INSA LYON

Building design: multidisciplinary approach

Optional Lecture GCU-S8-M8

Civil Engineering and Urban Planning Department

Structure part

D. BERTRAND

2023-2024

Abstract

This lesson is part of the optional modulus GCU-S8-M8 dedicated to design a **simple building** accounting for hydraulic, energetic, geotechnical and structural aspects. This document focuses on **structural design**. Students will be able to adapt a finite element code (using MATLAB) to simulate the response of a **2D beam-post structure loaded by static** or dynamic loadings (Optional). Only **elastic materials** are described. In addition, all the developments are performed under the **small strains and displacement assumptions**.

The formulation of beam finite elements (two nodes) formulated under ***Euler-Bernoulli* kinematic assumptions** is presented. It allows to calculate the **displacement field** of the structure. The **internal stresses** within the finite elements are calculated under the form of generalized stresses (internal forces axial force (N), shear force (T) and bending moment (M)).

From MNT internal efforts and displacement field, **Ultimate Limit State** (ULS) and **Serviceability Limit State** (SLS) can be used to **define cross section dimensions**. The ULS and SLS criteria depend on the slenderness of the considered structural element, the material used, the nature of the loading (axial effort, bending, shear, torsion, *etc.*).

Contents

Chapter 1

Introduction

The introduction is largely inspired from the reference book of Martin and Purkiss (1996)

Structures (made of Steel, reinforced concrete or even wood) can take a lot of shapes : Low rise and high rise buildings, bridges, geotechnic works *etc.* These structures are often essentially composed of load bearing frames and members dedicated to resist to actions coming for it environment (*e.g.* self weight, dead loads and external imposed loads such as wind, snow, traffic, earthquake, impact, *etc.*). An example is given in figure 1.1. Civil engineers have to choose how to model (or simulated) the structure response realistically. The proposed model have to describe the physics involved[1] and in the same time should not be to complex to allow fast computations and thus allow to enhance the design of structures.

Whatever the model used, the results should give a lower bound limit for the internal forces[2] in all critical elements within the structure in order to be on the safety side. It allows to perform the **Ultimate Limit State analysis** which is related to the structure strength characterization. In addition, designers need to know the **maximal displacements** and also the **natural frequencies** that can develop the structure. It allows to perform the **Serviceability analysis**. Moreover, the common mathematical material behaviour models can be elastic, elastic with redistribution, plastic (with or without second order effects due to additional moments induced by large displacements). **Within the framework of this class, only elastic mate-**

[1] Material behaviours, structure geometry, scale of description (cross-section, material point,*etc.*), *quasi*-static or dynamic framework, *etc.*.

[2] Bending moments, shear forces and axial forces.

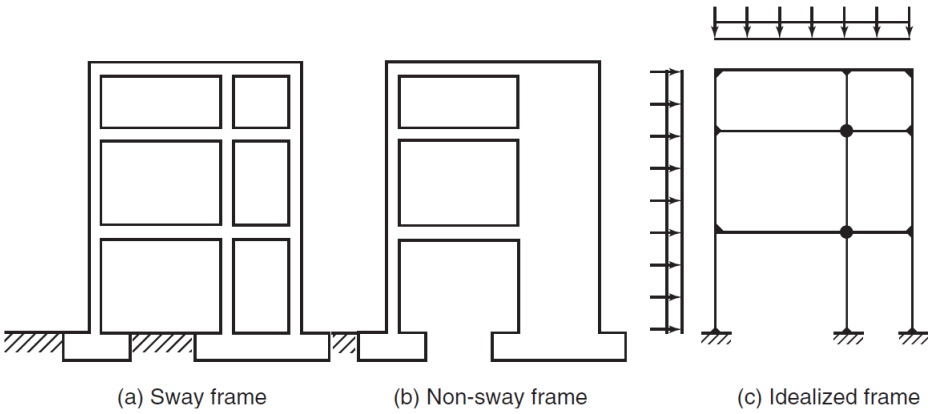

Figure 1.1: Typical load bearing frame (from Martin and Purkiss (1996)).

rials and small displacements will be considered for the calculation of the displacements and internal forces.

1.1 Limit State Design

It is rather evident that a structure should be *safe* during its lifetime, *i.e.* free from the risk of collapse. However, there are other risks associated with a structure and the term "safe" is now replaced by the term **serviceable**. During its lifetime, a structure should not become **unserviceable**, *i.e.* it should not reach a significant risk of collapse, rapid deterioration, fire, cracking, excessive deflection *etc.*

Ideally, it should be possible to assess the risk involved in structural safety based on the variation in strengths of the material and variation in terms of loading. **Eurocodes** (European standards for structural design) have introduced an elegant and powerful concept so called **structural reliability**. Methods have been devised whereby engineering judgement and experience can be combined with statistical analysis for the rational computation of partial safety factors in codes of practice. However, in the absence of complete understanding and data concerning the aspects of structural behaviour, absolute values of reliability cannot be determined. It is not conceivable, nor is it economically possible, to design a structure that will never fail. There is always a chance, as small as it can be, that the structure has

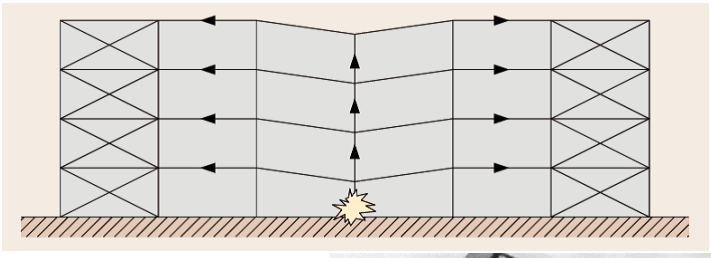

Figure 1.2: Progressive collapse example (from www.steelconstruction.info) and Ronan Point apartment building progressive collapse - 16^{th} may 1968.

been built with materials having a weaker strength than required or that it will be subject to loads greater than the expected design loads. Thus, it is communally accepted that there is a chance of 5% that the material strength of the structure is below the *design strength*, and 5% of chance that the applied loads are greater than the *design loads*. A very low probability exists that the weak material and overloading will combine simultaneously to produce collapse. Therefore, it means that structure collapse can occur but it is extremely unlikely.

The philosophy and objectives must be translated into a tangible form using calculations. A structure should be designed to be safe under all conditions of its life and to ensure that this is accomplished given performance requirements, called **limit states**. The method of limit state design recognizes the variability of loads, materials construction methods and approximations in the theory and calculations. Limit states may be at **any stage of the life of a structure**, or at any stage of loading. The limit states which are important for the design of civil structures are **ultimate and serviceability limit states**. Calculations for limit states involve loads and load factors, and material factors and strengths.

Ultimate Limit State (ULS)

Whatever the method used to determine the internal forces, the actions applied to the structure must be factored using those values pertaining to ULS, whether for single or multiple variable loads. Full plastic analysis may not be used for frames unless the plastic hinges, other than the last one to form, occur in the beams. This is because columns with high axial loads will not have any ductile capacity. This therefore means that frames are normally analysed elastically and the moment field then adjusted using redistribution to simulate plastic behaviour.

Stability, an ultimate limit state, is the ability of a structure or part of a structure, to resist overturning, overall failure and sway. Calculations should consider the worst realistic combination of loads at all stages of construction.

All structures, and parts of structures, should be **capable of resisting sway forces**, *e.g.* by the use of bracing, rigid joints or shear walls. Sway forces arise from horizontal loads, *e.g.* winds, and also from practical imperfections, *e.g.* lack of verticality.

Also involved in limit state design is the **concept of structural integrity**. Essentially this means that the structure should be tied together as a whole,

but if damage occurs, it should be localized. This was illustrated in a tower block (Figure 1.2) when a gas explosion in one flat caused the **progressive collapse** of other flats on one corner of the building.

Serviceability Limit State

In general, a serviceability analysis must be performed using an elastic analysis and contain all actions applied to the structure including any deformations or settlement effects. It is not generally necessary to consider time-dependant effects such as creep, except where they will cause substantial changes in internal forces. This last will occur in hyperstatic structures such as continuous bridge decks.

Deflection is a serviceability limit state. Deflections should not impair the efficiency of a structure or its components, nor cause damage to the finishes. Generally the worst realistic combination of unfactored imposed loads is used to calculate **elastic deflections**. These values are compared with limit states of deformation.

Dynamic effects to be considered at the serviceability limit state are vibrations caused by machines, and oscillations caused by harmonic resonance, e.g. wind gusts on buildings. The natural frequency of the building should be different from the exciting source to avoid resonance.

1.2 Contents of the class

The purpose of this document is to give to GCU students the fundamental theory to be able to model and to analyse the response of **structures composed of posts and beam under static and dynamic conditions of loadings**. The framework assumptions are

- The material behaviour is supposed elastic and linear
- The problem formulation is 2D
- The geometry does not undergo large displacements/strains

The project proposed within the optional modulus `GCU-S8-M8` aims at make the students develop their own modelling tool to be able to assess the displacements and internal forces (NTM). Based on these quantities, they are able to propose cross section dimensions to unsure the non collapse of the structure under the design loading case.

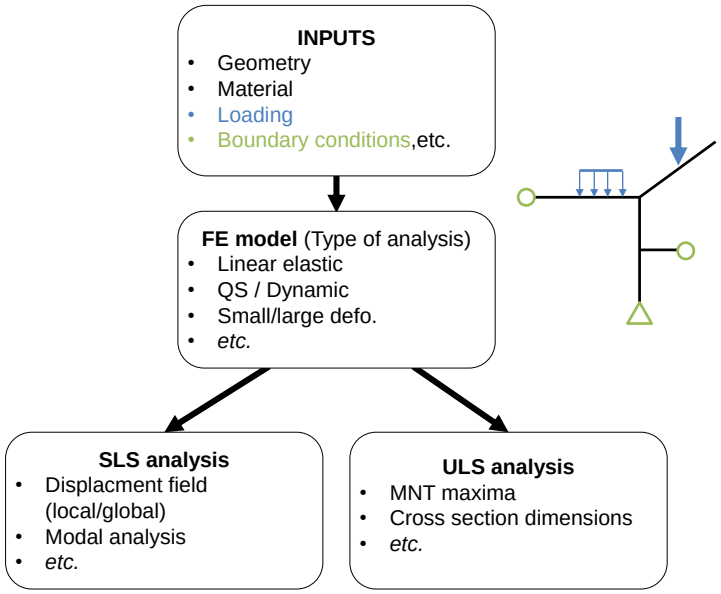

Figure 1.3: Simplified flowchart for the structural design.

In the next chapters, the analysis in static and dynamic conditions of a structure composed of beams (in a general sense) is described. The latter problems are solved by the finite element method (also called Finite Element Analysis - FEA). The formulation of *Euler-Bernoulli* finite element beams (`EBFE`) is developed, first in *quasi* static conditions and then extended to dynamic conditions. The results of the FEA is the nodal displacement field which can be directly use to perform the Serviceability analysis. The obtention of the internal forces (NTM) are derived from the knowledge of the displacement field and allows to get information to perform the ultimate limit state analysis (sizing of cross-sections function of the supposed material of the structure). Figure 1.3 presents a simplified flowchart for the structural design analysis.

The final goal of the project is to propose an **integrated design** of a simple post-beam structure which account for Hydraulic, Geotechnic, Energetic and structural aspects.

Chapter 2

FEA in static conditions

2.1 Problem formulation

The formulation of the finite element EBFE si proposed. The beginning point is the fundamental hypothesis of *Euler-Bernoulli* related to the displacement field of the cross-section. The latter which is supposed to remain plan and orthogonal to the longitudinal axis of the beam (Figure 2.1). Then, from the generalized displacement field (u_0, v_0 and β) can be obtain and thus all others quantities (strains, stresses internal forces, *etc.*). The formulation leads to the obtention of the elementary stiffness and mass matrix of the EBFE which as 3 DOF per nodes. Then the assembling of the global system allows to obtain the nodal displacements.

2.1.1 Displacements field (u_0, v_0 and β)

Under the *Euler-Bernoulli* (EB) kinematic assumptions, $u(x, y)$ and $v(x, y)$ are derived from the displacement of the centre of gravity of the cross section ($u_0(x)$ and $v_0(x)$) and the rotation $\beta(x)$ of the cross section. In addition, we know that $\beta = \frac{\partial v_0}{\partial x}$. The interpolating functions associated with the EB Beam Finite Element (EBFE) used to describe the displacement field related to the center of gravity are expressed as

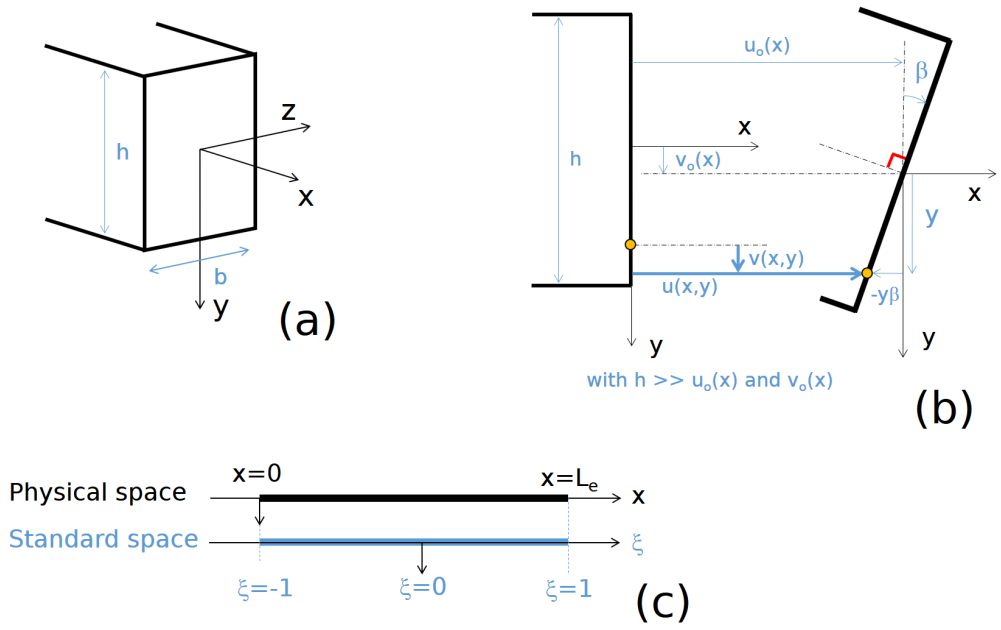

Figure 2.1: Example of rectangular cross section (a), *Euler-Bernoulli* kinematic assumption related to the cross section displacement (b) and EBFE geometry expressed in physical and standard space.

$$\begin{bmatrix} u_0(x) \\ v_0(x) \\ \beta(x) \end{bmatrix} = \begin{bmatrix} N_1(x) & 0 & 0 & N_2(x) & 0 & 0 \\ 0 & H_1(x) & H_2(x) & 0 & H_3(x) & H_4(x) \\ 0 & \frac{\partial H_1}{\partial x} & \frac{\partial H_2}{\partial x} & 0 & \frac{\partial H_3}{\partial x} & \frac{\partial H_4}{\partial x} \end{bmatrix} \begin{bmatrix} u_1 \\ v_1 \\ \beta_1 \\ u_2 \\ v_2 \\ \beta_2 \end{bmatrix} \quad (2.1)$$

where

$$\mathbb{N} = \begin{bmatrix} N_1 & 0 & 0 & N_2 & 0 & 0 \\ 0 & H_1 & H_2 & 0 & H_3 & H_4 \\ 0 & H_1' & H_2' & 0 & H_3' & H_4' \end{bmatrix} \quad (2.2)$$

is the matrix of interpolating functions and where $q_e^T = [u_1\, v_1\, \beta_1\, u_2\, v_2\, \beta_2]$ is the nodal displacement vector (two nodes called 1 and 2) of the EBFE, L_e is it length. We get finally

$$\begin{cases} N_1(x) = 1 - \frac{x}{L_e} \\ H_1(x) = 1 - \frac{3x^2}{L_e^2} + \frac{2x^3}{L_e^3} \\ H_2(x) = x - \frac{2x^2}{L_e} + \frac{x^3}{L_e^2} \\ N_2(x) = \frac{x}{L_e} \\ H_3(x) = \frac{3x^2}{L_e^2} - \frac{2x^3}{L_e^3} \\ H_4(x) = -\frac{x^2}{L_e} - \frac{x^3}{L_e^2} \end{cases} \quad (2.3)$$

with the associated derivatives (first and second)

$$\begin{cases} N_1'(x) = -\frac{1}{L_e} \\ H_1'(x) = -\frac{6x}{L_e^2} + \frac{6x^2}{L_e^3} \\ H_2'(x) = 1 - \frac{4x}{L_e} + \frac{3x^2}{L_e^2} \\ N_2'(x) = \frac{1}{L_e} \\ H_3'(x) = \frac{6x}{L_e^2} - \frac{6x^2}{L_e^3} \\ H_4'(x) = -\frac{2x}{L_e} + \frac{3x^2}{L_e^2} \end{cases} \quad \begin{cases} N_1''(x) = 0 \\ H_1''(x) = -\frac{6}{L_e^2} + \frac{12x}{L_e^3} \\ H_2''(x) = -\frac{4}{L_e} + \frac{6x}{L_e^2} \\ N_2''(x) = 0 \\ H_3''(x) = \frac{6}{L_e^2} - \frac{12x}{L_e^3} \\ H_4''(x) = -\frac{2}{L_e} + \frac{6x}{L_e^2} \end{cases} \quad (2.4)$$

2.1.2 Principle of Virtual Work (PVW*)

In a general context, the PVW* allows to write ($\forall u^\star$) on all the considered domain which is supposed in static equilibrium

$$\int_V (\epsilon^\star)^T \sigma dV = \int_V (u^\star)^T f_b \, dV + \int_{\partial V} (u^\star)^T F_s \, ds \tag{2.5}$$

In the context of the **beam theory**, it can be written as

$$\int_\Gamma (\epsilon_b^\star)^T \sigma_b d\Gamma = \int_\Gamma (u_b^\star)^T f_v \, d\Gamma + \sum_{\partial\Gamma} (u_b^\star)^T F_{bs} \tag{2.6}$$

where Γ is the longitudinal abscissa of the structure assimilated to a beam, f_v the body forces expressed in N/ml, F_{bs} are the punctual forces and the expressions of the stress and strain tensors (generalized version[1]) are

$$\sigma_b = \mathbb{C}\epsilon_b \Leftrightarrow \begin{bmatrix} N \\ M(x) \end{bmatrix} = \mathbb{C} \begin{bmatrix} \epsilon_0 \\ \chi(x) \end{bmatrix} \tag{2.7}$$

where ϵ_0 is the axial strain and χ is the beam curvature.

Remark : If the material behavior is supposed linear, we have

$$\mathbb{C} = \begin{bmatrix} ES & 0 \\ 0 & EI \end{bmatrix}$$

where E is the *Young* modulus, S the cross section area and I its inertia.

In matrix form and from the nodal displacements, we have

$$\epsilon_b = \begin{bmatrix} \epsilon_0 \\ \chi(x) \end{bmatrix} = \begin{bmatrix} \frac{\partial u_0}{\partial x} \\ \frac{\partial^2 v_0}{\partial x^2} \end{bmatrix} = \mathbb{B} q_e \tag{2.8}$$

where

$$\mathbb{B} = \begin{bmatrix} N_1' & 0 & 0 & N_2' & 0 & 0 \\ 0 & H_1'' & H_2'' & 0 & H_3'' & H_4'' \end{bmatrix} \tag{2.9}$$

Moreover

$$u_b = \begin{bmatrix} u_0 \\ v_0 \\ \beta \end{bmatrix} = \mathbb{N} q_e \quad \text{and} \quad f_v = \begin{bmatrix} f_x^v \\ f_y^v \\ f_{Rot}^v = 0 \end{bmatrix} \tag{2.10}$$

[1] "Generalized" means integrated over the cross section area

Remark : Formulated in this way, the interpolating function's matrix is preserved ($\mathbb{N}$). If we add a $3^r d$ component (f^v_{Rot}) which is not null, then a rotational body force is accounted (in the case of magnetic fields for example). In civil engineering, it is always zero. As we will see, at the elementary scale, the calculation of nodal forces due to a body force is written in the FE reference frame: $f_{ve} = \int_0^{L_e} \mathbb{N}^T f_v \, ds$

Integrating on each FE, we have

$$\sum_{EF} \int_{\Gamma_e} (\epsilon_b^\star)^T \sigma_b dx = \sum_{EF} \left(\int_{\Gamma_e} (u_b^\star)^T f_v \, dx + \sum_{\partial\Gamma_e} (u_b^\star)^T F_{bs} \right) \tag{2.11}$$

which is equivalent to

$$\sum_{EF} \int_0^{L_e} (q_e^\star)^T \mathbb{B}^T \sigma_b dx = \sum_{EF} \left(\int_0^{L_e} (q_e^\star)^T \mathbb{N}^T f_v \, dx + \sum_{\partial\Gamma_e} (q_e^\star)^T \mathbb{N}^T F_{bs} \right) \tag{2.12}$$

After the assembling phase (symbolized by the operator $\mathcal{A}_{EF}$), requiring the passage of expressions from FE local frames to the global common frame, it comes

$$(U^\star)^T \mathcal{A}_{EF} \int_0^{L_e} \mathbb{B}^T \sigma_b dx = (U^\star)^T \mathcal{A}_{EF} \left(\int_0^{L_e} \mathbb{N}^T f_v \, dx + \sum_{\partial\Gamma_e} \mathbb{N}^T F_{bs} \right) \tag{2.13}$$

where U is the vector of nodal displacements in the global frame

Remark : In local (resp. global) frame, lower-case (resp. upper-case) letters are used for the quantities (example: q_e (resp. Q_e))

Finally, we find the classical expression of a mechanical balance of a given system (whether or not material none linearities develop)

$$\mathcal{A}_{EF} \int_0^{L_e} \mathbb{B}^T \sigma_b dx = \mathcal{A}_{EF} \left(\int_0^{L_e} \mathbb{N}^T f_v \, dx + \sum_{\partial\Gamma_e} \mathbb{N}^T F_{bs} \right) \tag{2.14}$$

where we define

$$\begin{cases} F_{int}(U) = \mathcal{A}_{EF} \int_0^{L_e} \mathbb{B}^T \sigma_b dx \\ F_{ext} = \mathcal{A}_{EF} \left(\int_0^{L_e} \mathbb{N}^T f_v \, dx + \sum_{\partial \Gamma_e} \mathbb{N}^T F_{bs} \right) \end{cases} \tag{2.15}$$

At the elementary level and given q_e, the goal is to find the contribution of the internal forces of each FE ($f_{int}(q_e) = \int_0^{L_e} \mathbb{B}^T \sigma_b(q_e) dx$). The determination of the stress tensor (σ_b) from the strain tensor ($\epsilon_b(q_e)$) is a function of the behaviour law which is supposed elastic in this class. The method of resolution varies according to the complexity of the stress-strain relationship. In case of non linear equations, the system $F_{int}(U) = F_{ext}$ is solved with a procedure of *Newton-Raphson* which is out of the scope of the document.

Here, because the behaviour law and small strains and displacements are supposed, the resolution is quite straightforward. The behavior law can be written as

$$\sigma_b = \mathbb{C} \underbrace{\mathbb{B} q_e}_{\epsilon_b} \tag{2.16}$$

Then, the assembled system can be transformed into

$$F_{int}(U) = \left[\mathcal{A}_{EF} \int_0^{L_e} \mathbb{B}^T \mathbb{C} \mathbb{B} dx \right] U \tag{2.17}$$

where $K_e = \int_0^{L_e} \mathbb{B}^T \mathbb{C} \mathbb{B} dx$ is the elementary stiffness matrix in the global frame, $\mathbb{K} = \mathcal{A}_{EF} \int_0^{L_e} \mathbb{B}^T \mathbb{C} \mathbb{B} dx$ is assembled stiffness matrix in the global frame. Assuming a linear structural analysis, the final system can be written

$$\mathbb{K} U = F_{ext} \tag{2.18}$$

2.1.3 From local to global frame and assembling

At the elementary level, all calculations are done in the FE frame ($\vec{x}$,$\vec{y}$). For assembly, all vectors and matrices must be expressed in the global frame ($\vec{X}$,$\vec{Y}$). In the case of a FE whose oriented angle formed by $\vec{x}$ and $\vec{X}$ is θ (θ defined from $\vec{X}$ to $\vec{x}$), we have the following passage matrices

$$q = \mathbb{R}Q \quad \text{where} \quad \mathbb{R} = \begin{bmatrix} \cos\theta & \sin\theta & 0 & 0 & 0 & 0 \\ -\sin\theta & \cos\theta & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & \cos\theta & \sin\theta & 0 \\ 0 & 0 & 0 & -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \text{ and } Q = \begin{bmatrix} U_1 \\ V_1 \\ \Theta_1 \\ U_2 \\ V_2 \\ \Theta_2 \end{bmatrix} \tag{2.19}$$

where Q is expressed in the global frame and q in the local frame. The angle can be expressed from the coordinates of the FE nodes : $\tan\theta = \frac{y_2 - y_1}{x_2 - x_1}$. We also have the following properties

$$Q = \mathbb{R}^T q \tag{2.20}$$

$$K_e = \mathbb{R}^T k_e \mathbb{R} \tag{2.21}$$

2.1.4 f_{int} integration

The calculation of f_{int} requires the integration of $\mathbb{B}^T\sigma_b$ over x (from 0 to L_e). In this class, because linear structural analysis is assumed, $\mathbb{B}^T\sigma_b = \mathbb{B}^T\mathbb{C}\mathbb{B}\,q_e$ where $\mathbb{B}^T\mathbb{C}\mathbb{B}$ is a function of x. The quadrature integration of *Gauss* is used and allows putting the integral in the general form

$$\int_{-1}^{+1} f(\xi)d\xi = \sum_{i=1}^{r} w_i f(\xi_i) \tag{2.22}$$

where r is number of Gauss points needed to obtain an exact integration of the function and w_i the weights of *Gauss*. For a polynomial function whose order is less than or equal to 3, if $r = 2$, the integration is exact. In this case, we have

$$\int_{-1}^{1} f(\xi)d\xi = f\Big(-1/\sqrt{3}\Big) + f\Big(1/\sqrt{3}\Big) \tag{2.23}$$

For our case, considering the interpolating functions used, $\mathbb{B}$ is linear in x, N is constant and M is a linear function of χ which is a linear function of x. Therefore, $\mathbb{B}^T\mathbb{C}\mathbb{B}$ is a quadratic polynomial function in x^2. We have finally with the following variable change $x = \frac{L_e}{2}(1+\xi) \quad \forall\xi \in (-1,\,1)$ and $dx = \frac{L_e}{2}d\xi$

$$\begin{cases} f_{int} = k_e\, q_e \quad \text{with} \quad k_e = \int_0^{L_e} \mathbb{B}^T \mathbb{C}\mathbb{B} dx \\ \\ k_e = \int_{-1}^{1} \mathbb{B}^T \mathbb{C}\mathbb{B}\, d\xi\, \frac{L_e}{2} = \frac{L_e}{2} \left(\mathbb{B}^T \mathbb{C}\mathbb{B}|_{PG_1} + \mathbb{B}^T \mathbb{C}\mathbb{B}|_{PG_2} \right) \end{cases} \tag{2.24}$$

where k_e is the elementary stiffness matrix expressed within the local frame of the EBFE. The values are expressed at the *Gauss* points (PG_i)

$$\begin{cases} \mathbb{B}^T \mathbb{C}\mathbb{B}|_{PGi} = \mathbb{B}^T(x_i) \mathbb{C}\mathbb{B}(x_i) \\ x_i = \frac{L_e}{2}(1+\xi_i) \quad \text{with} \quad \xi_1 = -\frac{1}{\sqrt{3}} \text{an}\, \xi_2 = \frac{1}{\sqrt{3}} \end{cases} \tag{2.25}$$

Remark : In the case of linear problems, the integration over the FE volume leads to the following elementary stiffness matrix expressed with the local frame of the EBFE

$$k_e = \begin{bmatrix} \frac{ES}{L_e} & 0 & 0 & -\frac{ES}{L_e} & 0 & 0 \\ 0 & \frac{12EI}{L_e^3} & \frac{6EI}{L_e^2} & 0 & -\frac{12EI}{L_e^3} & \frac{6EI}{L_e^2} \\ 0 & \frac{6EI}{L_e^2} & \frac{4EI}{L_e} & 0 & -\frac{6EI}{L_e^2} & \frac{2EI}{L_e} \\ -\frac{ES}{L_e} & 0 & 0 & \frac{ES}{L_e} & 0 & 0 \\ 0 & -\frac{12EI}{L_e^3} & -\frac{6EI}{L_e^2} & 0 & \frac{12EI}{L_e^3} & -\frac{6EI}{L_e^2} \\ 0 & \frac{6EI}{L_e^2} & \frac{2EI}{L_e} & 0 & -\frac{6EI}{L_e^2} & \frac{4EI}{L_e} \end{bmatrix} \tag{2.26}$$

2.2 Validation example

2.2.1 Cantilever beam

In order to validate the FE code, the case of a cantilever beam is considered and compared to classical beam theory. A punctual force is applied on the free end of the beam. The details of the parameters values can be found with the MATLAB script (section 2.2.2). The results are in very good agreement with the theory (Figure 2.2).

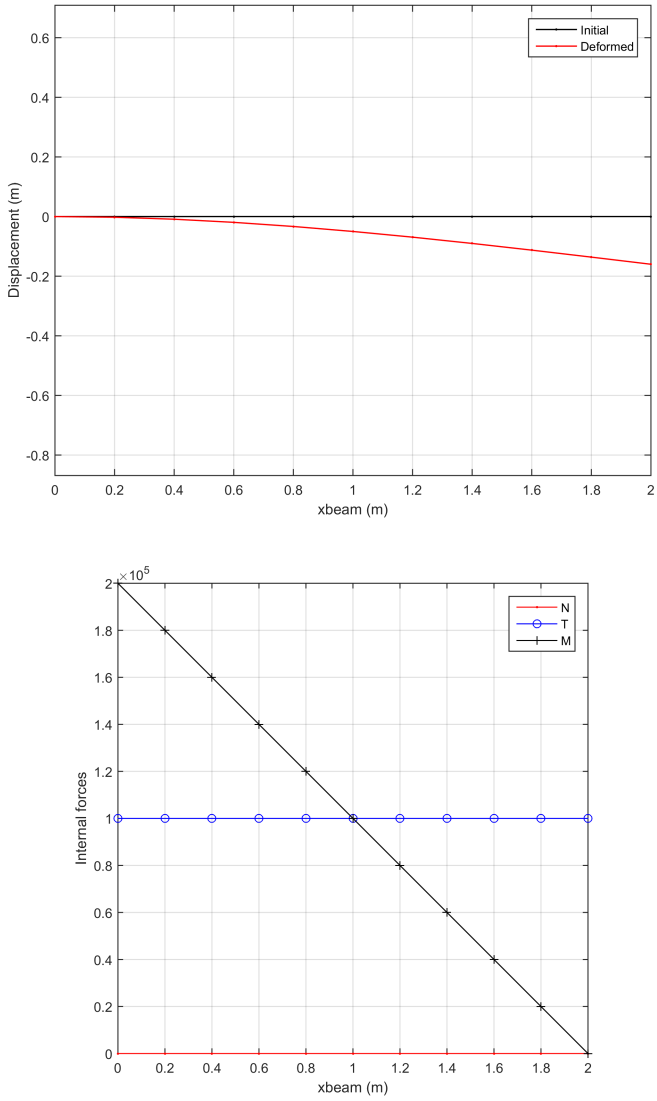

Figure 2.2: Validation example of the MATLAB Finite Element code. Cantilever beam case. Deformed shape (upper graph), Internal forces (lower graph).

2.2.2 MATLAB Algorithms

2.2.2.1 Main file

Main file `Main_QS_L.m`

```
clearvars ; clc ; format short ;

%% INPUTS
INPUTS.Material.E = 200e9 ;          % Young modulus
INPUTS.Material.rho = 7500 ;        % density

INPUTS.Geometry.L = 2 ;             % Beam length
INPUTS.Geometry.b = 0.1 ;           % cross section geometry
INPUTS.Geometry.h = 0.1 ;           % cross section geometry

INPUTS.Load.g = [0 ; -9.81] ;   % gravity
INPUTS.Load.Fchgt = [10 ; -50] ;       % Ponctual force magnitude
INPUTS.Load.Position = [INPUTS.Geometry.L ; 0] ;       % Ponctual loading Position

INPUTS.Mesh.nFE = 15;       % number of FE for the beam

%% MESHING
[XY,TC,DOF,INPUTS] = MESHING(INPUTS) ;

%% FINITE ELEMENT
[ElemData,INPUTS] = FE_DATA(INPUTS) ;

%% MATRICES ASSEMBLING
[K,FVol,INPUTS] = ASSEMBLING(ElemData,INPUTS,'VisuOFF') ;

%% PONCTUAL FORCE
F = LOADING(FVol,INPUTS) ;

%% BOUNDARY CONDITIONS
[KCL,FCL,INPUTS] = BOUNDARY_CONDITIONS(K,F,INPUTS) ;

%% RESOLUTION in QS
UCL=KCL\FCL;

%% POST PROCESSING
%% POST PROCESSING
%% POST PROCESSING

%% DEFORMED SHAPE
U = DEFORMED_SHAPE(UCL,INPUTS) ;

%% INTERAL FORCES
[NTMAllFE,Re,thetae] = InternalForcesAllFE(U,INPUTS) ;

ampN = -1/50 ; % these coeff amplify/decease the graph magnitude
ampT = -1/700 ;
ampM = -1/700 ;
InternalForcesAllFEVISU(NTMAllFE,TC,Re,ampN,ampT,ampM) ;

%%
% savefig(h1,'Cb1_Deformed.fig')
% print('Cb1_Deformed.eps','-depsc')
% print('Cb1_Deformed.png','-dpng')
% savefig(h2,'Cb2_NTM.fig')
```

2.2.2.2 Used Functions

Function `MESHING.m`

```
function [XY,TC,DOF,INPUTS] = MESHING(INPUTS)

L   = INPUTS.Geometry.L ;
nFE = INPUTS.Mesh.nFE ;

%% XY Table : Nodes coordinates
% Beam
iNode = 0 ;
Li = 0 ;
Le = L/nFE ;
for i=1:nFE+1
    iNode = iNode+1 ;
    XY(iNode,:)=[Li 0];
    Li = Le + Li ;
end
nNodes = length(XY) ;

%% TC Table : Connectivity Table
iEF = 0 ;
for i=1:nFE
    iEF = iEF+1 ;
    TC{iEF} = [iEF iEF+1] ;
end

%% DOF Table : Degree of freedom Table
for iNode=1:nNodes
    DOF{iNode} = [(3*iNode-2) (3*iNode-1) 3*iNode] ;
end

%% Data storage
INPUTS.Mesh.nNodes = nNodes ;
INPUTS.Mesh.XY = XY ;
INPUTS.Mesh.TC = TC ;
INPUTS.Mesh.DOF = DOF ;
end
```

Function `FE_DATA.m`

```
function [ElemData,INPUTS] = FE_DATA(INPUTS)

L   = INPUTS.Geometry.L ;
h   = INPUTS.Geometry.h ;
b   = INPUTS.Geometry.b ;
E   = INPUTS.Material.E ;

nFE = length(INPUTS.Mesh.TC) ;

S = b*h ;
I = b*h^3/12 ;

% beam geom
Le = L/nFE ; % Finite element length
iEF = 0 ;
for i=1:nFE
    iEF = iEF+1 ;
    ElemData{iEF}.E   = E ;
    ElemData{iEF}.S   = S ;
    ElemData{iEF}.I   = I ;
    ElemData{iEF}.Le  = Le ;
end

%INPUTS.Geometry.S = S ;
%INPUTS.Geometry.I = I ;

end
```

Function `ASSEMBLING.m`

```
%% Assembling of the finite element contributions

function [K,FVol,INPUTS] = ASSEMBLING(ElemData,INPUTS,Visu)

nFE    = INPUTS.Mesh.nFE ;
nNodes = INPUTS.Mesh.nNodes ;

XY  = INPUTS.Mesh.XY ;
TC  = INPUTS.Mesh.TC ;
DOF = INPUTS.Mesh.DOF ;

%% Elementary stiffness matrices
Ke    = zeros(6,6,nFE) ;
FeVol = zeros(6,nFE) ;
for iFE = 1:nFE
    xyzelem = XY(TC{iFE},:);
    Disp = zeros(6,1) ;
    EFData = ElemData{iFE} ;
    %[Fint(:,iFE),Ke(:,:,iFE)] = BEAM_NB_LINE(xyzelem,EFData,Disp);
    [~,Ke(:,:,iFE)] = BEAM_NB_LINE(xyzelem,EFData,Disp);

    %% Body forces
    FeVol(:,iFE) = BodyForces(xyzelem,EFData,INPUTS) ;
end

%% Assembling
nDOF = 3*nNodes ;
K = zeros(nDOF,nDOF) ;
FVol = zeros(nDOF,1);

for iFE=1:nFE
    numDDL = [DOF{TC{iFE}(1)} DOF{TC{iFE}(2)}] ;
    K(numDDL,numDDL) = K(numDDL,numDDL) + Ke(:,:,iFE) ;
    FVol(numDDL) = FVol(numDDL) + FeVol(:,iFE) ;

    if strcmp(Visu,'VisuON')
        fig2 = figure(10) ; clf ;
        fig2.WindowStyle = "docked" ;
        spy(K)
    end

%% Data storage
INPUTS.Numeric.Ke = Ke ;
INPUTS.Numeric.FeVol = FeVol ;

end
```

Function `BodyForces.m`

```
%% Built the elementary nodal force vector coming form body forces

function FeVol = BodyForces(xyzelem,EFData,INPUTS)
g   = INPUTS.Load.g ;
rho = INPUTS.Material.rho ;

Le  = EFData.Le ;
S   = EFData.S ;

Ve = S*Le ;
qVolG = rho*Ve*g/Le ; % Transfo in lineic loading (own weight in N/ml) global frame

[~,thetae] = FromGlobal2LocalFrame(xyzelem) ;
Rdim2 = [cos(thetae) sin(thetae) ; -sin(thetae) cos(thetae)] ; % From Global to local

qVol = Rdim2*qVolG ; % local frame

qVolx = qVol(1) ; qVoly = qVol(2) ;

feVol = [qVolx*Le/2 ; qVoly*Le/2 ; qVoly*Le^2/12  ; ...
         qVolx*Le/2 ; qVoly*Le/2 ; -qVoly*Le^2/12 ] ; %local frame

%% Nodal forces (body forces) expressed within the global frame
% Rotation matrix
[R,~] = FromGlobal2LocalFrame(xyzelem) ;
FeVol = R'*feVol ; % global frame

end
```

Function `FromGlobal2LocalFrame.m`

```
function [R,thetae] = FromGlobal2LocalFrame(xyz)
x1 = xyz(1,1) ;
x2 = xyz(2,1) ;
y1 = xyz(1,2) ;
y2 = xyz(2,2) ;
thetae = atan((y2-y1)/(x2-x1)) ; % Oriented angle in respect to horizontal (node1 at the origin)
if y1>y2
    thetae = -abs(thetae) ;
elseif x2<x1
    thetae = -abs(thetae) ;
end
% thetae*180/pi ;
R = [cos(thetae)  sin(thetae) 0 0 0 0 ; ...
    -sin(thetae) cos(thetae) 0 0 0 0 ; ...
    0 0 1 0 0 0 ;
    0 0 0 cos(thetae) sin(thetae) 0; ...
    0 0 0 -sin(thetae) cos(thetae) 0 ;...
    0 0 0 0 0 1] ; % Rotation matrix (from local to global referential)
end
```

Function `LOADING.m`

```
function [F] = LOADING(FVol,INPUTS)

XY  = INPUTS.Mesh.XY ;
DOF = INPUTS.Mesh.DOF ;

% Load position on the beam
xP = INPUTS.Load.Position(1) ;
yP = INPUTS.Load.Position(2) ;

iNodeChgt = find(XY(:,1)>=xP*0.99999 & XY(:,2)==yP,1) ; % node number where the load is applied
numDDL    = DOF{iNodeChgt}(1:2) ; % degree of freedom (1 and 2) of the node

F = FVol ;
F(numDDL) = FVol(numDDL) + INPUTS.Load.Fchgt ;

end
```

Function BOUNDARY_CONDITIONS.m

```
function [KCL,FCL,INPUTS] = BOUNDARY_CONDITIONS(K,F,INPUTS)

TC  = INPUTS.Mesh.TC ;
DOF = INPUTS.Mesh.DOF ;

%dofCL = [DOF{TC{1}(1)}(1:2) DOF{TC{nFE}(2)}(1:2)]; % articulation on the rigth and the left ↩
      supports

numEFCL = 1 ;
numNodeCL = TC{numEFCL}(1) ; % number of the first node of FE numEFCL

dofCL = DOF{numNodeCL}(1:3) ; % clamped left support

K(dofCL,:)=[];
K(:,dofCL)=[];
F(dofCL)=[];

%% stiffness matrix and force vector with boundary conditions application
KCL = K ;
FCL = F ;

%% data storage
INPUTS.Mesh.dofCL = dofCL ;

end
```

Function DEFORMED_SHAPE.m

```
function U = DEFORMED_SHAPE(UCL,INPUTS)

dofCL = INPUTS.Mesh.dofCL ;
nDOF  = 3*INPUTS.Mesh.nNodes ;
TC    = INPUTS.Mesh.TC ;
XY    = INPUTS.Mesh.XY ;
DOF   = INPUTS.Mesh.DOF ;

VecDDL        = cell2mat(DOF) ;
VecDDL(dofCL) = [] ;

U         = zeros(nDOF,1);
U(VecDDL) = UCL;

fig1 = figure(1) ; clf;
fig1.WindowStyle = 'docked' ;

for iFE=1:length(TC)
    indicesNodes = TC{iFE} ;

    % Initiale configuration
    x = XY(indicesNodes,1) ;
    y = XY(indicesNodes,2) ;
    plot(x,y,'.-','LineWidth',1,'Color','k') ; hold on ;

    %  Deformed configuraion
    indicesDDLx = [] ;
    indicesDDLy = [] ;
    for i=1:2
        indicesDDLx = [indicesDDLx DOF{indicesNodes(i)}(1)] ;
        indicesDDLy = [indicesDDLy DOF{indicesNodes(i)}(2)] ;
    end
    x = XY(indicesNodes,1)+U(indicesDDLx) ;
    y = XY(indicesNodes,2)+U(indicesDDLy) ;
    plot(x,y,'.-','LineWidth',1,'Color','r') ; hold on ;
end

axis equal ;
axis([-0.3 2.3 -0.25 0.25])
grid on ;

end
```

Function `InternalForcesAllFE.m`

```
function [NTM,Re,thetae] = InternalForcesAllFE(U,INPUTS)

% Data recovering
TC  = INPUTS.Mesh.TC ;
XY  = INPUTS.Mesh.XY ;
DOF = INPUTS.Mesh.DOF ;

Ke  = INPUTS.Numeric.Ke ;

nFE = INPUTS.Mesh.nFE ;

% Initialisation
thetae = zeros(nFE,1) ;
Re     = zeros(6,6,nFE) ;
NTM    = struct([]) ;

for iFE=1:nFE
    node1    = TC{iFE}(1) ;
    node2    = TC{iFE}(2) ;

    dofnode1 = DOF{node1} ;
    dofnode2 = DOF{node2} ;

    KeG      = Ke(:,:,iFE) ;
    Ue       = U([dofnode1 dofnode2]) ;
    Fe       = KeG*Ue ;

    xyz(1,1) = XY(node1,1) ;
    xyz(2,1) = XY(node2,1) ;
    xyz(1,2) = XY(node1,2) ;
    xyz(2,2) = XY(node2,2) ;

    % Rotation (from Global to local) matrix for all FE
    [Re(:,:,iFE),thetae(iFE)] = FromGlobal2LocalFrame(xyz) ;

    % From Global to Local frame
    fe = Re(:,:,iFE)*Fe ;

    % internal forces at node 1 (local numbering)
    NTM(iFE).node1.valueFint = fe(1:3) ; % expressed in the local frame
    NTM(iFE).node1.numNode = node1 ;
    NTM(iFE).node1.position = XY(node1,:) ;

    % internal forces at node 2 (local numbering)
    NTM(iFE).node2.valueFint = -fe(4:6) ; % expressed in the local frame
    NTM(iFE).node2.numNode = node2 ;
    NTM(iFE).node2.position = XY(node2,:) ;
end
```

Function `InternalForcesAllFEVISU.m`

```
function InternalForcesAllFEVISU(NTM,TC,Re,ampN,ampT,ampM)

figure(2) ; clf ; hold on ;

for iFE=1:length(TC)

    % Position initiales des nodes
    PosNodes = [NTM(iFE).node1.position ; NTM(iFE).node2.position] ;
    xNodes = PosNodes(:,1) ;
    yNodes = PosNodes(:,2) ;

    % Defo initiale sur chaque subplot
    subplot(1,3,1) ; hold on ;
    plot(xNodes,yNodes,'.-k') ;
    subplot(1,3,2) ; hold on ;
    plot(xNodes,yNodes,'.-k') ;
    subplot(1,3,3) ; hold on ;
    plot(xNodes,yNodes,'.-k') ;

    Le = sqrt(diff(xNodes)^2+diff(yNodes)^2) ;
    R = Re(1:2,1:2,iFE) ; % Matrice de rotation du repere global vers le local
                          % attention Re est une matrix 6x6 (deux nodes traites en meme temps en ↩
                               3D) il faut la reduire a 2x2 (1 point traite et 2D)

    % Position du point 1 associe aux efforts internes
    X1Y1 = PosNodes(1,:) ;

    % Efforts normal tranchant moment du node1
    N1 = NTM(iFE).node1.valueFint(1) ;
    T1 = NTM(iFE).node1.valueFint(2) ;
    M1 = NTM(iFE).node1.valueFint(3) ;
    PN1xy = [0 ; ampN*N1] ; % repere local) Position point effort normal dans l'espace du ↩
        maillage
    PN1XY = X1Y1' + R'*PN1xy ; % exprime dans repere global
    PT1xy = [0 ; ampT*T1] ; % repere local) Position point effort tranchant dans l'espace du ↩
        maillage
    PT1XY = X1Y1' + R'*PT1xy ; % exprime dans repere global
    PM1xy = [0 ; ampM*M1] ; % repere local) Position point effort moment dans l'espace du ↩
        maillage
    PM1XY = X1Y1' + R'*PM1xy ; % exprime dans repere global

    % Effort normal du node2
    N2 = NTM(iFE).node2.valueFint(1) ;
    T2 = NTM(iFE).node2.valueFint(2) ;
    M2 = NTM(iFE).node2.valueFint(3) ;
    PN2xy = [Le ; ampN*N2] ; % repere local) Position point effort normal dans l'espace du ↩
        maillage
    PN2XY = X1Y1' + R'*PN2xy ; % exprime dans repere global
    PT2xy = [Le ; ampT*T2] ; % repere local) Position point effort normal dans l'espace du ↩
        maillage
    PT2XY = X1Y1' + R'*PT2xy ; % exprime dans repere global
    PM2xy = [Le ; ampM*M2] ; % repere local) Position point effort normal dans l'espace du ↩
        maillage
    PM2XY = X1Y1' + R'*PM2xy ; % exprime dans repere global


    xNNodes = [PN1XY(1) ; PN2XY(1)] ;
    yNNodes = [PN1XY(2) ; PN2XY(2)] ;

    xTNodes = [PT1XY(1) ; PT2XY(1)] ;
    yTNodes = [PT1XY(2) ; PT2XY(2)] ;

    xMNodes = [PM1XY(1) ; PM2XY(1)] ;
    yMNodes = [PM1XY(2) ; PM2XY(2)] ;

    subplot(1,3,1) ; hold on ;
    plot(xNNodes,yNNodes,'.-r') ;
    xlabel('x (m)')
    ylabel('y (m)')
    title('N distribution')
```

```
64      axis square ;
65      grid on ;
66      xlim([-0.5 2.5])
67      ylim([-2.5 2.5])
68
69      subplot(1,3,2) ; hold on ;
70      plot(xTNodes,yTNodes,'.-r') ;
71      xlabel('x (m)')
72      ylabel('y (m)')
73      title('T distribution')
74      axis square ;
75      grid on ;
76      xlim([-0.5 2.5])
77      ylim([-2.5 2.5])
78
79      subplot(1,3,3) ; hold on ;
80      plot(xMNodes,yMNodes,'.-r') ;
81      xlabel('x (m)')
82      ylabel('y (m)')
83      title('M distribution')
84      axis square ;
85      grid on ;
86      xlim([-0.5 2.5])
87      ylim([-2.5 2.5])
88
89
90  end
91
92
93
94  end
```

Chapter 3

FEA in dynamic conditions (Optional)

3.1 Problem formulation

The extension of the previous *quasi* static problem to dynamic problems is proposed. The same `EBFE` is use under the same kinematic hypothesis. The inertial forces are now accounted for where the effect of the mass acceleration appear within the PVW*. The problem should now be solved through space and time.

3.1.1 Principle of Virtual Work (PVW*)

Within a dynamic framework and for *Euler-Bernoulli* beams, the PVW* is written as

$$\underbrace{\int_V (u^\star)^T \rho\, \ddot{u}\, dV}_{\text{Work of inertial forces}} + \int_\Gamma (\epsilon_b^\star)^T \sigma_b d\Gamma = \int_\Gamma (u_b^\star)^T f_v\, d\Gamma + \sum_{\partial\Gamma} (u_b^\star)^T F_{bs} \qquad (3.1)$$

where $\ddot{u} = \frac{\partial^2 u}{\partial t^2}$ is the acceleration of a given material point of the beam. ρ is the density of the material. The virtual work of the inertial forces should now be expressed.

Under the kinematic assumptions framework of *Euler-Bernoulli*, the displacement[1] of a given material point can be obtained from the displacement of the center of gravity and from the cross section rotation.

[1] and thus also the velocity and the acceleration

$$\begin{bmatrix} u^\star \\ v^\star \end{bmatrix} = \begin{bmatrix} u_0^\star - y\,\beta^\star \\ v_0^\star \end{bmatrix} \quad \text{and} \quad \begin{bmatrix} \ddot{u} \\ \ddot{v} \end{bmatrix} = \begin{bmatrix} \ddot{u_0} - y\,\ddot{\beta} \\ \ddot{v_0} \end{bmatrix} \tag{3.2}$$

In order to make the elementary appears (integration over the FE volumes), the inertial term can be written as

$$\int_V (u^\star)^T \rho\,\ddot{u}\,dV = \sum_{EF} \left(\int_{V_e} (u^\star)^T \rho\,\ddot{u}\,dV \right) \tag{3.3}$$

Injecting the displacement field in the equation 3.2, it comes

$$\int_{V_e} (u^\star)^T \rho\,\ddot{u}\,dV = \int_{V_e} \rho(u^\star\ddot{u} + v^\star\ddot{v})\,dV \tag{3.4}$$

which leads to

$$\int_{V_e} \rho(u^\star\ddot{u} + v^\star\ddot{v})\,dV = \int_{V_e} \rho \left[(u_0^\star - y\,\beta^\star)(\ddot{u_0} - y\,\ddot{\beta}) + v_0^\star\ddot{v_0} \right] dV \tag{3.5}$$

The integration is performed first over the cross-section area and then over the FE length. We have $dV = b\,dy\,dx$ where $S = b\,h$ is the cross section area which here is supposed to have a square shape (h thickness along y axis of the beam and b along z axis) but it is not a mandatory condition.

$$= \rho \int_0^{L_e} \left(\int_{-\frac{h}{2}}^{\frac{h}{2}} u_0^\star\ddot{u_0} - y\left(u_0^\star\ddot{\beta} + \ddot{u_0}\beta^\star\right) + y^2\left(\beta^\star\ddot{\beta}\right) + v_0^\star\ddot{v_0} \right) b\,dy\,dx \tag{3.6}$$

thus

$$\rho \int_0^{L_e} \left(S\,u_0^\star\ddot{u_0} + S\,v_0^\star\ddot{v_0} + I_{Gz}\,\beta^\star\ddot{\beta} \right) dx = \rho \int_0^{L_e} u_b^\star\,\mathbb{C}\,\ddot{u}_b\,dx \tag{3.7}$$

where

$$\mathbb{C}_m = \begin{bmatrix} S & 0 & 0 \\ 0 & S & 0 \\ 0 & 0 & I_{Gz} \end{bmatrix} \tag{3.8}$$

knowing that $u_b = \mathbb{N}q_e$ and $\ddot{u_b} = \mathbb{N}\ddot{q_e}$, the final form is

$$\int_{V_e} (u^\star)^T \rho\, \ddot{u}\, dV = \rho \int_0^{L_e} (q_e^\star)^T\, \mathbb{N}^T\, \mathbb{C}_m\, \mathbb{N}\, \ddot{q}_e\, dx \tag{3.9}$$

Finally, the elementary mass matrix of EBFE can be written as

$$m_e = \rho \int_0^{L_e} \mathbb{N}^T\, \mathbb{C}_m\, \mathbb{N}\, dx \tag{3.10}$$

Remark : The previous equation is easy but long. An alternative can be to use the formal computation tool of MATLAB. Moreover, at the end of the calculation, it is possible to distinguish two contributions in terms of inertial forces. The first one is translationnal (related to S) and the second one is rotationnal, related to the cross section inertia I.

Finally, we have

$$\sum_{EF} (q_e^\star)^T \left(\int_0^{L_e} \rho \mathbb{N}^T \mathbb{C}_m \mathbb{N}\, dx\, \ddot{q}_e + \int_0^{L_e} \mathbb{B}^T \sigma_b dx \right) = \sum_{EF} (q_e^\star)^T \left(\int_0^{L_e} \mathbb{N}^T f_v\, dx + \sum_{\partial\Gamma_e} \mathbb{N}^T F_{bs} \right) \tag{3.11}$$

After the assembling, it comes where U is the nodal displacement vector expressed in the global frame

$$(U^\star)^T \left[\mathcal{A}_{EF} \left(\int_0^{L_e} \rho \mathbb{N}^T \mathbb{C}_m \mathbb{N}\, dx \right) \ddot{U} + \mathcal{A}_{EF} \left(\int_0^{L_e} \mathbb{B}^T \sigma_b dx \right) \right] = (U^\star)^T \mathcal{A}_{EF} \left(\int_0^{L_e} \mathbb{N}^T f_v\, dx + \sum_{\partial\Gamma_e} \mathbb{N}^T F_{bs} \right) \tag{3.12}$$

which takes the form (without damping $C = [0]$)

$$M\, \ddot{U}(t) + F_{int}(U(t)) = F_{ext}(t) \tag{3.13}$$

The mass matrix and the inertial forces contributions appear at the global scale such as

$$M = \mathcal{A}_{EF}\left(\int_0^{L_e} \rho \mathbb{N}^T \mathbb{C}_m \mathbb{N}\, dx \right) \tag{3.14}$$

$$F_{int} = \mathcal{A}_{EF}\left(\int_0^{L_e} \mathbb{B}^T \sigma_b dx \right) \tag{3.15}$$

Within the local frame and at the elementary scale, the EBFE has a mass matrix which can be decomposed in two contributions.

$$m_e = m_e^{\mathtt{tra}} + m_e^{\mathtt{rot}} \tag{3.16}$$

The translational contribution can be written as

$$m_e^{\mathtt{tra}} = \frac{\rho S L_e}{420} \begin{bmatrix} 140 & 0 & 0 & 70 & 0 & 0 \\ 0 & 156 & 22L_e & 0 & 54 & -13L_e \\ 0 & 22L_e & 4L_e^2 & 0 & 13L_e & -3L_e^2 \\ 70 & 0 & 0 & 140 & 0 & 0 \\ 0 & 54 & 13L_e & 0 & 156 & -22L_e \\ 0 & -13L_e & -3L_e^2 & 0 & -22L_e & 4L_e^2 \end{bmatrix} \tag{3.17}$$

where rotational terms are neglected (I is supposed zero for the computation of m_e). When rotational terms are taken into account, the rotation of the cross section generates additional inertial forces described by

$$m_e^{\mathtt{rot}} = \frac{\rho\, I_{Gz}}{30 L_e} \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 36 & 3L_e & 0 & -36 & 3L_e \\ 0 & 3L_e & 4L_e^2 & 0 & -3L_e & -L_e^2 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -36 & -3L_e & 0 & 36 & -3L_e \\ 0 & 3L_e & -L_e^2 & 0 & -3L_e & 4L_e^2 \end{bmatrix} \tag{3.18}$$

which is added to $m_e^{\mathtt{tra}}$.

3.1.2 Equations of motion

The problem should be solved through time. For a given timestep, the objective is to solve the following equation

$$P(U_{i+1}, \dot{U}_{i+1}, \ddot{U}_{i+1}) = F_{ext}^{i+1} \tag{3.19}$$

where P is the internal forces vector which can be written in this case

$$P(U_{i+1}, \dot{U}_{i+1}, \ddot{U}_{i+1}) = M\ddot{U}_{i+1} + C\dot{U}_{i+1} + F_{int}(U_{i+1}) \tag{3.20}$$

The discretization of the time is performed based on a *Newmark* integration scheme. The supposed unknowns are the displacement (*i.e.* all kinematic quantities are expressed from U_{i+1})

$$\begin{cases} \dot{U}_{i+1} = \dot{U}_i + (1-\gamma)\Delta t\ddot{U}_i + \dfrac{\gamma}{\beta\Delta t}\left[U_{i+1} - U_i - \Delta t\dot{U}_i - (\dfrac{1}{2} - \beta)\Delta t^2\ddot{U}_i\right] \\ \ddot{U}_{i+1} = \dfrac{1}{\beta\Delta t^2}\left[U_{i+1} - U_i - \Delta t\dot{U}_i - (\dfrac{1}{2} - \beta)\Delta t^2\ddot{U}_i\right] \end{cases} \tag{3.21}$$

Thus, it makes it possible to get the new displacements at time t_{i+1} from the converged values at time t_i.

$$\begin{aligned} \left(\frac{\gamma}{\beta\Delta t}C + \frac{1}{\beta\Delta t^2}M\right)U_{i+1} + F_{int}(U_{i+1}) = \\ F_{ext}^{i+1} + \\ \left(\frac{\gamma}{\beta\Delta t}C + \frac{1}{\beta\Delta t^2}M\right)U_i + \\ \left(C(\frac{\gamma}{\beta} - 1) + \frac{1}{\beta\Delta t}M\right)\dot{U}_i + \\ \left(C\Delta t\left(\frac{\gamma}{\beta}(\frac{1}{2} - \beta) - (1-\gamma)\right) + \frac{\frac{1}{2} - \beta}{\beta}M\right)\ddot{U}_i \end{aligned} \tag{3.22}$$

For **linear** system, we have $F_{int}(U_{i+1}) = KU_{i+1}$ which leads to

$$\underbrace{\left(K + \frac{\gamma}{\beta\Delta t}C + \frac{1}{\beta\Delta t^2}M\right)}_{S_{pf}} U_{i+1} =$$

$$f_{i+1} \quad \begin{cases} F_{ext}^{i+1} + \left(\frac{\gamma}{\beta\Delta t}C + \frac{1}{\beta\Delta t^2}M\right) U_i + \left(C(\frac{\gamma}{\beta} - 1) + \frac{1}{\beta\Delta t}M\right) \dot{U}_i + \\ \left(C\Delta t\left(\frac{\gamma}{\beta}(\frac{1}{2} - \beta) - (1 - \gamma)\right) + \frac{\frac{1}{2} - \beta}{\beta}M\right) \dot{U} \end{cases} \tag{3.23}$$

or in matrix form

$$S_{pf} U_{i+1} = f_{i+1} \tag{3.24}$$

The equation 3.22 can be written

$$R(U_{i+1}) = -P(U_{i+1}) + F_{ext}^{i+1} = 0 \tag{3.25}$$

where

$$P(U_{i+1}) = F_{int}(U_{i+1}) + a_{NM}\,(U_{i+1} - U_i) - b_{NM}\,\dot{U}_i - c_{NM}\,\ddot{U}_i \tag{3.26}$$

où (3.27)

$$\begin{cases} a_{NM} = \left(\frac{\gamma}{\beta\Delta t}C + \frac{1}{\beta\Delta t^2}M\right) \\ b_{NM} = \left(C(\frac{\gamma}{\beta} - 1) + \frac{1}{\beta\Delta t}M\right) \\ c_{NM} = \left(C\Delta t\left(\frac{\gamma}{\beta}(\frac{1}{2} - \beta) - (1 - \gamma)\right) + \frac{\frac{1}{2} - \beta}{\beta}M\right) \end{cases} \tag{3.28}$$

The next consists to obtain the velocities and the accelerations at $i + 1$ thank to the equation (3.21).

3.2 Validation example

3.2.1 Cantilever beam

The same example has been chosen. A cantilever beam is considered and is subjected to an heavyside type of loading through time imposed by a punctual force at the free end of the beam.

The results are in very good agreement with the theory. In the presented case, the maximal displacement of the beam is 25cm (*cf.* a detail of the parameter values in the `MATLAB` script) in static conditions. If an heaviside force is applied, the displacement oscillates around the quasi-static value (from 0 to 50cm).

3.2.2 MATLAB Algorithms

Main file `Main_Dy_L_WithoutNR.m`

```
%% Example of validation
% Cantilever Beam loaded by a punctual load at the free end
% Dynamic condition : tree types of time evolutions :
%        'RealEarthquake', 'Triangular' and 'Heavyside'
clearvars ; clc ; format short ;
%% INPUTS
E = 200e9;          % Young modulus
L = 5.;              % Beam length
g = 0*[0 ; -9.81] ; % gravity
b = 0.1 ;           % cross section geometry
h = 0.1 ;           % cross section geometry
rho = 2500 ;        % density
ksi = 0.0 ;        % Rayleigh damping
nEFPout = 6 ;       % number of FE for the beam
dt = 1e-2 ;         % Timestep
% Newmark parameters
GammaNM = 1/2 ; BetaNM = 1/4 ; % constante acceleration (unconditionaly stable)

% Load type and description
%    RealEarthquake : accelerogram application
%    Heavyside : Ponctual force, Heavyside time evolution
%    Triangular : Ponctual force, Triangular time evolution
LoadingCase = 'Heavyside' ;
if strcmp(LoadingCase,'RealEarthquake')==1
    %% Real Earthquake accelerogram
    filenameAccelero = 'Seisme_EC8_1_EW.txt' ;
    [TpsAcc, Acc] = textread(filenameAccelero,'%f %f') ;
    Acc = Acc*1 ; % Sismic signal reduction
    Tfin = TpsAcc(end) ;
elseif strcmp(LoadingCase,'Triangular')==1
    %% Ponctual Force application at node iNodeChgt
    Tfin = 2. ;
    T1 = Tfin/10 ; % if Triangular time evolution of the loading is chosen
    FchgtX = 0. ;
    FchgtY = -10000. ;
    xFyF = [L 0] ; % Ponctual force position
elseif strcmp(LoadingCase,'Heavyside')==1
    %% Ponctual Force application at node iNodeChgt
    Tfin = 2. ;
    FchgtX = 0. ;
    FchgtY = -10000. ;
    xFyF = [L 0] ; % Ponctual force position
end

% Derived data
S      = b*h ;
I      = b*h^3/12 ;
t      = 0:dt:Tfin ; % Time discretisation
Ntps   = length(t) ;

%% MESHING
% Nodes coordinates
% Beam
iNode = 0 ;
Li = 0 ;
Le = L/nEFPout ;
for i=1:nEFPout+1
    iNode = iNode+1 ;
    XY(iNode,:)=[Li 0];
    Li = Le + Li ;
end
nNodes = length(XY) ;
nDDL   = 3*nNodes ;
% Connectivity Table
iEF = 0 ;
for i=1:nEFPout
    iEF = iEF+1 ;
```

```
    TC{iEF} = [iEF iEF+1] ;
end
nEF = length(TC) ;
% Degree of fredom Table
for iNode=1:nNodes
    DOF{iNode} = [(3*iNode-2) (3*iNode-1) 3*iNode] ;
end


%% FINITE ELEMENT DATA AFFECTATION
iEF = 0 ;
% beam
Le = L/nEFPout ;
for i=1:nEFPout
    iEF = iEF+1 ;
    ElemData{iEF}.E = E ;
    ElemData{iEF}.S = S ;
    ElemData{iEF}.I = I ;
    ElemData{iEF}.Le = Le ;
    ElemData{iEF}.g = g ;
    ElemData{iEF}.rho = rho ;
    ElemHist{iEF} = [] ;
    ElemDisp{iEF} = [0 ; 0 ; 0 ; 0 ; 0 ; 0] ;
end

%% BOUNDARY CONDITION APPLICATION
ddlCL = DOF{TC{1}(1)}(1:3) ;

%% MATRICES ASSEMBLING
% elementary mass and stiffness matrices
for iEF=1:nEF
    xyzelem=XY(TC{iEF},:);
    [Me(:,:,iEF),Ke(:,:,iEF),Feint(:,iEF),FeVol(:,iEF),ElemHist{iEF}] = ...
        BEAM_NB_LINE(xyzelem,ElemData{iEF},ElemDisp{iEF},ElemHist{iEF});
end
% Assemblage following the DOF numbering
K=zeros(3*nNodes,3*nNodes) ;
M=zeros(3*nNodes,3*nNodes) ;
FextVol=zeros(3*nNodes,1);
for iEF=1:nEF
    numDDL = [DOF{TC{iEF}(1)} DOF{TC{iEF}(2)}] ;
    K(numDDL,numDDL) = K(numDDL,numDDL) + Ke(:,:,iEF) ;
    M(numDDL,numDDL) = M(numDDL,numDDL) + Me(:,:,iEF) ;
    FextVol(numDDL) = FextVol(numDDL) + FeVol(:,iEF) ;
end
MnoCL = M ; % for the construction fo the vector f=-MnoCL*iota*Acc

%% BOUNDARY CONDITION APPLICATION
K(ddlCL,:)=[];
K(:,ddlCL)=[];
M(ddlCL,:)=[];
M(:,ddlCL)=[];

%% INITIALISATION OF U, Upt and U2pt
U = cell(Ntps,1) ;
Upt = cell(Ntps,1) ;
U2pt = cell(Ntps,1) ;
U{1} = zeros(nDDL,1) ;
Upt{1} = zeros(nDDL,1) ;
U2pt{1} = zeros(nDDL,1) ;
% Boundary condition application
for it =1:Ntps
    U{it} = zeros(nDDL,1) ;
    Upt{it} = zeros(nDDL,1) ;
    U2pt{it} = zeros(nDDL,1) ;
    U{it}(ddlCL) = [] ;
    Upt{it}(ddlCL) = [] ;
    U2pt{it}(ddlCL) = [] ;
end

%% DAMPING (RAYLEIGH type)
% Modale analysis
[P,V]=eig(K^-1*M);
```

```
% None null eigen values
for k=1:size(V,2)
    aa(k)=sum(V(:,k));
end
bb=find(aa~=0); % location of none null eigen values

for k=1:length(bb)
    omega(k)=1/sqrt(V(bb(k),bb(k)));
end
% Eigenvectors
Phi=P(:,bb);
disp(['Natural Pulsation for the FE model : ' num2str(omega(1)) ' /s'])
% Rayleigh Damping type (global definition)
omega_th(1) = 1.8751^2*sqrt(E*I/(rho*S*L^4)) ;
omega_th(2) = 4.69409^2*sqrt(E*I/(rho*S*L^4)) ;
omega_th(3) = 7.85473^2*sqrt(E*I/(rho*S*L^4)) ;
omega_th(4) = ((2*4-1)*pi/2)^2*sqrt(E*I/(rho*S*L^4)) ;
omega_sorted = sort(omega) ;
for i=1:4
    disp(['Theorical and FE model Natural Frequency (in /s) : '...
      num2str(omega_th(i)/(2*pi)) ' --- ' num2str(omega_sorted(i)/(2*pi))])
end

omega1= 0 ;
omega2= omega_sorted(4) ;
alpRay=2*ksi*omega1*omega2/(omega1+omega2);
betaRay=2*ksi/(omega1+omega2);
C= alpRay*M+betaRay*K ;

%% LOADING
% External loading creation (function of time)
Fext = cell(Ntps,1) ;
ForceEvol = [] ;

if strcmp(LoadingCase,'RealEarthquake')==1
   % Real ground acceleration
    Acc = interp1(TpsAcc,Acc,t);
    iota = zeros(nDDL,1) ;
    iota(1:2:end) = 1 ;
    for it=1:length(t)
        Fext{it} = FextVol -MnoCL*iota*Acc(it) ; % External force
        Fext{it}(ddlCL)=[]; % BC appli
    end
else
    iNodeChgt = find(abs(XY(:,1)-xFyF(1))<0.00001 & abs(XY(:,2)-xFyF(2))<0.00001) ;
    ddlForExt = DOF{iNodeChgt}(1:2) ; % appli. ponc. force on FE nEF (second node) in dir. 1 and ↩
        2 (x and y)
    for it=1:Ntps % Triangular loading through time
        vecforce = zeros(nDDL,1) ;
        if strcmp(LoadingCase,'Triangular')==1
            if it*dt < T1
                forcet = (it*dt)/T1*[FchgtX ; FchgtY] ;
                vecforce(ddlForExt) = forcet ;
            elseif T1 <= it*dt && it*dt <= 2*T1
                forcet = [FchgtX ; FchgtY] - (it*dt-T1)/(2*T1-T1)*[FchgtX ; FchgtY] ;
                vecforce(ddlForExt) = forcet ;
            elseif it*dt > 2*T1
                vecforce(ddlForExt) = [0 ; 0] ;
            end
        elseif strcmp(LoadingCase,'Heavyside')==1
            forcet = [FchgtX ; FchgtY] ;
            vecforce(ddlForExt) = forcet ;
        end
        ForceEvol = [ForceEvol forcet] ;
        Fext{it} = FextVol + vecforce ; % ponc. force and body forces (coming from gravity)
        Fext{it}(ddlCL)=[]; % BC appli
    end
end


% figure(1) ; clf ;
%% plot(t,ForceEvol(2,:)/1000) ;
%% ylabel('Force (kN)')
```

```
% plot(t,Acc)
% ylabel('Acceletation (m/s^2)')
% xlabel('Time (s)')
% grid on ; axis square ;
% return

%% RESOLUTION (Dynamic case)
% equivalent stiffness matrix in NM algo
Spf = K+GammaNM/(BetaNM*dt)*C+1/(BetaNM*dt^2)*M ;
cptAff = 0 ; VecDDL = cell2mat(DOF) ; VecDDL(ddlCL) = [] ; f = cell(Ntps,1) ;
for it=1:Ntps-1
    % Displacements
    aNM = GammaNM/(BetaNM*dt)*C+1/(BetaNM*dt^2)*M ;
    bNM = C*(GammaNM/BetaNM-1)+M/(dt*BetaNM) ;
    cNM = C*dt*(GammaNM/BetaNM*(1/2-BetaNM)-(1-GammaNM))+(1/2-BetaNM)/(BetaNM)*M ;
    f{it+1} = Fext{it+1} + aNM*U{it} + bNM*Upt{it} + cNM*U2pt{it} ;
    U{it+1} = Spf\f{it+1} ;
    % Velocities and accelerations
    Upt{it+1}=Upt{it} + (1-GammaNM)*dt*U2pt{it}+ ...
        GammaNM/(BetaNM*dt)*(U{it+1}-U{it}-dt*Upt{it}-(1/2-BetaNM)*dt^2*U2pt{it}) ;
    U2pt{it+1}=1/(BetaNM*dt^2)*(U{it+1}-U{it}-dt*Upt{it}-(1/2-BetaNM)*dt^2*U2pt{it}) ;
    % Remaing computation time
    if cptAff == 100
        disp(' ')
        disp(['Niveau de calcul = ' sprintf('%1.1f',it/Ntps*100) ' %'])
        disp(['Time = ' num2str(norm(it*dt)) ' s'])
        cptAff = 1 ;
    else
        cptAff = cptAff +1 ;
    end
end

%% WORKSPACE SAVING
save('Result_EF_NB_L_Dyna.mat')



%% POST PROCESSING
% Visulisation
VecDDL = cell2mat(DOF) ;
VecDDL(ddlCL) = [] ;
UCL = cell(Ntps,1) ;
for it=1:Ntps
    UCL{it}=zeros(3*nNodes,1);
    UCL{it}(VecDDL)=U{it};
end
U = UCL ;
amp = 2 ;
for it=1:1:Ntps
    figure(1) ; clf;
    for iEF=1:nEF
        indicesNoeu = TC{iEF} ;
        indiceDDL = DOF{indicesNoeu} ;
        % configuration initiale
        x = XY(indicesNoeu,1) ;
        y = XY(indicesNoeu,2) ;
        plot(x,y,'.-','LineWidth',1,'Color','k') ; hold on ;
        % configuraion deformee
        indicesDDLx = [] ;
        indicesDDLy = [] ;
        for i=1:2
            indicesDDLx = [indicesDDLx DOF{indicesNoeu(i)}(1)] ;
            indicesDDLy = [indicesDDLy DOF{indicesNoeu(i)}(2)] ;
        end
        x = XY(indicesNoeu,1)+amp*U{it}(indicesDDLx) ;
        y = XY(indicesNoeu,2)+amp*U{it}(indicesDDLy) ;
        plot(x,y,'.-','LineWidth',1,'Color','r') ; hold on ;
    end
    axis equal ; axis([-1 6 -1 2])
    grid on ;
    pause(0.01)
end

```

```
%% VALIDATION
disp(['CHECK IF THE BC and the loading are OK (case of a contilever beam'])
disp(['Cantilever beam theory :'])
disp(['RA :' num2str(-FchgtY) ' N'])
disp(['MA :' num2str(FchgtY*L) ' N.m'])
disp(['dep max :' num2str(FchgtY*L^3/(3*E*I))  ' m'])
disp(['rot max :' num2str(FchgtY*L^2/(2*E*I))  ' rad'])
```

Function BEAM_NB_LINE.m

```
%% Euler-Bernoulli beam finite element (2 nodes)
function [Me,Ke,Fint,FeVol,Hist]=BEAM_NB_LINE(xyz,ElemData,Disp,Hist)
rho = ElemData.rho ;
E = ElemData.E ;
S = ElemData.S ;
I = ElemData.I ;
Le = ElemData.Le ;
g = ElemData.g ;

x1 = xyz(1,1) ;
x2 = xyz(2,1) ;
y1 = xyz(1,2) ;
y2 = xyz(2,2) ;
% Rotation angle (from local to global referential)
thetae = atan((y2-y1)/(x2-x1)) ; % angle oriented in relation to the horizontal (node1 at the origine)
% Rotation matrix (from local to global referential)
R = [cos(thetae)  sin(thetae) 0 0 0 0 ; ...
     -sin(thetae) cos(thetae) 0 0 0 0 ; ...
     0 0 1 0 0 0 ;
     0 0 0 cos(thetae) sin(thetae) 0; ...
     0 0 0 -sin(thetae) cos(thetae) 0 ;...
     0 0 0 0 0 1] ;

% Body forces (in )
Ve = S*Le ;
qVolG = rho*Ve*g/Le ; % lineic load transfo - dead weight (in N/ml) global frame
qVol = [cos(thetae) sin(thetae) ; -sin(thetae) cos(thetae)]*qVolG ; % local frame
qVolx = qVol(1) ;
qVoly = qVol(2) ;
feVol = [qVolx*Le/2 ; qVoly*Le/2 ; qVoly*Le^2/12 ; ...
    qVolx*Le/2 ; qVoly*Le/2 ; -qVoly*Le^2/12] ;


Ue = Disp ; % Nodes displacement within the global frame
ue = R*Ue ; % to the local frame
ke = [E*S/Le  0 0 -E*S/Le 0 0 ; ...
      0       12*E*I/Le^3 6*E*I/Le^2 0 -12*E*I/Le^3 6*E*I/Le^2 ; ...
      0       6*E*I/Le^2 4*E*I/Le 0 -6*E*I/Le^2 2*E*I/Le ; ...
      -E*S/Le 0 0 E*S/Le 0 0 ; ...
      0       -12*E*I/Le^3 -6*E*I/Le^2 0 12*E*I/Le^3 -6*E*I/Le^2 ; ...
      0 6*E*I/Le^2 2*E*I/Le 0 -6*E*I/Le^2 4*E*I/Le] ;

me = (rho*S*Le)/420*[140 0 0 70 0 0 ; ...
    0 156 22*Le 0 54 -13*Le ; ...
    0 22*Le 4*Le^2 0 13*Le -3*Le^2 ; ...
    70 0 0 140 0 0 ; ...
    0 54 13*Le 0 156 -22*Le ; ...
    0 -13*Le -3*Le^2 0 -22*Le 4*Le^2] ; % mass matrix within the local frame

fint = ke*ue ; % internal forces within the local frame

% Elementary matrix expressed within the global frame
Me = R'*me*R ;
Ke = R'*ke*R ;
Fint = R'*fint ;

% Nodales body forces within the global frame
FeVol = R'*feVol ;

end
```

Chapter 4

Example of a Frame structure

From the previous MATLAB scripts, structures composed of beams can modelled. An example of a simple geometry of frame structure (two posts and one beam) is presented in the next sections. The presented algorithms are written under *functions-form* in order to incease the adaptablility and clearness of the code.

4.1 *Quasi* static conditions

4.1.1 Main file

Main file `Main_QS_L.m`

```
clearvars ; clc ; format short ;

%% INPUTS
INPUTS.Material.E = 20e6 ;          % Young modulus
INPUTS.Material.rho = 2500 ;        % density

INPUTS.Geometry.H = 1 ;             % post height
INPUTS.Geometry.L = 2 ;             % beam length
INPUTS.Geometry.b = 0.1 ;           % cross section geometry
INPUTS.Geometry.h = 0.1 ;           % cross section geometry

INPUTS.Load.g = 0*[0 ; -9.81] ;   % gravity
INPUTS.Load.Fchgt = 100*[1 ; -1] ;       % Ponctual force magnitude
INPUTS.Load.Position = [0 ; 1] ;       % Ponctual loading Position

INPUTS.Mesh.nFEPot1 = 8 ;       % number of FE for post 1
INPUTS.Mesh.nFEPout = 20 ;       % number of FE for the beam
INPUTS.Mesh.nFEPot2 = 8 ; % number of FE for post 2

%% MESHING
[XY,TC,DOF,INPUTS] = MESHING(INPUTS) ;

%% FINITE ELEMENT
[ElemData,INPUTS] = FE_DATA(INPUTS) ;

%% MATRICES ASSEMBLING
[K,FVol,INPUTS] = ASSEMBLING(ElemData,INPUTS,'VisuOFF') ;

%% PONCTUAL FORCE
F = LOADING(FVol,INPUTS) ;

%% BOUNDARY CONDITIONS
[KCL,FCL,INPUTS] = BOUNDARY_CONDITIONS(K,F,INPUTS) ;

%% RESOLUTION in QS
UCL=KCL\FCL;

%% POST PROCESSING
%% POST PROCESSING
%% POST PROCESSING

%% DEFORMED SHAPE
U = DEFORMED_SHAPE(UCL,INPUTS) ;

%% INTERAL FORCES

%% INTERAL FORCES
[NTMAllFE,Re,thetae] = InternalForcesAllFE(U,INPUTS) ;

ampN = 0.005 ; % these coeff amplify/decease the graph magnitude
ampT = 0.005 ;
ampM = -0.005 ;
InternalForcesAllFEVISU(NTMAllFE,TC,Re,ampN,ampT,ampM) ;

%%
% savefig(h1,'Cb1_Deformed.fig')
% print('Cb1_Deformed.eps','-depsc')
% print('Cb1_Deformed.png','-dpng')
% savefig(h2,'Cb2_NTM.fig')
```

4.1.2 Used Functions

Function `MESHING.m`

```
function [XY,TC,DOF,INPUTS] = MESHING(INPUTS)

H       = INPUTS.Geometry.H ;
L       = INPUTS.Geometry.L ;

nFEPot1 = INPUTS.Mesh.nFEPot1 ;
nFEPout = INPUTS.Mesh.nFEPout ;
nFEPot2 = INPUTS.Mesh.nFEPot2 ;

%% XY Table : Nodes coordinates
% Post 1
iNode = 0 ;
Li = 0 ;
Le = H/nFEPot1 ;
for i=1:nFEPot1+1
    iNode = iNode+1 ;
    XY(iNode,:)=[0 Li];
    Li = Le + Li ;
end
% Beam
Le = L/nFEPout ;
Li = Le ;
for i=1:nFEPout
    iNode = iNode+1 ;
    XY(iNode,:)=[Li H];
    Li = Le + Li ;
end
% Post 2
Le = H/nFEPot2 ;
Li = Le ;
for i=1:nFEPot2
    iNode = iNode+1 ;
    XY(iNode,:)=[L (H-Li)];
    Li = Le + Li ;
end

nNodes = length(XY) ;

%% TC Table : Connectivity Table
iEF = 0 ;
nFE = nNodes-1 ;
for i=1:nFE
    iEF = iEF+1 ;
    TC{iEF} = [iEF iEF+1] ;
end

%% DOF Table : Degree of freedom Table
for iNode=1:nNodes
    DOF{iNode} = [(3*iNode-2) (3*iNode-1) 3*iNode] ;
end

%% Data storage
INPUTS.Mesh.nFE    = length(TC) ;
INPUTS.Mesh.nNodes = length(XY) ;
INPUTS.Mesh.XY     = XY ;
INPUTS.Mesh.TC     = TC ;
INPUTS.Mesh.DOF    = DOF ;
end
```

Function `FE_DATA.m`

```
function [ElemData,INPUTS] = FE_DATA(INPUTS)

H = INPUTS.Geometry.H ;
L = INPUTS.Geometry.L ;
h = INPUTS.Geometry.h ;
b = INPUTS.Geometry.b ;

E = INPUTS.Material.E ;

nFEPot1 = INPUTS.Mesh.nFEPot1 ;
nFEPout = INPUTS.Mesh.nFEPout ;
nFEPot2 = INPUTS.Mesh.nFEPot2 ;

S = b*h ;
I = b*h^3/12 ;

iEF = 0 ;
% Post 1
Le = H/nFEPot1 ;
for i=1:nFEPot1
    iEF = iEF+1 ;
    ElemData{iEF}.E = E ;
    ElemData{iEF}.S = S ;
    ElemData{iEF}.I = I ;
    ElemData{iEF}.Le = Le ;
end
% beam
Le = L/nFEPout ;
for i=1:nFEPout
    iEF = iEF+1 ;
    ElemData{iEF}.E = E ;
    ElemData{iEF}.S = S ;
    ElemData{iEF}.I = I ;
    ElemData{iEF}.Le = Le ;
end
% Post 2
Le = H/nFEPot2 ;
for i=1:nFEPot2
    iEF = iEF+1 ;
    ElemData{iEF}.E = E ;
    ElemData{iEF}.S = S ;
    ElemData{iEF}.I = I ;
    ElemData{iEF}.Le = Le ;
end

%% Data storage
INPUTS.Geometry.S = S ;
INPUTS.Geometry.I = I ;

end
```

Function `ASSEMBLING.m`

```
%% Assembling of the finite element contributions

function [K,FVol,INPUTS] = ASSEMBLING(ElemData,INPUTS,Visu)

nFE    = INPUTS.Mesh.nFE ;
nNodes = INPUTS.Mesh.nNodes ;

XY  = INPUTS.Mesh.XY ;
TC  = INPUTS.Mesh.TC ;
DOF = INPUTS.Mesh.DOF ;

%% Elementary stiffness matrices
Ke    = zeros(6,6,nFE) ;
FeVol = zeros(6,nFE) ;
for iFE = 1:nFE
    xyzelem = XY(TC{iFE},:);
    Disp = zeros(6,1) ;
    EFData = ElemData{iFE} ;
    %[Fint(:,iFE),Ke(:,:,iFE)] = BEAM_NB_LINE(xyzelem,EFData,Disp);
    [~,Ke(:,:,iFE)] = BEAM_NB_LINE(xyzelem,EFData,Disp);

    %% Body forces
    FeVol(:,iFE) = BodyForces(xyzelem,EFData,INPUTS) ;
end

%% Assembling
nDOF = 3*nNodes ;
K = zeros(nDOF,nDOF) ;
FVol = zeros(nDOF,1);
for iFE=1:nFE
    numDDL = [DOF{TC{iFE}(1)} DOF{TC{iFE}(2)}] ;
    K(numDDL,numDDL) = K(numDDL,numDDL) + Ke(:,:,iFE) ;
    FVol(numDDL) = FVol(numDDL) + FeVol(:,iFE) ;

    if strcmp(Visu,'VisuON')
        fig2 = figure(10) ; clf ;
        fig2.WindowStyle = "docked" ;
        spy(K)
    end

%% Data storage
INPUTS.Numeric.Ke = Ke ;
INPUTS.Numeric.FeVol = FeVol ;

end
```

Function `BodyForces.m`

```
%% Built the elementary nodal force vector coming form body forces

function FeVol = BodyForces(xyzelem,EFData,INPUTS)
g   = INPUTS.Load.g ;
rho = INPUTS.Material.rho ;

Le  = EFData.Le ;
S   = EFData.S ;

Ve = S*Le ;
qVolG = rho*Ve*g/Le ; % Transfo in lineic loading (own weight in N/ml) global frame

[~,thetae] = FromGlobal2LocalFrame(xyzelem) ;
Rdim2 = [cos(thetae) sin(thetae) ; -sin(thetae) cos(thetae)] ; % From Global to local

qVol = Rdim2*qVolG ; % local frame

qVolx = qVol(1) ; qVoly = qVol(2) ;

feVol = [qVolx*Le/2 ; qVoly*Le/2 ; qVoly*Le^2/12  ; ...
         qVolx*Le/2 ; qVoly*Le/2 ; -qVoly*Le^2/12 ] ; %local frame

%% Nodal forces (body forces) expressed within the global frame
% Rotation matrix
[R,~] = FromGlobal2LocalFrame(xyzelem) ;
FeVol = R'*feVol ; % global frame

end
```

Function `FromGlobal2LocalFrame.m`

```
function [R,thetae] = FromGlobal2LocalFrame(xyz)
x1 = xyz(1,1) ;
x2 = xyz(2,1) ;
y1 = xyz(1,2) ;
y2 = xyz(2,2) ;
thetae = atan((y2-y1)/(x2-x1)) ; % Oriented angle in respect to horizontal (node1 at the origin)
if y1>y2
    thetae = -abs(thetae) ;
elseif x2<x1
    thetae = -abs(thetae) ;
end
% thetae*180/pi ;
R = [cos(thetae)  sin(thetae) 0 0 0 0 ; ...
    -sin(thetae) cos(thetae) 0 0 0 0 ; ...
    0 0 1 0 0 0 ;
    0 0 0 cos(thetae) sin(thetae) 0; ...
    0 0 0 -sin(thetae) cos(thetae) 0 ;...
    0 0 0 0 0 1] ; % Rotation matrix (from local to global referential)
end
```

Function `LOADING.m`

```
function [F] = LOADING(FVol,INPUTS)

XY  = INPUTS.Mesh.XY ;
DOF = INPUTS.Mesh.DOF ;

% Load position on the beam
xP = INPUTS.Load.Position(1) ;
yP = INPUTS.Load.Position(2) ;
iNodeChgt = find(abs(XY(:,1)-xP)<0.001 & abs(XY(:,2)-yP)<0.001) ;
%iNodeChgt = find(XY(:,1)>=xP & XY(:,2)>=yP,1) ;
numDDL = DOF{iNodeChgt}(1:2) ;

F = zeros(length(FVol),1) ;
F(numDDL) = FVol(numDDL) + INPUTS.Load.Fchgt ;
end
```

Function BOUNDARY_CONDITIONS.m

```
function [KCL,FCL,INPUTS] = BOUNDARY_CONDITIONS(K,F,INPUTS)

TC  = INPUTS.Mesh.TC ;
DOF = INPUTS.Mesh.DOF ;

nFE = INPUTS.Mesh.nFE ;
dofCL = [DOF{TC{1}(1)}(1:2) DOF{TC{nFE}(2)}(1:2)]; % articulation on the rigth and the left ↩
     supports
%dofCL = DOF{TC{1}(1)}(1:3) ; % clamped left support

K(dofCL,:)=[];
K(:,dofCL)=[];
F(dofCL)=[];

%% stiffness matrix and force vector with boundary conditions application
KCL = K ;
FCL = F ;

%% data storage
INPUTS.Mesh.dofCL = dofCL ;
end
```

Function DEFORMED_SHAPE.m

```
function U = DEFORMED_SHAPE(UCL,INPUTS)

dofCL = INPUTS.Mesh.dofCL ;
nDOF  = 3*INPUTS.Mesh.nNodes ;
TC    = INPUTS.Mesh.TC ;
XY    = INPUTS.Mesh.XY ;
DOF   = INPUTS.Mesh.DOF ;

VecDDL        = cell2mat(DOF) ;
VecDDL(dofCL) = [] ;

U         = zeros(nDOF,1);
U(VecDDL) = UCL;

fig1 = figure(1) ; clf;
fig1.WindowStyle = 'docked' ;

for iFE=1:length(TC)
    indicesNodes = TC{iFE} ;

    % Initiale configuration
    x = XY(indicesNodes,1) ;
    y = XY(indicesNodes,2) ;
    plot(x,y,'.-','LineWidth',1,'Color','k') ; hold on ;

    %  Deformed configuraion
    indicesDDLx = [] ;
    indicesDDLy = [] ;
    for i=1:2
        indicesDDLx = [indicesDDLx DOF{indicesNodes(i)}(1)] ;
        indicesDDLy = [indicesDDLy DOF{indicesNodes(i)}(2)] ;
    end
    x = XY(indicesNodes,1)+U(indicesDDLx) ;
    y = XY(indicesNodes,2)+U(indicesDDLy) ;
    plot(x,y,'.-','LineWidth',1,'Color','r') ; hold on ;
end

axis equal ;
axis([-0.3 2.3 -0.1 1.1])
grid on ;

end
```

Function `InternalForcesAllFE.m`

```
function [NTM,Re,thetae] = InternalForcesAllFE(U,INPUTS)

% Data recovering
TC  = INPUTS.Mesh.TC ;
XY  = INPUTS.Mesh.XY ;
DOF = INPUTS.Mesh.DOF ;

Ke  = INPUTS.Numeric.Ke ;

nFE = INPUTS.Mesh.nFE ;

% Initialisation
thetae = zeros(nFE,1) ;
Re     = zeros(6,6,nFE) ;
NTM    = struct([]) ;

for iFE=1:nFE
    node1    = TC{iFE}(1) ;
    node2    = TC{iFE}(2) ;

    dofnode1 = DOF{node1} ;
    dofnode2 = DOF{node2} ;

    KeG      = Ke(:,:,iFE) ;
    Ue       = U([dofnode1 dofnode2]) ;
    Fe       = KeG*Ue ;

    xyz(1,1) = XY(node1,1) ;
    xyz(2,1) = XY(node2,1) ;
    xyz(1,2) = XY(node1,2) ;
    xyz(2,2) = XY(node2,2) ;

    % Rotation (from Global to local) matrix for all FE
    [Re(:,:,iFE),thetae(iFE)] = FromGlobal2LocalFrame(xyz) ;

    % From Global to Local frame
    fe = Re(:,:,iFE)*Fe ;

    % internal forces at node 1 (local numbering)
    NTM(iFE).node1.valueFint = fe(1:3) ; % expressed in the local frame
    NTM(iFE).node1.numNode = node1 ;
    NTM(iFE).node1.position = XY(node1,:) ;

    % internal forces at node 2 (local numbering)
    NTM(iFE).node2.valueFint = -fe(4:6) ; % expressed in the local frame
    NTM(iFE).node2.numNode = node2 ;
    NTM(iFE).node2.position = XY(node2,:) ;
end
```

Function `InternalForcesAllFEVISU.m`

```
function InternalForcesAllFEVISU(NTM,TC,Re,ampN,ampT,ampM)

fig = figure(2) ; clf ; hold on ;
fig.WindowStyle = "docked" ;

for iFE=1:length(TC)

    % Position initiales des nodes
    PosNodes = [NTM(iFE).node1.position ; NTM(iFE).node2.position] ;
    xNodes = PosNodes(:,1) ;
    yNodes = PosNodes(:,2) ;

    % Defo initiale sur chaque subplot
    subplot(1,3,1) ; hold on ;
    plot(xNodes,yNodes,'.-k') ;
    subplot(1,3,2) ; hold on ;
    plot(xNodes,yNodes,'.-k') ;
    subplot(1,3,3) ; hold on ;
    plot(xNodes,yNodes,'.-k') ;

    Le = sqrt(diff(xNodes)^2+diff(yNodes)^2) ;
    R = Re(1:2,1:2,iFE) ; % Matrice de rotation du repere global vers le local
                          % attention Re est une matrix 6x6 (deux nodes traites en meme temps en ↩
                               3D) il faut la reduire a 2x2 (1 point traite et 2D)

    % Position du point 1 associe aux efforts internes
    X1Y1 = PosNodes(1,:) ;

    % Efforts normal tranchant moment du node1
    N1 = NTM(iFE).node1.valueFint(1) ;
    T1 = NTM(iFE).node1.valueFint(2) ;
    M1 = NTM(iFE).node1.valueFint(3) ;
    PN1xy = [0 ; ampN*N1] ; % repere local) Position point effort normal dans l'espace du ↩
        maillage
    PN1XY = X1Y1' + R'*PN1xy ; % exprime dans repere global
    PT1xy = [0 ; ampT*T1] ; % repere local) Position point effort tranchant dans l'espace du ↩
        maillage
    PT1XY = X1Y1' + R'*PT1xy ; % exprime dans repere global
    PM1xy = [0 ; ampM*M1] ; % repere local) Position point effort moment dans l'espace du ↩
        maillage
    PM1XY = X1Y1' + R'*PM1xy ; % exprime dans repere global

    % Effort normal du node2
    N2 = NTM(iFE).node2.valueFint(1) ;
    T2 = NTM(iFE).node2.valueFint(2) ;
    M2 = NTM(iFE).node2.valueFint(3) ;
    PN2xy = [Le ; ampN*N2] ; % repere local) Position point effort normal dans l'espace du ↩
        maillage
    PN2XY = X1Y1' + R'*PN2xy ; % exprime dans repere global
    PT2xy = [Le ; ampT*T2] ; % repere local) Position point effort normal dans l'espace du ↩
        maillage
    PT2XY = X1Y1' + R'*PT2xy ; % exprime dans repere global
    PM2xy = [Le ; ampM*M2] ; % repere local) Position point effort normal dans l'espace du ↩
        maillage
    PM2XY = X1Y1' + R'*PM2xy ; % exprime dans repere global


    xNNodes = [PN1XY(1) ; PN2XY(1)] ;
    yNNodes = [PN1XY(2) ; PN2XY(2)] ;

    xTNodes = [PT1XY(1) ; PT2XY(1)] ;
    yTNodes = [PT1XY(2) ; PT2XY(2)] ;

    xMNodes = [PM1XY(1) ; PM2XY(1)] ;
    yMNodes = [PM1XY(2) ; PM2XY(2)] ;

    subplot(1,3,1) ; hold on ;
    plot(xNNodes,yNNodes,'.-r') ;
    xlabel('x (m)')
    ylabel('y (m)')
```

```
64      title('N distribution')
65      axis square ;
66      grid on ;
67      xlim([-0.5 2.5])
68      ylim([-2.5 2.5])
69
70      subplot(1,3,2) ; hold on ;
71      plot(xTNodes,yTNodes,'.-r') ;
72      xlabel('x (m)')
73      ylabel('y (m)')
74      title('T distribution')
75      axis square ;
76      grid on ;
77      xlim([-0.5 2.5])
78      ylim([-2.5 2.5])
79
80      subplot(1,3,3) ; hold on ;
81      plot(xMNodes,yMNodes,'.-r') ;
82      xlabel('x (m)')
83      ylabel('y (m)')
84      title('M distribution')
85      axis square ;
86      grid on ;
87      xlim([-0.5 2.5])
88      ylim([-2.5 2.5])
89
90
91  end
92
93
94
95  end
```

4.2 Dynamic conditions (Optional)

4.2.1 Main file

Main file `Main_Dy_L_WithoutNR.m`

```
clearvars ; clc ; format short ;

%% CURRENT FOLDER VERIFICATION
chm = 'D:\Enseignements\ModulesOptionnels\M8\CM\Handout\Figures\Dyna\EB_ComplexeGeom\002_CompactedAlgo';
if strcmp(chm,pwd) ~=1
    error('ATTENTION ; not in the right folder chm ~= pwd')
else
    addpath(genpath(chm))
end

%% REMINDER OF THE MODEL
Img = imread('ModelHypothesis.jpeg') ;
figure(1) ; clf;
imshow(Img) ;
pause
close figure 1 ;

%% INPUTS
INPUTS.Material.E = 200e9;          % Young modulus
INPUTS.Material.rho = 2500 ;        % density

INPUTS.Geometry.L = 5;              % Beam length
INPUTS.Geometry.H = 1;              % Beam height
INPUTS.Geometry.S = 0.1*0.1 ;  % Cross section area
INPUTS.Geometry.I = 0.1^4/12 ; % Cross section second moment of Inertia

% Loading case choice
%    RealEarthquake : accelerogram application
%    Triangular : Ponctual force, Triangular time evolution
%    Heavyside : Ponctual force, Heavyside time evolution
INPUTS.Load.LoadingCase = 'Triangular' ;
INPUTS.Load.g = 0*[0 ; -9.81] ; % gravity
INPUTS.Load.FchgtPonc = [10000 0] ;

INPUTS.Mesh.nEFPot1 = 6 ;       % number of FE for post 1
INPUTS.Mesh.nEFPout = 10 ;       % number of FE for the beam
INPUTS.Mesh.nEFPot2 = 6 ; % number of FE for post 2

INPUTS.Numeric.Tend = 5. ;
INPUTS.Numeric.dt   = 1e-2 ;         % Timestep
INPUTS.Numeric.GammaNM = 1/2 ;
INPUTS.Numeric.BetaNM = 1/4 ; % constante acceleration (unconditionaly stable)

INPUTS.dampRay.ksi = 0.01 ;  % damping from 0 to "freqRange" th natural frequency
INPUTS.dampRay.freqRange = 4 ;        % Rayleigh damping

%% DERIVED DATA
INPUTS.Numeric.t = 0:INPUTS.Numeric.dt:INPUTS.Numeric.Tend ;
INPUTS.Numeric.Ntps = length(INPUTS.Numeric.t) ;

%% MESHING
[XY,TC,DOF,INPUTS] = Meshing(INPUTS) ;

%% FINITE ELEMENT DATA
[ElemData,ElemHist] = ElemDataHistAffectation(XY,TC,INPUTS) ;

%% BOUNDARY CONDITIONS
[ddlCL,INPUTS] = BoundaryConditions(XY,DOF,INPUTS) ;

%% INITIALISATION
[U,Upt,U2pt,ElemDisp] = Initialisation(TC,DOF,INPUTS) ;

```

```
%% MATRICES ASSEMBLING
[M,K,FextVol,Fint] = Assembling(XY,TC,DOF,ElemData,ElemDisp,ElemHist) ;
MnoCL = M ; % for the construction fo the vector f=-MnoCL*iota*Acc

%% MODAL ANALYSIS - Damping (Raylegh type)
[C,INPUTS] = ModalAnalysisDampRay(M,K,INPUTS) ;

%% LOADING
[Fext] = LoadingTimeEvol(XY,DOF,FextVol,INPUTS) ;

%% TIME RESOLUTION
[U,Upt,U2pt] = TimeResolution(XY,TC,DOF,M,K,C,Fext,U,Upt,U2pt,ElemData,ElemDisp,ElemHist,'Results.mat',INPUTS) ;

%% Animation
%% POST PROCESSING
% Visualisation
PostProcVISU(1000,3,XY,TC,DOF,U,INPUTS) ;
```

4.2.2 Used Functions

Function `Meshing.m`

```
function [XY,TC,DOF,INPUTS] = Meshing(INPUTS)
H = INPUTS.Geometry.H ;
L = INPUTS.Geometry.L ;
nEFPot1 = INPUTS.Mesh.nEFPot1 ;
nEFPout = INPUTS.Mesh.nEFPout ;
nEFPot2 = INPUTS.Mesh.nEFPot2 ;

nEF = nEFPot1+nEFPout+nEFPot1 ;
nNodes = nEF+1 ;

% Post 1
iNode = 0 ;
Li = 0 ;
Le = H/nEFPot1 ;
for i=1:nEFPot1+1
    iNode = iNode+1 ;
    XY(iNode,:)=[0 Li];
    Li = Le + Li ;
end
% Beam
Le = L/nEFPout ;
Li = Le ;
for i=1:nEFPout
    iNode = iNode+1 ;
    XY(iNode,:)=[Li H];
    Li = Le + Li ;
end
% Post 2
Le = H/nEFPot2 ;
Li = Le ;
for i=1:nEFPot2
    iNode = iNode+1 ;
    XY(iNode,:)=[L (H-Li)];
    Li = Le + Li ;
end



%% CONNECTIVITY TABLE : from Finite Elements numbers gives the associated Node numbers
iEF = 0 ;
for i=1:nEF
    iEF = iEF+1 ;
    TC{iEF} = [iEF iEF+1] ;
end
%% DOF Table : from Node numbers gives the associated DOF
for iNode=1:nNodes
    DOF{iNode} = [(3*iNode-2) (3*iNode-1) 3*iNode] ;
end

%%
INPUTS.Mesh.nEF = nEF ;
INPUTS.Mesh.nNodes = nNodes ;
INPUTS.Mesh.nDDL = 3*nNodes ;

end
```

Function `ElemDataHistAffectation.m`

```
function [ElemData,ElemHist] = ElemDataHistAffectation(XY,TC,INPUTS)
H = INPUTS.Geometry.H ;
L = INPUTS.Geometry.L ;
S = INPUTS.Geometry.S ;
I = INPUTS.Geometry.I ;

E = INPUTS.Material.E ;
rho = INPUTS.Material.rho ;

g = INPUTS.Load.g ;

nEFPot1 = INPUTS.Mesh.nEFPot1 ;
nEFPout = INPUTS.Mesh.nEFPout ;
nEFPot2 = INPUTS.Mesh.nEFPot2 ;

nEF = length(TC) ;

iEF = 0 ;
% Post num 1
Le = H/nEFPot1 ;
for i=1:nEFPot1
    iEF = iEF+1 ;
    ElemData{iEF}.E = E ;
    ElemData{iEF}.S = S ;
    ElemData{iEF}.I = I ;
    Node1 = TC{iEF}(1) ;
    Node2 = TC{iEF}(2) ;
    Le = norm(XY(Node2,:)-XY(Node1,:)) ;
    ElemData{iEF}.Le = Le ;
    ElemData{iEF}.g = g ;
    ElemData{iEF}.rho = rho ;
end
% Beam
Le = L/nEFPout ;
for i=1:nEFPout
    iEF = iEF+1 ;
    ElemData{iEF}.E = E ;
    ElemData{iEF}.S = S ;
    ElemData{iEF}.I = I ;
    Node1 = TC{iEF}(1) ;
    Node2 = TC{iEF}(2) ;
    Le = norm(XY(Node2,:)-XY(Node1,:)) ;
    ElemData{iEF}.Le = Le ;
    ElemData{iEF}.g = g ;
    ElemData{iEF}.rho = rho ;
end
% Post n2
Le = H/nEFPot2 ;
for i=1:nEFPot2
    iEF = iEF+1 ;
    ElemData{iEF}.E = E ;
    ElemData{iEF}.S = S ;
    ElemData{iEF}.I = I ;
    Node1 = TC{iEF}(1) ;
    Node2 = TC{iEF}(2) ;
    Le = norm(XY(Node2,:)-XY(Node1,:)) ;
    ElemData{iEF}.Le = Le ;
    ElemData{iEF}.g = g ;
    ElemData{iEF}.rho = rho ;
end

for iEF=1:nEF
    ElemHist{iEF} = 0 ;
end

end
```

Function `BoundaryConditions.m`

```
function [ddlCL,INPUTS] = BoundaryConditions(XY,DOF,INPUTS)
BCPositions = [0 0 ; INPUTS.Geometry.L 0] ;

% First BC
xPyP = BCPositions(1,:) ;
inode = find(abs(XY(:,1)-xPyP(1))<0.00001 & ...
             abs(XY(:,2)-xPyP(2))<0.00001,1) ;
ddlCL1 = DOF{inode}(1:3) ; % Blocked node
% Second BC
xPyP = BCPositions(2,:) ;
inode = find(abs(XY(:,1)-xPyP(1))<0.00001 & ...
             abs(XY(:,2)-xPyP(2))<0.00001,1) ;
ddlCL2 = DOF{inode}(1:3) ; % Blocked node
% All BC
ddlCL = [ddlCL1 ddlCL2] ;

INPUTS.Mesh.dofBC = ddlCL ;
end
```

Function `Initialisation.m`

```
function [U,Upt,U2pt,ElemDisp] = Initialisation(TC,DOF,INPUTS)
Ntps = INPUTS.Numeric.Ntps ;
nEF = INPUTS.Mesh.nEF ;
nDDL = INPUTS.Mesh.nDDL ;
ddlCL = INPUTS.Mesh.dofBC ;
%% Initialisation of U, Upt and U2pt
U = cell(Ntps,1) ;
Upt = cell(Ntps,1) ;
U2pt = cell(Ntps,1) ;
U{1} = zeros(nDDL,1) ;
Upt{1} = zeros(nDDL,1) ;
U2pt{1} = zeros(nDDL,1) ;
it=1;
for iEF=1:nEF
    ElemDisp{iEF} = [U{it}(DOF{TC{iEF}(1)}) ; U{it}(DOF{TC{iEF}(2)})] ;
end
% Boundary condition application
for it =1:Ntps
    U{it} = zeros(nDDL,1) ;
    Upt{it} = zeros(nDDL,1) ;
    U2pt{it} = zeros(nDDL,1) ;
    U{it}(ddlCL) = [] ;
    Upt{it}(ddlCL) = [] ;
    U2pt{it}(ddlCL) = [] ;
end
end
```

Function `Assembling.m`

```
function [M,K,FextVol,Fint] = Assembling(XY,TC,DOF,ElemData,ElemDisp,ElemHist)
nEF    = length(TC) ;
nNodes = length(XY) ;
nDDL   = 3*nNodes ;
%% elementary mass and stiffness matrices
for iEF=1:nEF
    xyzelem=XY(TC{iEF},:);
    [Me(:,:,iEF),Ke(:,:,iEF),Feint(:,iEF),FeVol(:,iEF),ElemHist{iEF}] = ...
        BEAM_NB_LINE(xyzelem,ElemData{iEF},ElemDisp{iEF},ElemHist{iEF});
end
%% Assemblage following the DOF numbering
K=zeros(3*nNodes,3*nNodes) ;
M=zeros(3*nNodes,3*nNodes) ;
FextVol =zeros(3*nNodes,1);
Fint    = zeros(nDDL,1);
for iEF=1:nEF
    numDDL = [DOF{TC{iEF}(1)} DOF{TC{iEF}(2)}] ;
    K(numDDL,numDDL) = K(numDDL,numDDL) + Ke(:,:,iEF) ;
    M(numDDL,numDDL) = M(numDDL,numDDL) + Me(:,:,iEF) ;
    FextVol(numDDL) = FextVol(numDDL) + FeVol(:,iEF) ;
    Fint(numDDL) = Fint(numDDL) + Feint(:,iEF) ;
end
end
```

Function `ModalAnalysisDampRay.m`

```
function [C,INPUTS] = ModalAnalysisDampRay(M,K,INPUTS)

freqRange = INPUTS.dampRay.freqRange ;
ksi       = INPUTS.dampRay.ksi ;
ddlCL     = INPUTS.Mesh.dofBC ;

%% Boundary conditions application
K(ddlCL,:)=[];
K(:,ddlCL)=[];
M(ddlCL,:)=[];
M(:,ddlCL)=[];

%% damping (Rayleigh)
% Modal Analysis
[P,V]=eig(K^-1*M);

% Search of None zero values within V vector
for k=1:size(V,2)
    aa(k)=sum(V(:,k));
end
bb=find(aa~=0); % Position of none zero eigen values
fprintf('================================\n')
fprintf('Modal Frequency of the FE model\n')
fprintf('================================\n')
for k=1:length(bb)
    omega(k)=1/sqrt(V(bb(k),bb(k)));
    fprintf('f%d = %5.1f Hz\n',k,omega(k)/(2*pi))
end
% Eigen vectors
Phi = P(:,bb);

% Eigen values
omega_sorted = sort(omega) ;
omega1 = omega_sorted(1) ;
omega2 = omega_sorted(freqRange) ;

% Rayleigh coefficients and damping matrix
alpRay  = 2*ksi*omega1*omega2/(omega1+omega2);
betaRay = 2*ksi/(omega1+omega2);
C = alpRay*M+betaRay*K ;

% saving
INPUTS.dampRay.alpRay = alpRay ;
INPUTS.dampRay.betaRay = betaRay ;
end
```

Function `LoadingTimeEvol.m`

```
function [Fext] = LoadingTimeEvol(XY,DOF,FextVol,INPUTS)

H = INPUTS.Geometry.H ;
L = INPUTS.Geometry.L ;

Ntps = INPUTS.Numeric.Ntps ;
dt = INPUTS.Numeric.dt ;

nDDL = INPUTS.Mesh.nDDL ;
ddlCL = INPUTS.Mesh.dofBC ;

LoadingCase= INPUTS.Load.LoadingCase ;
Fchgt =  INPUTS.Load.FchgtPonc ;
FchgtX = Fchgt(1) ;
FchgtY = Fchgt(2) ;

%% Loading
if strcmp(LoadingCase,'RealEarthquake')==1
    %% Real Earthquake accelerogram
    filenameAccelero = 'Accelerometers\Seisme_EC8_1_EW.txt' ;
    %    filenameAccelero = 'Seisme_EC8_1_EW_Cut.txt' ; %=> marche pas !!!! pourquoi ?
    %    filenameAccelero = 'Seisme_Nice52.txt' ;
    %    filenameAccelero = 'SynchrFable.txt' ; % => deplacement nuls a la fin de l accelero ?!? bizarre
    [TpsAcc, Acc] = textread(filenameAccelero,'%f %f') ;
    Acc = Acc*1 ; % Sismic signal reduction
    INPUTS.Numeric.Tend = TpsAcc(end) ;
else
    %% Ponctual Force application at node iNodeChgt
    T1 = INPUTS.Numeric.Tend/10 ; % if Triangular time evolution of the loading is chosen
    iNodeChgt = find(abs(XY(:,2)-H)<0.00001 & abs(XY(:,1)-L)<0.00001) ;
    ddlForExt = DOF{iNodeChgt}(1:2) ; % appli. ponc. force on FE nEF (second node) in dir. 1 and 2 (x and y)
end


%% External loading creation (function of time)
% Constant force Application on top of post 1
Fext = cell(Ntps,1) ;
ForceEvol = [] ;

if strcmp(LoadingCase,'RealEarthquake')==1
    % Real ground acceleration
    Acc = interp1(TpsAcc,Acc,t);
    iota = zeros(nDDL,1) ;
    iota(1:3:end) = 1 ;
    for it=1:length(t)
        Fext{it} = FextVol -MnoCL*iota*Acc(it) ; % External force
        Fext{it}(ddlCL)=[]; % BC appli
    end
else
    for it=1:Ntps % Triangular loading through time
        vecforce = zeros(nDDL,1) ;
        if strcmp(LoadingCase,'Triangular')==1
            if it*dt < T1
                forcet = (it*dt)/T1*[FchgtX ; FchgtY] ;
                vecforce(ddlForExt) = forcet ;
            elseif T1 <= it*dt && it*dt <= 2*T1
                forcet = [FchgtX ; FchgtY] - (it*dt-T1)/(2*T1-T1)*[FchgtX ; FchgtY] ;
                vecforce(ddlForExt) = forcet ;
            elseif it*dt > 2*T1
                vecforce(ddlForExt) = [0 ; 0] ;
            end
        elseif strcmp(LoadingCase,'Heavyside')==1
            forcet = [FchgtX ; FchgtY] ;
            vecforce(ddlForExt) = forcet ;
        end
        ForceEvol = [ForceEvol forcet] ;
        Fext{it} = FextVol + vecforce ; % ponc. force and body forces (coming from gravity)
        Fext{it}(ddlCL)=[]; % BC appli
```

```
69      end
70  end
71
72  % iEFChgt
73  % ddlForExt
74  %
75  % figure(1) ;
76  % plot(t,ForceEvol(2,:))
77  % plot(t,Acc)
78  % return
79  end
```

Function `TimeResolution.m`

```
function [U,Upt,U2pt] = TimeResolution(XY,TC,DOF,M,K,C,Fext,U,Upt,U2pt,ElemData,ElemDisp,ElemHist
    ,filenameResult,INPUTS)

ddlCL = INPUTS.Mesh.dofBC ;
GammaNM = INPUTS.Numeric.GammaNM ;
BetaNM = INPUTS.Numeric.BetaNM ;
Ntps = INPUTS.Numeric.Ntps ;
dt = INPUTS.Numeric.dt ;

%% Boundary conditions application
M(ddlCL,:)=[];
M(:,ddlCL)=[];
K(ddlCL,:)=[];
K(:,ddlCL)=[];

%% equivalent stiffness matrix in NM algo
Spf = K+GammaNM/(BetaNM*dt)*C+1/(BetaNM*dt^2)*M ;

%% Newmark Resolution through time
cptAff = 0 ; VecDDL = cell2mat(DOF) ; VecDDL(ddlCL) = [] ; f = cell(Ntps,1) ;
for it=1:Ntps-1
    % Displacements
    aNM = GammaNM/(BetaNM*dt)*C+1/(BetaNM*dt^2)*M ;
    bNM = C*(GammaNM/BetaNM-1)+M/(dt*BetaNM) ;
    cNM = C*dt*(GammaNM/BetaNM*(1/2-BetaNM)-(1-GammaNM))+(1/2-BetaNM)/(BetaNM)*M ;
    f{it+1} = Fext{it+1} + aNM*U{it} + bNM*Upt{it} + cNM*U2pt{it} ;
    U{it+1} = Spf\f{it+1} ;
    % Velocities and accelerations
    Upt{it+1}=Upt{it} + (1-GammaNM)*dt*U2pt{it}+ ...
        GammaNM/(BetaNM*dt)*(U{it+1}-U{it}-dt*Upt{it}-(1/2-BetaNM)*dt^2*U2pt{it}) ;
    U2pt{it+1}=1/(BetaNM*dt^2)*(U{it+1}-U{it}-dt*Upt{it}-(1/2-BetaNM)*dt^2*U2pt{it}) ;
    % Remaing computation time
    if cptAff == 100
        disp(' ')
        disp(['Niveau de calcul = ' sprintf('%1.1f',it/Ntps*100) ' %'])
        disp(['Time = ' num2str(norm(it*dt)) ' s'])
        cptAff = 1 ;
    else
        cptAff = cptAff +1 ;
    end
end

% Workspace saving
save(filenameResult)

end
```

Function `PostProcVISU.m`

```
function PostProcVISU(amp,update,XY,TC,DOF,U,INPUTS)

ddlCL = INPUTS.Mesh.dofBC ;
nNodes = INPUTS.Mesh.nNodes ;
nEF = INPUTS.Mesh.nEF ;
Ntps = INPUTS.Numeric.Ntps ;

% reaffectation des ddl pour le trace
VecDDL = cell2mat(DOF) ;
VecDDL(ddlCL) = [] ;
UCL = cell(Ntps,1) ;
for it=1:Ntps
    UCL{it}=zeros(3*nNodes,1);
    UCL{it}(VecDDL)=U{it};
end
U = UCL ;

filename = 'testAnimated.gif';
for it=1:update:50%Ntps
    h=figure(1) ; clf ;
    for iEF=1:nEF
        indicesNoeu = TC{iEF} ;
        indiceDDL = DOF{indicesNoeu} ;
        % configuration initiale
        x = XY(indicesNoeu,1) ;
        y = XY(indicesNoeu,2) ;
        plot(x,y,'.-','LineWidth',1,'Color','k') ; hold on ;
        % configuraion deformee
        indicesDDLx = [] ;
        indicesDDLy = [] ;
        for i=1:2
            indicesDDLx = [indicesDDLx DOF{indicesNoeu(i)}(1)] ;
            indicesDDLy = [indicesDDLy DOF{indicesNoeu(i)}(2)] ;
        end
        x = XY(indicesNoeu,1)+amp*U{it}(indicesDDLx) ;
        y = XY(indicesNoeu,2)+amp*U{it}(indicesDDLy) ;
        plot(x,y,'.-','LineWidth',1,'Color','r') ; hold on ;
    end
    axis equal ; axis([-1 6 -1 2])
    grid on ;


        % Capture the plot as an image
      frame = getframe(h);
      im = frame2im(frame);
      [imind,cm] = rgb2ind(im,256);
      % Write to the GIF File
      if it == 1
          imwrite(imind,cm,filename,'gif', 'Loopcount',inf,'DelayTime',0.2);
      else
          imwrite(imind,cm,filename,'gif','WriteMode','append','DelayTime',0.2);
      end


    %pause(0.01)
end
```

Chapter 5

Project Aims

- **Global project scale**
  - Design a building using the fundamental principles of structural design
  - Propose 1/200 scale plans
  - Develop the ability to implement calculation methods in algorithms
  - Optimize solutions according to multiple criteria
  - Collaborate to solve complex problems
- **Structure scale**
  - Design and dimensioning of a post-beam structure
  - Development of the FE model in linear elasticity (2D) from **Euler-Bernoulli** type beam elements (under MATLAB).
  - Maximum internal forces and displacements assessment and cross sections sizing according to the material considered.
  - Analysis of the effect of different loading scenarii (dead weight, wind, operating load, *etc.*) on sizing.

Some helpful references :

- Biggs (1964) : Structural dynamics

- Chopra (1995) : Structural dynamics (Earthquake Enginerring)
- Hoadley (2000) : Wood material
- Benoit et al. (2008) : Wood structures (EC5)
- Sorensen (2004) : : Structural reliability
- Mosley et al. (2007) : Concrete structures
- Roux (2009) : Concrete structures
- Beeby and Narayanan (2005) : Concrete structures
- Moy (1981) : Limit analysis for steel and concrete structures

Bibliography

L. Martin and J. Purkiss. *Concrete design to EN 1992*. 1996.

J. Biggs. *Introduction to structural dynamics*. McGraw-Hill, 1964.

A. Chopra. *Dynamics of structures - Theory and Application to Earthquake Engineering*. Prentice Hall, 1995.

B. Hoadley. *Understanding wood*. The Taunton Press, 2000.

Y. Benoit, B. Legrand, and V. Tastet. *Calcul des structures en bois - Guide d'application*. EYROLLES, 2008.

J. Sorensen. *Notes in Structural Reliability Theory and Risk Analysis*. 2004.

B. Mosley, J. Bungey, and R. Hulse. *Reinforced concrete design to Eurocode 2*. Palgrave Macmillan, 2007.

J. Roux. *Maitrise de l'eurocode 2 - Guide d'application*. 2009.

A. Beeby and R. Narayanan. *Designers' guide to EN 1992-1-1 and EN 1992-1-2, EuroCode 2: Design of concrete structures. General rules and rules for buildings and structural fire design*. 2005.

S. Moy. *Plastic methods for steel and concrete Structures*. Macmillan, 1981.