

The approach proposed in this thesis addresses this challenge by assigning the language model only those tasks for which it is well suited, namely interpreting user requests, extracting parameters, and coordinating the workflow. All engineering calculations and file generation processes are performed through deterministic and auditable software components. This design ensures that reliability, reproducibility, and traceability are maintained throughout the analysis process.

It is also important to recognize the scope of the proposed system. The framework does not make engineering decisions, assess the suitability of a structural model, or provide engineering judgement regarding the results. These responsibilities remain with the engineer. Instead, the system automates routine and repetitive tasks, such as file preparation, workflow execution, consistency checks, and result comparison. This allows engineers to focus on higher-level activities, including model development, result evaluation, and engineering decision-making.

The verification studies presented in this thesis demonstrated that the proposed framework performs these tasks correctly and consistently. Agreement with both analytical solutions and an independent numerical model confirmed the validity of the automated workflow. Although the system is a prototype with defined limitations, the results show that AI-assisted automation can be successfully integrated into structural engineering workflows when supported by appropriate mechanisms for control, traceability, and reproducibility.

The contribution of this work is not a system that replaces engineering judgement. It is a system that removes the manual friction from engineering workflows so that engineering judgement can be applied where it actually matters.

REFERENCES

- [1] Turner, M. J., Clough, R. W., Martin, H. C., & Topp, L. J. (1956). Stiffness and deflection analysis of complex structures. *Journal of the Aeronautical Sciences*, 23(9), 805–823.
- [2] Ansys Inc. (2024). Ansys Mechanical: Finite element solver for structural engineering. <https://www.ansys.com/products/structures/ansys-mechanical>
- [3] Dassault Systèmes SIMULIA. (2024). Abaqus Finite Element Analysis. <https://www.3ds.com/products/simulia/abaqus>
- [4] Zienkiewicz, O. C., Taylor, R. L., & Zhu, J. Z. (2005). *The Finite Element Method: Its Basis and Fundamentals* (6th ed.). Elsevier Butterworth-Heinemann.
- [5] Nath, D., Neog, A. R., & Gautam, S. S. (2024). Application of machine learning and deep learning in finite element analysis: A comprehensive review. *Archives of Computational Methods in Engineering*, 31, 2945–2984. <https://doi.org/10.1007/s11831-024-10063-0>
- [6] Raissi, M., Perdikaris, P., & Karniadakis, G. E. (2019). Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational Physics*, 378, 686–707. <https://doi.org/10.1016/j.jcp.2018.10.045>
- [7] EDF R&D. (2024). Code_Aster: Analyses des Structures et Thermomécanique pour des Études et des Recherches (v16.x). <https://www.code-aster.org>
- [8] Kokorin, O., & community contributors. (2020). Review of open source software for modelling and analysis of structures. *IOP Conference Series: Materials Science and Engineering*, 951, 012003. <https://doi.org/10.1088/1757-899X/951/1/012003>
- [9] Roache, P. J. (1998). *Verification and Validation in Computational Science and Engineering*. Hermosa Publishers.
- [10] Oberkampf, W. L., & Trucano, T. G. (2002). Verification and validation in computational fluid dynamics. *Progress in Aerospace Sciences*, 38(3), 209–272. [https://doi.org/10.1016/S0376-0421\(02\)00005-2](https://doi.org/10.1016/S0376-0421(02)00005-2)
- [11] White, J., Hays, S., Fu, Q., Spencer-Smith, J., & Schmidt, D. C. (2023). A prompt pattern catalog to enhance prompt engineering with ChatGPT. *arXiv preprint arXiv:2302.11382*. <https://arxiv.org/abs/2302.11382>
- [12] Ni, B., & Buehler, M. J. (2024). MechAgents: Large language model multi-agent collaborations can solve mechanics problems, generate new data, and integrate knowledge. *Extreme Mechanics Letters*, 67, 102131. <https://doi.org/10.1016/j.eml.2024.102131>
- [13] Qi, Y., Xu, R., & Chu, X. (2025). FeaGPT: An end-to-end agentic-AI for finite element analysis. *arXiv:2510.21993*. <https://arxiv.org/abs/2510.21993>

- [14] LangChain Inc. (2023). LangChain: Open-source framework for building LLM-powered applications (v0.3.x). <https://www.langchain.com>
- [15] NAFEMS. (2023). Python for FEA: Automation and Optimization. NAFEMS e-Learning Course. <https://www.nafems.org>
- [16] Dassault Systèmes SIMULIA. (2024). Abaqus Scripting User's Guide. Abaqus 2024 Documentation.
- [17] Phuor et al, I., et al. (2025). Computational investigation of long free-span submarine pipelines with buoyancy modules using an automated Python–Abaqus framework. *Mathematics*, 13(9), 1387. <https://doi.org/10.3390/math13091387>
- [18] Geuzaine, C., & Remacle, J.-F. (2009). Gmsh: A 3-D finite element mesh generator with built-in pre- and post-processing facilities. *International Journal for Numerical Methods in Engineering*, 79(11), 1309–1331. <https://doi.org/10.1002/nme.2579>
- [19] Pallets Projects. (2026). Jinja Documentation. <https://jinja.palletsprojects.com/>
- [20] EDF R&D. (2024). Code_Aster: Analyses des Structures et Thermomécanique pour des Études et des Recherches (v16.x). <https://www.code-aster.org>
- [21] Tsvetkova, T., & Kerenchev, N. (2020). Review of open source software for modelling and analysis of structures. *IOP Conference Series: Materials Science and Engineering*, 951, 012003. <https://doi.org/10.1088/1757-899X/951/1/012003>
- [22] Nath, D., Neog, A. R., & Gautam, S. S. (2024). Application of machine learning and deep learning in finite element analysis: A comprehensive review. *Archives of Computational Methods in Engineering*, 31, 2945–2984. <https://doi.org/10.1007/s11831-024-10063-0>
- [23] Raissi, M., Perdikaris, P., & Karniadakis, G. E. (2019). Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational Physics*, 378, 686–707. <https://doi.org/10.1016/j.jcp.2018.10.045>
- [24] Karniadakis, G. E., Kevrekidis, I. G., Lu, L., Perdikaris, P., Wang, S., & Yang, L. (2021). Physics-informed machine learning. *Nature Reviews Physics*, 3(6), 422–440. <https://doi.org/10.1038/s42254-021-00314-5>
- [25] Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., ... Amodei, D. (2020). Language models are few-shot learners. *Advances in Neural Information Processing Systems*, 33, 1877–1901.
- [26] OpenAI. (2023). GPT-4 Technical Report. arXiv:2303.08774. <https://doi.org/10.48550/arXiv.2303.08774>
- [27] Schick, T., Dwivedi-Yu, J., Dessì, R., Raileanu, R., Lomeli, M., Zettlemoyer, L., Cancedda, N., & Scialom, T. (2023). Toolformer: Language models can teach themselves to use tools. *Advances in Neural Information Processing Systems*, 36. arXiv:2302.04761

- [28] Yao, S., Zhao, J., Yu, D., Du, N., Shafran, I., Narasimhan, K., & Cao, Y. (2023). ReAct: Synergizing reasoning and acting in language models. *International Conference on Learning Representations (ICLR 2023)*. arXiv:2210.03629
- [29] LangChain Inc. (2023). LangChain: Open-source framework for building LLM-powered applications (v0.3.x). <https://www.langchain.com>
- [30] Ni, B., & Buehler, M. J. (2024). MechAgents: Large language model multi-agent collaborations can solve mechanics problems, generate new data, and integrate knowledge. *Extreme Mechanics Letters*, 67, 102131. <https://doi.org/10.1016/j.eml.2024.102131>
- [31] Qi, Y., Xu, R., & Chu, X. (2025). FeaGPT: An end-to-end agentic-AI for finite element analysis. arXiv:2510.21993. <https://arxiv.org/abs/2510.21993>
- [32] Deotale, R., et al. (2026).). ALL-FEM: Agentic large language models fine-tuned for finite element methods. ResearchGate preprint. <https://www.researchgate.net/publication/400381610>
- [33] Oberkampf, W. L., & Trucano, T. G. (2002). Verification and validation in computational fluid dynamics. *Progress in Aerospace Sciences*, 38(3), 209–272. [https://doi.org/10.1016/S0376-0421\(02\)00005-2](https://doi.org/10.1016/S0376-0421(02)00005-2)
- [34] Roache, P. J. (1998). *Verification and Validation in Computational Science and Engineering*. Hermosa Publishers.
- [35] Timoshenko, S. P., & Gere, J. M. (1972). *Mechanics of Materials*. Van Nostrand Reinhold.
- [36] Roy, C. J., & Oberkampf, W. L. (2011). A comprehensive framework for verification, validation, and uncertainty quantification in scientific computing. *Computer Methods in Applied Mechanics and Engineering*, 200(25–28), 2131–2144. <https://doi.org/10.1016/j.cma.2011.03.016>
- [37] American Society of Mechanical Engineers. (2006). *Guide for Verification and Validation in Computational Solid Mechanics (ASME V&V 10-2006)*. ASME.

APPENDIX I: CASE STUDY 1 – CANTILEVER BEAM WITH TIP LOAD

This appendix presents the input files and the corresponding numerical results produced by Code_Aster for the analysis of a cantilever beam subjected to a concentrated tip load.

.geo file:

```
// ---- 1D Beam geometry (OpenCASCADE) ----
SetFactory("OpenCASCADE");

// Mesh settings to preserve points as nodes in MED
Mesh.MedFileMinorVersion = 0;
Mesh.MeshSizeFromPoints = 1;
Mesh.MeshSizeFromCurvature = 0;
Mesh.MeshSizeExtendFromBoundary = 0;

// Parameters
// Length = 4000.0 (mm)
Point(1) = {0, 0, 0};
Point(2) = {4000.0, 0, 0};
Line(1) = {1, 2};

// Transfinite Line definition
// n_nodes = n_elems + 1 = 17
Transfinite Line {1} = 17;

// Physical groups
Physical Point("BASE") = {1};
Physical Point("TIP") = {2};
Physical Curve("BEAM") = {1};
```

.med file is a binary file that can't be represented here.

.comm file:

```
# _____ Start _____  
DEBUT(PAR_LOT='NON');  
  
# _____ Read the mesh _____  
mesh = LIRE_MALLAGE(FORMAT='MED', UNITE=20);  
  
# _____ Define node groups (BASE, TIP, BEAM) _____  
mesh = DEFI_GROUP(  
  reuse=mesh,  
  CREA_GROUP_NO=(  
    _F(GROUP_MA=('BEAM',), CRIT_NOEUD='TOUS'),  
    _F(GROUP_MA=('BASE',), CRIT_NOEUD='TOUS'),  
    _F(GROUP_MA=('TIP',), CRIT_NOEUD='TOUS'),  
  ),  
  MAILLAGE=mesh  
);  
  
# _____ Define mechanical model _____  
model = AFFE_MODELE(  
  MAILLAGE=mesh,  
  AFFE=_F(  
    GROUP_MA=('BEAM',),  
    PHENOMENE='MECANIQUE',  
    MODELISATION='POU_D_E'  
  )  
);  
  
# _____ Define material _____  
mat = DEFI_MATERIAU(ELAS=_F(E=210000.0, NU=0.3, RHO=7850.0));  
  
mater = AFFE_MATERIAU(  
  MAILLAGE=mesh,  
  AFFE=_F(GROUP_MA=('BEAM',), MATER=mat)  
);  
  
# _____ Define beam section geometry (GENERALE, mm) _____
```

```

cara = AFFE_CARA_ELEM(
  MODELE=model,
  POUTRE=_F(
    GROUP_MA=( 'BEAM', ),
    SECTION='GENERALE',
    CARA=( 'A', 'IY', 'IZ', 'JX', 'AY', 'AZ' ),
    VALE=(2009.0, 8693000.0, 8693000.0, 35300.0, 1214.0, 966.0),
  ),
);

```

```

# _____ Boundary condition (BASE) _____
constr = AFFE_CHAR_MECA(
  MODELE=model,
  DDL_IMPO=_F(
    GROUP_NO=( 'BASE', ),
    DX=0.0, DY=0.0, DZ=0.0,
    DRX=0.0, DRY=0.0, DRZ=0.0
  )
);

```

```

# _____ Applied load on TIP _____
Load = AFFE_CHAR_MECA(
  MODELE=model,
  FORCE_NODALE=_F(
    GROUP_NO=( 'TIP', ),
    FY=-1500.0
  )
);

```

```

# _____ Static analysis _____
res_stat = MECA_STATIQUE(
  MODELE=model,
  CHAM_MATER=mater,
  CARA_ELEM=cara,
  EXCIT=(
    _F(CHARGE=constr),
    _F(CHARGE=Load)
  ),
);

```

```

SOLVEUR=_F(
    METHODE='MUMPS',
    STOP_SINGULIER='OUI',
    RENUM='METIS',
    NPREC=8,
    ELIM_LAGR='NON'
)
);

# _____ Compute fields: reactions, displacements, stresses _____
res_stat = CALC_CHAMP(
    reuse=res_stat,
    RESULTAT=res_stat,
    FORCE=('REAC_NODA',),
    CONTRAINTE=('EFGE_ELNO', 'SIEF_ELNO')
);

# _____ Create result tables (reactions, displacements) _____
tab_reac = CREA_TABLE(
    RESU=_F(
        GROUP_NO=('BASE',),
        NOM_CHAM='REAC_NODA',
        RESULTAT=res_stat,
        TOUT_CMP='OUI'
    ),
    TITRE='Nodal Reactions'
);

tab_disp = CREA_TABLE(
    RESU=_F(
        TOUT='OUI',
        NOM_CHAM='DEPL',
        RESULTAT=res_stat,
        TOUT_CMP='OUI'
    ),
    TITRE='Nodal Displacements'
);

```

```

# _____ Print tables to text _____
IMPR_TABLE(TABLE=tab_reac, UNITE=18, SEPARATEUR=' ', FORMAT='TABLEAU', FORMAT_R='E12.5');
IMPR_TABLE(TABLE=tab_disp, UNITE=19, SEPARATEUR=' ', FORMAT='TABLEAU', FORMAT_R='E12.5');

# _____ Print displacement field _____
IMPR_RESU(
    FORMAT='RESULTAT',
    UNITE=80,
    RESU=_F(RESULTAT=res_stat, NOM_CHAM='DEPL')
);

# _____ Print local coordinate systems _____
IMPR_CONCEPT(
    CONCEPT=_F(CARA_ELEM=cara, MODELE=model, REPERE_LOCAL='ELEM'),
    FORMAT='RESULTAT',
    UNITE=53
);

# _____ Internal forces table _____
SIEFTAB = POST_RELEVE_T(
    ACTION=_F(
        INTITULE='BeamForces',
        RESULTAT=res_stat,
        NOM_CHAM='SIEF_ELNO',
        GROUP_NO=('BEAM',),
        OPERATION='EXTRACTION',
        TOUT_ORDRE='OUI',
        TOUT_CMP='OUI'
    )
);

IMPR_TABLE(TABLE=SIEFTAB, UNITE=21, SEPARATEUR=';', FORMAT='TABLEAU', FORMAT_R='E12.5');

# _____ End _____
FIN();

```

.export file:

```

P actions make_etude
P debug nodebug
P memjob 4194304
P memory_limit 4096.0
P mode interactif
P mpi_nbcpu 1
P mpi_nbnoeud 1
P ncpus 1
P nomjob beam_job
P origine CodeAster_Agent
P rep_trav C:\Users\X\AppData\Local\Temp\X-PC-SONIA-interactif_2073
P server localhost
P time_limit 1800.0
P tpsjob 31
P version stable
A memjeveux 512.0
A tpmx 1800.0
F comm C:\Users\X\workspace\runs\20260515_123207_beam_23448fec\inputs\model.comm D 1
F libr
C:\Users\Michele\training\workspace\runs\20260515_123207_beam_23448fec\inputs\model.med D
20
F mess C:\Users\X\workspace\runs\20260515_123207_beam_23448fec\outputs\beam_job.mess R 6
F resu C:\Users\X\workspace\runs\20260515_123207_beam_23448fec\outputs\beam_job.resu R 80
F libr C:\Users\X\workspace\runs\20260515_123207_beam_23448fec\outputs\tab_reac.txt R 18
F libr C:\Users\X\workspace\runs\20260515_123207_beam_23448fec\outputs\tab_disp.txt R 19
F libr C:\Users\X\workspace\runs\20260515_123207_beam_23448fec\outputs\local_axes.txt R 53
F libr C:\Users\X\workspace\runs\20260515_123207_beam_23448fec\outputs\tab_sie.txt R 21
R base C:\Users\X\workspace\runs\20260515_123207_beam_23448fec\outputs\beam_job_base R 0

```

The following output files were generated after executing the numerical analysis in Code_Aster. These files contain the computed structural response, including displacement, internal forces, and post-processing data associated with the cantilever beam model.

.mess file:

```

# -----
-

```

Ouverture en écriture du fichier ./vola.1

<INFO> Démarrage de l'exécution.

```

  _ _ _ _ _
 / _ | _ | | _ / \ _ | | _ _ _ \ \ / ( ) _ _ | | _ _ _
 | ( _ _ V _ ` / - ) / _ \ ( - < _ / - ) ' _ | \ V V / | | ' V _ ` / _ \ V V ( _ - <
 \ _ \ / \ , _ \ | / / \ V _ / \ _ \ | | \ \ / | _ | | \ _ , _ \ / \ / \ / \ /
                                     https://code-aster-windows.com

```

-- CODE_ASTER -- VERSION: EXPLOITATION (stable) --

Modified 14.4.0 version 28/06/2019

the revision 952c0751ea6d - branch v14

Copyright EDF R & D 1991 - 2026

Exécution of: Fri May 15 14:32:09 2026

Name of the machine: PC-SONIA

Structure: 64bit

CPU

type: AMD64

Operating system: Windows-10-10.0.19041-SP0

Language of the messages: en (cp1252)

Python version: 3.7.2

NumPy version: 1.21.6

```

!-----!
! <A> <SUPERVIS2_2> !
! ! !
! You use an old version of Code_Aster. !
! ! !
! By upgrading your version, you will profit from the last improvements !
! made to the code !
! since 15 month. !
! If you have private developments, you are likely to have an important !

```

```

! work of bearing if you do not follow the
! upgraded ones.
!
!
! This is a warning. If you do not understand the meaning of this
! warning, you can obtain unexpected results!
!-----!

```

```

Parallelism MPI: idle
OpenMP parallelism: Nombre
credit of processes used: 1
Version of the HDF5 Library : 1.10.3
Version of the MED Library : 4.0.0
Librairie MFront : non disponible
MUMPS Library version: 5.1.2
PETSc Library: not available
Version of the SCOTCH Library : 6.0.4

```

```

Mémoire limite pour l'exécution : 4096.00 Mo
consommée par l'initialisation : 45.84 Mo

```

```

par les objets du jeu de commandes : 0.00 Mo
reste pour l'allocation dynamique : 4045.78 Mo
Cut limiting files of exchange: 48.00 Go

```

```

.. __stg1_txt3

```

```

# -----
---
# Commands No: 0001 Concept of the type: -
# -----
---

```

```

DEBUT(PAR_LOT='NON',
      IMPR_MACRO='NON',
      DEBUG=_F(JXVERI='NON',
               SDVERI='NON',
               JEVEUX='NON',
               HIST_ETAPE='NON',),

```

[illegible]

91

Initial value of maximum time CPU = 1800 second

Valeur of the maximum time CPU placed to the orders = 1620 second

Réserve CPU

envisaged = 180 seconds

Ouverture en écriture du fichier ./glob.1

Ouverture en écriture du fichier ./vola.1

Ouverture en lecture du fichier C:\STADATA\CodeAster19\v2019\14.4\Lib\aster\elem.1

Nom de la base : ELEMBASE

Créée avec la version : 14.04.00

Nombre

d'enregistrements utilisés : 43

Nombre d'enregistrements maximum : 512

Longueur d'enregistrement (octets)

: 819200

Nombre d'identificateurs utilisés : 123

Taille maximum du répertoire : 300

Pourcentage d'utilisation du répertoire : 41 %

Ouverture en lecture du fichier C:\STADATA\CodeAster19\v2019\14.4\Lib\aster\elem.1

Nom de la base : ELEMBASE

Nombre d'enregistrements utilisés : 43

Nombre

d'enregistrements maximum : 512

Longueur d'enregistrement (octets) : 819200

Nombre total d'accès en lecture

: 60

Volume des accès en lecture : 46.88 Mo.

Nombre total d'accès en écriture : 0

Volume des accès en écriture : 0.00 Mo.

Nombre d'identificateurs utilisés : 123

Taille maximum
du répertoire : 300
Pourcentage d'utilisation du répertoire : 41 %

Second reading of the catalogues of the elements made.

End of reading (lasted 0.000000 S.)

Memory (Mo): 96.51 / 88.75 / 44.30 / 22.92 (VmPeak / VmSize / Optimum / Minimum)

Fin orders No: 0001 user+syst: 0.05S (syst: 0.05S, Elaps: 0.00S)

.. __stg1_txt6

Commands No: 0002 Concept of the type: maillage_sdaster


```
mesh = LIRE_MAILLAGE(FORMAT='MED',
                     UNITE=20,
                     INFO_MED=1,
                     VERI_MAIL=_F(VERIF='OUI',
                                   APLAT=1.E-3, ),
                     INFO=1, )
```

Vérification du maillage.

```
!-----!
! <A> <MODELISA8_14> !
! ! !
! La maille POI1 M1 est incluse dans une autre. !
! ! !
! ! !
! This is a warning. If you do not understand the meaning of this !
! warning, you can obtain unexpected results! !
!-----!
```

```

!-----!
! <A> <MODELISA8_14> !
! !
! La maille POI1 M2 est incluse dans une autre. !
! !
! !
! This is a warning. If you do not understand the meaning of this !
! warning, you can obtain unexpected results! !
!-----!

```

----- MAILLAGE mesh - IMPRESSIONS NIVEAU 1 -----

MED file generated by Gmsh

NOMBRE DE NOEUDS 17

NOMBRE DE MAILLES 18

POI1	2
SEG2	16

NOMBRE DE GROUPES DE MAILLES 3

BASE	1
TIP	1
BEAM	16

Memory (Mo): 96.51 / 91.35 / 44.30 / 22.92 (VmPeak / VmSize / Optimum / Minimum)

Fin orders No: 0002 user+syst: 0.00S (syst: 0.00S, Elaps: 0.00S)

.. __stg1_txt16

Commands No: 0004 Concept of the type: maillage_sdaster

```
# -----
---
mesh = DEFI_GROUP(reuse=mesh,
                  MAILLAGE=mesh,
                  CREA_GROUP_NO=(_F(GROUP_MA=('BEAM', ),
                                     CRIT_NOEUD='TOUS', ),
                                _F(GROUP_MA=('BASE', ),
                                     CRIT_NOEUD='TOUS', ),
                                _F(GROUP_MA=('TIP', ),
                                     CRIT_NOEUD='TOUS', ),
                                ),
                  ALARME='OUI',)
```

NOMBRE DE GROUPES DE NOEUDS CREES : 3

=====

```
-----
!          NOM DU GROUPE          ! NBRE DE NOEUDS DU !
!          NOEUDS                  !   GROUPE_NO      !
-----
! BEAM                             !      17          !
! BASE                             !      1           !
! TIP                              !      1           !
-----
```

Memory (Mo): 96.51 / 91.48 / 44.30 / 22.92 (VmPeak / VmSize / Optimum / Minimum)

Fin orders No: 0004 user+syst: 0.02S (syst: 0.00S, Elaps: 0.00S)

```
# -----
---
```

.. __stg1_txt25

```
# -----
---
```

Commands No: 0005 Concept of the type: modele_sdaster

```
# -----
---
```

```

model = AFFE_MODELE(MAILLAGE=mesh,
                    AFPE=_F(GROUP_MA=('BEAM', ),
                            PHENOMENE='MECANIQUE',
                            MODELISATION=('POU_D_E', ),),
                    INFO=1,
                    DISTRIBUTION=_F(PARTITIONNEUR='METIS',
                                    METHODE='SOUS_DOMAINE',),
                    VERI_JACOBIEN='OUI',)

```

Sur les 18 mailles du maillage mesh, on a demandé l'affectation de 16, on a pu en affecter 16

Modélisation	Type maille	Élément fini	Nombre
POU_D_E	SEG2	MECA_POU_D_E	16

#2 Calculs elementaires et assemblages CPU (USER+SYST/SYST/ELAPS):
0.00 0.00 0.00

Memory (Mo): 96.51 / 92.89 / 44.30 / 33.82 (VmPeak / VmSize / Optimum / Minimum)

Fin orders No: 0005 user+syst: 0.00S (syst: 0.00S, Elaps: 0.00S)

```

# -----
---
```

.. __stg1_txt30

```

# -----
---
```

Commands No: 0006 Concept of the type: mater_sdaster

```

# -----
---
```

```

mat = DEFI_MATERIAU(ELAS=_F(E=2.10000E5,
                              NU=0.3,
                              RHO=7850.0,
                              COEF_AMOR=1.0,)),
                    INFO=1,)

```

Propriétés définies dans le matériau: ELAS.

# Memory (Mo):	96.51 /	93.00 /	44.30 /	33.82 (VmPeak / VmSize / Optimum / Minimum)
# Fin orders No: 0006 user+syst:	0.00S (syst:	0.00S, Elaps:	0.00S)	

```

# -----
---
```

```

.. __stg1_txt34
# -----
---
# Commands No: 0007 Concept of the type: cham_mater
# -----
---
mater = AFFE_MATERIAU(MAILLAGE=mesh,
                      AFFE=_F(GROUP_MA=('BEAM', ),
                              MATER=(mat, ),),
                      INFO=1,)

# Memory (Mo):      96.51 /      93.07 /      44.30 /      33.82 (VmPeak / VmSize / Optimum /
Minimum)

# Fin orders No: 0007 user+syst:      0.00S (syst:      0.00S, Elaps:
0.00S)
# -----
---

.. __stg1_txt44
# -----
---
# Commands No: 0008 Concept of the type: cara_elem
# -----
---
cara = AFFE_CARA_ELEM(MODELE=model,
                      POUTRE=_F(SECTION='GENERALE',
                                GROUP_MA=('BEAM', ),
                                CARA=('A', 'IY', 'IZ', 'JX', 'AY', 'AZ'),
                                VALE=(2009.0, 8.693000E6, 8.693000E6, 3.5300E4, 1214.0,
966.0),
                                VARI_SECT='CONSTANT',),
                      INFO=1,)

LE MODELE model    CONTIENT :

                                16 ELEMENT(S) POUTRE
#2      Calculs elementaires et assemblages      CPU (USER+SYST/SYST/ELAPS):
0.00      0.00      0.00

# Memory (Mo):      96.51 /      93.66 /      44.30 /      33.82 (VmPeak / VmSize / Optimum /
Minimum)

```

```

# Fin orders No: 0008 user+syst:      0.02S (syst:      0.00S, Elaps:
0.00S)

# -----
---

.. __stg1_txt54

# -----
---

# Commands No: 0010 Concept of the type: char_meca

# -----
---

constr = AFFE_CHAR_MECA(MODELE=model,
                        DDL_IMPO=_F(GROUP_NO=( 'BASE', ),
                                     DX=0.0,
                                     DY=0.0,
                                     DZ=0.0,
                                     DRX=0.0,
                                     DRY=0.0,
                                     DRZ=0.0, ),
                        VERI_NORM='OUI',
                        DOUBLE_LAGRANGE='OUI',
                        INFO=1, )

# Memory (Mo):      96.51 /      94.21 /      44.30 /      33.82 (VmPeak / VmSize / Optimum /
Minimum)

# Fin orders No: 0010 user+syst:      0.00S (syst:      0.00S, Elaps:
0.00S)

# -----
---

.. __stg1_txt63

# -----
---

# Commands No: 0011 Concept of the type: char_meca

# -----
---

Load = AFFE_CHAR_MECA(MODELE=model,
                      FORCE_NODALE=_F(GROUP_NO=( 'TIP', ),
                                       FY=-1500.0, ),
                      VERI_NORM='OUI',
                      DOUBLE_LAGRANGE='OUI',

```

```

INFO=1, )

# Memory (Mo):      96.51 /      94.23 /      44.30 /      33.82 (VmPeak / VmSize / Optimum /
Minimum)

# Fin orders No:  0011 user+syst:      0.00S (syst:      0.00S, Elaps:
0.00S)

# -----
---

.. __stg1_txt81

# -----
---

# Commands No:  0012 Concept of the type:  evol_elas

# -----
---

res_stat = MECA_STATIQUE(MODELE=model,
                          CHAM_MATER=mater,
                          CARA_ELEM=cara,
                          EXCIT=( _F(CHARGE=constr,
                                     TYPE_CHARGE='FIXE_CSTE', ),
                                _F(CHARGE=load,
                                     TYPE_CHARGE='FIXE_CSTE', ),
                                ),
                          SOLVEUR=_F(METHODE='MUMPS',
                                     RENUM='METIS',
                                     NPREC=8,
                                     ELIM_LAGR='NON',
                                     STOP_SINGULIER='OUI',
                                     TYPE_RESOL='AUTO',
                                     ACCELERATION='AUTO',
                                     LOW_RANK_SEUIL=0.0,
                                     PRETRAITEMENTS='AUTO',
                                     POSTTRAITEMENTS='AUTO',
                                     PCENT_PIVOT=20,
                                     RESI_RELA=1.E-06,
                                     GESTION_MEMOIRE='AUTO',
                                     MATR_DISTRIBUEE='NON', ),
                          OPTION='SIEF_ELGA',
                          INFO=1, )

```

Le système linéaire à résoudre a 114 degrés de liberté:

- 102 sont des degrés de liberté physiques

(ils sont portés par 17

noeuds du maillage)

- 12 sont les couples de paramètres de Lagrange associés

aux 6 relations linéaires dualisées.

La matrice est de taille 114 équations.

Elle contient 1023 termes non nuls si elle est symétrique et 1932 termes non nuls si elle n'est pas symétrique.

Soit un taux de remplissage de 14.866 %.

Field stored DEPL at time 0.000000000000e+00 for the sequence number 1

#1 Resolution des systemes lineaires CPU (USER+SYST/SYST/ELAPS):
0.03 0.00 0.00

#2 Calculs elementaires et assemblages CPU (USER+SYST/SYST/ELAPS):
0.00 0.00 0.00

Memory (Mo): 100.44 / 100.43 / 44.30 / 33.82 (VmPeak / VmSize / Optimum / Minimum)

Fin orders No: 0012 user+syst: 0.03S (syst: 0.02S, Elaps: 0.00S)

.. __stg1_txt90

Commands No: 0013 Concept of the type: evol_elas


```
res_stat = CALC_CHAMP(reuse=res_stat,  
                      RESULTAT=res_stat,  
                      PRECISION=1.E-06,  
                      CONTRAINTE=('EFGE_ELNO', 'SIEF_ELNO'),  
                      FORCE=('REAC_NODA', ),  
                      CRITERE='RELATIF',  
                      INFO=1,)
```

Le système linéaire à résoudre a 114 degrés de liberté:

- 102 sont des degrés de liberté physiques

(ils sont portés par 17

noeuds du maillage)

- 12 sont les couples de paramètres de Lagrange associés

```

    aux 6 relations linéaires dualisées.

#2      Calculs elementaires et assemblages      CPU (USER+SYST/SYST/ELAPS):
0.00      0.00      0.00

# Memory (Mo):      100.73 /      100.72 /      44.30 /      33.95 (VmPeak / VmSize / Optimum /
Minimum)

# Fin orders No: 0013 user+syst:      0.02S (syst:      0.00S, Elaps:
0.00S)

# -----
---

.. __stg1_txt101

# -----
---

# Commands No: 0014 Concept of the type: table_sdaster

# -----
---

tab_reac = CREA_TABLE(TITRE='Nodal Reactions',
                      RESU=_F(RESULTAT=res_stat,
                              TOUT_CMP='OUI',
                              GROUP_NO=( 'BASE', ),
                              NOM_CHAM='REAC_NODA',
                              PRECISION=1.E-06,
                              CRITERE='RELATIF',),
                      TYPE_TABLE='TABLE',)

# Memory (Mo):      100.80 /      100.80 /      44.30 /      33.95 (VmPeak / VmSize / Optimum /
Minimum)

# Fin orders No: 0014 user+syst:      0.00S (syst:      0.00S, Elaps:
0.00S)

# -----
---

.. __stg1_txt111

# -----
---

# Commands No: 0015 Concept of the type: table_sdaster

# -----
---

tab_disp = CREA_TABLE(TITRE='Nodal Displacements',
                      RESU=_F(RESULTAT=res_stat,
                              TOUT_CMP='OUI',
                              TOUT='OUI',

```

```

        NOM_CHAM='DEPL',
        PRECISION=1.E-06,
        CRITERE='RELATIF',),
TYPE_TABLE='TABLE',)

# Memory (Mo):    100.80 /    100.80 /    44.30 /    33.95 (VmPeak / VmSize / Optimum /
Minimum)

# Fin orders No:  0015 user+syst:          0.00S (syst:          0.00S, Elaps:
0.00S)

# -----
---

.. __stg1_txt115

# -----
---

# Commands No:  0016 Concept of the type:  -
# -----
---

IMPR_TABLE(TABLE=tab_reac,
            FORMAT='TABLEAU',
            FORMAT_R='E12.5',
            UNITE=18,
            SEPARATEUR=' ',
            FIN_TABLE='',
            COMMENTAIRE='#',
            COMM_PARA='',
            DEBUT_LIGNE='',
            FIN_LIGNE='\n',
            IMPR_FONCTION='NON',
            INFO=1,)

# Memory (Mo):    101.09 /    101.09 /    44.30 /    33.95 (VmPeak / VmSize / Optimum /
Minimum)

# Fin orders No:  0016 user+syst:          0.02S (syst:          0.00S, Elaps:
0.00S)

# -----
---

.. __stg1_txt116

# -----
---

```

```

# Commands No: 0019 Concept of the type: -
# -----
---

IMPR_TABLE(TABLE=tab_disp,
            FORMAT='TABLEAU',
            FORMAT_R='E12.5',
            UNITE=19,
            SEPARATEUR=' ',
            FIN_TABLE='',
            COMMENTAIRE='#',
            COMM_PARA='',
            DEBUT_LIGNE='',
            FIN_LIGNE='\n',
            IMPR_FONCTION='NON',
            INFO=1,)

# Memory (Mo):    101.09 /    101.09 /    44.30 /    33.95 (VmPeak / VmSize / Optimum /
Minimum)

# Fin orders No: 0019 user+syst:          0.00S (syst:          0.00S, Elaps:
0.00S)

# -----
---

.. __stg1_txt122

# -----
---

# Commands No: 0022 Concept of the type: -
# -----
---

IMPR_RESU(FORMAT='RESULTAT',
          UNITE=80,
          RESU=_F(RESULTAT=res_stat,
                  NOM_CHAM=('DEPL', ),
                  FORM_TABL='NON',
                  IMPR_COOR='NON',
                  FORMAT_R='1PE21.14',),
          PROC0='OUI',
          INFO=1,)

# Memory (Mo):    101.29 /    101.29 /    44.30 /    33.95 (VmPeak / VmSize / Optimum /
Minimum)

```

Fin orders No: 0022 user+syst: 0.025 (syst: 0.005, Elaps: 0.005)

.. __stg1_txt129

Commands No: 0023 Concept of the type: -

IMPR_CONCEPT(FORMAT='RESULTAT',
UNITE=53,
CONCEPT=_F(CARA_ELEM=cara,
REPERE_LOCAL='ELEM',
MODELE=model,),
INFO=1,)

IMPRESSION D'UN CHAMP DE CONCEPT : Caracteristiques generales des poutres

IMPRESSION D'UN CHAMP DE CONCEPT : Caracteristiques generales des poutres

NOM DU CHAMP : cara _CARGENPO

NOM DU CHAMP : cara _CARGENPO

CORRESPONDANCE VALEUR <-> CONTENU :

CORRESPONDANCE VALEUR <-> CONTENU :

VALEUR = 1.

VALEUR = 1.

VALEUR = 2.

VALEUR = 2.

A1 = 2.009D+03

A1 = 2.009D+03

IY1 = 8.693D+06

IY1 = 8.693D+06

IZ1 = 8.693D+06

IZ1 = 8.693D+06

AY1 = 0.000D+00

AY1 = 0.000D+00

AZ1 = 0.000D+00

AZ1	=	0.000D+00
EY1	=	0.000D+00
EY1	=	0.000D+00
EZ1	=	0.000D+00
EZ1	=	0.000D+00
JX1	=	3.530D+04
JX1	=	3.530D+04
RY1	=	1.000D+00
RY1	=	1.000D+00
RZ1	=	1.000D+00
RZ1	=	1.000D+00
RT1	=	1.000D+00
RT1	=	1.000D+00
AI1	=	0.000D+00
AI1	=	0.000D+00
JG1	=	0.000D+00
JG1	=	0.000D+00
IYR21	=	0.000D+00
IYR21	=	0.000D+00
IZR21	=	0.000D+00
IZR21	=	0.000D+00
A2	=	2.009D+03
A2	=	2.009D+03
IY2	=	8.693D+06
IY2	=	8.693D+06
IZ2	=	8.693D+06
IZ2	=	8.693D+06
AY2	=	0.000D+00
AY2	=	0.000D+00
AZ2	=	0.000D+00
AZ2	=	0.000D+00
EY2	=	0.000D+00
EY2	=	0.000D+00
EZ2	=	0.000D+00
EZ2	=	0.000D+00
JX2	=	3.530D+04
JX2	=	3.530D+04
RY2	=	1.000D+00

RY2 = 1.000D+00
 RZ2 = 1.000D+00
 RZ2 = 1.000D+00
 RT2 = 1.000D+00
 RT2 = 1.000D+00
 AI2 = 0.000D+00
 AI2 = 0.000D+00
 JG2 = 0.000D+00
 JG2 = 0.000D+00
 IYR22 = 0.000D+00
 IYR22 = 0.000D+00
 IZR22 = 0.000D+00
 IZR22 = 0.000D+00
 TVAR = 0.000D+00
 TVAR = 0.000D+00

IMPRESSION D'UN CHAMP DE CONCEPT : Caracteristiques geom. des poutres

IMPRESSION D'UN CHAMP DE CONCEPT : Caracteristiques geom. des poutres

NOM DU CHAMP : cara _CARGEOPO

NOM DU CHAMP : cara _CARGEOPO

CORRESPONDANCE VALEUR <-> CONTENU :

CORRESPONDANCE VALEUR <-> CONTENU :

VALEUR = 1.

VALEUR = 1.

IMPRESSION D'UN CHAMP DE CONCEPT : Orientation des coques, poutres, barres, cables

IMPRESSION D'UN CHAMP DE CONCEPT : Orientation des coques, poutres, barres, cables

NOM DU CHAMP : cara _CARORIEN

NOM DU CHAMP : cara _CARORIEN

CORRESPONDANCE VALEUR <-> CONTENU :

CORRESPONDANCE VALEUR <-> CONTENU :

VALEUR = 1.

VALEUR = 1.

ALPHA = 0.000D+00

ALPHA = 0.000D+00

BETA = 0.000D+00

```

    BETA      =    0.000D+00

    GAMMA     =    0.000D+00

    GAMMA     =    0.000D+00

#2      Calculs elementaires et assemblages          CPU (USER+SYST/SYST/ELAPS):
0.00      0.00      0.00

    # Memory (Mo):    101.48 /    101.47 /    44.30 /    33.95 (VmPeak / VmSize / Optimum /
Minimum)

    # Fin orders No:  0023 user+syst:          0.00S (syst:          0.00S, Elaps:
0.00S)

    # -----
    ---

.. __stg1_txt141

    # -----
    ---

    # Commands No:  0024 Concept of the type:  table_sdaster

    # -----
    ---

    SIEFTAB = POST_RELEVE_T(ACTION=_F(OPERATION=('EXTRACTION', ),
                                     INTITULE='BeamForces',
                                     RESULTAT=res_stat,
                                     NOM_CHAM='SIEF_ELNO',
                                     TOUT_ORDRE='OUI',
                                     PRECISION=1.E-06,
                                     CRITERE='RELATIF',
                                     GROUP_NO=('BEAM', ),
                                     TOUT_CMP='OUI',
                                     REPERE='GLOBAL',
                                     MOYE_NOEUD='OUI',
                                     FORMAT_C='MODULE', ),
                             INFO=1,)

    # Memory (Mo):    101.79 /    101.78 /    44.30 /    33.95 (VmPeak / VmSize / Optimum /
Minimum)

    # Fin orders No:  0024 user+syst:          0.02S (syst:          0.00S, Elaps:
0.00S)

    # -----
    ---

.. __stg1_txt145

    # -----
    ---

```

```

# Commands No: 0025 Concept of the type: -
# -----
---
IMPR_TABLE(TABLE=SIEFTAB,
            FORMAT='TABLEAU',
            FORMAT_R='E12.5',
            UNITE=21,
            SEPARATEUR=';',
            FIN_TABLE='',
            COMMENTAIRE='#',
            COMM_PARA='',
            DEBUT_LIGNE='',
            FIN_LIGNE='\n',
            IMPR_FONCTION='NON',
            INFO=1,)

# Memory (Mo):    101.79 /    101.79 /    44.30 /    33.95 (VmPeak / VmSize / Optimum /
Minimum)

# Fin orders No: 0025 user+syst:    0.00S (syst:    0.00S, Elaps:
0.00S)
# -----
---

.. __stg1_txt148
# -----
---

# Commands No: 0028 Concept of the type: -
# -----
---
FIN(FORMAT_HDF='NON',
    RETASSAGE='NON',
    INFO_RESU='OUI',
    PROC0='OUI',
    )

!-----!
! <I> <CATAMESS_89> !
! !
! List of warnings emitted during the execution of computation. !

```

```

!
! Warnings which you chose to ignore of are preceded by (*).
!
! Number of occurrences for each warning:
!
!          MODELISA8_14          emitted    2 time
!          SUPERVIS2_2           emitted    1 time
!          SUPERVIS_1            emitted    1 time
!-----!

```

=====>

STRUCTURE DU CONCEPT *res_stat* CALCULE POUR 1 NUMERO D'ORDRE

LISTE DES NOMS SYMBOLIQUES:

```

! ----- !-----!-----!-----!-----!-----
-----!
! NUME_ORDRE !      DEPL      !  EFGE_ELNO  !  SIEF_ELGA  !  SIEF_ELNO  !
REAC_NODA    !
! ----- !-----!-----!-----!-----!-----
-----!
!          1 !      DEPL_R      !      SIEF_R      !      SIEF_R      !      SIEF_R      !
DEPL_R      !
! ----- !-----!-----!-----!-----!-----
-----!

```

LISTE DES NOMS DE VARIABLES D'ACCES:

INST DE TYPE R

LISTE DES NOMS DE PARAMETRES:

```

! ----- !-----!-----!-----!-----!-----
-----!-----!-----!-----!-----!
! NUME_ORDRE !      CARAELEM      !      CHAMPMAT      !      EXCIT      !      ITER_GCPC      !
METHODE      !      MODELE      !      RENUM      !      STOCKAGE      !
! ----- !-----!-----!-----!-----!-----
-----!-----!-----!-----!
!          1 !      K8      !      K8      !      K24      !      I      !
K16          !      K8      !      K16      !      K16      !
! ----- !-----!-----!-----!-----!-----
-----!-----!-----!-----!

```

Concepts de La base: G

<i>Nom</i>	<i>Type</i>	<i>Taille (Mo)</i>	<i>Nombre d'objets</i>	<i>Nombre de segments</i>
<i>TOTAL</i>		<i>27.36</i>	<i>296</i>	<i>463</i>
<i>mesh</i>	<i>MAILLAGE_SDASTER</i>	<i>0.01</i>	<i>36</i>	<i>63</i>
<i>model</i>	<i>MODELE_SDASTER</i>	<i>0.00</i>	<i>8</i>	<i>13</i>
<i>mat</i>	<i>MATER_SDASTER</i>	<i>0.00</i>	<i>5</i>	<i>5</i>
<i>mater</i>	<i>CHAM_MATER</i>	<i>0.00</i>	<i>6</i>	<i>11</i>
<i>cara</i>	<i>CARA_ELEM</i>	<i>0.01</i>	<i>22</i>	<i>42</i>
<i>tab_disp</i>	<i>TABLE_SDASTER</i>	<i>0.00</i>	<i>31</i>	<i>31</i>
<i>SIEFTAB</i>	<i>TABLE_SDASTER</i>	<i>0.01</i>	<i>37</i>	<i>37</i>
<i>Load</i>	<i>CHAR_MECA</i>	<i>0.00</i>	<i>14</i>	<i>29</i>
<i>constr</i>	<i>CHAR_MECA</i>	<i>0.01</i>	<i>23</i>	<i>43</i>
<i>res_stat</i>	<i>EVOL_ELAS</i>	<i>0.03</i>	<i>38</i>	<i>52</i>
<i>tab_reac</i>	<i>TABLE_SDASTER</i>	<i>0.00</i>	<i>31</i>	<i>31</i>
<i>&FOZERO</i>		<i>0.00</i>	<i>2</i>	<i>2</i>
<i>&&_NUM_C</i>		<i>0.00</i>	<i>1</i>	<i>1</i>
<i>&CATA.AC</i>		<i>0.00</i>	<i>2</i>	<i>4</i>
<i>&CATA.CL</i>		<i>0.60</i>	<i>1</i>	<i>3</i>
<i>&CATA.GD</i>		<i>0.14</i>	<i>4</i>	<i>11</i>
<i>&CATA.ME</i>		<i>0.20</i>	<i>2</i>	<i>4</i>
<i>&CATA.OP</i>		<i>0.30</i>	<i>4</i>	<i>19</i>
<i>&CATA.PH</i>		<i>0.00</i>	<i>1</i>	<i>1</i>
<i>&CATA.PR</i>		<i>0.00</i>	<i>2</i>	<i>4</i>
<i>&CATA.TE</i>		<i>26.03</i>	<i>17</i>	<i>42</i>
<i>&CATA.TH</i>		<i>0.01</i>	<i>2</i>	<i>4</i>
<i>&CATA.TM</i>		<i>0.01</i>	<i>7</i>	<i>11</i>

Nom de la base : *GLOBALE*

Nombre d'enregistrements utilisés : *46*

Nombre

d'enregistrements maximum : *62914*

Longueur d'enregistrement (octets) : *819200*

Nombre total d'accès en

Lecture : *7*

Volume des accès en lecture : *5.47 Mo.*

Nombre total d'accès en écriture :
 62
 Volume des accès en écriture : 48.44 Mo.
 Nombre d'identificateurs utilisés : 533
 Taille
 maximum du répertoire : 2000
 Pourcentage d'utilisation du répertoire : 26 %

Nom de la base : VOLATILE
 Nombre d'enregistrements utilisés : 3
 Nombre
 d'enregistrements maximum : 62914
 Longueur d'enregistrement (octets) : 819200
 Nombre total d'accès en
 Lecture : 0
 Volume des accès en lecture : 0.00 Mo.
 Nombre total d'accès en écriture :

6
 Volume des accès en écriture : 4.69 Mo.
 Nombre d'identificateurs utilisés : 985
 Taille
 maximum du répertoire : 2000
 Pourcentage d'utilisation du répertoire : 49 %

<FIN> Arrêt normal dans "FIN".

<I> <FIN> ARRET NORMAL DANS "FIN" PAR APPEL A "JEFINI".

The memory requested from launching is over-estimated, it is of 4096.00 Mo.

The peak report used is of 101.89 Mo.

Nom de la base : ELEMbase
 Nombre d'enregistrements utilisés : 43
 Nombre
 d'enregistrements maximum : 512
 Longueur d'enregistrement (octets) : 819200

Nombre total d'accès en lecture

: 60

Volume des accès en lecture : 46.88 Mo.

Nombre total d'accès en écriture : 0

Volume des accès en écriture : 0.00 Mo.

Nombre d'identificateurs utilisés : 123

Taille maximum

du répertoire : 300

Pourcentage d'utilisation du répertoire : 41 %

<I> <FIN> MEMOIRE JEVEUX MINIMALE REQUISE POUR L'EXECUTION : 33.95 Mo

<I> <FIN> MEMOIRE JEVEUX OPTIMALE REQUISE POUR L'EXECUTION : 44.30 Mo

<I> <FIN> MAXIMUM DE MEMOIRE UTILISEE PAR LE PROCESSUS LORS DE L'EXECUTION : 101.89 Mo

<I> FERMETURE DES BASES EFFECTUEE

STATISTIQUES CONCERNANT L'ALLOCATION DYNAMIQUE :

TAILLE CUMULEE MAXIMUM : 44 Mo.

TAILLE CUMULEE LIBREE : 6 Mo.

NOMBRE TOTAL D'ALLOCATIONS : 9494

NOMBRE TOTAL DE LIBERATIONS : 9474

APPELS AU MECANISME DE LIBERATION : 0

TAILLE MEMOIRE CUMULEE RECUPEREE : 0 Mo.

VOLUME DES LECTURES : 0 Mo.

VOLUME DES ECRITURES : 0 Mo.

MEMOIRE JEVEUX MINIMALE REQUISE POUR L'EXECUTION : 33.95 Mo

- IMPOSE DE NOMBREUX ACCES DISQUE

- RALENTIT LA VITESSE D'EXECUTION

MEMOIRE JEVEUX OPTIMALE REQUISE POUR L'EXECUTION : 44.30 Mo

- LIMITE LES ACCES DISQUE

- AMELIORE LA VITESSE D'EXECUTION

MAXIMUM DE MEMOIRE UTILISEE PAR LE PROCESSUS : 101.89 Mo

- COMPREND LA MEMOIRE CONSOMMEE PAR JEVEUX,
LE SUPERVISEUR PYTHON, LES LIBRAIRIES EXTERNES

<I> FIN D'EXECUTION LE : VE-15-MAI -2026 14:32:09

<I> Informations sur les temps d'exécution

Temps cpu user total 0.45 s
 Temps cpu systeme total 0.31 s
 Temps elapsed total 0.00 s
 Temps restant 1620.00 s

The signup of the saved base is (with the address 0):

bd786ae1df53c9ab3e45b958b8518873dc0d776b

```
*****
* COMMAND          :      USER :      SYSTEM :      USER+SYS :      ELAPSED *
*****
* init (jdc)       :      0.48 :      0.31 :      0.80 :      0.00 *
* . compile        :      0.00 :      0.00 :      0.00 :      0.00 *
* . exec_compile   :      0.22 :      0.14 :      0.36 :      0.00 *
* DEBUT           :      0.00 :      0.05 :      0.05 :      0.00 *
* LIRE_MALLAGE     :      0.00 :      0.00 :      0.00 :      0.00 *
* DEFI_GROUP       :      0.02 :      0.00 :      0.02 :      0.00 *
* AFFE_MODELE     :      0.00 :      0.00 :      0.00 :      0.00 *
* DEFI_MATERIAU    :      0.00 :      0.00 :      0.00 :      0.00 *
* AFFE_MATERIAU    :      0.00 :      0.00 :      0.00 :      0.00 *
* AFFE_CARA_ELEM   :      0.02 :      0.00 :      0.02 :      0.00 *
* AFFE_CHAR_MECA   :      0.00 :      0.00 :      0.00 :      0.00 *
* AFFE_CHAR_MECA   :      0.00 :      0.00 :      0.00 :      0.00 *
* MECA_STATIQUE    :      0.02 :      0.02 :      0.03 :      0.00 *
* CALC_CHAMP       :      0.02 :      0.00 :      0.02 :      0.00 *
* CREA_TABLE       :      0.00 :      0.00 :      0.00 :      0.00 *
* CREA_TABLE       :      0.00 :      0.00 :      0.00 :      0.00 *
* IMPR_TABLE       :      0.02 :      0.00 :      0.02 :      0.00 *
* IMPR_TABLE       :      0.00 :      0.00 :      0.00 :      0.00 *
* IMPR_RESU        :      0.02 :      0.00 :      0.02 :      0.00 *
* IMPR_CONCEPT   :      0.00 :      0.00 :      0.00 :      0.00 *
* POST_RELEVE_T    :      0.02 :      0.00 :      0.02 :      0.00 *
* IMPR_TABLE       :      0.00 :      0.00 :      0.00 :      0.00 *
* FIN              :      0.05 :      0.02 :      0.06 :      0.00 *
```

```

* . part Superviseur      :      0.34 :      0.28 :      0.62 :      0.00 *
* . part Fortran          :      0.14 :      0.03 :      0.17 :      0.00 *
*****
* TOTAL_JOB               :      0.45 :      0.31 :      0.77 :      0.00 *
*****

```

--- End of the execution

EXECUTION_CODE_ASTER_EXIT_2073=0

.resu file:

```

-----
ASTER 14.04.00 CONCEPT res_stat CALCULE LE 15/05/2026 A 14:32:09 DE TYPE
EVOL_ELAS

```

=====>

----->

CHAMP AUX NOEUDS DE NOM SYMBOLIQUE DEPL

NUMERO D'ORDRE: 1 INST: 0.000000000000E+00

NOEUD DRX	DX DRY	DY DRZ	DZ
N1	0.000000000000E+00	9.93873355430001E-25	0.000000000000E+00
	0.000000000000E+00	0.000000000000E+00	-4.84378490450601E-21
N2	0.000000000000E+00	-1.75291559163334E+01	0.000000000000E+00
	0.000000000000E+00	0.000000000000E+00	-6.57343346862557E-03
N3	0.000000000000E+00	-1.00570108406676E-01	0.000000000000E+00
	0.000000000000E+00	0.000000000000E+00	-7.96001709091144E-04
N4	0.000000000000E+00	-3.93721275464440E-01	0.000000000000E+00
	0.000000000000E+00	0.000000000000E+00	-1.54064846920870E-03
N5	0.000000000000E+00	-8.66614763929893E-01	0.000000000000E+00
	0.000000000000E+00	0.000000000000E+00	-2.23394028035267E-03
N6	0.000000000000E+00	-1.50641183655964E+00	0.000000000000E+00
	0.000000000000E+00	0.000000000000E+00	-2.87587714252305E-03
N7	0.000000000000E+00	-2.30027375611029E+00	0.000000000000E+00
	0.000000000000E+00	0.000000000000E+00	-3.46645905571985E-03
N8	0.000000000000E+00	-3.23536178533843E+00	0.000000000000E+00
	0.000000000000E+00	0.000000000000E+00	-4.00568601994305E-03
N9	0.000000000000E+00	-4.29883718700068E+00	0.000000000000E+00
	0.000000000000E+00	0.000000000000E+00	-4.49355803519264E-03

N10 0.00000000000000E+00 -5.47786122385362E+00 0.00000000000000E+00
 0.00000000000000E+00 0.00000000000000E+00 -4.93007510146863E-03

N11 0.00000000000000E+00 -6.75959515865385E+00 0.00000000000000E+00
 0.00000000000000E+00 0.00000000000000E+00 -5.31523721877097E-03

N12 0.00000000000000E+00 -8.13120025415797E+00 0.00000000000000E+00
 0.00000000000000E+00 0.00000000000000E+00 -5.64904438709972E-03

N13 0.00000000000000E+00 -9.57983777312258E+00 0.00000000000000E+00
 0.00000000000000E+00 0.00000000000000E+00 -5.93149660645487E-03

N14 0.00000000000000E+00 -1.10926689783043E+01 0.00000000000000E+00
 0.00000000000000E+00 0.00000000000000E+00 -6.16259387683634E-03

N15 0.00000000000000E+00 -1.26568551324596E+01 0.00000000000000E+00
 0.00000000000000E+00 0.00000000000000E+00 -6.34233619824414E-03

N16 0.00000000000000E+00 -1.42595574983452E+01 0.00000000000000E+00
 0.00000000000000E+00 0.00000000000000E+00 -6.47072357067827E-03

N17 0.00000000000000E+00 -1.58879373387176E+01 0.00000000000000E+00
 0.00000000000000E+00 0.00000000000000E+00 -6.54775599413875E-03

APPENDIX II: CASE STUDY 2 – SIMPLY SUPPORTED BEAM WITH MIDPOINT LOAD

This appendix presents the input files and the corresponding numerical results produced by Code_Aster for the analysis of a Simply Supported Beam with Midpoint Load.

.geo file:

```
// ---- 1D Beam geometry (OpenCASCADE) ----
SetFactory("OpenCASCADE");

// Mesh settings to preserve points as nodes in MED
Mesh.MedFileMinorVersion = 0;
Mesh.MeshSizeFromPoints = 1;
Mesh.MeshSizeFromCurvature = 0;
Mesh.MeshSizeExtendFromBoundary = 0;

// Parameters
// Length = 4000.0 (mm)
Point(1) = {0, 0, 0};
Point(2) = {2000.0, 0, 0};
Point(3) = {4000.0, 0, 0};
Line(1) = {1, 2};
Line(2) = {2, 3};

// Transfinite Line definition
// n_nodes = n_elems + 1 = 17
Transfinite Line {1} = 9;
Transfinite Line {2} = 9;

// Physical groups
Physical Point("BASE") = {1};
Physical Point("MID") = {2};
Physical Point("TIP") = {3};
Physical Curve("BEAM") = {1, 2};
```

.comm file:

```
# _____ Start _____  
DEBUT(PAR_LOT='NON');  
  
# _____ Read the mesh _____  
mesh = LIRE_MALLAGE(FORMAT='MED', UNITE=20);  
  
# _____ Define node groups (BASE, MID, TIP, BEAM) _____  
mesh = DEFI_GROUP(  
  reuse=mesh,  
  CREA_GROUP_NO=(  
    _F(GROUP_MA=('BEAM',), CRIT_NOEUD='TOUS'),  
    _F(GROUP_MA=('BASE',), CRIT_NOEUD='TOUS'),  
    _F(GROUP_MA=('MID',), CRIT_NOEUD='TOUS'),  
    _F(GROUP_MA=('TIP',), CRIT_NOEUD='TOUS'),  
  ),  
  MAILLAGE=mesh  
);  
  
# _____ Define mechanical model _____  
model = AFFE_MODELE(  
  MAILLAGE=mesh,  
  AFFE=_F(  
    GROUP_MA=('BEAM',),  
    PHENOMENE='MECANIQUE',  
    MODELISATION='POU_D_E'  
  )  
);  
  
# _____ Define material _____  
mat = DEFI_MATERIAU(ELAS=_F(E=210000.0, NU=0.3, RHO=7850.0));  
  
mater = AFFE_MATERIAU(  
  MAILLAGE=mesh,  
  AFFE=_F(GROUP_MA=('BEAM',), MATER=mat)
```

```

);

# _____ Define beam section geometry (GENERALE, mm) _____
cara = AFFE_CARA_ELEM(
    MODELE=model,
    POUTRE=_F(
        GROUP_MA=('BEAM',),
        SECTION='GENERALE',
        CARA=('A', 'IY', 'IZ', 'JX', 'AY', 'AZ'),
        VALE=(2009.0, 8693000.0, 8693000.0, 35300.0, 1214.0, 966.0),
    ),
);

# _____ Boundary conditions (pin at BASE, roller at TIP) _____
constr = AFFE_CHAR_MECA(
    MODELE=model,
    DDL_IMPO=(
        _F(
            GROUP_NO=('BASE',),
            DX=0.0, DY=0.0, DZ=0.0, DRX=0.0
        ),
        _F(
            GROUP_NO=('TIP',),
            DY=0.0, DZ=0.0
        )
    )
);

# _____ Applied load at MID _____
Load = AFFE_CHAR_MECA(
    MODELE=model,
    FORCE_NODALE=_F(
        GROUP_NO=('MID',),
        FY=-1500.0
    )
);

# _____ Static analysis _____

```

```

res_stat = MECA_STATIQUE(
    MODELE=model,
    CHAM_MATER=mater,
    CARA_ELEM=cara,
    EXCIT=(
        _F(CHARGE=constr),
        _F(CHARGE=Load)
    ),
    SOLVEUR=_F(
        METHODE='MUMPS',
        STOP_SINGULIER='OUI',
        RENUM='METIS',
        NPREC=8,
        ELIM_LAGR='NON'
    )
);

# _____ Compute fields: reactions, displacements, stresses _____
res_stat = CALC_CHAMP(
    reuse=res_stat,
    RESULTAT=res_stat,
    FORCE=('REAC_NODA',),
    CONTRAINTE=('EFGE_ELNO', 'SIEF_ELNO')
);

# _____ Create result tables (reactions, displacements) _____
tab_reac = CREA_TABLE(
    RESU=_F(
        GROUP_NO=('BASE', 'TIP'),
        NOM_CHAM='REAC_NODA',
        RESULTAT=res_stat,
        TOUT_CMP='OUI'
    ),
    TITRE='Nodal Reactions'
);

tab_disp = CREA_TABLE(
    RESU=_F(

```

```

TOUT='OUI',
NOM_CHAM='DEPL',
RESULTAT=res_stat,
TOUT_CMP='OUI'
),
TITRE='Nodal Displacements'
);

# _____ Print tables to text _____
IMPR_TABLE(TABLE=tab_reac, UNITE=18, SEPARATEUR=' ', FORMAT='TABLEAU', FORMAT_R='E12.5');
IMPR_TABLE(TABLE=tab_disp, UNITE=19, SEPARATEUR=' ', FORMAT='TABLEAU', FORMAT_R='E12.5');

# _____ Print displacement field _____
IMPR_RESU(
    FORMAT='RESULTAT',
    UNITE=80,
    RESU=_F(RESULTAT=res_stat, NOM_CHAM='DEPL')
);

# _____ Print local coordinate systems _____
IMPR_CONCEPT(
    CONCEPT=_F(CARA_ELEM=cara, MODELE=model, REPERE_LOCAL='ELEM'),
    FORMAT='RESULTAT',
    UNITE=53
);

# _____ Internal forces table _____
SIEFTAB = POST_RELEVE_T(
    ACTION=_F(
        INTITULE='BeamForces',
        RESULTAT=res_stat,
        NOM_CHAM='SIEF_ELNO',
        GROUP_NO=('BEAM',),
        OPERATION='EXTRACTION',
        TOUT_ORDRE='OUI',
        TOUT_CMP='OUI'
    )
);

```

```
IMPR_TABLE(TABLE=SIEFTAB, UNITE=21, SEPARATEUR=';', FORMAT='TABLEAU', FORMAT_R='E12.5');
```

```
# _____ End _____
```

```
FIN());
```

.export file:

```
P actions make_etude
```

```
P debug nodebug
```

```
P memjob 4194304
```

```
P memory_limit 4096.0
```

```
P mode interactif
```

```
P mpi_nbcpu 1
```

```
P mpi_nbnoeud 1
```

```
P ncpus 1
```

```
P nomjob beam_job
```

```
P origine CodeAster_Agent
```

```
P rep_trav C:\X\AppData\Local\Temp\Michele-PC-SONIA-interactif_395
```

```
P server localhost
```

```
P time_limit 1800.0
```

```
P tpsjob 31
```

```
P version stable
```

```
A memjeveux 512.0
```

```
A tpmx 1800.0
```

```
F comm C:\Users\X\workspace\runs\20260515_123513_beam_d10458d6\inputs\model.comm D 1
```

```
F libr C:\Users\X\workspace\runs\20260515_123513_beam_d10458d6\inputs\model.med D 20
```

```
F mess C:\Users\X\workspace\runs\20260515_123513_beam_d10458d6\outputs\beam_job.mess R 6
```

```
F resu C:\Users\X\workspace\runs\20260515_123513_beam_d10458d6\outputs\beam_job.resu R 80
```

```
F libr C:\Users\X\workspace\runs\20260515_123513_beam_d10458d6\outputs\tab_reac.txt R 18
```

```
F libr C:\Users\X\workspace\runs\20260515_123513_beam_d10458d6\outputs\tab_disp.txt R 19
```

```
F libr C:\Users\X\workspace\runs\20260515_123513_beam_d10458d6\outputs\local_axes.txt R 53
```

```
F libr C:\Users\X\workspace\runs\20260515_123513_beam_d10458d6\outputs\tab_sie.txt R 21
```

```
R base C:\Users\X\workspace\runs\20260515_123513_beam_d10458d6\outputs\beam_job_base R 0
```

.mess file:

<INFO> Démarrage de l'exécution.

```

!-----!
! <A> <SUPERVIS2_2>
!
!
```

```

! You use an old version of Code_Aster.
!
! By upgrading your version, you will profit from the last improvements
! made to the code
! since 15 month.
! If you have private developments, you are likely to have an important
! work of bearing if you do not follow the
! upgraded ones.
!
!
! This is a warning. If you do not understand the meaning of this
! warning, you can obtain unexpected results!
!-----!

```

```

Parallelism MPI: idle
OpenMP parallelism: Nombre
credit of processes used: 1
Version of the HDF5 Library : 1.10.3
Version of the MED Library : 4.0.0
Librairie MFront : non disponible
MUMPS Library version: 5.1.2
PETSc Library: not available
Version of the SCOTCH Library : 6.0.4

```

```

Mémoire limite pour l'exécution : 4096.00 Mo
consommée par l'initialisation : 45.98 Mo

```

```

par les objets du jeu de commandes : 0.00 Mo
reste pour l'allocation dynamique : 4045.59 Mo
Cut limiting files of exchange: 48.00 Go

```

```

.. __stg1_txt3

```

```

# -----
---
# Commands No: 0001 Concept of the type: -
# -----
---
```

[illegible]

124

! !
! This is a warning. If you do not understand the meaning of this !
! warning, you can obtain unexpected results! !
!-----!

Initial value of maximum time CPU = 1800 second

Valeur of the maximum time CPU placed to the orders = 1620 second

Réserve CPU

envisaged = 180 seconds

Ouverture en écriture du fichier ./glob.1

Ouverture en écriture du fichier ./vola.1

Ouverture en lecture du fichier C:\STADATA\CodeAster19\v2019\14.4\Lib\aster/elem.1

Nom de la base : ELEMBASE
Créée avec la version : 14.04.00
Nombre
d'enregistrements utilisés : 43
Nombre d'enregistrements maximum : 512
Longueur d'enregistrement (octets)
: 819200
Nombre d'identificateurs utilisés : 123
Taille maximum du répertoire : 300

Pourcentage d'utilisation du répertoire : 41 %

Ouverture en lecture du fichier C:\STADATA\CodeAster19\v2019\14.4\Lib\aster/elem.1

Nom de la base : ELEMBASE
Nombre d'enregistrements utilisés : 43
Nombre
d'enregistrements maximum : 512
Longueur d'enregistrement (octets) : 819200
Nombre total d'accès en lecture

```

      : 60
Volume des accès en lecture      :      46.88 Mo.
Nombre total d'accès en écriture :      0

Volume des accès en écriture      :      0.00 Mo.
Nombre d'identificateurs utilisés :    123
Taille maximum
du répertoire      :    300
Pourcentage d'utilisation du répertoire :  41 %

Second reading of the catalogues of the elements made.

End of reading (lasted 1.000000 S.)

# Memory (Mo):      96.70 /      88.94 /      44.30 /      22.92 (VmPeak / VmSize / Optimum /
Minimum)

# Fin orders No: 0001 user+syst:      0.06S (syst:      0.06S, Elaps:
0.00S)

# -----
---

.. __stg1_txt6

# -----
---

# Commands No: 0002 Concept of the type: maillage_sdaster

# -----
---

mesh = LIRE_MAILLAGE(FORMAT='MED',
                     UNITE=20,
                     INFO_MED=1,
                     VERI_MAIL=_F(VERIF='OUI',
                                   APLAT=1.E-3, ),
                     INFO=1, )

Vérification du maillage.

!-----!
! <A> <MODELISA8_14> !
! !
! La maille POI1 M1 est incluse dans une autre. !
! !

```

! !
 ! This is a warning. If you do not understand the meaning of this !
 ! warning, you can obtain unexpected results! !
 !-----!

!-----!
 ! <A> <MODELISA8_14> !
 ! !
 ! La maille POI1 M2 est incluse dans une autre. !
 ! !
 ! !
 ! This is a warning. If you do not understand the meaning of this !
 ! warning, you can obtain unexpected results! !
 !-----!

!-----!
 ! <A> <MODELISA8_14> !
 ! !
 ! La maille POI1 M3 est incluse dans une autre. !
 ! !
 ! !
 ! This is a warning. If you do not understand the meaning of this !
 ! warning, you can obtain unexpected results! !
 !-----!

----- MAILLAGE mesh - IMPRESSIONS NIVEAU 1 -----

MED file generated by Gmsh

NOMBRE DE NOEUDS 17

NOMBRE DE MAILLES 19

POI1 3

SEG2 16

NOMBRE DE GROUPE DE MAILLES 4

BASE	1
MID	1
TIP	1
BEAM	16

Memory (Mo): 96.70 / 91.57 / 44.30 / 22.92 (VmPeak / VmSize / Optimum / Minimum)

Fin orders No: 0002 user+syst: 0.025 (syst: 0.005, Elaps: 0.005)

.. __stg1_txt17

Commands No: 0004 Concept of the type: maillage_sdaster


```
mesh = DEFI_GROUP(reuse=mesh,
                  MAILLAGE=mesh,
                  CREA_GROUP_NO=(_F(GROUP_MA=('BEAM', ),
                                     CRIT_NOEUD='TOUS', ),
                                _F(GROUP_MA=('BASE', ),
                                     CRIT_NOEUD='TOUS', ),
                                _F(GROUP_MA=('MID', ),
                                     CRIT_NOEUD='TOUS', ),
                                _F(GROUP_MA=('TIP', ),
                                     CRIT_NOEUD='TOUS', ),
                                ),
                  ALARME='OUI', )
```

NOMBRE DE GROUPE DE NOEUDS CREES : 4

=====

!	NOM DU GROUPE	!	NBRE DE NOEUDS DU	!
!	NOEUDS	!	GROUPE_NO	!

!	BEAM	!	17	!
!	BASE	!	1	!
!	MID	!	1	!
!	TIP	!	1	!

Memory (Mo): 96.70 / 91.70 / 44.30 / 22.92 (VmPeak / VmSize / Optimum / Minimum)

Fin orders No: 0004 user+syst: 0.00S (syst: 0.00S, Elaps: 0.00S)

.. __stg1_txt26

Commands No: 0005 Concept of the type: modele_sdaster


```
model = AFPE_MODELE(MAILLAGE=mesh,
                    AFPE=_F(GROUP_MA=('BEAM', ),
                            PHENOMENE='MECANIQUE',
                            MODELISATION=('POU_D_E', ),),
                    INFO=1,
                    DISTRIBUTION=_F(PARTITIONNEUR='METIS',
                                     METHODE='SOUS_DOMAINE',),
                    VERI_JACOBIEN='OUI',)
```

Sur Les 19 mailles du maillage mesh, on a demandé l'affectation de 16, on a pu en affecter 16

Modélisation	Type maille	Élément fini	Nombre
POU_D_E	SEG2	MECA_POU_D_E	16

#2 Calculs elementaires et assemblages CPU (USER+SYST/SYST/ELAPS):
0.00 0.00 0.00

Memory (Mo): 96.70 / 93.06 / 44.30 / 33.82 (VmPeak / VmSize / Optimum / Minimum)

```

# Fin orders No: 0005 user+syst:      0.00S (syst:      0.00S, Elaps:
0.00S)

# -----
---

.. __stg1_txt31

# -----
---

# Commands No: 0006 Concept of the type: mater_sdaster

# -----
---

mat = DEFI_MATERIAU(ELAS=_F(E=2.10000E5,
                             NU=0.3,
                             RHO=7850.0,
                             COEF_AMOR=1.0, ),
                    INFO=1, )

Propriétés définies dans Le matériau: ELAS.

# Memory (Mo):      96.70 /      93.22 /      44.30 /      33.82 (VmPeak / VmSize / Optimum /
Minimum)

# Fin orders No: 0006 user+syst:      0.00S (syst:      0.00S, Elaps:
0.00S)

# -----
---

.. __stg1_txt35

# -----
---

# Commands No: 0007 Concept of the type: cham_mater

# -----
---

mater = AFFE_MATERIAU(MAILLAGE=mesh,
                      AFFE=_F(GROUP_MA=('BEAM', ),
                               MATER=(mat, ), ),
                      INFO=1, )

# Memory (Mo):      96.70 /      93.30 /      44.30 /      33.82 (VmPeak / VmSize / Optimum /
Minimum)

# Fin orders No: 0007 user+syst:      0.00S (syst:      0.00S, Elaps:
0.00S)

# -----
---
```

```

.. __stg1_txt45
# -----
---
# Commands No: 0008 Concept of the type: cara_elem
# -----
---
cara = AFFE_CARA_ELEM(MODELE=model,
                      POUTRE=_F(SECTION='GENERALE',
                                GROUP_MA=('BEAM', ),
                                CARA=('A', 'IY', 'IZ', 'JX', 'AY', 'AZ'),
                                VALE=(2009.0, 8.693000E6, 8.693000E6, 3.5300E4, 1214.0,
966.0),
                                VARI_SECT='CONSTANT', ),
                      INFO=1,)

LE MODELE model    CONTIENT :

                                16 ELEMENT(S) POUTRE
#2      Calculs elementaires et assemblages      CPU (USER+SYST/SYST/ELAPS):
0.00      0.00      0.00
# Memory (Mo):      96.70 /      93.89 /      44.30 /      33.82 (VmPeak / VmSize / Optimum /
Minimum)
# Fin orders No: 0008 user+syst:      0.02S (syst:      0.00S, Elaps:
0.00S)
# -----
---

.. __stg1_txt59
# -----
---
# Commands No: 0010 Concept of the type: char_meca
# -----
---
constr = AFFE_CHAR_MECA(MODELE=model,
                        DDL_IMPO=(_F(GROUP_NO=('BASE', ),
                                      DX=0.0,
                                      DY=0.0,
                                      DZ=0.0,
                                      DRX=0.0, ),
                        _F(GROUP_NO=('TIP', ),
                          DY=0.0,

```

```

                                DZ=0.0, ),
                                ),
                                VERI_NORM='OUI',
                                DOUBLE_LAGRANGE='OUI',
                                INFO=1, )

# Memory (Mo):      96.70 /      94.44 /      44.30 /      33.82 (VmPeak / VmSize / Optimum /
Minimum)

# Fin orders No:  0010 user+syst:      0.00S (syst:      0.00S, Elaps:
0.00S)

# -----
---

.. __stg1_txt69

# -----
---

# Commands No:  0011 Concept of the type:  char_meca

# -----
---

Load = AFFE_CHAR_MECA(MODELE=model,
                        FORCE_NODALE=_F(GROUP_NO=( 'MID', ),
                        FY=-1500.0, ),
                        VERI_NORM='OUI',
                        DOUBLE_LAGRANGE='OUI',
                        INFO=1, )

# Memory (Mo):      96.70 /      94.46 /      44.30 /      33.82 (VmPeak / VmSize / Optimum /
Minimum)

# Fin orders No:  0011 user+syst:      0.00S (syst:      0.00S, Elaps:
0.00S)

# -----
---

.. __stg1_txt87

# -----
---

# Commands No:  0012 Concept of the type:  evol_elas

# -----
---

res_stat = MECA_STATIQUE(MODELE=model,
                           CHAM_MATER=mater,
                           CARA_ELEM=cara,

```

```

EXCIT=(_F(CHARGE=constr,
          TYPE_CHARGE='FIXE_CSTE',),
       _F(CHARGE=Load,
          TYPE_CHARGE='FIXE_CSTE',),
       ),
SOLVEUR=_F(METHODE='MUMPS',
           RENUM='METIS',
           NPREC=8,
           ELIM_LAGR='NON',
           STOP_SINGULIER='OUI',
           TYPE_RESOL='AUTO',
           ACCELERATION='AUTO',
           LOW_RANK_SEUIL=0.0,
           PRETRAITEMENTS='AUTO',
           POSTTRAITEMENTS='AUTO',
           PCENT_PIVOT=20,
           RESI_RELA=1.E-06,
           GESTION_MEMOIRE='AUTO',
           MATR_DISTRIBUEE='NON',),
OPTION='SIEF_ELGA',
INFO=1,)

```

Le système linéaire à résoudre a 114 degrés de liberté:

- 102 sont des degrés de liberté physiques

(ils sont portés par 17

noeuds du maillage)

- 12 sont les couples de paramètres de Lagrange associés

aux 6 relations linéaires dualisées.

La matrice est de taille 114 équations.

Elle contient 1023 termes non nuls si elle est symétrique et 1932 termes non nuls si elle n'est pas symétrique.

Soit un taux de remplissage de 14.866 %.

Field stored DEPL at time 0.000000000000e+00 for the sequence number 1

#1	Resolution des systemes lineaires	CPU (USER+SYST/SYST/ELAPS):
0.04	0.00 0.00	

#2	Calculs elementaires et assemblages	CPU (USER+SYST/SYST/ELAPS):
0.00	0.00 0.00	

# Memory (Mo):	100.68 /	100.67 /	44.30 /	33.82 (VmPeak / VmSize / Optimum /
Minimum)				

```

# Fin orders No: 0012 user+syst:      0.06S (syst:      0.02S, Elaps:
0.00S)

# -----
---

.. __stg1_txt96

# -----
---

# Commands No: 0013 Concept of the type: evol_elas
# -----
---

res_stat = CALC_CHAMP(reuse=res_stat,
                      RESULTAT=res_stat,
                      PRECISION=1.E-06,
                      CONTRAINTE=('EFGE_ELNO', 'SIEF_ELNO'),
                      FORCE=('REAC_NODA', ),
                      CRITERE='RELATIF',
                      INFO=1,)

Le système linéaire à résoudre a 114 degrés de liberté:
- 102 sont des degrés de liberté physiques
  (ils sont portés par 17
noeuds du maillage)
- 12 sont les couples de paramètres de Lagrange associés
  aux 6 relations linéaires dualisées.

#2      Calculs elementaires et assemblages      CPU (USER+SYST/SYST/ELAPS):
0.00      0.00      0.00

# Memory (Mo):   101.00 /   101.00 /   44.30 /   33.95 (VmPeak / VmSize / Optimum /
Minimum)

# Fin orders No: 0013 user+syst:      0.02S (syst:      0.00S, Elaps:
0.00S)

# -----
---

.. __stg1_txt107

# -----
---

# Commands No: 0014 Concept of the type: table_sdaster
# -----
---

tab_reac = CREA_TABLE(TITRE='Nodal Reactions',
                      RESU=_F(RESULTAT=res_stat,

```

```

        TOUT_CMP='OUI',
        GROUP_NO=('BASE', 'TIP'),
        NOM_CHAM='REAC_NODA',
        PRECISION=1.E-06,
        CRITERE='RELATIF',),
    TYPE_TABLE='TABLE',)

# Memory (Mo):    101.07 /    101.07 /    44.30 /    33.95 (VmPeak / VmSize / Optimum /
Minimum)

# Fin orders No:  0014 user+syst:          0.00S (syst:          0.00S, Elaps:
0.00S)

# -----
---

.. __stg1_txt117

# -----
---

# Commands No:  0015 Concept of the type:  table_sdaster

# -----
---

tab_disp = CREA_TABLE(TITRE='Nodal Displacements',
    RESU=_F(RESTAT=res_stat,
        TOUT_CMP='OUI',
        TOUT='OUI',
        NOM_CHAM='DEPL',
        PRECISION=1.E-06,
        CRITERE='RELATIF',),
    TYPE_TABLE='TABLE',)

# Memory (Mo):    101.07 /    101.07 /    44.30 /    33.95 (VmPeak / VmSize / Optimum /
Minimum)

# Fin orders No:  0015 user+syst:          0.00S (syst:          0.00S, Elaps:
0.00S)

# -----
---

.. __stg1_txt121

# -----
---

# Commands No:  0016 Concept of the type:  -

# -----
---

```

```

IMPR_TABLE(TABLE=tab_reac,
            FORMAT='TABLEAU',
            FORMAT_R='E12.5',
            UNITE=18,
            SEPARATEUR=' ',
            FIN_TABLE='',
            COMMENTAIRE='#',
            COMM_PARA='',
            DEBUT_LIGNE='',
            FIN_LIGNE='\n',
            IMPR_FONCTION='NON',
            INFO=1,)

# Memory (Mo):    101.43 /    101.43 /    44.30 /    33.95 (VmPeak / VmSize / Optimum /
Minimum)

# Fin orders No:  0016 user+syst:          0.00S (syst:          0.00S, Elaps:
0.00S)

# -----
---

.. __stg1_txt122

# -----
---

# Commands No:  0019 Concept of the type:  -

# -----
---

IMPR_TABLE(TABLE=tab_disp,
            FORMAT='TABLEAU',
            FORMAT_R='E12.5',
            UNITE=19,
            SEPARATEUR=' ',
            FIN_TABLE='',
            COMMENTAIRE='#',
            COMM_PARA='',
            DEBUT_LIGNE='',
            FIN_LIGNE='\n',
            IMPR_FONCTION='NON',
            INFO=1,)

```

Memory (Mo): 101.43 / 101.43 / 44.30 / 33.95 (VmPeak / VmSize / Optimum / Minimum)

Fin orders No: 0019 user+syst: 0.025 (syst: 0.005, Elaps: 0.005)

.. __stg1_txt128

Commands No: 0022 Concept of the type: -

IMPR_RESU(FORMAT='RESULTAT',
UNITE=80,
RESU=_F(RESULTAT=res_stat,
NOM_CHAM=('DEPL',),
FORM_TABL='NON',
IMPR_COOR='NON',
FORMAT_R='1PE21.14',),
PROC0='OUI',
INFO=1,)

Memory (Mo): 101.57 / 101.56 / 44.30 / 33.95 (VmPeak / VmSize / Optimum / Minimum)

Fin orders No: 0022 user+syst: 0.005 (syst: 0.005, Elaps: 0.005)

.. __stg1_txt135

Commands No: 0023 Concept of the type: -

IMPR_CONCEPT(FORMAT='RESULTAT',
UNITE=53,
CONCEPT=_F(CARA_ELEM=cara,
REPERE_LOCAL='ELEM',
MODELE=model,),
INFO=1,)

IMPRESSION D'UN CHAMP DE CONCEPT : Caracteristiques generales des poutres

IMPRESSION D'UN CHAMP DE CONCEPT : Caracteristiques generales des poutres

NOM DU CHAMP : cara _CARGENPO

NOM DU CHAMP : cara _CARGENPO

CORRESPONDANCE VALEUR <-> CONTENU :

CORRESPONDANCE VALEUR <-> CONTENU :

VALEUR = 1.

VALEUR = 1.

VALEUR = 2.

VALEUR = 2.

A1 = 2.009D+03

A1 = 2.009D+03

IY1 = 8.693D+06

IY1 = 8.693D+06

IZ1 = 8.693D+06

IZ1 = 8.693D+06

AY1 = 0.000D+00

AY1 = 0.000D+00

AZ1 = 0.000D+00

AZ1 = 0.000D+00

EY1 = 0.000D+00

EY1 = 0.000D+00

EZ1 = 0.000D+00

EZ1 = 0.000D+00

JX1 = 3.530D+04

JX1 = 3.530D+04

RY1 = 1.000D+00

RY1 = 1.000D+00

RZ1 = 1.000D+00

RZ1 = 1.000D+00

RT1 = 1.000D+00

RT1 = 1.000D+00

AI1 = 0.000D+00

AI1 = 0.000D+00

JG1 = 0.000D+00

JG1	=	0.000D+00
IYR21	=	0.000D+00
IYR21	=	0.000D+00
IZR21	=	0.000D+00
IZR21	=	0.000D+00
A2	=	2.009D+03
A2	=	2.009D+03
IY2	=	8.693D+06
IY2	=	8.693D+06
IZ2	=	8.693D+06
IZ2	=	8.693D+06
AY2	=	0.000D+00
AY2	=	0.000D+00
AZ2	=	0.000D+00
AZ2	=	0.000D+00
EY2	=	0.000D+00
EY2	=	0.000D+00
EZ2	=	0.000D+00
EZ2	=	0.000D+00
JX2	=	3.530D+04
JX2	=	3.530D+04
RY2	=	1.000D+00
RY2	=	1.000D+00
RZ2	=	1.000D+00
RZ2	=	1.000D+00
RT2	=	1.000D+00
RT2	=	1.000D+00
AI2	=	0.000D+00
AI2	=	0.000D+00
JG2	=	0.000D+00
JG2	=	0.000D+00
IYR22	=	0.000D+00
IYR22	=	0.000D+00
IZR22	=	0.000D+00
IZR22	=	0.000D+00
TVAR	=	0.000D+00
TVAR	=	0.000D+00

IMPRESSION D'UN CHAMP DE CONCEPT : Caracteristiques geom. des poutres

IMPRESSION D'UN CHAMP DE CONCEPT : Caracteristiques geom. des poutres

NOM DU CHAMP : cara _CARGEOPO

NOM DU CHAMP : cara _CARGEOPO

CORRESPONDANCE VALEUR <-> CONTENU :

CORRESPONDANCE VALEUR <-> CONTENU :

VALEUR = 1.

VALEUR = 1.

IMPRESSION D'UN CHAMP DE CONCEPT : Orientation des coques, poutres, barres, cables

IMPRESSION D'UN CHAMP DE CONCEPT : Orientation des coques, poutres, barres, cables

NOM DU CHAMP : cara _CARORIEN

NOM DU CHAMP : cara _CARORIEN

CORRESPONDANCE VALEUR <-> CONTENU :

CORRESPONDANCE VALEUR <-> CONTENU :

VALEUR = 1.

VALEUR = 1.

ALPHA = 0.000D+00

ALPHA = 0.000D+00

BETA = 0.000D+00

BETA = 0.000D+00

GAMMA = 0.000D+00

GAMMA = 0.000D+00

#2 Calculs elementaires et assemblages

CPU (USER+SYST/SYST/ELAPS):

0.00 0.00 0.00

Memory (Mo): 101.76 / 101.76 / 44.30 / 33.95 (VmPeak / VmSize / Optimum / Minimum)

Fin orders No: 0023 user+syst: 0.02S (syst: 0.00S, Elaps: 0.00S)

.. __stg1_txt147

Commands No: 0024 Concept of the type: table_sdaster


```

SIEFTAB = POST_RELEVE_T(ACTION=_F(OPERATION=('EXTRACTION', ),
                                INITITULE='BeamForces',
                                RESULTAT=res_stat,
                                NOM_CHAM='SIEF_ELNO',
                                TOUT_ORDRE='OUI',
                                PRECISION=1.E-06,
                                CRITERE='RELATIF',
                                GROUP_NO=('BEAM', ),
                                TOUT_CMP='OUI',
                                REPERE='GLOBAL',
                                MOYE_NOEUD='OUI',
                                FORMAT_C='MODULE', ),
                        INFO=1,)

# Memory (Mo):    102.07 /    102.07 /    44.30 /    33.95 (VmPeak / VmSize / Optimum /
Minimum)

# Fin orders No:  0024 user+syst:    0.02S (syst:    0.00S, Elaps:
0.00S)

# -----
---

.. __stg1_txt151

# -----
---

# Commands No:  0025 Concept of the type:  -

# -----
---

IMPR_TABLE(TABLE=SIEFTAB,
            FORMAT='TABLEAU',
            FORMAT_R='E12.5',
            UNITE=21,
            SEPARATEUR=';',
            FIN_TABLE=' ',
            COMMENTAIRE='#',
            COMM_PARA=' ',
            DEBUT_LIGNE=' ',
            FIN_LIGNE='\n',
            IMPR_FONCTION='NON',
            INFO=1,)

```

Memory (Mo): 102.08 / 102.07 / 44.30 / 33.95 (VmPeak / VmSize / Optimum / Minimum)

Fin orders No: 0025 user+syst: 0.00S (syst: 0.00S, Elaps: 0.00S)

.. __stg1_txt154

Commands No: 0028 Concept of the type: -


```
FIN(FORMAT_HDF='NON',  
    RETASSAGE='NON',  
    INFO_RESU='OUI',  
    PROC0='OUI',  
    )
```

```
!-----!  
! <I> <CATAMESS_89> !  
! !  
! List of warnings emitted during the execution of computation. !  
! !  
! Warnings which you chose to ignore of are preceded by (*). !  
! !  
! Number of occurrences for each warning: !  
!          MODELISA8_14          emitted    3 time !  
!          SUPERVIS2_2           emitted    1 time !  
!          SUPERVIS_1            emitted    1 time !  
!-----!
```

=====>

STRUCTURE DU CONCEPT res_stat CALCULE POUR 1 NUMERO D'ORDRE

LISTE DES NOMS SYMBOLIQUES:

```

! ----- !-----!-----!-----!-----!-----
-----!

! NUME_ORDRE !      DEPL      !  EFGE_ELNO  !  SIEF_ELGA  !  SIEF_ELNO  !
REAC_NODA    !

! ----- !-----!-----!-----!-----!-----
-----!

!      1 !      DEPL_R      !      SIEF_R      !      SIEF_R      !      SIEF_R      !
DEPL_R      !

! ----- !-----!-----!-----!-----!-----
-----!

```

LISTE DES NOMS DE VARIABLES D'ACCES:

INST DE TYPE R

LISTE DES NOMS DE PARAMETRES:

```

! ----- !-----!-----!-----!-----!-----
-----!-----!-----!-----!-----!

! NUME_ORDRE !      CARAELEM      !      CHAMPMAT      !      EXCIT      !      ITER_GCPC      !
METHODE      !      MODELE      !      RENUM      !      STOCKAGE      !

! ----- !-----!-----!-----!-----!-----
-----!-----!-----!-----!-----!

!      1 !      K8      !      K8      !      K24      !      I      !
K16      !      K8      !      K16      !      K16      !

! ----- !-----!-----!-----!-----!-----
-----!-----!-----!-----!-----!

-----

```

Concepts de la base: G

Nom	Type	Taille (Mo)	Nombre d'objets	Nombre de segments
TOTAL		27.36	296	465
mesh	MAILLAGE_SDASTER	0.01	36	65
model	MODELE_SDASTER	0.00	8	13
mat	MATER_SDASTER	0.00	5	5
mater	CHAM_MATER	0.00	6	11
cara	CARA_ELEM	0.01	22	42
tab_disp	TABLE_SDASTER	0.00	31	31
SIEFTAB	TABLE_SDASTER	0.01	37	37
Load	CHAR_MECA	0.00	14	29
constr	CHAR_MECA	0.01	23	43
res_stat	EVOL_ELAS	0.03	38	52
tab_reac	TABLE_SDASTER	0.00	31	31

&FOZERO	0.00	2	2
&&_NUM_C	0.00	1	1
&CATA.AC	0.00	2	4
&CATA.CL	0.60	1	3
&CATA.GD	0.14	4	11
&CATA.ME	0.20	2	4
&CATA.OP	0.30	4	19
&CATA.PH	0.00	1	1
&CATA.PR	0.00	2	4
&CATA.TE	26.03	17	42
&CATA.TH	0.01	2	4
&CATA.TM	0.01	7	11

Nom de la base : GLOBALE

Nombre d'enregistrements utilisés : 46

Nombre
d'enregistrements maximum : 62914

Longueur d'enregistrement (octets) : 819200

Nombre total d'accès en
Lecture : 7

Volume des accès en lecture : 5.47 Mo.

Nombre total d'accès en écriture :
62

Volume des accès en écriture : 48.44 Mo.

Nombre d'identificateurs utilisés : 533

Taille
maximum du répertoire : 2000

Pourcentage d'utilisation du répertoire : 26 %

Nom de la base : VOLATILE

Nombre d'enregistrements utilisés : 3

Nombre
d'enregistrements maximum : 62914

Longueur d'enregistrement (octets) : 819200

Nombre total d'accès en

Lecture : 0

Volume des accès en lecture : 0.00 Mo.

Nombre total d'accès en écriture :

6

Volume des accès en écriture : 4.69 Mo.

Nombre d'identificateurs utilisés : 985

Taille

maximum du répertoire : 2000

Pourcentage d'utilisation du répertoire : 49 %

<FIN> Arrêt normal dans "FIN".

<I> <FIN> ARRET NORMAL DANS "FIN" PAR APPEL A "JEFINI".

The memory requested from launching is over-estimated, it is of 4096.00 Mo.

The peak report used is of 102.11 Mo.

Nom de la base : ELEMbase

Nombre d'enregistrements utilisés : 43

Nombre

d'enregistrements maximum : 512

Longueur d'enregistrement (octets) : 819200

Nombre total d'accès en lecture

: 60

Volume des accès en lecture : 46.88 Mo.

Nombre total d'accès en écriture : 0

Volume des accès en écriture : 0.00 Mo.

Nombre d'identificateurs utilisés : 123

Taille maximum

du répertoire : 300

Pourcentage d'utilisation du répertoire : 41 %

<I> <FIN> MEMOIRE JEUX MINIMALE REQUISE POUR L'EXECUTION : 33.95 Mo

<I> <FIN> MEMOIRE JEUX OPTIMALE REQUISE POUR L'EXECUTION : 44.30 Mo

<I> <FIN> MAXIMUM DE MEMOIRE UTILISEE PAR LE PROCESSUS LORS DE L'EXECUTION : 102.11 Mo

<I> FERMETURE DES BASES EFFECTUEE

STATISTIQUES CONCERNANT L'ALLOCATION DYNAMIQUE :

TAILLE CUMULEE MAXIMUM	:	44 Mo.
TAILLE CUMULEE LIBEREE	:	6 Mo.
NOMBRE TOTAL D'ALLOCATIONS	:	9434
NOMBRE TOTAL DE LIBERATIONS	:	9414
APPELS AU MECANISME DE LIBERATION	:	0
TAILLE MEMOIRE CUMULEE RECUPEREE	:	0 Mo.
VOLUME DES LECTURES	:	0 Mo.
VOLUME DES ECRITURES	:	0 Mo.

MEMOIRE JEVEUX MINIMALE REQUISE POUR L'EXECUTION : 33.95 Mo

- IMPOSE DE NOMBREUX ACCES DISQUE
- RALENTIT LA VITESSE D'EXECUTION

MEMOIRE JEVEUX OPTIMALE REQUISE POUR L'EXECUTION : 44.30 Mo

- LIMITE LES ACCES DISQUE
- AMELIORE LA VITESSE D'EXECUTION

MAXIMUM DE MEMOIRE UTILISEE PAR LE PROCESSUS : 102.11 Mo

- COMPREND LA MEMOIRE CONSOMMEE PAR JEVEUX,
LE SUPERVISEUR PYTHON, LES LIBRAIRIES EXTERNES

<I> FIN D'EXECUTION LE : VE-15-MAI -2026 14:35:15

<I> Informations sur les temps d'exécution

Temps cpu user total	0.56 s
Temps cpu systeme total	0.31 s
Temps elapsed total	0.00 s
Temps restant	1620.00 s

The signup of the saved base is (with the address 0):

adbe8d2b2c190dd04a9af4e1bc9855e7dc341e29

* COMMAND : USER : SYSTEM : USER+SYS : ELAPSED *

```

* init (jdc)           :      0.59 :      0.31 :      0.91 :      0.00 *
*   . compile          :      0.02 :      0.00 :      0.02 :      0.00 *
*   . exec_compile     :      0.31 :      0.16 :      0.47 :      0.00 *
* DEBUT                :      0.00 :      0.06 :      0.06 :      0.00 *
* LIRE_MALLAGE         :      0.02 :      0.00 :      0.02 :      0.00 *
* DEFI_GROUP           :      0.00 :      0.00 :      0.00 :      0.00 *
* AFFE_MODELE          :      0.00 :      0.00 :      0.00 :      0.00 *
* DEFI_MATERIAU        :      0.00 :      0.00 :      0.00 :      0.00 *
* AFFE_MATERIAU        :      0.00 :      0.00 :      0.00 :      0.00 *
* AFFE_CARA_ELEM       :      0.02 :      0.00 :      0.02 :      0.00 *
* AFFE_CHAR_MECA        :      0.00 :      0.00 :      0.00 :      0.00 *
* AFFE_CHAR_MECA        :      0.00 :      0.00 :      0.00 :      0.00 *
* MECA_STATIQUE         :      0.05 :      0.02 :      0.06 :      0.00 *
* CALC_CHAMP           :      0.02 :      0.00 :      0.02 :      0.00 *
* CREA_TABLE           :      0.00 :      0.00 :      0.00 :      0.00 *
* CREA_TABLE           :      0.00 :      0.00 :      0.00 :      0.00 *
* IMPR_TABLE           :      0.00 :      0.00 :      0.00 :      0.00 *
* IMPR_TABLE           :      0.02 :      0.00 :      0.02 :      0.00 *
* IMPR_RESU            :      0.00 :      0.00 :      0.00 :      0.00 *
* IMPR_CONCEPT        :      0.02 :      0.00 :      0.02 :      0.00 *
* POST_RELEVE_T        :      0.02 :      0.00 :      0.02 :      0.00 *
* IMPR_TABLE           :      0.00 :      0.00 :      0.00 :      0.00 *
* FIN                  :      0.05 :      0.03 :      0.08 :      0.00 *
*   . part Superviseur :      0.44 :      0.27 :      0.70 :      0.00 *
*   . part Fortran      :      0.16 :      0.05 :      0.20 :      0.00 *
*****
* TOTAL_JOB            :      0.56 :      0.31 :      0.88 :      0.00 *
*****

```

--- End of the execution

EXECUTION_CODE_ASTER_EXIT_395=0

.resu file:

```

-----
ASTER 14.04.00 CONCEPT res_stat CALCULE LE 15/05/2026 A 14:35:15 DE TYPE
EVOL_ELAS

```

=====>

----->

CHAMP AUX NOEUDS DE NOM SYMBOLIQUE DEPL

NUMERO D'ORDRE: 1 INST: 0.000000000000E+00

NOEUD DRX	DX DRY	DY DRZ	DZ
N1	0.000000000000E+00	2.67268536181749E-25	0.000000000000E+00
0.000000000000E+00	0.000000000000E+00	-8.21679183579306E-04	
N2	0.000000000000E+00	-1.09557224477242E+00	0.000000000000E+00
0.000000000000E+00	0.000000000000E+00	-2.00916312963016E-17	
N3	0.000000000000E+00	4.59336281603319E-25	0.000000000000E+00
0.000000000000E+00	0.000000000000E+00	8.21679183579329E-04	
N4	0.000000000000E+00	-2.04349901124541E-01	0.000000000000E+00
0.000000000000E+00	0.000000000000E+00	-8.08840446335880E-04	
N5	0.000000000000E+00	-4.02280433627369E-01	0.000000000000E+00
0.000000000000E+00	0.000000000000E+00	-7.70324234605600E-04	
N6	0.000000000000E+00	-5.87372228886770E-01	0.000000000000E+00
0.000000000000E+00	0.000000000000E+00	-7.06130548388468E-04	
N7	0.000000000000E+00	-7.53205918281032E-01	0.000000000000E+00
0.000000000000E+00	0.000000000000E+00	-6.16259387684485E-04	
N8	0.000000000000E+00	-8.93362133188441E-01	0.000000000000E+00
0.000000000000E+00	0.000000000000E+00	-5.00710752493645E-04	
N9	0.000000000000E+00	-1.00142150498728E+00	0.000000000000E+00
0.000000000000E+00	0.000000000000E+00	-3.59484642815955E-04	
N10	0.000000000000E+00	-1.07096466505585E+00	0.000000000000E+00
0.000000000000E+00	0.000000000000E+00	-1.92581058651417E-04	
N11	0.000000000000E+00	-1.07096466505586E+00	0.000000000000E+00
0.000000000000E+00	0.000000000000E+00	1.92581058651388E-04	
N12	0.000000000000E+00	-1.00142150498730E+00	0.000000000000E+00
0.000000000000E+00	0.000000000000E+00	3.59484642815940E-04	
N13	0.000000000000E+00	-8.93362133188458E-01	0.000000000000E+00
0.000000000000E+00	0.000000000000E+00	5.00710752493639E-04	
N14	0.000000000000E+00	-7.53205918281049E-01	0.000000000000E+00
0.000000000000E+00	0.000000000000E+00	6.16259387684489E-04	
N15	0.000000000000E+00	-5.87372228886785E-01	0.000000000000E+00
0.000000000000E+00	0.000000000000E+00	7.06130548388482E-04	
N16	0.000000000000E+00	-4.02280433627379E-01	0.000000000000E+00
0.000000000000E+00	0.000000000000E+00	7.70324234605619E-04	
N17	0.000000000000E+00	-2.04349901124547E-01	0.000000000000E+00
0.000000000000E+00	0.000000000000E+00	8.08840446335902E-04	