POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

Springs and BEM

Final Report of
Advanced Computational Mechanics - G. Novati

Master Degree in
Mathematical Engineering

Author: **Matteo Bonfadini**

Student ID: XXXXXX
Academic Year: 2023-24

Contents

1 | System of Nonlinear Springs

1.1. Introduction

Many engineering applications show nonlinear behaviours, and linear systems cannot provide an acceptable solution in more and more situations. In solid mechanics, such a situation usually occurs when the deformation is large, material response is complex, boundary conditions vary, etc. In this context, linear systems could be seen as approximation of nonlinear systems under limited conditions, e.g., with small deformation.

In general there is no analytical way of finding the solution of a system of nonlinear equations, and this explains why computational sciences have become increasingly in demand.

In the following section we discuss a variation of a problem presented during the course and in the book [2].

1.2. System of Nonlinear Springs

Consider three non-linear springs connected in series and in parallel, as shown in Figure 1.1.

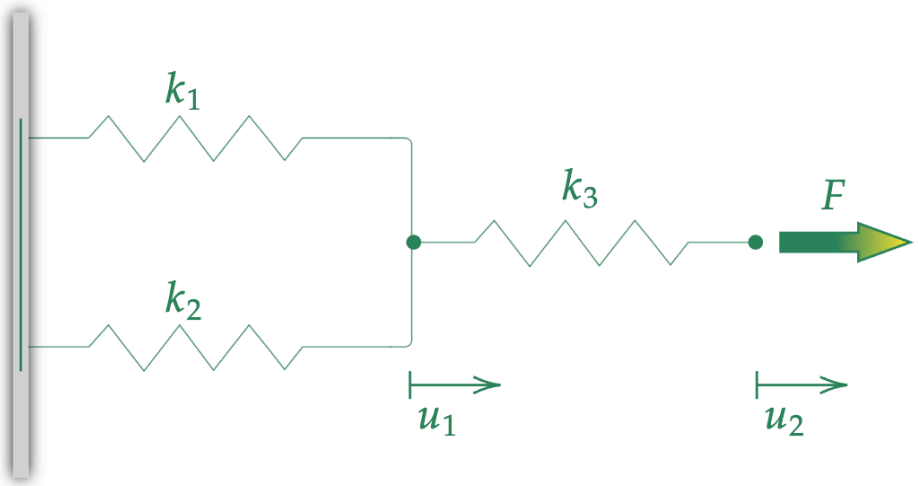


Figure 1.1: System of nonlinear springs.

The stiffness of the springs depends on the elongation of springs such that

$$k_i = k_i(e_i) = \alpha_i + \beta_i \cdot e_i \quad [\text{N/mm}], \qquad i = 1, 2, 3, \tag{1.1}$$

with e_i being the elongation of the i-th spring and coefficients (α_i, β_i) being listed in Table 1.2. In this way, although the geometry of the problem is linear, it is ruled by nonlinear costitutive laws.

In Figure 1.1 is also shown the load application $F = 100$ [N] at the tip.

Our interest lies in calculating u_1 and u_2, the nodal displacements at the two nodes.

i	α_i	β_i
1	500	50
2	200	100
3	500	100

Table 1.1: Springs coefficients.

From Figure 1.2 one can notice that the stiffness (the local tangent) is increasing for increased elongations.

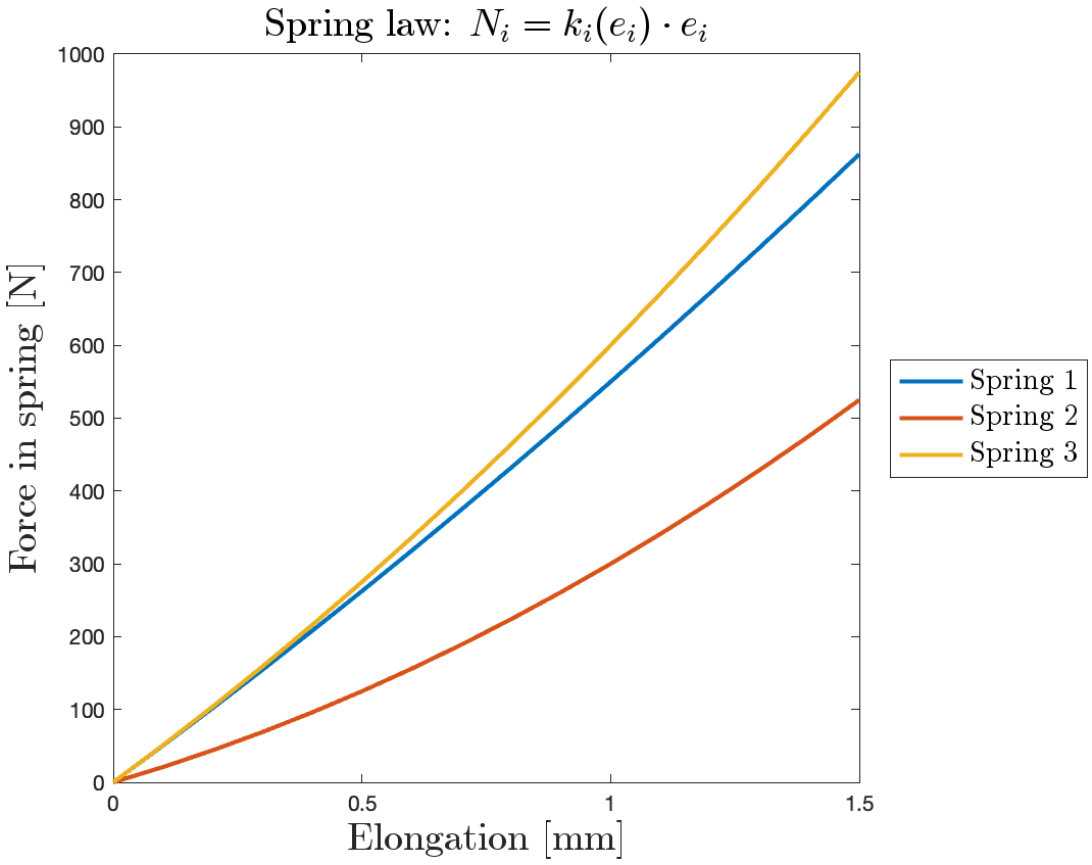


Figure 1.2: Costitutive law plot.

Since springs 1 and 2 are fixed on the wall, their elongation is equivalent to u_1, while for spring 3, the elongation is $u_2 - u_1$. It's very easy to find out the equilibrium of the two nodes:

$$\begin{cases} N_1 + N_2 - N_3 = 0, \\ N_3 = F. \end{cases} \tag{1.2}$$

It follows that the matrix notation of the system in term of unknown displacement u is

$$\mathbf{P}(\mathbf{u}) = \mathbf{f} \tag{1.3}$$

or, in a more physical notation,

$$\mathbf{F^{int}}(\mathbf{u}) = \mathbf{F^{ext}}, \tag{1.4}$$

hence, the system is the following:

$$\begin{bmatrix} k_1 + k_2 + k_3 & -k_3 \\ -k_3 & +k_3 \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \end{bmatrix} = \begin{bmatrix} 0 \\ F \end{bmatrix}. \tag{1.5}$$

Note that, as we said before, the above equations are not linear as the stiffness matrix contains unknown variables.

Using the given stiffness of springs, the general matrix equation becomes

$$\begin{cases} (\beta_1 + \beta_2 - \beta_3)\, u_1^2 + (\alpha_1 + \alpha_2 + \alpha_3)\, u_1 + 2\beta_3 u_1 u_2 - \alpha_3 u_2 - \beta_3 u_2^2 = 0 \\ \beta_3 u_1^2 - \alpha_3 u_1 - 2\beta_3 u_1 u_2 + \alpha_3 u_2 + \beta_3 u_2^2 = F \end{cases}, \tag{1.6}$$

which, for the sake of generality, leads to

$$\begin{cases} \mathcal{A} u_1^2 + \mathcal{B} u_1 + \mathcal{C} u_1 u_2 + \mathcal{D} u_2 + \mathcal{E} u_2^2 = 0 \\ \mathcal{F} u_1^2 + \mathcal{G} u_1 + \mathcal{H} u_1 u_2 + \mathcal{I} u_2 + \mathcal{L} u_2^2 = F \end{cases}. \tag{1.7}$$

In this way it will be easier to build the system in MATLAB, namely with

$$\begin{bmatrix} \mathcal{A} & \mathcal{B} & \mathcal{C} & \mathcal{D} & \mathcal{E} \\ \mathcal{F} & \mathcal{G} & \mathcal{H} & \mathcal{I} & \mathcal{L} \end{bmatrix} \begin{bmatrix} u_1^2 \\ u_1 \\ u_1 u_2 \\ u_2 \\ u_2^2 \end{bmatrix}. \tag{1.8}$$

Finally, the assembled system is

$$\begin{cases} 50u_1^2 + 1200u_1 + 200u_1 u_2 - 500u_2 - 100u_2^2 = 0 \\ 100u_1^2 - 500u_1 - 200u_1 u_2 + 500u_2 + 100u_2^2 = 100 \end{cases}. \tag{1.9}$$

In the following section, a method of solving the system of nonlinear equations is discussed.

1.3. Numerical Solution

Before discussing systems, it is important to understand the problem of numerical approximation of the zeros of a real-valued function of one variable, that is

$$\boxed{\text{Given } f : (a,b) \to \mathbb{R}, \text{ find } \alpha \in \mathbb{R} \text{ s.t. } f(\alpha) = 0} \tag{1.10}$$

There are several iterative methods to solve this rootfinding problem, such as the bisection method, the chord method, the secant method, fixed point methods, etc. For a more complete treatment of these, see for example [8].

In the following we analyze another algorithm: the Newton's method, also known as the **Newton–Raphson method**.

Assuming that $f \in \mathcal{C}^1\big((a,b)\big)$ and that α is a simple root of f, if we let

$$q_k = f'(x^k), \qquad \forall k \geq 0 \tag{1.11}$$

and assign the initial value x^0, we obtain the Newton scheme

$$x^{k+1} = x^k - \frac{f(x^k)}{q_k}, \qquad k \geq 0. \tag{1.12}$$

The idea is clarified by looking at the first-order series expansion of $f(x)$ in the neighborhood of x^k

$$f(x) \approx f(x^k) + \left[\frac{\mathrm{d}f}{\mathrm{d}x}(x^k)\right](x - x^k) \qquad \xrightarrow{x=x^{k+1}} \qquad \text{Equation (1.12)} \tag{1.13}$$

and by Figure 1.3.

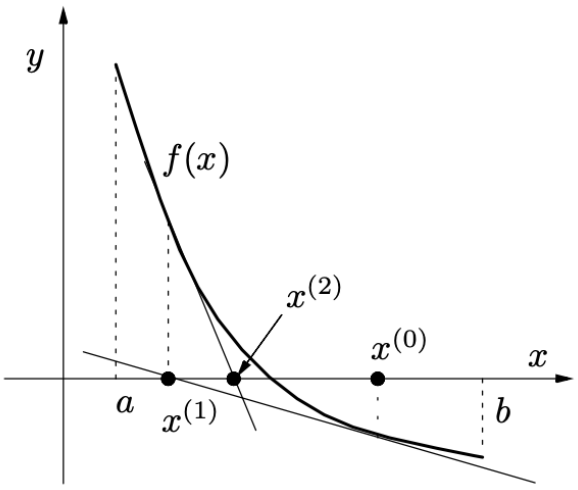


Figure 1.3: Geometric approach of Newton-Raphson method: first two steps.

This method has an high order convergence, namely $p = 2$, i.e.

$$\exists C > 0 \qquad \text{s.t.} \qquad \lim_{k\to\infty} \frac{\left|x^{k+1} - \alpha\right|}{\left|x^k - \alpha\right|^2} = C, \tag{1.14}$$

but α should be a simple root and the initial value x^0 should be *close* to the solution, because the method does not always guarantee convergence (actually, sometimes the solution may diverge or oscillate between two points).

An immediate extension of Newton-Raphson's method (1.12) for the nonlinear system (1.3) can be recursively formulated as

$$\mathbf{u}^{k+1} = \mathbf{u}^k + \frac{\mathbf{f} - \mathbf{P}(\mathbf{u}^k)}{\mathbf{K}_T^k} \qquad \text{with} \qquad \mathbf{K}_T^k = \frac{\partial \mathbf{P}}{\partial \mathbf{u}}(\mathbf{u}^k), \tag{1.15}$$

which leads to the following algorithm:

$$\begin{aligned} &\text{given } \mathbf{u}^0, \text{ for } k = 0, 1, \ldots, \text{ until convergence:} \\ &\text{compute} \quad \mathbf{K}_T^k \text{ and } \mathbf{R}^k = \mathbf{f} - \mathbf{P}(\mathbf{u}^k), && (1.16a) \\ &\text{solve} \quad \mathbf{K}_T^k \, \Delta\mathbf{u}^k = \mathbf{R}^k, && (1.16b) \\ &\text{set} \quad \mathbf{u}^{k+1} = \mathbf{u}^k + \Delta\mathbf{u}^k. && (1.16c) \end{aligned}$$

In general, this solution will not satisfy the system of nonlinear equations exactly and there will be some residual (or unbalance force in a physical point of view). If the k-th residual is smaller than a given tolerance, the solution $\mathbf{u}^{k+1}$ can be accepted as the accurate solution, and the process stops. The stopping criterion is expressed as follows

$$\text{conv}^{\,k+1} = \frac{\displaystyle\sum_{j=1}^{n} \left(\mathrm{R}_j^{k+1}\right)^2}{1 + \displaystyle\sum_{j=1}^{n} \left(\mathrm{f}_j\right)^2}, \tag{1.17}$$

and, in order to show quadratic convergence as in Equation (1.14), it is often enough to show that **conv** reduces quadratically at each iteration.

! From Equation (1.16b), one can notice that at each step k the solution of a linear system with matrix $\mathbf{K}_T^k$ is required, thus the matrix cannot be singular. We will use the *backslash* MATLAB command to solve the system, but the computational cost increases as n^2. There are several ways to solve this, for example use the LU decomposition and keep it fixed for some iterations, or just keep $\mathbf{K}_T^k$ fixed. See Secion 1.4 for further details.

1.4. Numerical Implementation

Here we present and explain the MATLAB scripts used to solve the problem. The algorithms are listed in the Subsection 1.4.1.

In `main11` (Algorithm 1.1) we implement the Newton-Raphson method previously described. Here two auxiliary functions are used to compute the internal force vector $\mathbf{P}(\mathbf{u})$ and its Jacobian matrix $\mathbf{K}_T$ at each iteration. These MATLAB functions `generate_int_force` (Algorithm 1.2) and `generate_jacobian` (Algorithm 1.3) are merely based on Equation (1.8).

The output of the code is the following:

```
**********************************************************************
RESULTS
**********************************************************************

iter   u1        u2        conv
   0   2.00000   2.00000   4.00960e+02
   1   0.53846   0.73846   1.00124e+01
   2   0.16655   0.35914   4.30187e-02
   3   0.13889   0.33147   1.31776e-06
   4   0.13873   0.33132   1.29140e-15

Convergence reached in 4 iterations

**********************************************************************
```

The initial guess choosen was $\mathbf{u}^0 = [2; 2]$, but we can easily repeat the process for other initial estimates. For example, with $\mathbf{u}^0 = [0; 0]$ we have

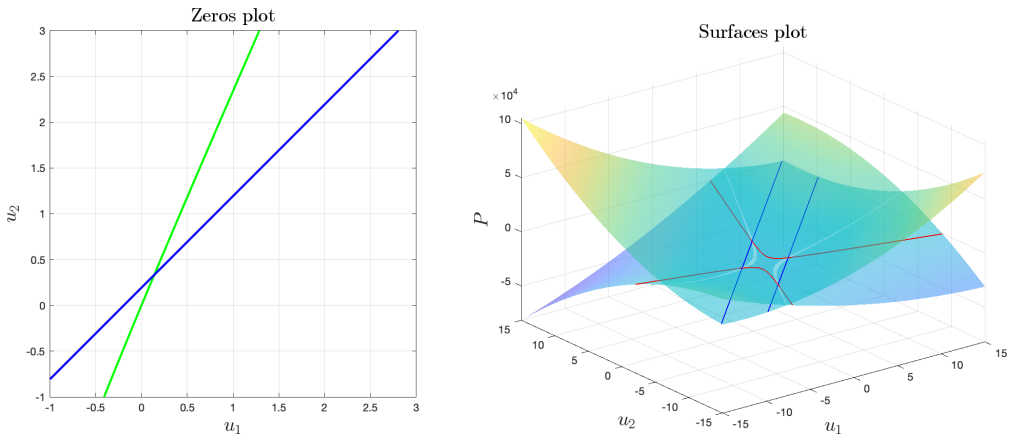
```
**********************************************************************
RESULTS
**********************************************************************

iter   u1        u2        conv
   0   0.00000   0.00000   9.99900e-01
   1   0.14286   0.34286   1.68796e-03
   2   0.13874   0.33133   3.87449e-09
   3   0.13873   0.33132   1.81925e-20

Convergence reached in 3 iterations

**********************************************************************
```

and with $\mathbf{u}^0 = [9; 9]$ we have

```
**********************************************************************
RESULTS
**********************************************************************

iter    u1          u2          conv
   0    9.00000     9.00000     3.40378e+04
   1    3.60294     3.80294     1.90534e+03
   2    1.14953     1.34212     8.15108e+01
   3    0.28541     0.47799     1.25440e+00
   4    0.14284     0.33542     9.29483e-04
   5    0.13874     0.33132     6.38409e-10

Convergence reached in 5 iterations

**********************************************************************
```

From all these outputs we can say that as MATLAB program iterates, the displacements converge to the numerical solution $\mathbf{u} = [0.13873; 0.33132]$ mm; secondly, the number of iterations depends on the starting point, however the residual reduction via convergence criterion (1.17) is approximately quadratic.

We can double check the accuracy of our solution via plotting the surfaces of the system of nonlinear equations with zero-level contour. The results we obtain with `main12` (Algorithm 1.4) ant the `plotzeros` function (Algorithm 1.5) are the following figures:



(a) Zero-level contour plot. (b) 3D Surface plot.

Figure 1.4: Zero-level contour and Surface plots.

Note that there exist four solutions to the problem, but only one of them, the one we found, is positive.

Thanks to script `main13` (Algorithm 1.6) we generate useful data for the subsequent processing in Abaqus (see Section 1.5) and Figure 1.2.

The Newton–Raphson method requires that at each iteration, the Jacobian matrix should be formed and the system of linearized equations should be solved for the increment of the solution. Computationally, these are expensive tasks. The **Modified Newton–Raphson method** is an attempt to make these procedures less expensive. Instead of formulating a new tangent stiffness matrix at each iteration, the initial tangent stiffness matrix is repeatedly used for all iterations.

Through `main14` (Algorithm 1.7) we implement the Modified Newton-Raphson method, gaining the following output:

```
**********************************************************************
RESULTS
**********************************************************************

iter   u1        u2        conv
   0   2.00000   2.00000   4.00960e+02
   1   0.53846   0.73846   1.00124e+01
   2   0.29199   0.48399   1.38037e+00
   3   0.20185   0.39448   2.24486e-01
   4   0.16538   0.35796   3.94896e-02
   5   0.15010   0.34268   7.13582e-03
   6   0.14360   0.33618   1.30509e-03
   7   0.14082   0.33340   2.39875e-04
   8   0.13963   0.33221   4.41837e-05
   9   0.13912   0.33170   8.14585e-06
  10   0.13890   0.33148   1.50238e-06
  11   0.13880   0.33139   2.77140e-07
  12   0.13876   0.33135   5.11268e-08
  13   0.13875   0.33133   9.43218e-09
  14   0.13874   0.33132   1.74013e-09
  15   0.13874   0.33132   3.21035e-10

Convergence reached in 15 iterations

**********************************************************************
```

The method usually requires a greater number of iterations for convergence than that of the regular Newton–Raphson method. However, the overall computational cost to obtain the solution can be made less because each iteration is much faster than that of the regular Newton–Raphson method. The method is also a little more stable and is not prone to divergence.

1.4.1. MATLAB scripts

Algorithm 1.1: main11

```
clear; clc; close;
% data
tol = 1.0e-9; iter = 0; max_iter = 100; u = [2;2]; old_u = u; F = 100;
a1 = 500; b1 = 50; a2 = 200; b2 = 100; a3 = 500; b3 = 100;
k1 = a1+b1*u(1); k2 = a2+b2*u(1); k3 = a3+b3*(u(2)-u(1));
% internal force vector
P = generate_int_force(u,a1,b1,a2,b2,a3,b3);
% external force vector
FF = [0; F];
% residual
R = FF - P;
% convergence parameter
conv = (R(1)^2+R(2)^2)/(1+FF(1)^2+FF(2)^2);
conv_rate = [];
fprintf('%s', repmat('*', 1, 69));
fprintf('\nRESULTS\n');
fprintf('%s', repmat('*', 1, 69));
fprintf('\n');
fprintf('\niter   u1        u2        conv');
fprintf('\n %3d   %7.5f   %7.5f   %7.5e',iter,u(1),u(2),conv);
while conv > tol && iter < max_iter
    % jacobian matrix of P
    Kt = generate_jacobian(u,a1,b1,a2,b2,a3,b3);
    % check if it is singular
    if det(Kt)==0
        fprintf('\n\n');
        error('Jacobian is singular!')
    end
    % solve the linear system
    delta_u = Kt\R;
    % update of the solution
    u = old_u + delta_u;
    % update of the internal force vector
    P = generate_int_force(u,a1,b1,a2,b2,a3,b3);
    % update of the residual
    R = FF - P;
    % update of the convergence parameter
    conv = (R(1)^2+R(2)^2)/(1+FF(1)^2+FF(2)^2);
    % save it in a vector
    conv_rate = [conv_rate; conv];
    % ready for a new iteration
    old_u = u;
```

```
    iter = iter + 1;
    % print of the results
    fprintf('\n %3d   %7.5f   %7.5f   %7.5e',iter,u(1),u(2),conv);
end
if iter==max_iter
    fprintf('\n\n');
    error('Convergence not reached!')
end
fprintf('\n\nConvergence reached in %d iterations',iter);
fprintf('\n\n');
fprintf('%s', repmat('*', 1, 69)); fprintf('\n\n');
% p = log2(conv_rate(1:end-1)./conv_rate(2:end));
```

Algorithm 1.2: generate int force

```
function P = generate_int_force(u,a1,b1,a2,b2,a3,b3)
A = b1+b2-b3; B = a1+a2+a3; C = 2*b3; D = -a3; E = -b3;
F = b3; G = -a3; H = -2*b3; I = a3; L = b3;
U = [u(1)^2;u(1);u(1)*u(2);u(2);u(2)^2];
P1 = [A,B,C,D,E;F,G,H,I,L];
P = P1*U;
end
```

Algorithm 1.3: generate jacobian

```
function J = generate_jacobian(u,a1,b1,a2,b2,a3,b3)
A = b1+b2-b3; B = a1+a2+a3; C = 2*b3; D = -a3; E = -b3;
F = b3; G = -a3; H = -2*b3; I = a3; L = b3;
U = [u(1);u(2);1];
J1 = [2*A,C,B;C,2*E,D;2*F,H,G;H,2*L,I];
J2 = J1*U;
J = [J2(1),J2(2);J2(3),J2(4)];
end
```

Algorithm 1.4: main12

```
clear; clc; close;
a1 = 500; b1 = 50; a2 = 200; b2 = 100; a3 = 500; b3 = 100; f = 100;
A = b1+b2-b3; B = a1+a2+a3; C = 2*b3; D = -a3; E = -b3;
F = b3; G = -a3; H = -2*b3; I = a3; L = b3;
fun = @(u) [A*u(1)^2+B*u(1)+C*u(1)*u(2)+D*u(2)+E*u(2)^2;
            F*u(1)^2+G*u(1)+H*u(1)*u(2)+I*u(2)+L*u(2)^2-f];
plotzeros(fun, [-1 3], [-1 3])
%
figure()
[X,Y] = meshgrid(-15:.5:15);
Z1 = (b1+b2-b3)*X.^2+(a1+a2+a3)*X+2*b3*X.*Y-a3*Y-b3*Y.^2;
s1 = surf(X,Y,Z1,'FaceAlpha',0.5);
s1.EdgeColor = 'none';
xlabel('$u_1$','Interpreter','latex',FontSize=18)
ylabel('$u_2$','Interpreter','latex',FontSize=18)
zlabel('$P$','Interpreter','latex',FontSize=18)
title('Surfaces plot','Interpreter','latex',FontSize=18)
hold on
Z2 = b3*X.^2-a3*X-2*b3*X.*Y+a3*Y+b3*Y.^2-F;
s2 = surf(X,Y,Z2,'FaceAlpha',0.5);
s2.EdgeColor = 'none';
contour(X,Y,Z1, [0 0], 'r', 'LineWidth', 1);
contour(X,Y,Z2, [0 0], 'b', 'LineWidth', 1);
```

Algorithm 1.5: plotzeros

```
function plotzeros(fun, x_limits, y_limits, varargin)
if nargin > 3
    sample_len = varargin{1};
else
    sample_len = 500;
end
v_dis_x = linspace(x_limits(1), x_limits(2), sample_len);
v_dis_y = linspace(y_limits(1), y_limits(2), sample_len);
n_dis_x = length(v_dis_x);
n_dis_y = length(v_dis_y);
Z_1 = zeros(n_dis_y, n_dis_x);
Z_2 = zeros(n_dis_y, n_dis_x);
for i = 1:n_dis_x
    for j = 1:n_dis_y
        funz = fun([v_dis_x(i); v_dis_y(j)]);
        Z_1(j,i) = funz(1); Z_2(j,i) = funz(2);
    end
end
```

```
contour(v_dis_x, v_dis_y, Z_1, [0 0], 'Linecolor',[0 1 0], 'Linewidth',
   2)
hold on
contour(v_dis_x, v_dis_y, Z_2, [0 0], 'Linecolor',[0 0 1], 'Linewidth',
   2)
axis equal
grid on
xlabel('$u_1$','Interpreter','latex',FontSize=18)
ylabel('$u_2$','Interpreter','latex',FontSize=18)
title('Zeros plot','Interpreter','latex',FontSize=18)
end
```

Algorithm 1.6: main13

```
clear; clc; close; format shorte
U = 0:0.1:1.5; Ut = U'
a1 = 500; b1 = 50; a2 = 200; b2 = 100; a3 = 500; b3 = 100;
k1 = a1+b1*Ut; k2 = a2+b2*Ut; k3 = a3+b3*Ut; kEQ = k1+k2;
F1 = k1.*Ut;
figure(1)
plot(Ut,F1,'linewidth',2)
xlabel('Elongation [mm]','Interpreter','latex',FontSize=18)
ylabel('Force in spring [N]','Interpreter','latex',FontSize=18)
title('Spring law: $N_i=k_i(e_i)\cdot e_i$','Interpreter','latex',
   FontSize=18)
hold on
F2 = k2.*Ut;
plot(Ut,F2,'linewidth',2)
F3 = k3.*Ut;
plot(Ut,F3,'linewidth',2)
legend('Spring 1','Spring 2','Spring 3','Interpreter','latex',Location='
   eastoutside',FontSize=14)
F12 = kEQ.*Ut
figure(2)
plot(Ut,F12,'linewidth',2)
xlabel('Elongation [mm]','Interpreter','latex',FontSize=18)
ylabel('Force in spring [N]','Interpreter','latex',FontSize=18)
title('Spring law: $N_i=k_i(e_i)\cdot e_i$','Interpreter','latex',
   FontSize=18)
hold on
F3 = k3.*Ut
plot(Ut,F3,'linewidth',2)
legend('Spring 1+2','Spring 3','Interpreter','latex',Location='
   eastoutside',FontSize=14)
```

Algorithm 1.7: main14

```
clear; clc; close;
tol = 1.0e-9; iter = 0; max_iter = 100; u = [2;2]; old_u = u; F = 100;
a1 = 500; b1 = 50; a2 = 200; b2 = 100; a3 = 500; b3 = 100;
k1 = a1+b1*u(1); k2 = a2+b2*u(1); k3 = a3+b3*(u(2)-u(1));
P = generate_int_force(u,a1,b1,a2,b2,a3,b3);
% jacobian matrix of P: FIXED
Kt = generate_jacobian(u,a1,b1,a2,b2,a3,b3);
FF = [0; F];
R = FF - P;
conv = (R(1)^2+R(2)^2)/(1+FF(1)^2+FF(2)^2);
conv_rate = [];
fprintf('%s', repmat('*', 1, 69));
fprintf('\nRESULTS\n');
fprintf('%s', repmat('*', 1, 69));
fprintf('\n');
fprintf('\niter   u1        u2        conv');
fprintf('\n %3d   %7.5f   %7.5f   %7.5e',iter,u(1),u(2),conv);
while conv > tol && iter < max_iter
    delta_u = Kt\R;
    u = old_u + delta_u;
    P = generate_int_force(u,a1,b1,a2,b2,a3,b3);
    R = FF - P;
    conv = (R(1)^2+R(2)^2)/(1+FF(1)^2+FF(2)^2);
    conv_rate = [conv_rate; conv];
    old_u = u;
    iter = iter + 1;
    fprintf('\n %3d   %7.5f   %7.5f   %7.5e',iter,u(1),u(2),conv);
end
if iter==max_iter
    fprintf('\n\n');
    error('Convergence not reached!')
end
fprintf('\n\nConvergence reached in %d iterations',iter);
fprintf('\n\n');
fprintf('%s', repmat('*', 1, 69)); fprintf('\n\n');
% p = log2(conv_rate(1:end-1)./conv_rate(2:end));
```

1.5. Abaqus Implementation

Here we solve the problem through the Abaqus Software by using spring-like connectors. The adopted units of measure are mm and N in order to be consistent, as Abaqus requires.

Before starting, let's remark that our nonlinear system shown in Figure 1.1 can be semplified in a system of two serial connected springs:

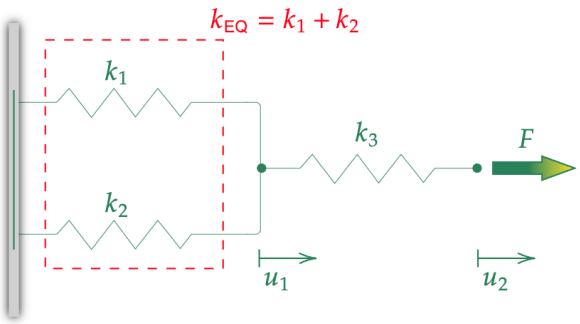


Figure 1.5: Equivalent system of nonlinear springs.

1.5.1. Interaction module

We define the nodes of the spring system by setting three colinear reference points (the distance between the nodes is not important) using the "Create Reference Point" button. Then we re-scale the view and the result is shown in Figure 1.6.

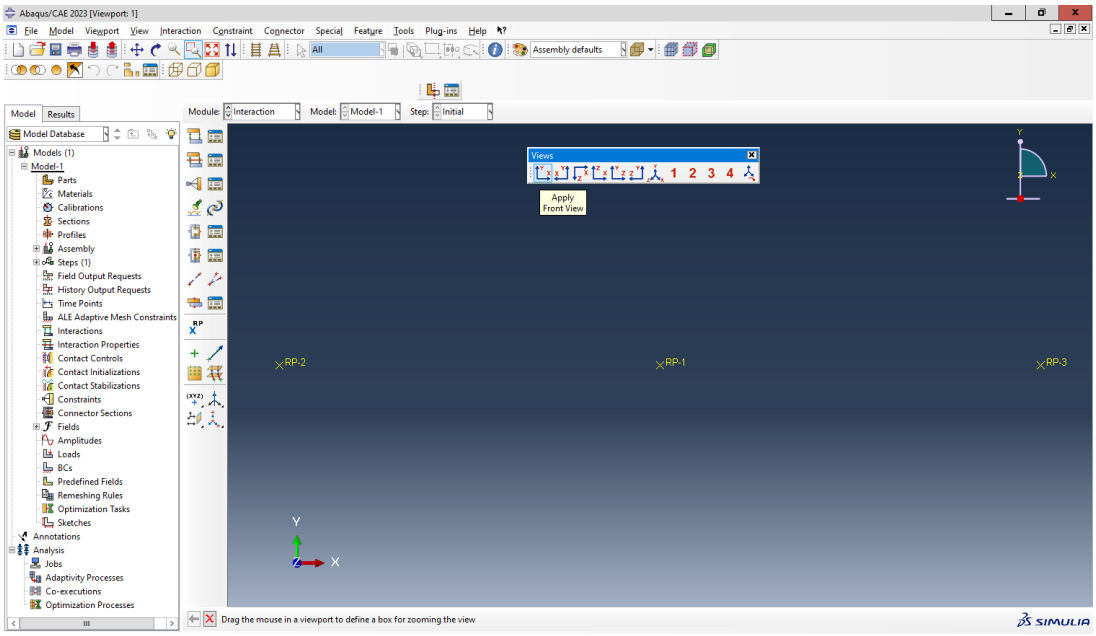


Figure 1.6: Reference points.

We proceed by creating the wire feature by clicking on the "Create Wire Feature"

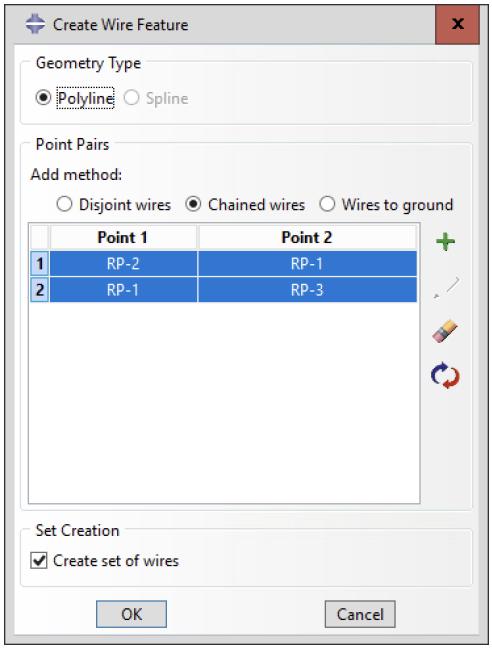


Figure 1.7: Wire features.

and the result is the following:

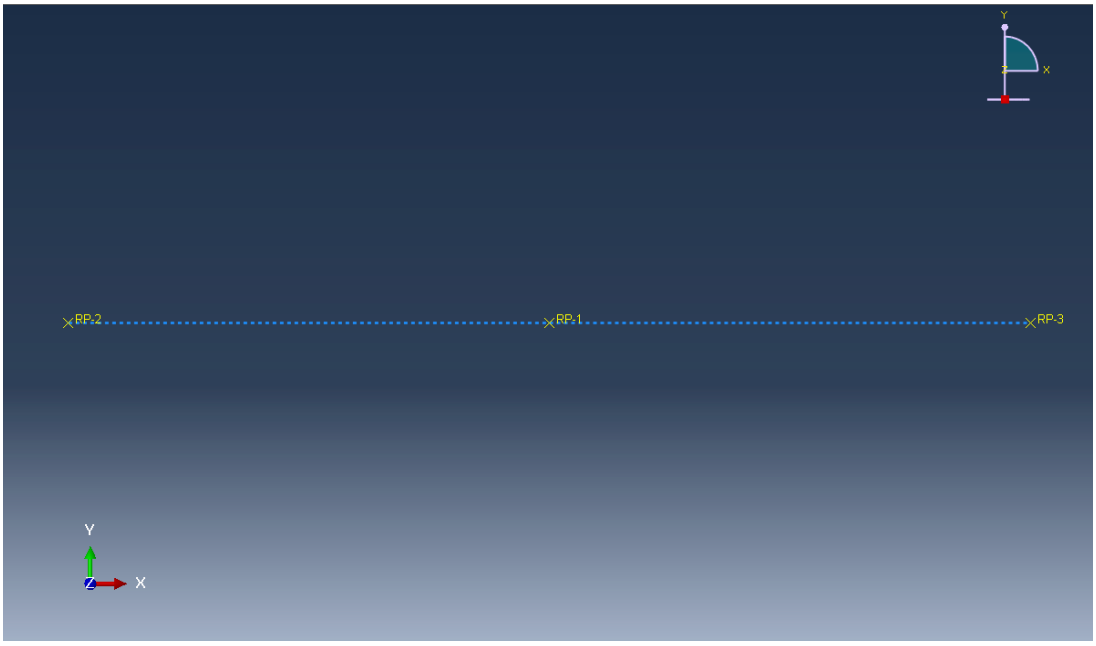


Figure 1.8: Middle result of Interaction module.

A new *section* should be defined for each springs. In the "Connector Section Manager", for both springs, we set the connection category as basic and the connection translational type as axial:

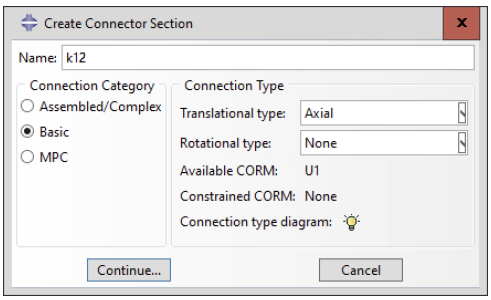
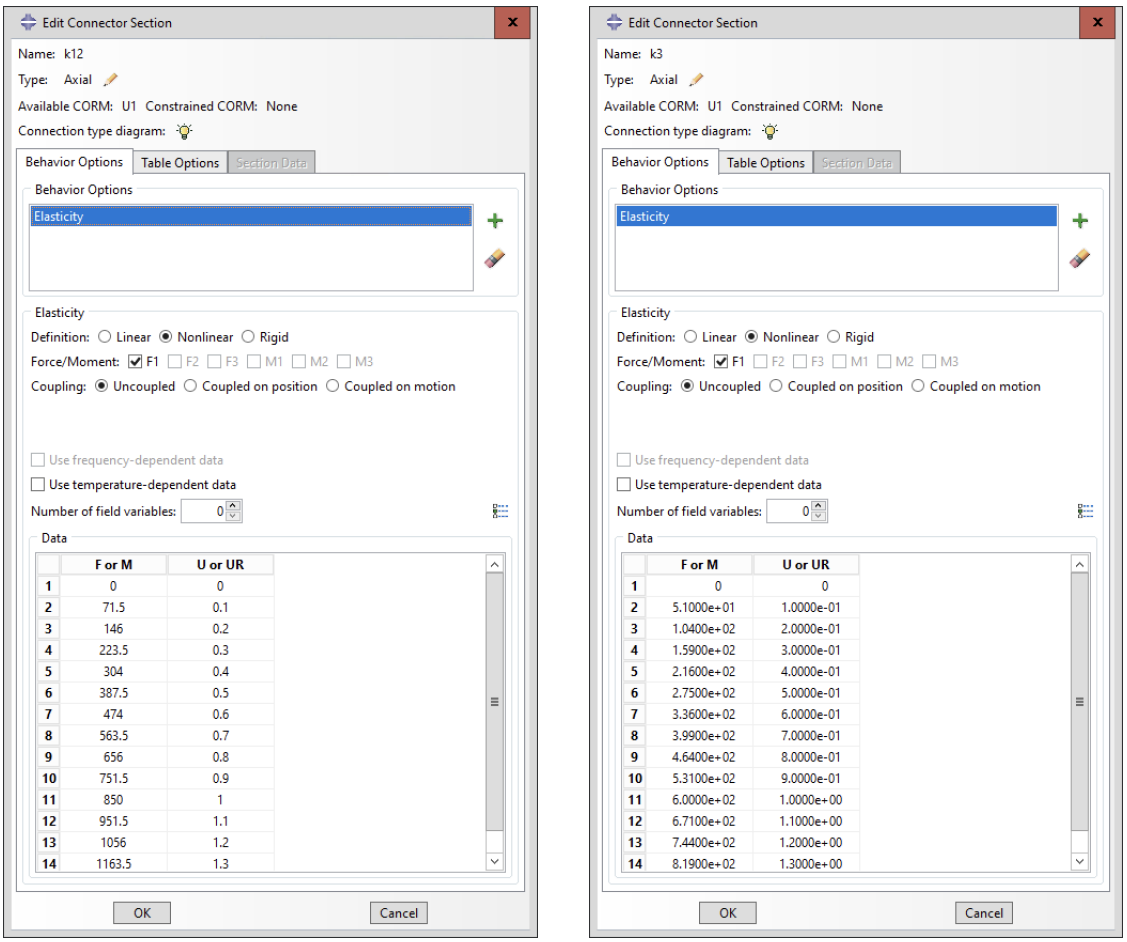


Figure 1.9: Connector Section features.

Here we manage to introduce nonlinearity by pasting the tables of values obtained with Algorithm 1.6 in MATLAB:



(a) Behavior options of Spring 1+2.

(b) Behavior options of Spring 3.

Figure 1.10: Behavior options of the springs.

Finally, we associate each spring with its respective mechanical property:

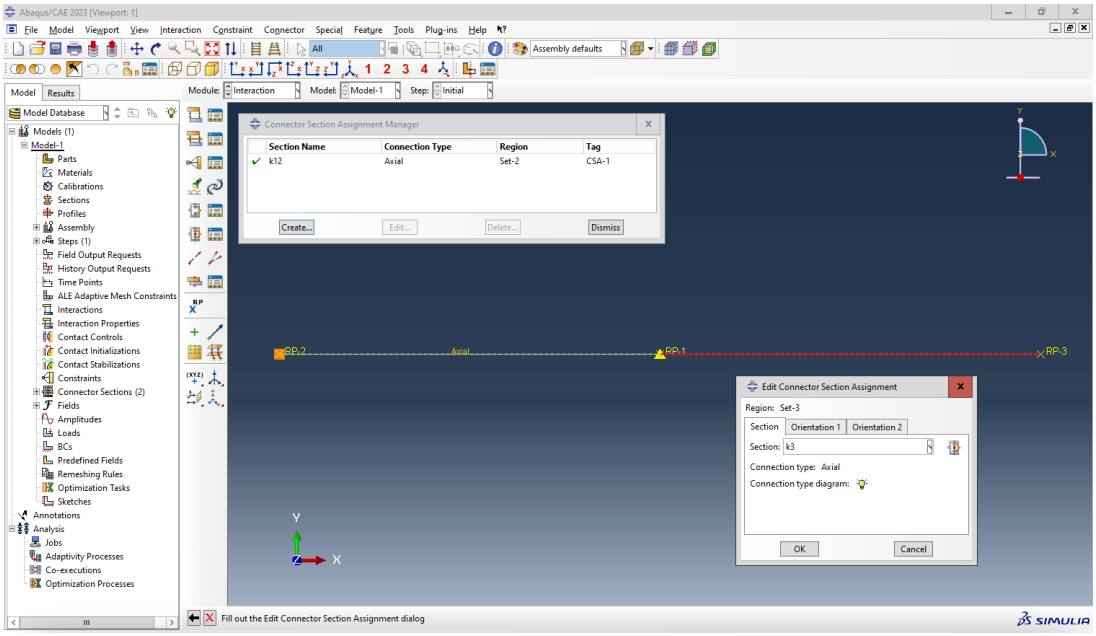


Figure 1.11: Connector Section Assignment.

The final result of this module is shown in Figure 1.12.

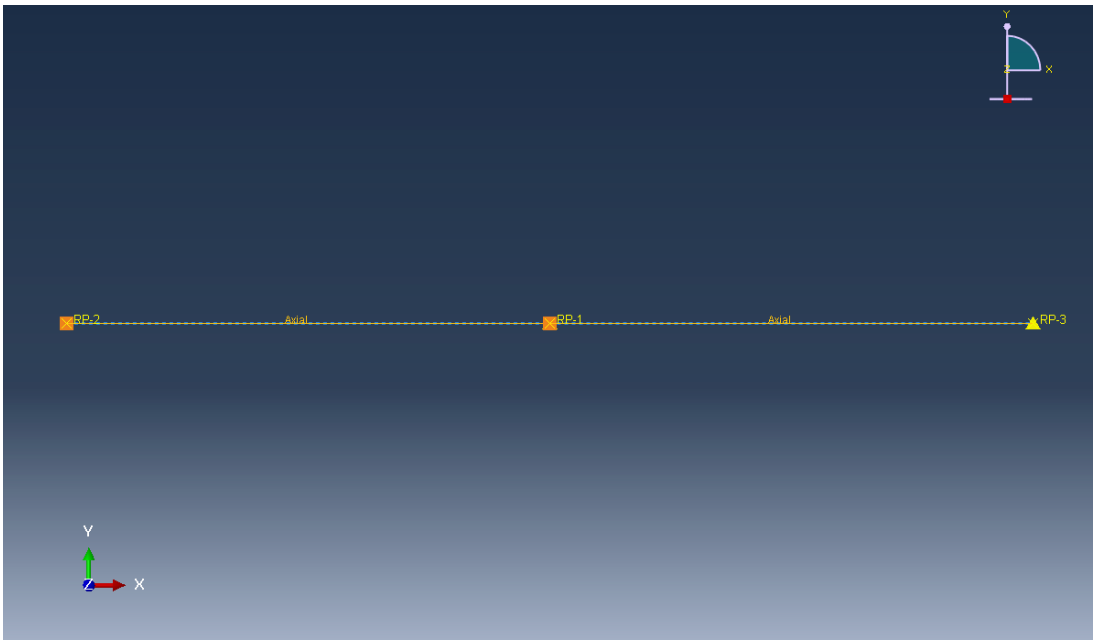


Figure 1.12: Final result of Interaction module.

1.5.2. Step module

We click on the "Step Manager" button, which looks like Figure 1.13,

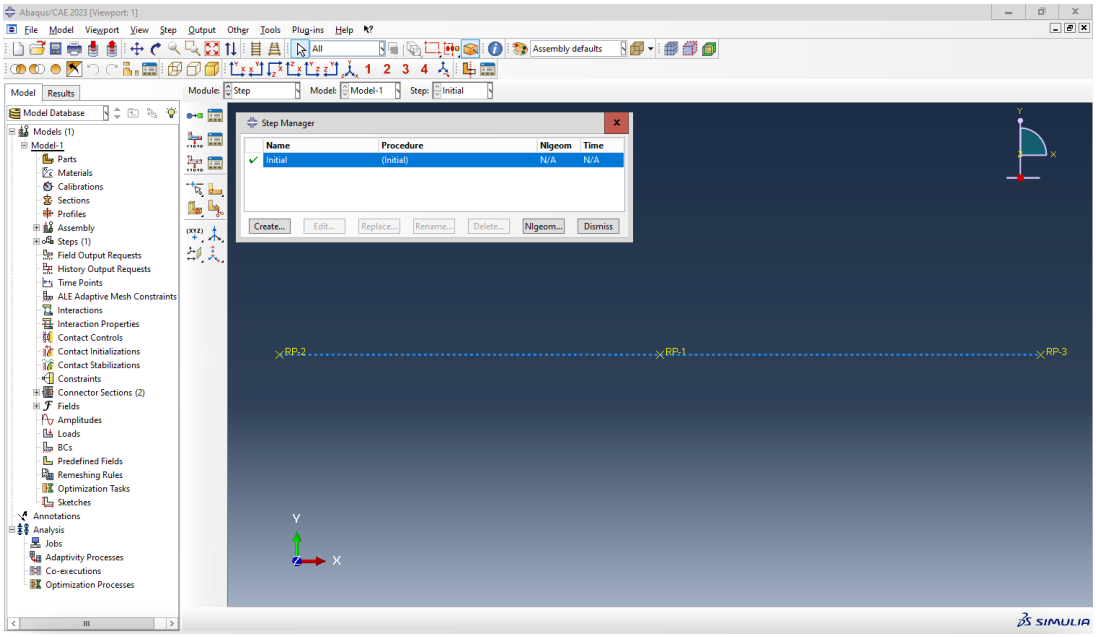


Figure 1.13: Step Manager.

and we create a new step turning on the Nlgeom option (in order to include the nonlinear effects of large displacements):

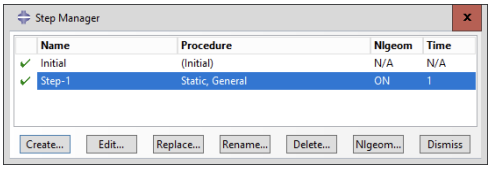


Figure 1.14: Final result of Step module.

1.5.3. Load module

In this module we first set the boundary conditions of the problem. For the node on the left we want to restrict the vertical, horizontal, and the out-of-plane displacements. Thus, the node on the left has U1, U2 and U3 restricted. Instead, the middle and the right nodes have just U2 and U3 restricted. Figure 1.15 shows the setting.

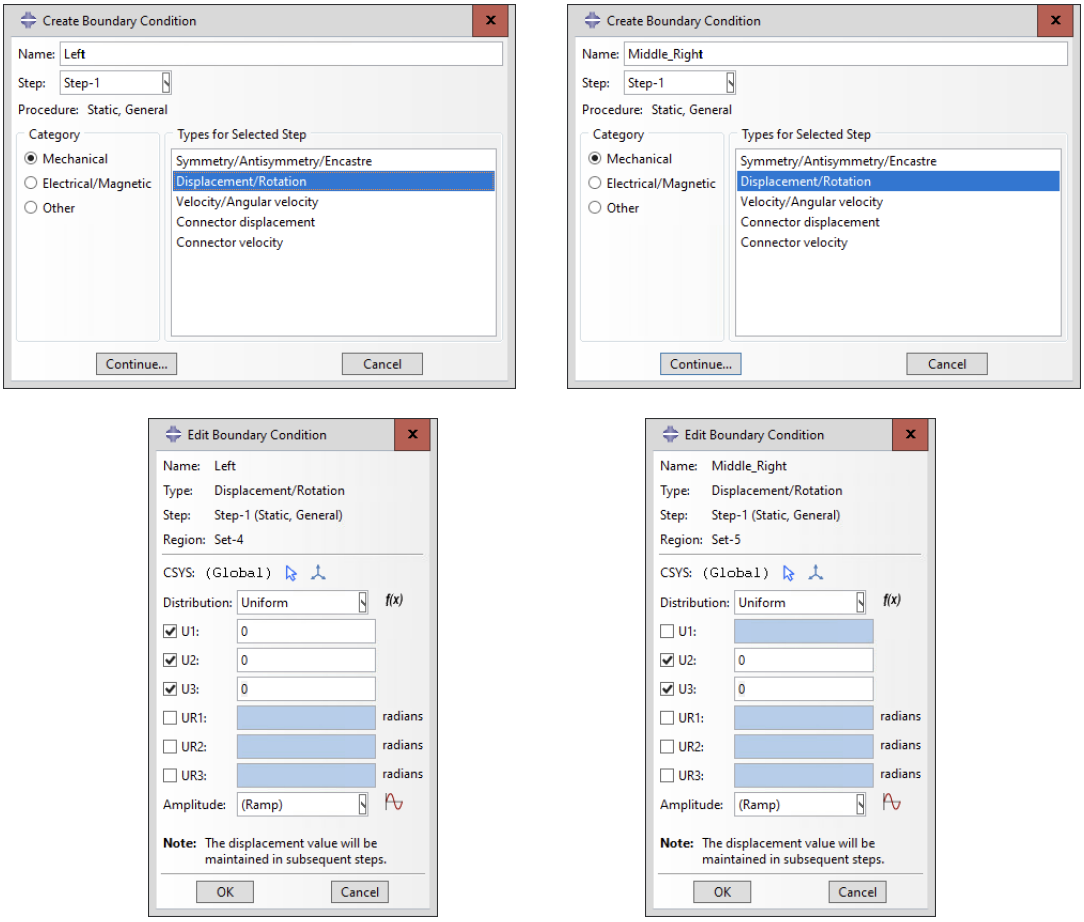


Figure 1.15: Boundary conditions of the problem.

Then we set a concentrated force of 100 N acting on the right node in the x direction with the "Create Load" symbol:

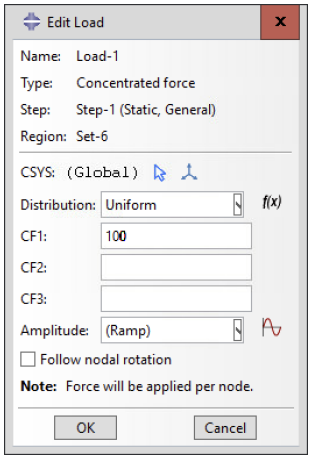


Figure 1.16: Load options.

Here there is the result:

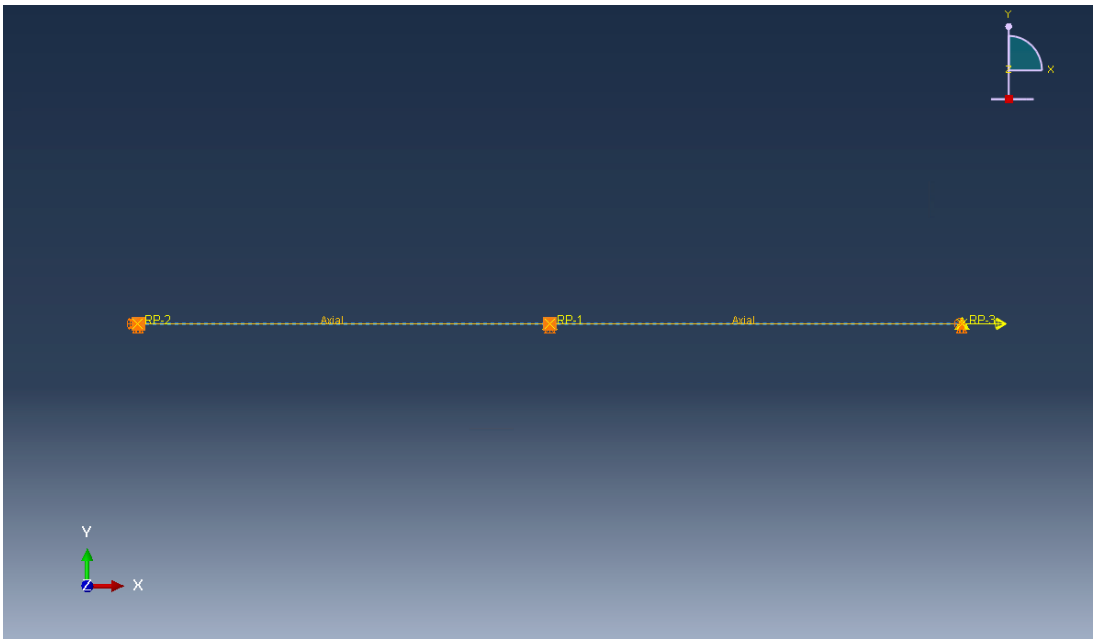


Figure 1.17: Final result of Load module.

1.5.4. Job module

We create a standard job in the "Job Manager" and we submit it:

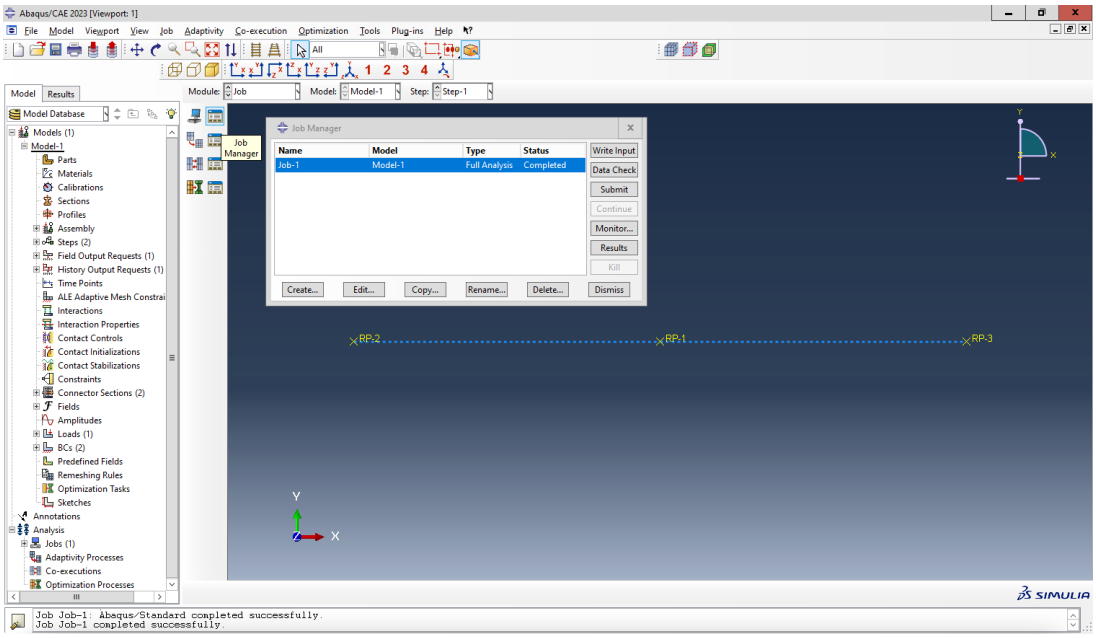


Figure 1.18: Final result of Job module.

1.5.5. Visualization module

First, we enable the connectors view in the ODB Display Options, as in Figure 1.19

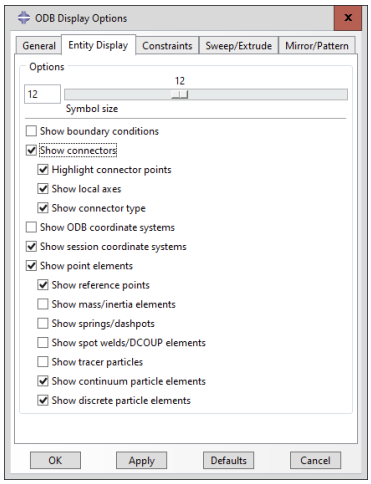


Figure 1.19: View ODB Display Options.

Here is the result we obtained:

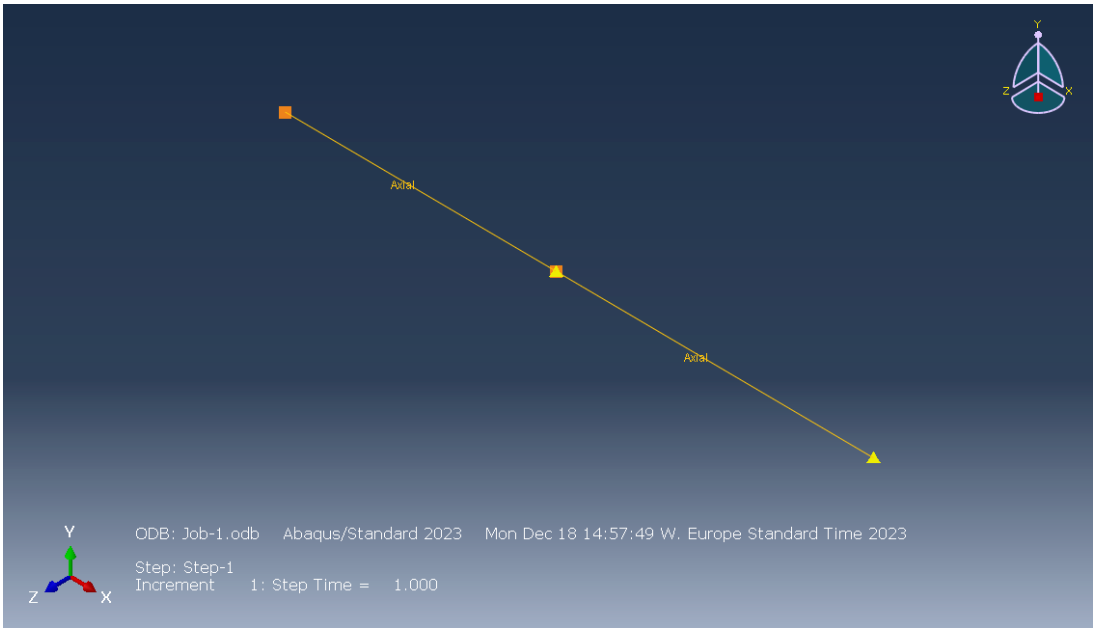


Figure 1.20: Final result.

We can explain U1 thanks to Figure 1.21.

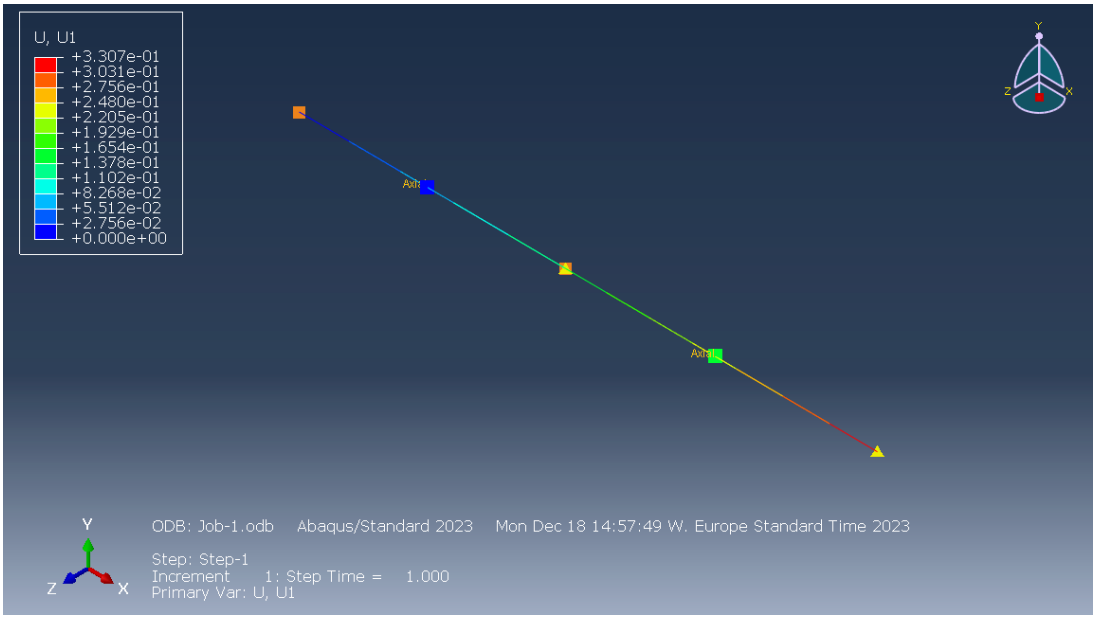


Figure 1.21: Final result in term of U1.

To be more precise, we proceed by clicking on the "Create XY Data" symbol and then we select the spatial displacement of U1 of the middle and right nodes (the ones we are interested in):

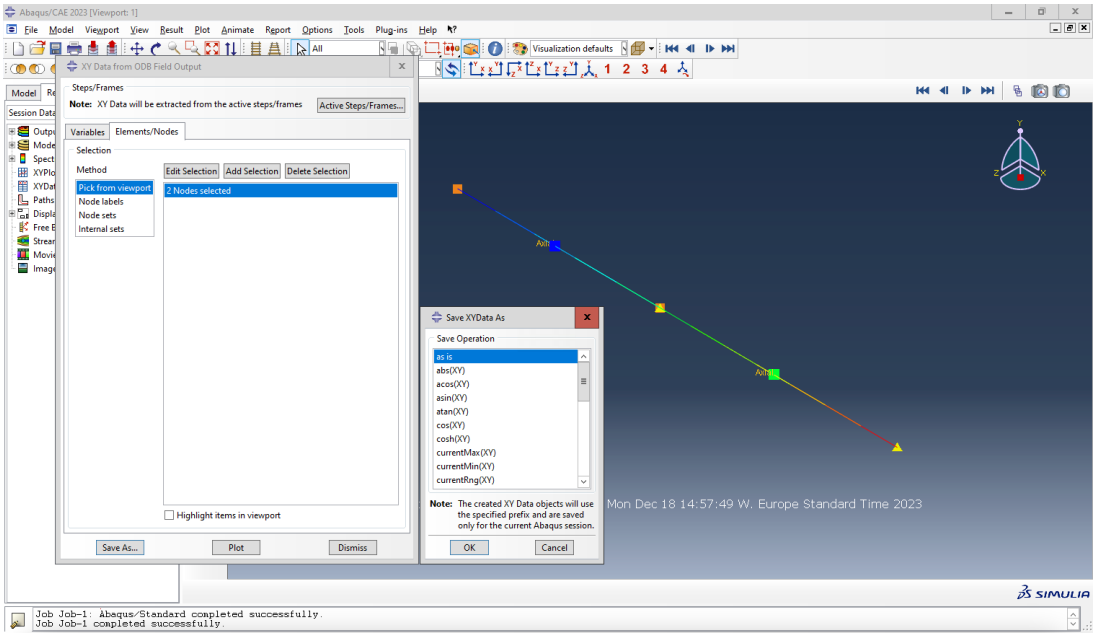


Figure 1.22: XY Data from ODB Field Output.

Finally, by going into the *edit* option of each displacement we can check the numerical result for each selected node:

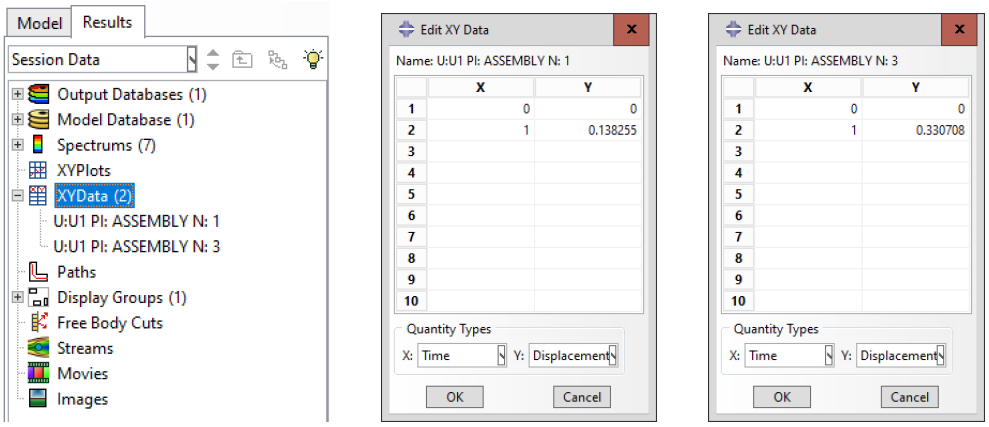


Figure 1.23: Numerical results with Abaqus.

1.6. Comparison between MATLAB and Abaqus

Now we compare the values previously obtained with MATLAB and Abaqus.

Quantity	Symbol	MATLAB	Abaqus	UoM
Horizontal middle displacement	u_1	0.13873	0.138255	mm
Horizontal right displacement	u_2	0.33132	0.330708	mm

Table 1.2: MATLAB and Abaqus results.

In conclusion, the analysis of the results obtained through simulation in MATLAB and Abaqus confirms the accuracy of the proposed model, highlighting a congruent agreement between the two software and the adopted theoretical methodology.

2 | Potential Problems with BEM

Hello

Internal Point	Name	Abaqus	MATLAB	UoM
1	NT11 N:21	38.6765	40.00000	°C
2	NT11 N:23	57.6329	57.88403	°C
3	NT11 N:25	69.0987	69.15628	°C
4	NT11 N:39	21.5550	22.11597	°C
5	NT11 N:41	39.5222	40.00000	°C
6	NT11 N:43	57.6329	57.88403	°C
7	NT11 N:57	10.5401	10.84372	°C
8	NT11 N:59	21.5550	22.11597	°C
9	NT11 N:61	38.6765	40.00000	°C

Table 2.1: Nodal temperature results of Problem 1.

simmetria.....

****
RESULTS
****

BOUNDARY NODES:

NODE	XM	YM	U	U_n
1	0.12500	0.00000	80.00000	470.36316
2	0.37500	0.00000	80.00000	116.43137
3	0.62500	0.00000	80.00000	66.83419
4	0.87500	0.00000	80.00000	18.38842
5	1.00000	0.12500	80.00000	18.38842
6	1.00000	0.37500	80.00000	66.83419
7	1.00000	0.62500	80.00000	116.43137
8	1.00000	0.87500	80.00000	470.36316
9	0.87500	1.00000	0.00000	-470.36316
10	0.62500	1.00000	0.00000	-116.43137
11	0.37500	1.00000	0.00000	-66.83419
12	0.12500	1.00000	0.00000	-18.38842
13	0.00000	0.87500	0.00000	-18.38842
14	0.00000	0.62500	0.00000	-66.83419
15	0.00000	0.37500	0.00000	-116.43137
16	0.00000	0.12500	0.00000	-470.36316

INTERNAL POINTS:

POINT	XIN	YIN	U
1	0.25000	0.25000	40.00000
2	0.50000	0.25000	57.88403
3	0.75000	0.25000	69.15628
4	0.25000	0.50000	22.11597
5	0.50000	0.50000	40.00000
6	0.75000	0.50000	57.88403
7	0.25000	0.75000	10.84372
8	0.50000	0.75000	22.11597
9	0.75000	0.75000	40.00000

Introduction

This document is intended to be both an example of the Polimi LaTeX template for Master Theses, as well as a short introduction to its use. It is not intended to be a general introduction to LaTeX itself, and the reader is assumed to be familiar with the basics of creating and compiling LaTeX documents (see [5, 7]).
The cover page of the thesis must contain all the relevant information: title of the thesis, name of the Study Programme and School, name of the author, student ID number, name of the supervisor, name(s) of the co-supervisor(s) (if any), academic year. The above information are provided by filling all the entries in the command `\puttitle{}` in the title page section of this template.
Be sure to select a title that is meaningful. It should contain important keywords to be identified by indexer. Keep the title as concise as possible and comprehensible even to people who are not experts in your field. The title has to be chosen at the end of your work so that it accurately captures the main subject of the manuscript.
Since a thesis might be a substantial document, it is convenient to break it into chapters. You can create a new chapter as done in this template by simply using the following command

```
\chapter{Title of the chapter}
```

followed by the body text.
Especially for long manuscripts, it is recommended to give each chapter its own file. In this case, you write your chapter in a separated `chapter_n.tex` file and then include it in the main file with the following command

```
\input{chapter_n.tex}
```

It is recommended to give a label to each chapter by using the command

```
\label{ch:chapter_name}%
```

where the argument is just a text string that you'll use to reference that part as follows: *Chapter 3 contains* AN INTRODUCTION TO ....
If necessary, an unnumbered chapter can be created by

```
\chapter*{Title of the unnumbered chapter}
```

3 | Chapter one

In this chapter additional useful information are reported.

3.1. Sections and subsections

Chapters are typically subdivided into sections and subsections, and, optionally, subsubsections, paragraphs and subparagraphs. All can have a title, but only sections and subsections are numbered. A new section is created by the command

```
\section{Title of the section}
```

The numbering can be turned off by using `\section*{}`.
A new subsection is created by the command

```
\subsection{Title of the subsection}
```

and, similarly, the numbering can be turned off by adding an asterisk as follows

```
\subsection*{}
```

3.2. Equations

This section gives some examples of writing mathematical equations in your thesis.

Maxwell's equations read:

$$\begin{cases} \nabla \cdot \boldsymbol{D} = \rho, & \text{(3.1a)} \\ \nabla \times \boldsymbol{E} + \dfrac{\partial \boldsymbol{B}}{\partial t} = \boldsymbol{0}, & \text{(3.1b)} \\ \nabla \cdot \boldsymbol{B} = 0, & \text{(3.1c)} \\ \nabla \times \boldsymbol{H} - \dfrac{\partial \boldsymbol{D}}{\partial t} = \boldsymbol{J}. & \text{(3.1d)} \end{cases}$$

Equation (3.1) is automatically labeled by `cleveref`, as well as Equation (3.1a) and Equation (3.1c). Thanks to the `cleveref` package, there is no need to use `\eqref`. Remember that Equations have to be numbered only if they are referenced in the text.

Equations (3.2), (3.3), (3.4), and (3.5) show again Maxwell's equations without brace:

$$\nabla \cdot \boldsymbol{D} = \rho, \tag{3.2}$$

$$\nabla \times \boldsymbol{E} + \frac{\partial \boldsymbol{B}}{\partial t} = \boldsymbol{0}, \tag{3.3}$$

$$\nabla \cdot \boldsymbol{B} = 0, \tag{3.4}$$

$$\nabla \times \boldsymbol{H} - \frac{\partial \boldsymbol{D}}{\partial t} = \boldsymbol{J}. \tag{3.5}$$

Equation (3.6) is the same as before, but with just one label:

$$\begin{cases} \nabla \cdot \boldsymbol{D} = \rho, \\ \nabla \times \boldsymbol{E} + \dfrac{\partial \boldsymbol{B}}{\partial t} = \boldsymbol{0}, \\ \nabla \cdot \boldsymbol{B} = 0, \\ \nabla \times \boldsymbol{H} - \dfrac{\partial \boldsymbol{D}}{\partial t} = \boldsymbol{J}. \end{cases} \tag{3.6}$$

3.3. Figures, Tables and Algorithms

Figures, Tables and Algorithms have to contain a Caption that describe their content, and have to be properly reffered in the text.

3.3.1. Figures

For including pictures in your text you can use `TikZ` for high-quality hand-made figures, or just include them as usual with the command

`\includegraphics[options]{filename.xxx}`

Here xxx is the correct format, e.g. `.png`, `.jpg`, `.eps`, ...

Figure 3.1: Caption of the Figure to appear in the List of Figures.

Thanks to the `\subfloat` command, a single figure, such as Figure 3.1, can contain multiple sub-figures with their own caption and label, e.g. Figure 3.2a and Figure 3.2b.

(a) One PoliMi logo. (b) Another one PoliMi logo.

Figure 3.2: This is a very long caption you don't want to appear in the List of Figures.

3.3.2. Tables

Within the environments `table` and `tabular` you can create very fancy tables as the one shown in Table 3.1.

Title of Table (optional)

	column 1	**column 2**	**column 3**
row 1	1	2	3
row 2	α	β	γ
row 3	alpha	beta	gamma

Table 3.1: Caption of the Table to appear in the List of Tables.

You can also consider to highlight selected columns or rows in order to make tables more readable. Moreover, with the use of `table*` and the option `bp` it is possible to align them at the bottom of the page. One example is presented in Table 3.2.

	column1	**column2**	**column3**	**column4**	**column5**	**column6**
row1	1	2	3	4	5	6
row2	a	b	c	d	e	f
row3	α	β	γ	δ	ϕ	ω
row4	alpha	beta	gamma	delta	phi	omega

Table 3.2: Highlighting the columns

	column1	column2	column3	column4	column5	column6
row1	1	2	3	4	5	6
row2	a	b	c	d	e	f
row3	α	β	γ	δ	ϕ	ω
row4	alpha	beta	gamma	delta	phi	omega

Table 3.3: Highlighting the rows

3.3.3. Algorithms

Pseudo-algorithms can be written in LaTeX with the `algorithm` and `algorithmic` packages. An example is shown in Algorithm 3.1.

Algorithm 3.1 Name of the Algorithm

```
Initial instructions
for for − condition do
    Some instructions
    if if − condition then
        Some other instructions
    end if
end for
while while − condition do
    Some further instructions
end while
Final instructions
```

3.4. Theorems, propositions and lists

3.4.1. Theorems

Theorems have to be formatted as:

Theorem 3.1. *Write here your theorem.*

Proof. If useful you can report here the proof.

3.4.2. Propositions

Propositions have to be formatted as:

Proposition 3.1. *Write here your proposition.*

3.4.3. Lists

How to insert itemized lists:

- first item;
- second item.

How to insert numbered lists:

1. first item;
2. second item.

3.5. Use of copyrighted material

Each student is responsible for obtaining copyright permissions, if necessary, to include published material in the thesis. This applies typically to third-party material published by someone else.

3.6. Plagiarism

You have to be sure to respect the rules on Copyright and avoid an involuntary plagiarism. It is allowed to take other persons' ideas only if the author and his original work are clearly mentioned. As stated in the Code of Ethics and Conduct, Politecnico di Milano *promotes the integrity of research, condemns manipulation and the infringement of intellectual property*, and gives opportunity to all those who carry out research activities to have an adequate training on ethical conduct and integrity while doing research. To be sure to respect the copyright rules, read the guides on Copyright legislation and citation styles available at:

`https://www.biblio.polimi.it/en/tools/courses-and-tutorials`

You can also attend the courses which are periodically organized on "Bibliographic citations and bibliography management".

3.7. Bibliography and citations

Your thesis must contain a suitable Bibliography which lists all the sources consulted on developing the work. The list of references is placed at the end of the manuscript after the chapter containing the conclusions. We suggest to use the BibTeX package and save the bibliographic references in the file `Thesis_bibliography.bib`. This is indeed a database containing all the information about the references. To cite in your manuscript, use the `\cite{}` command as follows:
Here is how you cite bibliography entries: [3], or multiple ones at once: [4, 6].
The bibliography and list of references are generated automatically by running BibTeX [1].

4 | Conclusions and future developments

A final chapter containing the main conclusions of your research/study and possible future developments of your work have to be inserted in this chapter.

Bibliography

[1] CTAN. BiBTeX documentation, 2017. URL `https://ctan.org/topic/bibtex-doc`.

[2] N.-H. Kim. *Introduction to Nonlinear Finite Element Analysis*. Springer, 2015.

[3] D. E. Knuth. Computer programming as an art. *Commun. ACM*, pages 667–673, 1974.

[4] D. E. Knuth. Two notes on notation. *Amer. Math. Monthly*, 99:403–422, 1992.

[5] S. Kottwitz. *LaTeX Cookbook*. Packt Publishing Ltd, 2015.

[6] L. Lamport. *LaTeX: A Document Preparation System*. Pearson Education India, 1994.

[7] T. Oetiker, H. Partl, I. Hyna, and E. Schlegl. The not so short introduction to latex2ε. *Electronic document available at http://www. tex. ac. uk/tex-archive/info/lshort*, 1995.

[8] A. Quarteroni, R. Sacco, and F. Saleri. *Numerical Mathematics*. Springer, 2007.

A | Appendix A

If you need to include an appendix to support the research in your thesis, you can place it at the end of the manuscript. An appendix contains supplementary material (figures, tables, data, codes, mathematical proofs, surveys, ...) which supplement the main results contained in the previous chapters.

B | Appendix B

It may be necessary to include another appendix to better organize the presentation of supplementary material.

List of Figures

List of Tables

List of Algorithm

List of Symbols

Variable	Description	SI unit
$\boldsymbol{u}$	solid displacement	m
$\boldsymbol{u}_f$	fluid displacement	m

Acknowledgements

Here you might want to acknowledge someone.