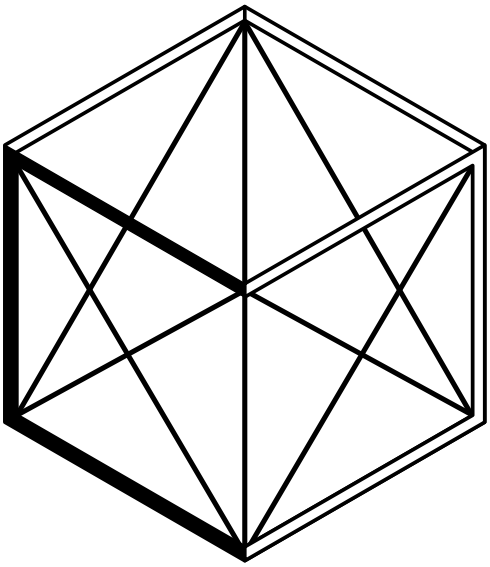
FEMaster

Finite **E**lements in **M**echanical **A**nalysis and **S**imulation for **T**echnical **E**ngineering **R**esearch



Finn Eggers
October 2024

Contents

Chapter 1

Introduction

1.1 Introduction

Welcome to the FEMaster User Manual. This document serves as a comprehensive guide to understanding and utilizing FEMaster, a finite element solver developed with a syntax similar to Abaqus but with several distinct features and improvements.
This manual is structured to help both new and experienced users navigate the solver's functionalities, input structure, and command syntax. Each section introduces core aspects of FEMaster, making it easier to dive into specific features for customized use.

1.1.1 Scope

FEMaster is designed for solving a variety of structural mechanics problems, including static, dynamic, and thermal analyses. It supports custom element formulations and allows for flexible input definitions tailored to complex geometries. The solver is open-source, lightweight, and primarily focused on solid mechanics applications, offering a streamlined experience for handling complex 3D models.

1.1.2 Features

FEMaster includes the following features:

- Custom input syntax inspired by Abaqus.
- Support for 3D structural elements.
- Multi-threading capabilities for efficient parallel processing.
- Utilisation of the graphical processing unit (GPU) for accelerated computations.
- A variety of constraints.
- Modular solver components for flexibility.
- Robust error handling and detailed logging.
- Integration with ParaView for post-processing.

1.2 Getting Started

This section provides a step-by-step guide to setting up and running a basic FEMaster analysis.

1.2.1 Installation

FEMaster is distributed as a standalone executable along with the necessary libraries. To install:

1. Download the latest release from the repository.
2. Extract the contents to your desired directory.
3. Set the environment variables as described in the README.
4. Ensure the solver's path is included in your system's PATH variable.

1.2.2 Basic Analysis Setup

To perform a basic analysis, users need to create an input file with nodes, elements, materials, and boundary conditions. Refer to Section 1.3.1 for the structure of a standard input deck.

1.3 Core Concepts

FEMaster is a lightweight, open-source finite element solver developed for solid mechanics problems. Its syntax is inspired by traditional FEM software like Abaqus but offers a streamlined and simplified structure that makes it easy to set up and run complex simulations. Designed to be user-friendly, fast, and highly scalable, FEMaster supports multi-threading capabilities to handle large-scale models efficiently. The solver primarily supports 3D solid elements and is ideal for researchers and engineers looking for a flexible tool to conduct static analyses. While shell and beam elements are not yet supported, these features are planned for future updates.

1.3.1 Element Types

FEMaster supports a range of 3D solid elements, each tailored for different types of analyses. The current set of supported elements includes:

- **C3D4**: A linear tetrahedral element with four nodes, used for general 3D applications.
- **C3D6**: A linear wedge element with six nodes, commonly used as a transition element between tetrahedral and hexahedral meshes.
- **C3D8**: A linear hexahedral element with eight nodes, suitable for most 3D problems.
- **C3D10**: A quadratic tetrahedral element with ten nodes, providing higher accuracy for curved geometries.
- **C3D15**: A quadratic wedge element with fifteen nodes, offering improved performance for complex geometries.
- **C3D20**: A quadratic hexahedral element with twenty nodes, offering excellent performance for higher-order solutions.
- **C3D20R**: A reduced integration version of the C3D20, minimizing computational cost while maintaining reasonable accuracy.

Note: Shell and beam elements are currently not supported but will be added in future updates.

1.3.2 Geometric Entities

FEMaster is built around three core geometric entities: **nodes**, **elements**, and **surfaces**. Each of these entities has its own ID-based referencing system to ensure efficient and straightforward model setup:

- **Nodes**: Represent discrete points in the structure, defined by their unique IDs and coordinates.

- **Elements**: Connect multiple nodes to form the fundamental building blocks of a FEM model. The element definitions vary based on the chosen element type (e.g., C3D8, C3D20).
- **Surfaces**: Automatically derived from element definitions. They play a crucial role in defining constraints, loads, and boundary conditions.

1.3.3 Material Properties

Defining accurate material properties is crucial for realistic simulations. FEMaster currently supports only isotropic materials, defined using properties such as Young's Modulus, Poisson's Ratio, and density. While other material types, including orthotropic and hyperelastic formulations, are planned for future versions, isotropic materials provide a solid foundation for most structural mechanics problems.

1.3.4 Constraints

FEMaster includes two primary types of constraints: **tie constraints** and **kinematic coupling constraints**. Tie constraints enable the rigid attachment of two surfaces, ensuring that they move together as a single unit. Kinematic coupling constraints tie the movement of a set of nodes to the movement of a reference node, allowing for coordinated motion.

1.3.5 Steps and Analysis Types

FEMaster supports the following step types for different types of analyses:

- **Linear Static Step**: This is the standard step type for performing static structural analysis under applied loads. It computes displacements and stresses for the given set of boundary conditions and constraints.
- **Linear Static Topology Step**: Similar to the linear static step, but it accepts element densities as input and returns optimized densities along with the sensitivity gradients for each element.
- **Eigenfrequency Step**: Solves for the natural frequencies and mode shapes of the structure. Currently, this step does not support constraints, but future updates are planned to include this capability.

1.3.6 Load Application

To define loads, FEMaster provides the following commands:

- **VLOAD**: Defines a volumetric load applied to a set of elements. Useful for modeling body forces such as gravitational effects.
- **DLOAD**: Defines a surface pressure load on element surfaces. Ideal for applying uniform or varying pressure distributions.
- **CLOAD**: Defines a concentrated point load applied directly to a node.

1.4 Why Use FEMaster?

FEMaster offers a balance between simplicity and power. It enables users to set up simulations with minimal configuration while still providing advanced capabilities like custom constraints and a robust material modeling framework. The tool is well-suited for:

- **Academic Research**: Its open-source nature and lightweight architecture make it ideal for extending and adapting to new element formulations or custom simulation techniques.
- **Industry Applications**: With support for multi-threading, FEMaster can quickly solve complex models, making it suitable for prototyping and optimization tasks.
- **Education**: FEMaster's clear syntax and straightforward configuration make it an excellent tool for teaching the principles of finite element analysis.

Chapter 2

Geometric Entities

FEMaster is built around three core geometric entities: **nodes**, **elements**, and **surfaces**. These entities serve as the building blocks for defining the geometry and topology of a finite element model. Each entity has a unique ID-based referencing system that allows for efficient and flexible model setup. This chapter details the structure, definition, and properties of each geometric entity, including their usage and syntax in the input file.

2.1 Nodes (*NODE)

Nodes are the fundamental points that define the geometry of a finite element model. Each node is associated with a set of coordinates that determine its position in space. Nodes are referenced by a unique ID and are used to construct elements and define boundary conditions. Additionally, each node can have rotational degrees of freedom (DOFs), although these are not specified directly in the node definitions. The translational DOFs are defined by the provided x, y, and z coordinates, while rotational DOFs are automatically included in the solution process if the specific element or constraint requires them.

2.1.1 Syntax

Nodes are defined using the following format in the input file:

```
*NODE, NSET=NODES
1, 0.0, 0.0, 0.0
2, 1.0, 0.0, 0.0
3, 0.0, 1.0, 0.0
4, 1.0, 1.0, 0.0
```

The first value corresponds to the node ID, followed by its x, y, and z coordinates. The optional parameter 'NSET' is used to group nodes into logical sets, which can be referenced later in the input file.
Values for node coordinates can be omitted if not explicitly needed, using commas to separate them. For example:

```
*NODE
5, , 5.0,
```

This syntax creates a node with ID 5, $x = 0$, $y = 5$, and $z = 0$. Omitted values are treated as zeros by default.

2.1.2 Node Sets (*NSET)

Node sets can also be created manually using the 'NSET' command, which allows for flexible grouping of node IDs. Both of the following commands are valid and equivalent:

```
*NSET, NAME=NODE_GROUP
1, 2, 3, 4
```

```
*NSET, NSET=NODE_GROUP
1, 2, 3, 4
```

Node sets can then be referenced in constraints, boundary conditions, or other commands that require specifying a group of nodes.

2.1.3 Rotational Degrees of Freedom

Even though the user only provides the translational coordinates in the input file, FEMaster supports rotational DOFs for nodes. These rotational DOFs are utilized in the solution when elements or constraints involve rotational terms, such as beam or shell elements. The results output file may include both translational and rotational solutions for each node, depending on the analysis type and the defined elements.

2.2 Elements (*ELEMENT)

Elements connect multiple nodes to form the primary components of a finite element model. Each element represents a region of the model's geometry, allowing for the computation of properties such as displacements, stresses, and strains. FEMaster supports a variety of solid elements for 3D analysis, providing flexibility and accuracy depending on the chosen element type.

2.2.1 Element Types

FEMaster currently supports the following 3D element types:

- **C3D4**: 4-node linear tetrahedral element.
- **C3D6**: 6-node linear triangular prism (wedge) element.
- **C3D8**: 8-node linear hexahedral (brick) element.
- **C3D10**: 10-node quadratic tetrahedral element.
- **C3D15**: 15-node quadratic triangular prism (wedge) element.
- **C3D20**: 20-node quadratic hexahedral (brick) element.
- **C3D20R**: 20-node reduced integration hexahedral element.

2.2.2 Syntax

Elements are defined using the '*ELEMENT' keyword in the input file, followed by the element type. Each new line will then define elements by an id and the node IDs that define the element. The format is as follows:

```
*ELEMENT, TYPE=C3D8, ELSET=PART1
1, 1, 2, 3, 4, 5, 6, 7, 8
2, 2, 6, 7, 3, 10, 11, 12, 8
```

For some elements, the amount of nodes can be excessively large, and the input file can become cluttered. To avoid this, line breaks during element definitions is allowed, as shown below:

```
*ELEMENT, TYPE=C3D20, ELSET=PART2
1, 1, 2, 3, 4, 5, 6, 7, 8,
   9, 10, 11, 12, 13, 14, 15, 16,
    17, 18, 19, 20
```

2.2.3 Element Sets (*ELSET)

Element sets can also be created manually using the 'ELSET' command, which allows for flexible grouping of element IDs. Both of the following commands are valid and equivalent:

```
*ELSET, NAME=ELEMENT_GROUP
1, 2, 3, 4
```

```
*ELSET, ELSET=ELEMENT_GROUP
1, 2, 3, 4
```

Element sets can then be referenced in constraints, boundary conditions, or other commands that require specifying a group of elements.

2.2.4 Theory

2.2.4.1 Element Definition and Shape Functions

A general element in FEMaster is defined using nodal coordinates and shape functions. Shape functions are mathematical functions used to interpolate the displacement, strain, or stress fields within an element. They are defined in terms of local coordinates (r, s, t), which map the element's geometry to a standard reference element. The shape functions are expressed as $N_i(r, s, t)$, where i refers to the shape function associated with the i-th node.
For an element with n nodes, the position of any point within the element can be interpolated using the nodal coordinates $\mathbf{x}_i$ and the shape functions $N_i(r, s, t)$:

$$\mathbf{x}(r, s, t) = \sum_{i=1}^{n} N_i(r, s, t)\, \mathbf{x}_i$$

where:

- $\mathbf{x}(r, s, t)$ is the global position of the point defined by local coordinates (r, s, t).
- $\mathbf{x}_i$ are the global coordinates of the i-th node.
- $N_i(r, s, t)$ are the shape functions, which vary depending on the element type.

2.2.4.2 Types of Shape Functions

FEMaster follows the general principles outlined in the Abaqus theory manual for defining shape functions. Depending on the element type, the shape functions can be linear (e.g., C3D4, C3D8) or quadratic (e.g., C3D10, C3D20). Linear shape functions provide a simpler interpolation scheme and are often used for coarse meshes, while quadratic shape functions offer improved accuracy for complex geometries.

2.2.4.3 Element Stiffness Matrix Computation

The element stiffness matrix $\mathbf{K}_e$ is a fundamental component in the finite element method. It describes the resistance of the element to deformation and is derived from the strain-displacement and material property relationships. The stiffness matrix for an element is computed as:

$$\mathbf{K}_e = \int_V \mathbf{B}^T \mathbf{D} \mathbf{B}\, dV$$

where:

- $\mathbf{B}$ is the strain-displacement matrix, derived from the shape function derivatives.
- $\mathbf{D}$ is the material stiffness matrix, which depends on material properties such as Young's modulus and Poisson's ratio.
- dV is the differential volume element, calculated as $dV = |\det(\mathbf{J})|\, dr\, ds\, dt$.
- $\mathbf{J}$ is the Jacobian matrix, defined by:

$$\mathbf{J} = \begin{bmatrix} \frac{\partial x}{\partial r} & \frac{\partial y}{\partial r} & \frac{\partial z}{\partial r} \\ \frac{\partial x}{\partial s} & \frac{\partial y}{\partial s} & \frac{\partial z}{\partial s} \\ \frac{\partial x}{\partial t} & \frac{\partial y}{\partial t} & \frac{\partial z}{\partial t} \end{bmatrix}$$

The Jacobian matrix transforms the local element coordinates (r, s, t) to global coordinates (x, y, z), allowing for the accurate evaluation of the volume element and other quantities.

2.2.4.4 Numerical Integration for Stiffness Matrix

For complex elements, the stiffness matrix is typically evaluated using numerical integration techniques such as Gaussian quadrature. The integral is approximated by a sum over a set of sample points (r_i, s_i, t_i) with corresponding weights w_i:

$$\mathbf{K}_e \approx \sum_{i=1}^{n} \mathbf{B}^T(r_i, s_i, t_i)\, \mathbf{D}\, \mathbf{B}(r_i, s_i, t_i)\, w_i\ |\det(\mathbf{J}(r_i, s_i, t_i))|$$

This formula enables the computation of the stiffness matrix even for elements with complex shapes and material properties, ensuring accurate representation of the element's behavior.

2.2.4.5 Summary of Stiffness Matrix Computation Steps

1. **Compute Shape Function Derivatives**: Calculate the first derivatives of the shape functions $\frac{\partial N_i}{\partial r}$, $\frac{\partial N_i}{\partial s}$, $\frac{\partial N_i}{\partial t}$.
2. **Calculate the Jacobian Matrix**: Use the shape function derivatives and nodal coordinates to form the Jacobian matrix $\mathbf{J}$.
3. **Evaluate the Strain-Displacement Matrix**: Compute the strain-displacement matrix $\mathbf{B}$ using the Jacobian matrix.
4. **Form the Stiffness Matrix**: Integrate $\mathbf{B}^T\mathbf{D}\mathbf{B}$ over the volume using numerical integration points.
5. **Assemble into the Global Stiffness Matrix**: Once the element stiffness matrices are computed, they are assembled into the global stiffness matrix for the entire model.

The elements implemented with the amount of integration points is listed below:

- **C3D4**: 1 integration point
- **C3D6**: 2 integration points
- **C3D8**: 8 integration points
- **C3D10**: 4 integration points
- **C3D15**: 9 integration points
- **C3D20**: 27 integration points
- **C3D20R**: 8 integration points

2.3 Surfaces (*SURFACE)

Surfaces are specialized geometric entities derived from the faces of 3D elements. They are used to apply boundary conditions, loads, and constraint definitions. Unlike nodes and elements, surface types (triangular or quadrilateral) cannot be directly chosen by the user. Instead, they are automatically created based on the connectivity of the underlying element and its associated face.
FEMaster creates the following surface types based on the element face configuration:

- **Surface3**: A 3-node triangular surface created from a triangular face of tetrahedral or wedge elements.
- **Surface4**: A 4-node quadrilateral surface created from a quadrilateral face of hexahedral or wedge elements.
- **Surface6**: A 6-node quadratic triangular surface created from a higher-order triangular face with mid-side nodes.
- **Surface8**: An 8-node quadratic quadrilateral surface created from a higher-order quadrilateral face with mid-side nodes.

2.3.1 Syntax

Surfaces are defined using the '*SURFACE' keyword in the input file. The format is as follows:

```
*SURFACE, SFSET=MASTER_SURFACE
1, 1, S2
2, 2, S3
```

Each surface definition includes:

- A unique surface ID.
- The element ID of the parent element.
- The element side ID, which specifies which face of the element is used (e.g., 'S1', 'S2', 'S3').

2.3.2 Surface Sets (*SFSET)

Surface sets can also be created manually using the 'SFSET' command, which allows for flexible grouping of surface IDs. Both of the following commands are valid and equivalent:

```
*SFSET, NAME=SURFACE_GROUP
1, 2, 3, 4
```

```
*SFSET, SFSET=SURFACE_GROUP
1, 2, 3, 4
```

Surface sets can then be referenced in constraints, boundary conditions, or other commands that require specifying a group of surfaces.

2.3.3 Theory

2.3.3.1 Shape Functions

Surfaces define the 3d point using interpolation of the corner nodes based on the local coordinates (r, s). The interpolation works by defining a shape function for each corner node and then interpolating the point using these shape functions. Sampling at some random point on the surface works by:

$$x(r,s) = \sum_{i=1}^{n} N_i(r,s) x_i$$

where x_i are the coordinates of the corner nodes and $N_i(r,s)$ are the shape functions. The surface type (e.g., Surface3, Surface4) is automatically determined based on the number of nodes on the selected element face.
Each surface type in FEMaster uses specific shape functions to interpolate values across its local coordinate system. These shape functions are crucial for defining displacement fields and mapping between local and global coordinates. Below is a detailed description of the shape functions, along with their first and second derivatives for each supported surface type.

Surface3 For a 3-node triangular surface, the shape functions are defined as:

$$N_1(r,s) = 1 - r - s, \quad N_2(r,s) = r, \quad N_3(r,s) = s$$

The first derivatives are:

$$\frac{\partial N}{\partial r} = \begin{bmatrix} -1 \\ 1 \\ 0 \end{bmatrix}, \quad \frac{\partial N}{\partial s} = \begin{bmatrix} -1 \\ 0 \\ 1 \end{bmatrix}$$

The second derivatives are:

$$\frac{\partial^2 N}{\partial r^2} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}, \quad \frac{\partial^2 N}{\partial s^2} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}, \quad \frac{\partial^2 N}{\partial r \partial s} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

Surface4 For a 4-node quadrilateral surface, the shape functions are defined as:

$$\begin{aligned} N_1(r,s) &= 0.25(1-r)(1-s) \\ N_2(r,s) &= 0.25(1+r)(1-s) \\ N_3(r,s) &= 0.25(1+r)(1+s) \\ N_4(r,s) &= 0.25(1-r)(1+s) \end{aligned}$$

The first derivatives are:

$$\frac{\partial N}{\partial r} = \begin{bmatrix} -0.25(1-s) \\ 0.25(1-s) \\ 0.25(1+s) \\ -0.25(1+s) \end{bmatrix}, \quad \frac{\partial N}{\partial s} = \begin{bmatrix} -0.25(1-r) \\ -0.25(1+r) \\ 0.25(1+r) \\ 0.25(1-r) \end{bmatrix}$$

The second derivatives are:

$$\frac{\partial^2 N}{\partial r^2} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}, \quad \frac{\partial^2 N}{\partial s^2} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}, \quad \frac{\partial^2 N}{\partial r \partial s} = \begin{bmatrix} 0.25 \\ -0.25 \\ 0.25 \\ -0.25 \end{bmatrix}$$

Surface6 For a 6-node quadratic triangular surface, the shape functions are defined as:

$$\begin{aligned} N_1(r,s) &= 1 - 3(r+s) + 2(r+s)^2 \\ N_2(r,s) &= r(2r-1) \\ N_3(r,s) &= s(2s-1) \\ N_4(r,s) &= 4r(1-r-s) \\ N_5(r,s) &= 4rs \\ N_6(r,s) &= 4s(1-r-s) \end{aligned}$$

The first derivatives are:

$$\frac{\partial N}{\partial r} = \begin{bmatrix} -3+4(r+s) \\ 4r-1 \\ 0 \\ 4-8r-4s \\ 4s \\ -4s \end{bmatrix}, \quad \frac{\partial N}{\partial s} = \begin{bmatrix} -3+4(r+s) \\ 0 \\ 4s-1 \\ -4r \\ 4r \\ 4-4r-8s \end{bmatrix}$$

The second derivatives are:

$$\frac{\partial^2 N}{\partial r^2} = \begin{bmatrix} 4 \\ 4 \\ 0 \\ -8 \\ 0 \\ 0 \end{bmatrix}, \quad \frac{\partial^2 N}{\partial s^2} = \begin{bmatrix} 4 \\ 0 \\ 4 \\ 0 \\ 0 \\ -8 \end{bmatrix}, \quad \frac{\partial^2 N}{\partial r \partial s} = \begin{bmatrix} 4 \\ 0 \\ 0 \\ -4 \\ 4 \\ -4 \end{bmatrix}$$

Surface8 For an 8-node quadratic quadrilateral surface, the shape functions are defined as:

$$\begin{aligned} N_1(r,s) &= 0.25(1-r)(1-s)(-1-r-s) \\ N_2(r,s) &= 0.25(1+r)(1-s)(-1+r-s) \\ N_3(r,s) &= 0.25(1+r)(1+s)(-1+r+s) \\ N_4(r,s) &= 0.25(1-r)(1+s)(-1-r+s) \\ N_5(r,s) &= 0.5(1-r^2)(1-s) \\ N_6(r,s) &= 0.5(1+r)(1-s^2) \\ N_7(r,s) &= 0.5(1-r^2)(1+s) \\ N_8(r,s) &= 0.5(1-r)(1-s^2) \end{aligned}$$

The first derivatives are:

$$\frac{\partial N}{\partial r} = \begin{bmatrix} 0.25(-2r-s)(s-1) \\ 0.25(-2r+s)(s-1) \\ 0.25(2r+s)(s+1) \\ 0.25(2r-s)(s+1) \\ r(s-1) \\ 0.5(1-s^2) \\ -r(1+s) \\ 0.5(s^2-1) \end{bmatrix}, \quad \frac{\partial N}{\partial s} = \begin{bmatrix} 0.25(-r-2s)(r-1) \\ 0.25(-r+2s)(r+1) \\ 0.25(r+2s)(r+1) \\ 0.25(r-2s)(r-1) \\ 0.5(r^2-1) \\ -s(r+1) \\ 0.5(1-r^2) \\ s(r-1) \end{bmatrix}$$

The second derivatives are:

$$\frac{\partial^2 N}{\partial r^2} = \begin{bmatrix} 0.5(1-s) \\ 0.5(1-s) \\ 0.5(s+1) \\ 0.5(s+1) \\ s-1 \\ 0 \\ -(s+1) \\ 0 \end{bmatrix}, \quad \frac{\partial^2 N}{\partial s^2} = \begin{bmatrix} 0.5(1-r) \\ 0.5(r+1) \\ 0.5(r+1) \\ 0.5(1-r) \\ 0 \\ -(r+1) \\ 0 \\ r-1 \end{bmatrix}, \quad \frac{\partial^2 N}{\partial r \partial s} = \begin{bmatrix} -0.5(r+s)+0.25 \\ -0.5(r-s)-0.25 \\ 0.5(r+s)+0.25 \\ 0.5(r-s)-0.25 \\ r \\ -s \\ -r \\ s \end{bmatrix}$$

2.3.3.2 Mapping Points to the Surface

Mapping a global point onto a surface involves finding the local (r, s) coordinates that minimize the distance to the point. FEMaster uses a Newton-Raphson method to iteratively find the best fit (r, s) values. The procedure involves calculating the first and second derivatives of the shape functions for the given surface type:

$$\frac{\partial N_i}{\partial r}, \quad \frac{\partial N_i}{\partial s}, \quad \frac{\partial^2 N_i}{\partial r^2}, \quad \frac{\partial^2 N_i}{\partial s^2}, \quad \frac{\partial^2 N_i}{\partial r \partial s}$$

These derivatives are used to construct a system of equations that minimize the residual distance between the surface and the target point. Boundary edges are represented by internal first and second-order line elements that are used to check if the Newton iteration goes out of bounds. In such cases, the algorithm switches to a boundary search using these line elements. The code implements the following pseudo-code to map a point to the surface:

Algorithm 1: Mapping Global Point to Surface

Input: Global point $\mathbf{p}$, Node coordinates node_coords_global, Element type N, Boolean *clip*
Output: Local coordinates (r, s) minimizing distance to $\mathbf{p}$

Step 1: Initialize
starting_coords_list $\leftarrow$ pick_starting_coords(N) ;
max_iter $\leftarrow 32$;
eps $\leftarrow 10^{-12}$;
min_distance_squared $\leftarrow \infty$;

Step 2: Perform Newton-Raphson Iteration for Each Initial Guess
for *each initial_guess* $\in$ *starting_coords_list* **do**
  $r, s \leftarrow$ initial_guess ;
  for *iter* $= 0$ ***to*** *max_iter* **do**
    Compute shape functions and derivatives ;
    $\mathbf{N} \leftarrow$ shape_function(r, s) ;
    $\frac{\partial \mathbf{N}}{\partial r}, \frac{\partial \mathbf{N}}{\partial s} \leftarrow$ shape_derivatives(r, s) ;
    $\frac{\partial^2 \mathbf{N}}{\partial r^2}, \frac{\partial^2 \mathbf{N}}{\partial s^2}, \frac{\partial^2 \mathbf{N}}{\partial r \partial s} \leftarrow$ shape_second_derivative(r, s) ;
    Compute surface position and derivatives ;
    $\mathbf{x}_{rs} = \sum_{i=1}^{N} \mathbf{x}_i N_i, \quad \frac{\partial \mathbf{x}}{\partial r} = \sum_{i=1}^{N} \mathbf{x}_i \frac{\partial N_i}{\partial r}, \quad \frac{\partial \mathbf{x}}{\partial s} = \sum_{i=1}^{N} \mathbf{x}_i \frac{\partial N_i}{\partial s}$, ;
    $\frac{\partial^2 \mathbf{x}}{\partial r^2} = \sum_{i=1}^{N} \mathbf{x}_i \frac{\partial^2 N_i}{\partial r^2}, \quad \frac{\partial^2 \mathbf{x}}{\partial s^2} = \sum_{i=1}^{N} \mathbf{x}_i \frac{\partial^2 N_i}{\partial s^2}, \quad \frac{\partial^2 \mathbf{x}}{\partial r \partial s} = \sum_{i=1}^{N} \mathbf{x}_i \frac{\partial^2 N_i}{\partial r \partial s}$;
    Compute distance and its derivatives ;
    diff $\leftarrow \mathbf{x}_{rs} - \mathbf{p}$;
    $\frac{\partial D}{\partial r} \leftarrow$ **diff** $\cdot \frac{\partial \mathbf{x}}{\partial r}$;
    $\frac{\partial D}{\partial s} \leftarrow$ **diff** $\cdot \frac{\partial \mathbf{x}}{\partial s}$;
    Compute the Hessian matrix ;
    $H_{11} \leftarrow \frac{\partial \mathbf{x}}{\partial r} \cdot \frac{\partial \mathbf{x}}{\partial r} +$ **diff** $\cdot \frac{\partial^2 \mathbf{x}}{\partial r^2}$;
    $H_{22} \leftarrow \frac{\partial \mathbf{x}}{\partial s} \cdot \frac{\partial \mathbf{x}}{\partial s} +$ **diff** $\cdot \frac{\partial^2 \mathbf{x}}{\partial s^2}$;
    $H_{12} \leftarrow \frac{\partial \mathbf{x}}{\partial r} \cdot \frac{\partial \mathbf{x}}{\partial s} +$ **diff** $\cdot \frac{\partial^2 \mathbf{x}}{\partial r \partial s}$;
    Solve the linear system ;
    $\Delta r, \Delta s \leftarrow H^{-1} \cdot \left[-\frac{\partial D}{\partial r}, -\frac{\partial D}{\partial s}\right]^T$;
    Update coordinates ;
    $r \leftarrow r + \Delta r$;
    $s \leftarrow s + \Delta s$;
    if $\|\Delta r, \Delta s\| <$ *eps* **then**
      break ;
  Check if (r, s) **is within bounds and update best coordinates** ;
  if *in_bounds*(r, s) ***or not*** *clip* **then**
    distance_squared $\leftarrow \|\mathbf{x}_{rs} - \mathbf{p}\|^2$;
    if *distance_squared* $<$ *min_distance_squared* **then**
      min_distance_squared $\leftarrow$ distance_squared ;
      best_coords $\leftarrow (r, s)$;
if *clip* **then**
  best_coords_edge $\leftarrow$ closest_point_on_boundary($\mathbf{p}$, node_coords_global) ;
  distance_squared $\leftarrow \|$local_to_global(best_coords_edge) $- \mathbf{p}\|^2$;
  if *distance_squared* $<$ *min_distance_squared* **then**
    best_coords $\leftarrow$ best_coords_edge ;
return *best_coords* ;

2.3.3.3 Integration across the surface

The integration across the surface is done using the Gaussian quadrature method. Based on the element, an appropriate number of integration points are chosen which can accurately integrate the respective shape functions. The amount of integration points is displayed in the following list:

- **Surface3**: 1 integration point
- **Surface4**: 1 integration points
- **Surface6**: 3 integration points
- **Surface8**: 4 integration points

Integration is then carried out by:

$$\int_{\text{surface}} f(\mathbf{x})\, dA \approx \sum_{i=1}^{n} f(\mathbf{x}(r_i, s_i))\, w_i \left| \frac{\partial \mathbf{x}}{\partial r} \times \frac{\partial \mathbf{x}}{\partial s} \right|_{(r_i, s_i)}$$

Chapter 3

Material Properties

Material properties are fundamental to the analysis and simulation of mechanical systems. They define how materials respond to external loads and are crucial in predicting the behavior of structures under various conditions. This chapter discusses the implementation of material models, focusing on isotropic and orthotropic elasticity. Currently, our system supports isotropic and orthotropic materials; however, orthotropic materials cannot be rotated yet.

3.1 Isotropic Elasticity (*ELASTIC, TYPE=ISOTROPIC)

An isotropic material has identical properties in all directions. This uniformity simplifies the stress-strain relationship, making isotropic materials widely used in engineering applications.

3.1.1 Constitutive Equations

The constitutive equation for isotropic elasticity relates stress $\boldsymbol{\sigma}$ and strain $\boldsymbol{\varepsilon}$ through Hooke's Law:

$$\boldsymbol{\sigma} = \mathbf{C} : \boldsymbol{\varepsilon}$$

where $\mathbf{C}$ is the fourth-order elasticity tensor. In matrix form for three-dimensional analysis, the stress-strain relationship becomes:

$$\begin{bmatrix} \sigma_{xx} \\ \sigma_{yy} \\ \sigma_{zz} \\ \sigma_{xy} \\ \sigma_{yz} \\ \sigma_{zx} \end{bmatrix} = \begin{bmatrix} C_{11} & C_{12} & C_{12} & 0 & 0 & 0 \\ C_{12} & C_{11} & C_{12} & 0 & 0 & 0 \\ C_{12} & C_{12} & C_{11} & 0 & 0 & 0 \\ 0 & 0 & 0 & C_{44} & 0 & 0 \\ 0 & 0 & 0 & 0 & C_{44} & 0 \\ 0 & 0 & 0 & 0 & 0 & C_{44} \end{bmatrix} \begin{bmatrix} \varepsilon_{xx} \\ \varepsilon_{yy} \\ \varepsilon_{zz} \\ 2\varepsilon_{xy} \\ 2\varepsilon_{yz} \\ 2\varepsilon_{zx} \end{bmatrix}$$

The stiffness coefficients are defined as:

$$C_{11} = \frac{E(1-\nu)}{(1+\nu)(1-2\nu)}$$
$$C_{12} = \frac{E\nu}{(1+\nu)(1-2\nu)}$$
$$C_{44} = \frac{E}{2(1+\nu)}$$

3.1.2 Syntax

To define an isotropic material in the input file, use the following syntax:

```
*MATERIAL, NAME=STEEL
*ELASTIC, TYPE=ISOTROPIC
```

```
E, v
*DENSITY
1
```

3.2 Orthotropic Elasticity (*ELASTIC, TYPE=ORTHOTROPIC)

Orthotropic materials have different properties along three mutually perpendicular axes. This anisotropy is common in composite materials and requires a more complex constitutive model.

3.2.1 Constitutive Equations

The stress-strain relationship for orthotropic materials in 3D is given by:

$$\boldsymbol{\sigma} = \mathbf{C}\boldsymbol{\varepsilon}$$

The stiffness matrix $\mathbf{C}$ for orthotropic materials is:

$$\mathbf{C} = \begin{bmatrix} C_{11} & C_{12} & C_{13} & 0 & 0 & 0 \\ C_{12} & C_{22} & C_{23} & 0 & 0 & 0 \\ C_{13} & C_{23} & C_{33} & 0 & 0 & 0 \\ 0 & 0 & 0 & C_{44} & 0 & 0 \\ 0 & 0 & 0 & 0 & C_{55} & 0 \\ 0 & 0 & 0 & 0 & 0 & C_{66} \end{bmatrix}$$

The stiffness coefficients are calculated using the material constants:

$$\begin{aligned} C_{11} &= \frac{1 - v_{23}v_{32}}{E_1 D} \\ C_{22} &= \frac{1 - v_{31}v_{13}}{E_2 D} \\ C_{33} &= \frac{1 - v_{12}v_{21}}{E_3 D} \\ C_{12} &= \frac{v_{21} + v_{31}v_{23}}{E_1 D} \\ C_{13} &= \frac{v_{31} + v_{21}v_{32}}{E_1 D} \\ C_{23} &= \frac{v_{32} + v_{31}v_{12}}{E_2 D} \\ C_{44} &= G_{23} \\ C_{55} &= G_{31} \\ C_{66} &= G_{12} \end{aligned}$$

where:

$$D = 1 - v_{12}v_{21} - v_{23}v_{32} - v_{31}v_{13} - 2v_{12}v_{23}v_{31}$$

3.2.2 Syntax

Define an orthotropic material using:

```
*MATERIAL, NAME=COMPOSITE
*ELASTIC, TYPE=ORTHOTROPIC
E1, E2, E3, G23, G31, G12, v23, v31, v12
*DENSITY
\rho
```

3.2.3 Limitations

Currently, orthotropic materials cannot be rotated. The principal material axes must align with the global coordinate system. Future updates may include the ability to define material orientations.

3.3 Conclusion

The accurate representation of material properties is essential for reliable simulation results. Isotropic materials offer simplicity and are suitable for homogeneous materials like metals. Orthotropic materials provide the flexibility to model anisotropic behaviors found in composites and wood. Understanding the constitutive equations and their implementation allows for the extension and customization of material models to meet specific analysis requirements.
Future developments will focus on enhancing the orthotropic material model by enabling rotation of material axes, allowing for more complex and realistic simulations of anisotropic materials.

Chapter 4

Constraints

Constraints in finite element analysis (FEA) are a critical component for defining how different regions of a model interact or how boundaries influence the system. They enforce certain relationships between nodes, elements, or surfaces to ensure that the solution adheres to predefined conditions, allowing complex interactions like load transfer or restricted movement. This chapter covers the two main types of constraints currently implemented: kinematic couplings and tie constraints.

4.1 Kinematic Couplings (*COUPLING, TYPE=KINEMATIC)

Kinematic couplings are used to control the motion of a group of nodes (the *slave set*) by connecting them to a single reference node (the *master set*). This type of constraint is often employed to model rigid body motion, where all nodes in the slave set move as if they are rigidly connected to the master node, preserving relative distances and rotations.

4.1.1 Kinematic Coupling Equations

The kinematic coupling equations ensure that each node $\mathbf{x}_i$ in the slave set follows the motion of the master node $\mathbf{x}_m$. This relationship can be expressed as:

$$\mathbf{x}_i = \mathbf{x}_m + \mathbf{R}_m \cdot \mathbf{r}_i$$

where:

- $\mathbf{x}_i$ is the position of the i-th slave node.
- $\mathbf{x}_m$ is the position of the master node.
- $\mathbf{R}_m$ is the rotation matrix defining the orientation of the master node.
- $\mathbf{r}_i$ is the relative position vector of the slave node in the local coordinate system of the master node.

The constraint can also be written in terms of displacements and rotations:

$$\mathbf{u}_i = \mathbf{u}_m + \boldsymbol{\omega}_m \times \mathbf{r}_i$$

where:

- $\mathbf{u}_i$ is the displacement of the slave node.
- $\mathbf{u}_m$ is the displacement of the master node.
- $\boldsymbol{\omega}_m$ is the rotation vector (angular displacement) of the master node.

4.1.2 Syntax

To define a kinematic coupling in the input file, use the following syntax:

```
*COUPLING, TYPE=KINEMATIC, MASTER=MASTER_SET, SLAVE=SLAVE_SET
1, 1, 1, , 1, 1
```

Both the master and slave must be defined as sets. The master set must contain exactly one node. The line following the '*COUPLING' command indicates which degrees of freedom (DOF) to couple, using 1 to include the DOF and leaving empty values to exclude specific DOFs. The DOFs are defined in the order: $u_x, u_y, u_z, \theta_x, \theta_y, \theta_z$. For example, to exclude rotation around the x-axis, the line would be: '1, 1, 1, , 0, 1'.

4.2 Tie Constraints (*TIE)

Tie constraints are used to connect two regions of a model by tying the motion of a *slave set* of nodes to a *master surface*. This constraint is often used in contact simulations or to couple non-matching meshes, where the nodes of the slave set may not correspond directly to the nodes of the master surface. Tie constraints ensure continuity of displacement across the interface, allowing for load transfer and interaction between regions.

4.2.1 Tie Constraint Equations

The tie constraint enforces that the displacement of a slave node matches the interpolated displacement of the master surface at the corresponding closest point:

$$\mathbf{u}_s = \mathbf{u}_m(\xi, \eta)$$

If a slave node does not lie exactly on the master surface, the constraint interpolates the motion of the master surface using shape functions:

$$\mathbf{u}_m(\xi, \eta) = \sum_{i=1}^{N} N_i(\xi, \eta) \mathbf{u}_{mi}$$

where:

- $N_i(\xi, \eta)$ are the shape functions of the master surface element.
- $\mathbf{u}_{mi}$ are the displacements of the i-th node of the master surface element.
- N is the number of nodes in the master element.

4.2.2 Syntax

To define a tie constraint in the input file, use the following syntax:

```
*TIE, MASTER=SURFACE_SET, SLAVE=NODE_SET, DISTANCE=0.2, ADJUST=YES
```

The master set must be defined as a surface set, while the slave set must be defined as a node set. The 'DISTANCE' parameter specifies the maximum distance between the slave node and the master surface for the constraint to be applied. To speed up computations, the mapping is done by first comparing to the nodes of the master surface. Only surfaces which has nodes that are within the distance will be even considered. It then takes the closest surface. The 'ADJUST' parameter determines whether the slave nodes are projected onto the master surface if they are outside the specified distance.

4.3 Conclusion

Kinematic couplings and tie constraints are essential tools for defining interactions and boundary conditions in finite element models. Kinematic couplings are best suited for modeling rigid body motion or attaching regions to a reference node. Tie constraints are ideal for connecting non-matching meshes or modeling bonded contact between parts. Understanding the mathematical foundations and practical considerations of these constraints is key to building accurate and efficient simulations.
Future developments may include additional constraint types, such as contact pairs or more sophisticated multi-point constraints, to further extend the capabilities of the solver.

Chapter 5

Boundary Conditions

Boundary conditions (BCs) are essential in finite element analysis to define how a structure interacts with its environment and how loads are applied to it. This chapter explains the three main types of loads supported—'VLOAD', 'CLOAD', and 'DLOAD'—as well as the syntax for applying constraints using '*SUPPORT'. Boundary conditions and loads are grouped into *collectors* for easy management and reusability, enabling consistent application across multiple steps in an analysis.

5.1 Load Types

5.1.1 Volume Load (*VLOAD)

'VLOAD' is used to apply distributed loads to elements. This type of load is defined using three components: F_x, F_y, F_z, which represent forces acting along the x, y, and z directions, respectively. 'VLOAD' can be applied either to individual element IDs or to entire element sets. It is suitable for defining gravitational loads, body forces, or thermal forces distributed over the volume of elements.

5.1.1.1 Syntax:

```
*VLOAD, LOAD_COLLECTOR=COLLECTOR_NAME
ELEMENT_ID, F_x, F_y, F_z
ELEMENT_SET_NAME, F_x, F_y, F_z
```

Example:

```
*VLOAD, LOAD_COLLECTOR=GRAVITY
10, 0, -9.81, 0
11, 0, -9.81, 0
ELSET1, 0, -9.81, 0
```

5.1.2 Concentrated Load (*CLOAD)

'CLOAD' is used to apply concentrated loads to nodes. It supports six components: three translational forces F_x, F_y, F_z and three optional rotational moments M_x, M_y, M_z.

5.1.2.1 Syntax

```
*CLOAD, LOAD_COLLECTOR=COLLECTOR_NAME
NODE_ID, F_x, F_y, F_z, M_x, M_y, M_z
```

Example:

```
*CLOAD, LOAD_COLLECTOR=LOADS
15, 0, 0, 1
16, 0, 0, 1
17, 0, 0, 1
18, 0, 0, 1
```

5.1.3 Distributed Load (*DLOAD)

‘DLOAD‘ is used to apply distributed loads to surfaces. This load type is suitable for defining pressure loads or other distributed forces acting over surface areas.

5.1.3.1 Syntax

```
*DLOAD, LOAD_COLLECTOR=COLLECTOR_NAME
SURFACE_ID, F_x, F_y, F_z
SURFACE_SET_NAME, F_x, F_y, F_z
```

Example:

```
*DLOAD, LOAD_COLLECTOR=PRESSURE
20, 0, 0, -10
SURFSET1, 0, 0, -10
```

5.2 Boundary Conditions (*SUPPORT)

Boundary conditions specify which degrees of freedom (DOF) are fixed or constrained at a particular node. The ‘*SUPPORT‘ keyword is used to define constraints on any combination of the six available DOFs: three translational (u_x, u_y, u_z) and three rotational $(\theta_x, \theta_y, \theta_z)$.

5.2.1 Syntax

```
*SUPPORT, SUPPORT_COLLECTOR=COLLECTOR_NAME
NODE_ID, u_x, u_y, u_z, \theta_x, \theta_y, \theta_z
```

Example:

```
*SUPPORT, SUPPORT_COLLECTOR=BCS
1, 0, 0, 0, 0, 0, 0
2, 0, 0, 0, 0, 0, 0
3, 0, 0, 0, 0, 0, 0
4, 0, 0, 0, 0, 0, 0
```

5.3 Load Case Management

Once loads and supports are grouped into collectors, they can be easily referenced in different load cases within a step. Load cases define how the loads and supports are applied, enabling different scenarios.

5.3.1 Syntax

```
*LOADCASE, TYPE=LINEAR STATIC
*LOAD
LOAD_COLLECTOR_1
LOAD_COLLECTOR_2
*SUPPORT
SUPPORT_COLLECTOR_1
```

```
SUPPORT_COLLECTOR_2
```

5.4 Conclusion

Boundary conditions and loads are fundamental to setting up a finite element analysis. The use of ‘VLOAD‘, ‘CLOAD‘, and ‘DLOAD‘ allows for a wide range of loading scenarios, while the ‘*SUPPORT‘ keyword enables precise control over node constraints. By grouping these into collectors, users can efficiently manage and apply conditions across multiple steps, enhancing flexibility and reusability in complex analyses.

Chapter 6

Load Cases

FEMaster supports three types of load cases for finite element analysis: **Linear Static**, **Linear Static Topo**, and **Eigenfrequency**. Each load case type serves a distinct purpose in analyzing and optimizing structural properties under various conditions. This chapter details the different load case types, the commands that can be defined for each, and the specific results they produce.

6.1 Available Commands

Each load case in FEMaster can be customized using a set of predefined commands. This section provides an overview of the available commands and their corresponding syntax.

6.1.1 *SUPPORT

The `*SUPPORT` command defines the support conditions for nodes. Supports are used to constrain specific degrees of freedom (DOF) at selected nodes, such as fixing a node in space or preventing rotation. Support conditions are typically grouped into support collectors for easier referencing.

```
*SUPPORT
SUP_COL_1
SUP_COL_2
```

Each line references a support collector that defines the DOFs constrained for a set of nodes.

6.1.2 *LOAD

The `*LOAD` command specifies the external loads applied to nodes or elements. Loads can include forces, moments, or distributed pressures. Similar to supports, load definitions are grouped into load collectors for referencing.

```
*LOAD
LOAD_COL_1
LOAD_COL_2
```

Each line references a load collector that contains the detailed load specifications, such as forces or moments.

6.1.3 *SOLVER

The `*SOLVER` command specifies the numerical solver to use for the current load case. It can include options for direct (`DIRECT`) or iterative solvers (`INDIRECT`) and hardware preferences such as CPU (`CPU`) or GPU (`GPU`).

```
*SOLVER, METHOD=DIRECT, DEVICE=CPU
```

The example above selects a direct solver using the CPU for computations.

6.1.4 *DENSITY

The `*DENSITY` command defines the material density values for elements during topology optimization. It is used in Linear Static Topo cases to compute density gradients and compliance values. Each line specifies the element ID and its associated density value.

```
*DENSITY
el_id_1, density_1
el_id_2, density_2
el_id_3, density_3
```

This format allows setting different density values for individual elements based on their IDs.

6.1.5 *EXPONENT

The `*EXPONENT` command sets the penalization exponent for the topology optimization formulation. The exponent influences the stiffness-density relationship, affecting the optimized material distribution. A higher exponent results in a stiffer material distribution.

```
*EXPONENT
3
```

This example sets the penalization exponent to 3, commonly used in SIMP (Solid Isotropic Material with Penalization) topology optimization.

6.1.6 *NUMEIGENVALUES

The `*NUMEIGENVALUES` command specifies the number of eigenvalues to compute for an eigenfrequency analysis. This determines the number of mode shapes and natural frequencies extracted from the structure.

```
*NUMEIGENVALUES
10
```

This example sets the number of eigenvalues to 10, meaning that the solver will compute the first 10 natural frequencies and their corresponding mode shapes.

6.2 Linear Static Analysis

The Linear Static analysis `*LOAD CASE, TYPE=LINEAR STATIC` is used to compute the static response of a structure under applied loads and support conditions. This analysis type solves for displacements, strains, and stresses throughout the structure, assuming linear elastic material behavior. The following commands are supported in this load case:

- `*SUPPORT`
- `*LOAD`
- `*SOLVER`

6.2.1 Output Fields

The results of a Linear Static analysis include the following fields, which are written to the results file:

- **DISPLACEMENT**: The nodal displacement values in each degree of freedom.

- **STRAIN**: The strain values computed at each element.
- **STRESS**: The stress values computed at each element.
- **DOF_LOADS**: The external loads applied at each degree of freedom.
- **DOF_SUPPORTS**: The boundary conditions at the constrained degrees of freedom.

6.3 Linear Static Topology Optimization

The Linear Static Topology Optimization analysis (`*LOAD CASE, TYPE=LINEAR STATIC TOPO`) extends the Linear Static case by introducing additional parameters and results related to material distribution. It is used to optimize the structure's material layout, minimizing compliance or maximizing stiffness for a given volume fraction. The following commands are supported:

- `*SUPPORT`
- `*LOAD`
- `*SOLVER`
- `*DENSITY`
- `*EXPONENT`

6.3.1 Output Fields

The output for Linear Static Topo includes some of the fields from the Linear Static analysis and additional fields related to the optimization results:

- **DISPLACEMENT**: The nodal displacement values in each degree of freedom.
- **STRAIN**: The strain values computed at each element.
- **STRESS**: The stress values computed at each element.
- **COMPLIANCE_RAW**: The raw compliance values for each element.
- **COMPLIANCE_ADJ**: The adjusted compliance values after penalization.
- **DENS_GRAD**: The density gradients used in the optimization.
- **VOLUME**: The volume of each element.
- **DENSITY**: The material density distribution across the structure.

6.4 Eigenfrequency Analysis

Eigenfrequency analysis (`*LOAD CASE, TYPE=EIGENFREQUENCY`) is used to determine the natural frequencies and mode shapes of a structure. This type of analysis is essential for understanding the dynamic behavior of a structure, such as identifying resonance frequencies. The following commands are supported:

- `*SUPPORT`
- `*NUMEIGENVALUES`

6.4.1 Output Fields

The results of an Eigenfrequency analysis include the following fields:

- **MODE_SHAPE_I**: The I-th mode shape vector, representing the deformation pattern of the structure at the i-th natural frequency.
- **EIGENVALUES**: The computed eigenvalues corresponding to the squared natural frequencies of the structure.
- **EIGENFREQUENCIES**: The natural frequencies of the structure in Hz, derived from the eigenvalues.