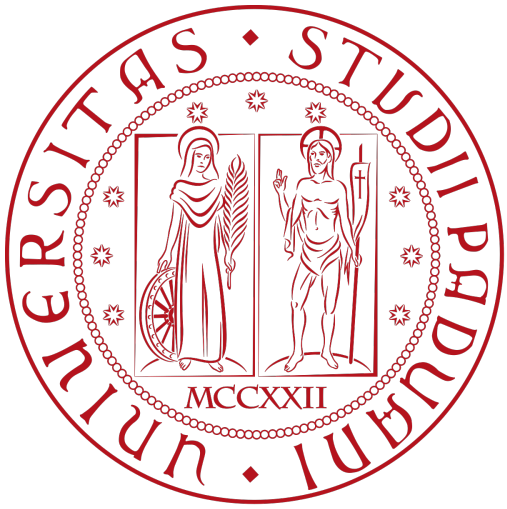


Advanced solid mechanics

Final project: Introductory FEM 2D program and Neural Networks to predict maximum displacement and maximum von Mises stress

Kenji Urazaki Junior
Student number: 2013056

Advanced solid mechanics
Università Degli Studi di Padova
February 2023

Contents

Abstract

The simulation to design and study physical structures is an important tool in many areas such as civil engineering, industrial components, naval and aerospace engineering. This work investigates a data approach in a 2D simulated structure.

The FEM formulation and structural model are briefly presented in this work in chapters 2 and 3, in a general form and the one in 2D using the 3-noded triangular form as elements. The FEM code is validated using a result from literature in section 6.1 and the convergence of displacement and stress with the increasing number of elements is discussed in section 6.3.1.

The simulations to generate data were performed in a cantilever beam structure with the base fixed (x = 0) and a vertical load applied in its tip. More than a hundred thousand simulations were performed by varying the geometry dimensions (length, height, thickness), Young modulus, and load magnitude. The results were explored and the linear relation of displacement and stresses with load magnitude was shown. The non-linear relation of displacement with geometry dimensions and Young modulus was shown as well as the independence of stress to the Young modulus.

The neural networks (NNs) used in the modeling are the feedforward with one hidden layer and its construction and training are briefly described in chapter 5. The NNs were built and selected using grid search and cross-validation with the analysis of the residuals of the best model briefly discussed. Two NNs were built: one to predict maximum displacement in the y direction, presented in section 6.4.1; the other one to predict the maximum von Mises stress, presented in section 6.4.2. The final relative error of the NN to predict displacement was 2.43%; and the one to predict von Mises was 2.33%.

The FEM simulations and NN models were run in Python as well as the construction of the result figures. The code can be accessed in [11], where the Jupyter notebook has the analysis of the simulations results and NNs training, while the main.py has the code of generation and validation of the FEM simulation.

Structural Model

{chapter:stru

The model considered to simulate the stress and strain in the rectangular form in this project is detailed and presented in this section. The geometry studied in this work was a plane of rectangular form, but the model described in this chapter was general to a 3D model, the simplifications to a 2D model are direct.

2.1 Kinematic description

{sec:kinemact

The model's mathematical treatments consider a continuous body composed of dimensionless material points. This body can assume different shapes.

The displacement u of a body is the change of position given a frame of reference of the materials points due to an external force. The strain is defined based on the displacement field, i.e. analyzing the whole body change of position it is possible to understand if given any two material points there was a change in the distance between them. The strain equation is presented in 2.1 and the formulation using a derivative operator is present in equation 2.2.

$$\varepsilon_{ij} = \frac{1}{2}\left[\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i}\right] \quad \text{where i,j = x, y, z} \tag{2.1}$$

{eq:strain}

$$\varepsilon = Lu \tag{2.2}$$

{eq:strain_de

where L is the derivative operator presented in equation 2.3.

$$L = \begin{bmatrix} \frac{\partial}{\partial x} & 0 & 0 \\ 0 & \frac{\partial}{\partial y} & 0 \\ 0 & 0 & \frac{\partial}{\partial z} \\ \frac{\partial}{\partial y} & \frac{\partial}{\partial x} & 0 \\ 0 & \frac{\partial}{\partial z} & \frac{\partial}{\partial y} \\ \frac{\partial}{\partial z} & 0 & \frac{\partial}{\partial x} \end{bmatrix} \tag{2.3}$$

{eq:der_opera

2.2 Equilibrium conditions

{sec:equilibr

Given a body subject to two types of external load: f contact forces in its boundary S and $\vec{F}$ the body forces (forces per unit volume/area) in its volume V. The total force in the body is computed by integrating the two forces and is represented in equation 2.4.

$$\int_S f dS = \vec{f}^{\,S} \quad and \quad \int_V F dV = \vec{F}^{\,V} \tag{2.4}$$

{eq:forces_in

The equilibrium equation is presented in 2.5, which considers that a body is in equilibrium when the total sum of the external forces is zero.

$$\vec{f}^{\,S} + \vec{F}^{\,V} = 0 \tag{2.5}$$

{eq:general_e

Inside the body, there is the presence of internal contact forces between the adjacent material points. These forces can be described by a stress field or stress tensor and are formalized by the Cauchy hypothesis and Theorem. In the next section, the stress is formalized in the condition of equilibrium.

2.3 Cauchy Theorem

{sec:cauchy_t

The Cauchy hypothesis on stress describes it as a surface density of force field $\boldsymbol{\sigma}(n, x, t)$ called stress field acting at x in time t on the normal n direction. This field is defined by the surface density of internal contact forces between points in an infinitesimal area dσ. Depending on the normal of the plane that is used to cut the body, the stress field is different and the stress vector follows the action-reaction law $\boldsymbol{\sigma}(-n) = -\boldsymbol{\sigma}(n)$. All definitions and equations in this section were adapted from [3].

Theorem 1 *The Dynamical balance laws of linear and angular momentum hold for a body in equilibrium if and only if exists a smooth tensor field σ called Cauchy Stress tensor such that:*

- $\boldsymbol{\sigma} = \sigma n$
- $\boldsymbol{\sigma}$ *is symmetric.* $\sigma_{i,j} = \sigma_{j,i}$
- $\boldsymbol{\nabla}\sigma + F = 0$

The Cauchy stress tensor is a tensor with three components in each direction of space, resulting in a 3 by 3 matrix. It's useful to write the statements of the theorem in expanded form. The equations 2.6 to 2.8 present the expanded form of the statements. Because of the symmetry of the stress tensor, it can be written in a vector form as in equation 2.7.

$$\boldsymbol{\sigma} = \sigma n = \begin{bmatrix} \boldsymbol{\sigma}_{nx} \\ \boldsymbol{\sigma}_{ny} \\ \boldsymbol{\sigma}_{nz} \end{bmatrix} = \begin{bmatrix} \sigma_x & \tau_{yx} & \tau_{zx} \\ \tau_{xy} & \sigma_y & \tau_{zy} \\ \tau_{xz} & \tau_{yz} & \sigma_z \end{bmatrix} \begin{bmatrix} n_x \\ n_y \\ n_z \end{bmatrix} \tag{2.6}$$ {eq:cauchy_te

$$\sigma = \begin{bmatrix} \sigma_x, & \sigma_y, & \sigma_z, & \tau_{xy}, & \tau_{zy} & \tau_{zx} \end{bmatrix} \tag{2.7}$$ {eq:cauchy_te

$$\begin{aligned} \frac{\partial \sigma_x}{\partial x} + \frac{\partial \tau_{yx}}{\partial y} + \frac{\partial \tau_{xz}}{\partial z} + F_x = 0 \\ \frac{\partial \tau_{xy}}{\partial x} + \frac{\partial \sigma_y}{\partial y} + \frac{\partial \tau_{zy}}{\partial z} + F_y = 0 \\ \frac{\partial \tau_{xz}}{\partial x} + \frac{\partial \tau_{yz}}{\partial y} + \frac{\partial \sigma_z}{\partial z} + F_z = 0 \end{aligned} \tag{2.8}$$ {eq:cauchy_eq

The strong form of the balance law for a body in equilibrium is presented in the equation 2.9. Two boundary conditions are restricting the problem, one as a prescribed displacement and the other as an external force in different parts of the boundary. Note that the boundary parts that have no boundary conditions explicitly defined are just assumed zero external force. Determining a boundary condition to the whole boundary S is needed to solve the problem.

$$\begin{cases} \nabla\sigma + F = 0 \; in \; V \\ f = \sigma n \; in \; S^f \\ u = \hat{u} \; in \; S^u \end{cases} \tag{2.9}$$ {eq:equilibri

where $\hat{u}$ is the prescribed displacement in the boundary S^u.

2.4 Principle of Virtual Work (Weak form)

{PVW_definiti

The Principle of Virtual Work (PVW) is a general procedure widely used to formulate the equilibrium equations for complex structures. It takes into account the equilibrium between forces acting in the joints and the displacement of the body. It's the integrated form of the strong equilibrium equation 2.9 and it is called weak form. The principle states that:

> "A structure is in equilibrium under a set of external loads if after imposing to the structure arbitrary (virtual) displacements compatible with the boundary conditions, the work performed by the external loads on the virtual displacements equals the work performed by the actual stresses on the strains induced by the virtual displacements". [6]

PVW is a necessary and sufficient condition for the equilibrium ([6]). The PVW for a generic body is presented in equation 2.10.

$$\int_V \sigma \delta\varepsilon dV = \int_S F\delta u dS + \int_S f\delta u dS \tag{2.10}$$ {eq:PVW}

where $\delta\varepsilon$ is the virtual strain, δu is the virtual displacement, V is the volume, F is a distributed external force, f is the external force applied in the boundary and S is the surface of the body. The left-hence side of the equation represents the internal work by the stresses on the strains and the right-hence side represents the work performed by the external loads

2.5 Constitutive equations

{sec:constitu

The constitutive equation is the formulation that connects the stress and strain in the problem. Material properties are considered in this project as linear elastic isotropic material and the small strain theory is considered, meaning the constitutive equation parameters remain constant until the end of the deformation. The hypothesis is that the deformation is not large enough to modify the behavior of the material.

The stress tensor for the material considered has the relation with the strain ϵ described by the Hooke Law:

$$\sigma = D\varepsilon \tag{2.11}$$ {eq:hooke_law

Where D is a matrix called constitutive, which is constructed of parameters related to the material.

The stress and strain tensors are symmetric so the Voigt notation can be used and the vectors are presented in 2.12. There are only 6 unknown stress and strain components because of the symmetry. The constitutive matrix parameters are reduced by the symmetry of these vectors and, because of the elasticity of the material, it's symmetric itself.

$$\begin{gathered}\sigma = \left[\sigma_x, \sigma_y, \sigma_z, \sigma_{xy}, \sigma_{zy}, \sigma_{zx}\right] \\ \varepsilon = \left[\varepsilon_x, \varepsilon_y, \varepsilon_z, \gamma_{xy}, \gamma_{zy}, \gamma_{zx}\right] \\ where\ \gamma_{i,j} = 2\varepsilon_{i,j}\end{gathered} \tag{2.12}$$ {vecs:strain_

The constitutive matrix for a isotropic linear elastic material is presented in 2.13.

$$\sigma = \begin{bmatrix} \lambda+2\mu & \lambda & \lambda & 0 & 0 & 0 \\ & \lambda+2\mu & \lambda & 0 & 0 & 0 \\ & & \lambda+2\mu & 0 & 0 & 0 \\ & & & \mu & 0 & 0 \\ & & & & \mu & 0 \\ & & & & & \mu \end{bmatrix} \varepsilon \tag{2.13}$$ {matrix:const

where:

$$\begin{gathered}\lambda = \frac{E\nu}{(1+\nu)(1-2\nu)} \\ \mu = \frac{E}{2(1+\nu)}\end{gathered} \tag{2.14}$$

where E is the Young modulus, which is the slope of the stress-strain curve measured during a uniaxial tension test; ν is the Poisson's ratio, which is the negative ratio between transverse and axial strain during a uniaxial tension test; μ is the Shear Modulus, which is the slope of the shear stress versus the shear strain.

The plane (2D) stress and strain problem constitutive parameters can be derived from 3D elasticity using the assumption that $\sigma_z = 0$ for plane stress and $\varepsilon_z = 0$ for plane strain. For both cases, the assumption $\gamma_{xz} = \gamma_{yz} = 0$ is applied. The parameters of the constitutive matrix 2.15 for an isotropic elastic material are presented in equation 2.16 and 2.17 for the plane stress and plane strain, respectively.

$$D = \begin{bmatrix} d_{11} & d_{12} & 0 \\ d_{21} & d_{22} & 0 \\ 0 & 0 & d_{33} \end{bmatrix} \tag{2.15}$$ {eq:2D_consti

$$D_{stress} = \begin{bmatrix} \frac{E}{1-v^2} & \frac{Ev}{1-v^2} & 0 \\ \frac{Ev}{1-v^2} & \frac{E}{1-v^2} & 0 \\ 0 & 0 & \frac{E}{2(1+v)} \end{bmatrix} \tag{2.16}$$ {eq:2D_stress

$$D_{strain} = \begin{bmatrix} \frac{E(1-v)}{(1+v)(1-2v)} & \frac{Ev}{(1+v)(1-2v)} & 0 \\ \frac{Ev}{(1+v)(1-2v)} & \frac{E(1-v)}{(1+v)(1-2v)} & 0 \\ 0 & 0 & \frac{E}{2(1+v)} \end{bmatrix} \tag{2.17}$$ {eq:2D_strain

2.6 Geometry and loads {geometry_loa

The geometry used in the simulations of this report is a cantilever beam with a vertical load applied to its tip. Figure 2.1 presents the generic geometry and the location of the load.

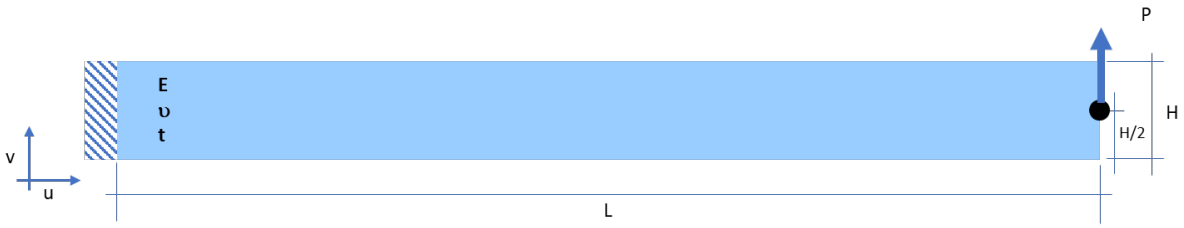


Figure 2.1: Geometry and applied load {fig:geometry

2.7 von Mises stress {von_mises_st

The failure of a material can be monitored by checking if the maximum principal stress reached a prescribed limit value (section 8.2.5, [6]). This monitoring is usually used to indicate fractures in the materials, but for most materials, the failure point is governed by the yield rule, which is expressed using stress invariants.

For 2D solids, the stress field is better represented by the two principal stresses σ_I and σ_{II}. The principal stresses are computed as the roots of the equation 2.18. The two roots for a 2D solid are presented in the equation 2.19 as expressed in [6]. The failure criteria are when the maximum stress σ_I reaches a prescribed limit. In this report, the principal stresses are computed using the nodal stresses.

$$det([\boldsymbol{\sigma}] - \lambda \mathbf{I}_2) = 0 \tag{2.18}$$ {eq:principal

where $\mathbf{I}_2$ is the identity matrix of order 2.

$$\begin{aligned} \lambda_1 = \sigma_I &= \frac{\sigma_x + \sigma_y}{2} + \frac{1}{2}\left[(\sigma_x - \sigma_y)^2 + 4\tau_{xy}^2\right]^{(1/2)} \\ \lambda_2 = \sigma_{II} &= \frac{\sigma_x + \sigma_y}{2} - \frac{1}{2}\left[(\sigma_x - \sigma_y)^2 + 4\tau_{xy}^2\right]^{(1/2)} \end{aligned} \tag{2.19}$$ {eq:principal

The von Mises equivalent stress for the plane stress problem was adapted from the article of [10], noting that the out-of-plane principal stress component is $\sigma_2 = 0$ for plane stress. The final expression used in this report is presented in equation 2.20.

$$\sigma_{Mises} = \frac{1}{\sqrt{2}}\sqrt{\sigma_I^2 + \sigma_{II}^2 + (\sigma_I - \sigma_{II})^2} \tag{2.20}$$ {eq:vonMises}

FEM formulation

{chapter: fe

Finding an analytical solution for the conceptual model described above is very difficult or impossible because of the complexities in the geometries, boundary conditions, material properties and external forces. The problem is normally solved using a numerical approximation technique called FEM. This method results in a solution in which the values of the variable of interest are defined in a discrete number of points. The higher the number of these points, the closer is the numerical solution to the exact analytical one.

The Finite Element Method (FEM) is a numerical technique to solve partial differential or integral equations. These kinds of equations are generally used to describe the behaviour of natural processes, for example, the flow of water in a pipe, so FEM is used to solve many engineering problems. In structural analysis, it is used to compute the displacements, stresses and strains of a structure under a set of external forces and arbitrary geometry.

The basic steps of the FEM analysis from the modelling to obtaining the results can be summarized in the following items extracted from [6].

- Step 1: Describe the geometry of the object, its supports and applied loads. For example, a bridge can be modelled as a 3D solid or a stiffened plate or a facet shell. Materials properties must be defined and the scope of the analysis (small or large displacements, static or dynamic analysis). This defines the equations that may be used and solved. For this project, a linear static analysis is done;
- Step 2: The discretization process is done when the geometrical structure is subdivided into non-intersecting domains called finite elements (final mesh). The displacement is approximated in each element using the values in the nodes. The nodes are points that can be in the boundary of the elements, but depending on the type of approximation they can be internal to the elements as well. The position of the node has no physical meaning, it is only a numerical feature. The higher the number of nodes, the higher the order of approximation;
- Step 3: The stiffness matrix K and the load vectors f are represented for each element;
- Step 4: The global stiffness matrix and the load vector for the structure are constructed assembling the results of each element obtained in step 3.
- Step 5: The system of equations is solved for the displacement;
- Step 6: The strain and stresses are computed using the found displacements.

After obtaining the results, an analysis has to be done to check the quality of the simulation. For example, sometimes may be necessary to increase the number of elements, or use more accurate finite elements in the mesh or the structural model need to be modified.

3.1 Finite element and discretized model

{sec:fem_disc

A finite element is a small part of the continuous body and the assembled collection of non-overlapping finite elements with a simple geometry forms the geometry that's being studied. The most common elements in 2D are triangles and quadrilaterals and in 3D tetrahedra and hexahedra.

The process of constructing these assembled finite elements to form the structure is called discretization and the final result is called a mesh. The number of elements to discretize the geometry sets a certain accuracy and degree of approximation for the mesh.

The errors in approximation of a FEM method according to [6] are three:

- modelling error: related to the conceptual and structural model to describe the real-world structure behaviour;
- discretization error: this error can be reduced by using a finer mesh, i.e. a higher number of smaller elements, or using higher order polynomials in the approximation of the element;
- numerical errors: the finite precision in the representation of numbers.

3.2 Discretization of displacement (Generic element)

{sec:discreti

The approximate displacement $\hat{u}(x)$ in the FEM method may be constructed using an interpolation of the values in the nodes for each element. This approximation is presented in equation 3.1.

$$u(x) = N_1^{(e)}(x)u_1^{(e)} + N_2^{(e)}(x)u_2^{(e)} + ... + N_n^{(e)}(x)u_n^{(e)} = \sum_{i=1}^{n} N_i^{(e)}(x)u_i^{(e)} \tag{3.1}$$ {eq:displacem

where $N_1^{(e)}...N_n^{(e)}$ are the interpolating functions defined in the domain of each element e and $u_i^{(e)}$ is the approximate value of the displacement in node i.

The interpolating function $N_i^{(e)}$ is called the shape function and it interpolates the value of the displacement of node "i" inside each element. It's clear that the value of the shape function "i" takes the value one in the node "i" and zero for all other nodes so that $u(x_i) = u_i^e$.

Using this approximate displacement for each element in the PVW, it's possible to write the equilibrium equations in terms of the displacement of the nodes in the finite element mesh.

3.3 Discretization of strain and equilibrium equations (Generic element)

{sec:discreti

Using the approximation explained in the previous section it is possible to write the virtual displacement and strain using the shape functions, which are presented in equations 3.2 and 3.3 respectively.

$$\delta u = N\delta a \tag{3.2}$$ {eq:virtual_d

$$\delta\varepsilon = L\delta u = LN\delta a = B\delta a \tag{3.3}$$ {eq:virtual_s

where a is the virtual displacement at the node, N is the matrix of the shape functions, L is the derivative operator to compute strain from displacement and B is the derivative matrix of the shape functions.

Substituting equations 3.2 and 3.3 in the PVW yields the equation 3.4 which can be simplified to 3.5.

$$\int_V \sigma B\delta a dV = \int_S FN\delta a dS + \int_S fN\delta a dS \tag{3.4}$$ {eq:PVW_eleme

$$\int_V B^T\sigma dV = \int_S N^T F dS + \int_S N^T f dS \tag{3.5}$$ {eq:PVW_eleme

The right-hence side of the equation 3.5 is the external load F^e and, the left-hence side, is the internal load F^i. If the loads are only applied on the nodes, equation 3.5 simplifies to 3.6.

$$\int_V B^T\sigma dV = \int_S F dS + \int_S f dS \tag{3.6}$$ {eq:PVW_eleme

The constitutive equation 2.11 can be expressed in terms of the shape functions and inserting this representation in the PVW equations yields the equation 3.7.

$$\int_V B^T DBadV = F^e \tag{3.7}$$ {eq:PVW_stiff

This equation 3.7 can be written in the standard form as equation 3.8.

$$Ka = F^e \tag{3.8}$$ {eq:PVW_stand

where K is a matrix called stiffness matrix defined as equation 3.9.

$$K = \int_V B^T DBdV \tag{3.9}$$ {eq:stiffness

The stiffness matrix of the structure depends on the element chosen (shape functions and geometry) and the mechanical properties of the material.

The PVW may be applied for each element in the mesh and the final global stiffness matrix can be assembled by an interactive method considering the global numbering of the nodes. The element integrals for 2D elements involve areas and for 3D elements involve volumes.

3.4 Solution for displacement

{sec:solution

The stiffness and force contributions of each finite element can be assembled in the global system of equations summing the contributions of the elements for each node.

The displacement at each node is then solved by finding the displacements which satisfy the final system of equations and the displacements initially prescribed. The representative equation of this solution is presented in equation 3.10. The strain and stresses can be computed for each element from the solved displacement using equations 3.3 and 3.11

$$a = K^{-1} f \tag{3.10}$$ {eq:system_so

$$\sigma = DBa \tag{3.11}$$ {eq:constitut

3.5 Model FEM 2D solids

{sec:model_fe

The FEM formulation for analyzing structures with 2D elasticity has practical applications for many problems and it can be split into two categories, depending on the geometric and load types:

- Plane stress problems: a prismatic structure is under plane stress when its thickness is much smaller than the other dimensions and all loads are located in the middle plane of the structure. This includes analysis of deep beams, plates, and walls under in-plane loading, for example, and the analysis is done for the middle section;
- Plane strain problems: a prismatic structure is under plane strain when its length is larger than the other dimensions and all loads are uniformly distributed along its length acting orthogonally to the longitudinal axis. This includes analysis of walls, gravity dams, pressurized pipes, and problems in geotechnical engineering (tunnels), for example, and the analysis is done for the cross-section of the longitudinal axis.

The assumptions to apply the analysis of plane stress and strain to a structure is that the transversal sections of the z-axis have the same displacement behavior and there is no displacement or it is negligible along the z-axis. Hence, the structure is analyzed by a generic transverse section of the plane x-y.

The displacement field $\mathbf{u}(x, y)$ is a 2D vector of the displacement in the x direction, $u(x, y)$, and the displacement in the y direction, $v(x, y)$, and is presented in equation 3.12.

$$\mathbf{u}(x,y) = \begin{bmatrix} u(x,y) \\ v(x,y) \end{bmatrix} \tag{3.12}$$ {eq:2D_displa

The strain related to the displacement field is presented in equation 3.13. The longitudinal strain ε_z doesn't need to be considered in the plane problem.

$$\varepsilon = \begin{bmatrix} \varepsilon_x \\ \varepsilon_y \\ \gamma_{xy} \end{bmatrix} = \begin{bmatrix} \frac{\partial u}{\partial x} \\ \frac{\partial v}{\partial y} \\ \frac{\partial u}{\partial y} + \frac{\partial v}{\partial x} \end{bmatrix} \tag{3.13}$$ {eq:2D_strain

The stress field for the 2D plane is presented in equation 3.14. As the simplification in the strain field, the shear stresses τ_{xz} and τ_{yz} are zero and the longitudinal stress σ_z is not considered but it does not contribute to the internal work.

$$\sigma = \begin{bmatrix} \sigma_x \\ \sigma_y \\ \tau_{xy} \end{bmatrix} \tag{3.14}$$ {eq:2D_stress

The equation 3.15 presents the relationship of stress-strain for the two cases. The constitutive parameters can be consulted in Chapter 2 about the Structural model.

$$\sigma = \begin{bmatrix} d_{11} & d_{12} & 0 \\ d_{21} & d_{22} & 0 \\ 0 & 0 & d_{33} \end{bmatrix} \varepsilon \tag{3.15}$$ {eq:2D_hooke}

3.5.1 Discretization for 3-noded triangular element

{sec:discreti

Figure 3.1 adapted from [6] presents a discretized domain as a mesh of 3-noded triangular elements. The element is defined by a set of nodes numbering and their coordinates x and y. The elements have a global and local numbering, which have a direct relationship that is used to assemble the final fem equations. The global numbering is presented as i, j, and k corresponding to the local numbering 1, 2, and 3 respectively.

Discretization of displacement field

The displacements u and v in the triangular element can be approximated by shape functions as in equation 3.16. The shape functions of x and y displacements are normally the same.

$$\begin{aligned} u &= N_1 u_1 + N_2 u_2 + N_3 u_3 \\ v &= N_1 v_1 + N_2 v_2 + N_3 v_3 \end{aligned} \tag{3.16}$$ {eq:2D_triang

where u_i, v_i are the horizontal and vertical displacements in the node "i", respectively, and N_i are the shape functions of node i.

In matrix form, the equation 3.16 is written as:

$$\mathbf{u} = \begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} N_1 & 0 & N_2 & 0 & N_3 & 0 \\ 0 & N_1 & 0 & N_2 & 0 & N_3 \end{bmatrix} \begin{bmatrix} u_1 \\ v_1 \\ u_2 \\ v_2 \\ u_3 \\ v_3 \end{bmatrix} \tag{3.17}$$ {eq:2D_triang

It can be expressed in compact form as 3.18 where $\mathbf{N}$ is expressed in equation 3.19; and $\mathbf{a}^{(e)}$, in equation 3.20

$$\mathbf{u} = \mathbf{N}\mathbf{a}^{(e)} \tag{3.18}$$ {eq:displacem

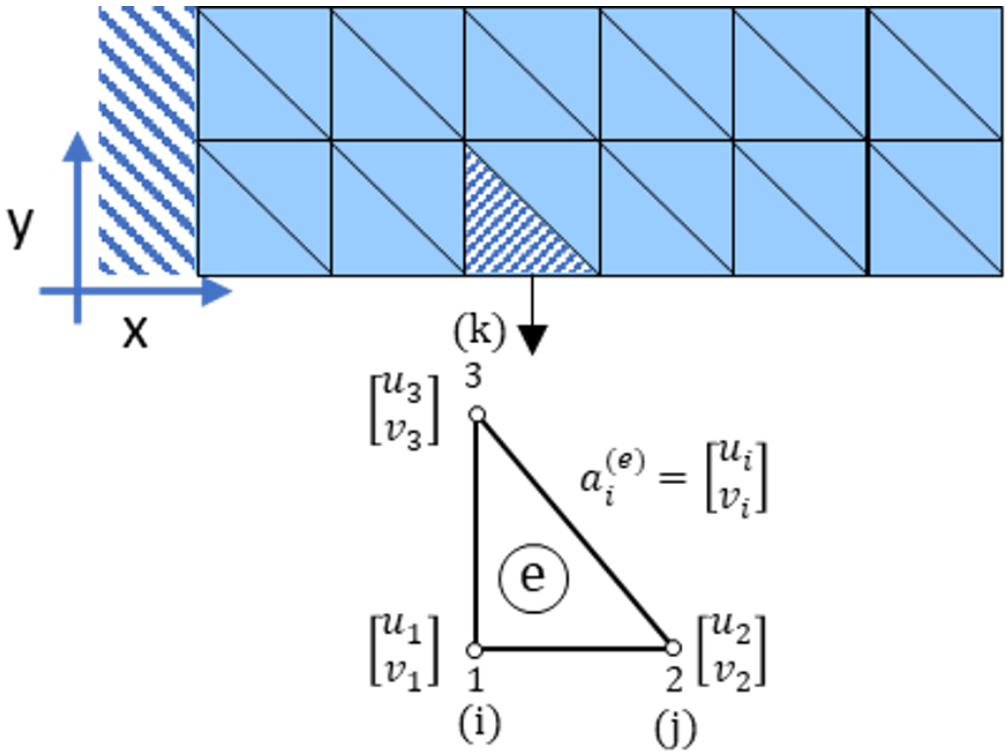


Figure 3.1: Discretized structure with nodal numbering and displacements identified for triangular elements {fig:discreti

$$\mathbf{N} = [\mathbf{N}_1, \mathbf{N}_2, \mathbf{N}_3] \quad with \quad \mathbf{N}_i = \begin{bmatrix} N_i & 0 \\ 0 & N_i \end{bmatrix} \tag{3.19}$$ {eq:shape_fun

Where $\mathbf{N}$ and $\mathbf{N}_i$ are the shape function matrices of the element and the ith node, respectively.

$$\mathbf{a}^{(e)} = \begin{bmatrix} \mathbf{a}_1^{(e)} \\ \mathbf{a}_2^{(e)} \\ \mathbf{a}_3^{(e)} \end{bmatrix} \quad with \quad \mathbf{a}_i^{(e)} = \begin{bmatrix} u_i \\ v_i \end{bmatrix} \tag{3.20}$$ {eq:nodal_dis

Where $\mathbf{a}^{(e)}$ and $\mathbf{a}_i^{(e)}$ are the nodal displacement vector of the element and ith node, respectively.

Discretization of strain

Using the discretized displacements presented in equation 3.16 and the definition of strain from displacement presented in equation 2.1, the discretized equations from strain can are expressed as equation 3.21.

$$\begin{aligned} \varepsilon_x &= \frac{\partial u}{\partial x} = \frac{\partial N_1}{\partial x} u_1 + \frac{\partial N_2}{\partial x} u_2 + \frac{\partial N_3}{\partial x} u_3 \\ \varepsilon_y &= \frac{\partial v}{\partial y} = \frac{\partial N_1}{\partial y} v_1 + \frac{\partial N_2}{\partial y} v_2 + \frac{\partial N_3}{\partial y} v_3 \\ \gamma_{xy} &= \frac{\partial u}{\partial y} + \frac{\partial v}{\partial x} = \frac{\partial N_1}{\partial y} u_1 + \frac{\partial N_1}{\partial x} v_1 + \frac{\partial N_2}{\partial y} u_2 + \frac{\partial N_2}{\partial x} v_2 + \frac{\partial N_3}{\partial y} u_3 + \frac{\partial N_3}{\partial x} v_3 \end{aligned} \tag{3.21}$$ {eq:discretiz

In matrix form, the strain set of equations can be expressed as equation 3.22 and in the compact form, it is presented in 3.23. The $\mathbf{B}$ is the element strain matrix and for the ith node is expressed as equation 3.24.

$$\boldsymbol{\varepsilon} = \begin{bmatrix} \frac{\partial u}{\partial x} \\ \frac{\partial v}{\partial y} \\ \frac{\partial u}{\partial y} + \frac{\partial v}{\partial x} \end{bmatrix} = \begin{bmatrix} \frac{\partial N_1}{\partial x} & 0 & \frac{\partial N_2}{\partial x} & 0 & \frac{\partial N_3}{\partial x} & \\ 0 & \frac{\partial N_1}{\partial y} & 0 & \frac{\partial N_2}{\partial y} & 0 & \frac{\partial N_3}{\partial y} \\ \frac{\partial N_1}{\partial y} & \frac{\partial N_1}{\partial x} & \frac{\partial N_2}{\partial y} & \frac{\partial N_2}{\partial x} & \frac{\partial N_3}{\partial y} & \frac{\partial N_3}{\partial x} \end{bmatrix} \begin{bmatrix} u_1 \\ v_1 \\ u_2 \\ v_2 \\ u_3 \\ v_3 \end{bmatrix} \tag{3.22}$$ {eq:discretiz

$$\begin{aligned} \boldsymbol{\varepsilon} &= \mathbf{B}\mathbf{a}^{(e)} \\ \text{where} \quad \mathbf{B} &= [\mathbf{B}_1, \mathbf{B}_2, \mathbf{B}_3] \end{aligned} \tag{3.23}$$ {eq:discretiz

$$\mathbf{B} = \begin{bmatrix} \frac{\partial N_i}{\partial x} & 0 \\ 0 & \frac{\partial N_i}{\partial y} \\ \frac{\partial N_i}{\partial y} & \frac{\partial N_i}{\partial x} \end{bmatrix} \tag{3.24}$$ {eq:strain_el

The element strain matrix for the 3-noded triangle, linear shape functions, is presented in equation 3.25.

$$\mathbf{B} = \frac{1}{2A(e)} \begin{bmatrix} b_1 & 0 & b_2 & 0 & b_3 & 0 \\ 0 & c_1 & 0 & c_2 & 0 & c_3 \\ c_1 & b_1 & c_2 & b_2 & c_3 & b_3 \end{bmatrix} \tag{3.25}$$ {eq:strain_el

Shape functions for the 3-noded triangular element

The shape functions for the 3-noded triangular element can be found from the definition of the linear displacement field presented in equation 3.26.

$$u = \alpha_1 + \alpha_2 x + \alpha_3 y v = \alpha_4 + \alpha_5 x + \alpha_6 y \tag{3.26}$$ {eq:linear_di

Assuming the same interpolation (same shape functions) for u and v and using the definition in equation 3.26, the parameters can be derived using one direction only. The horizontal nodal displacements can be expressed as equations 3.27.

$$u_1 = \alpha_1 + \alpha_2 x_1 + \alpha_3 y_1 u_2 = \alpha_1 + \alpha_2 x_2 + \alpha_3 y_2 u_3 = \alpha_1 + \alpha_2 x_3 + \alpha_3 y_3 \tag{3.27}$$ {eq:nodal_dis

Solving the system for the parameters α_i and inserting them into equation 3.26 yield the equation 3.28.

$$u = \frac{1}{2A^{(e)}} \left[(a_1 + b_1 x + c_1 y) u_1 + (a_2 + b_2 x + c_2 y) u_2 + (a_3 + b_3 x + c_3 y) u_3 \right] \tag{3.28}$$ {eq:horizonta

Where $A^{(e)}$ is the element area and

$$a_i = x_j y_k - x_k y_j \ , \ b_i = y_j - y_k \ , \ c_i = x_k - x_j \quad where \quad i, j, k = 1, 2, 3 \tag{3.29}$$

The parameters are obtained by a cyclic permutation of the i, j, and k indexes.

Finally, from the approximation presented in equation 3.1 and the result in equation 3.28, the linear shape function expression is deduced as equation 3.30.

$$N_i = \frac{1}{2A^{(e)}} (a_i + b_i x + c_i y) \quad , \ i = 1, 2, 3 \tag{3.30}$$ {eq:shape_fun

It can be checked that the shape function Ni takes the value one at node "i" and zero at the other two nodes. The format of the shape function is presented in figure 3.2.

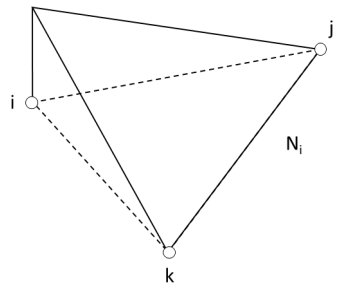


Figure 3.2: Shape function "i" for 3-noded triangular element {fig:shape_fu

Discretization of the stress field

The stress field discretized equation is derived from the equation 2.11 substituting the discretized strain into the equation. The expression for the discretized stress field within the element is presented in 3.31.

$$\boldsymbol{\sigma} = \mathbf{DBa}^{(e)} \tag{3.31}$$ {eq:discretiz

3.5.2 Discretized equilibrium equations - 2D PVW

{sec:discreti

The expression for the PVW for the 2D elasticity problem is presented in equation 3.32 extracted from [6].

$$\iint_A (\delta\varepsilon_x\sigma_x + \delta\varepsilon_y\sigma_y + \delta\gamma_{xy}\tau_{xy})tdA = \iint_A (\delta u b_x + \delta v b_y)tdA + \oint_l (\delta u t_x + \delta v t_y)tds + \sum_i (\delta u_i P_{x_i} + \delta v_i P_{y_i}) \tag{3.32}$$ {eq:PVW_2D}

where δu and δv are the virtual displacements and δu_i and δv_i are the virtual displacements at point i. The terms in the right-hence side are the body forces (forces per unit area) b_x, b_y; surface tractions t_x, t_y; and external point loads P_{x_i}, P_{y_i} and the terms in the left-hence side are the stresses σ_x, σ_y, τ_{xy} and strains $\delta\varepsilon_x$, $\delta\varepsilon_y$, $\delta\gamma_{xy}$. A is the area and l is the boundary of the transverse section and t is the solid thickness. The value of t in the plane stress problem is the actual thickness of the solid and for the plane strain problem, it is equal to one.

Usually in the FEM, the forces acting in an element are defined in the nodes, therefore the nodal point loads F_{x_i} and F_{y_i} are defined and the PVW for the 3-noded triangular element is presented in 3.33 in a compact form. The point loads F_{x_i} and F_{y_i} balances the element in the horizontal and vertical directions, respectively.

$$\iint_{A^{(e)}} \delta\boldsymbol{\varepsilon}^T\boldsymbol{\sigma}tdA = \iint_{A^{(e)}} \delta\mathbf{u}^T\mathbf{b}tdA + \oint_{l^{(e)}} \delta\mathbf{u}^T\mathbf{t}tds + \sum_{i=1}^{3} (\delta u_i F_{x_i} + \delta v_i F_{y_i}) \tag{3.33}$$ {eq:PVW_nodal

The virtual work performed by the nodal point loads is obtained by rearranging the equation 3.33 and is expressed in a compact form in equation 3.34.

$$\iint_{A^{(e)}} \delta\boldsymbol{\varepsilon}^T\boldsymbol{\sigma}tdA - \iint_{A^{(e)}} \delta\mathbf{u}^T\mathbf{b}tdA - \oint_{l^{(e)}} \delta\mathbf{u}^T\mathbf{t}tds = [\delta\mathbf{a}^{(e)}]^T\mathbf{q}^{(e)} \tag{3.34}$$ {eq:PVW_solve

where $\delta\mathbf{a}^{(e)}$ are the nodal virtual displacements of the element and $\mathbf{q}^{(e)}$ are the equilibrating nodal forces acting in the element.

The virtual displacements, strain and stress in the terms of the nodal values are discretized as equations 3.18, 3.23 and 3.31, respectively. Substituting the discretized virtual equations in the PVW yields equation 3.35.

$$\left[\iint_{A^{(e)}} \mathbf{B}^T\mathbf{DB}tdA\right]\mathbf{a}^{(e)} - \iint_{A^{(e)}} \mathbf{N}^T\mathbf{b}tdA - \oint_{l^{(e)}} \mathbf{N}^T\mathbf{t}tds = \mathbf{q}^{(e)} \tag{3.35}$$ {eq:discretiz

Where the first integral represents the element deformation forces; the second integral, the body forces; and the third integral, the surface tractions. The discretized element equilibrium equation can be expressed in compact form as equation 3.36. The **K** matrix is defined as the stiffness matrix of the element e and its expression is presented in 3.37.

$$\mathbf{K}^{(e)}\mathbf{a}^{(e)} - \mathbf{f}^{(e)} = \mathbf{q}^{(e)} \tag{3.36}$$ {eq:PVW_discr

$$\mathbf{K}^{(e)} = \iint_{A^{(e)}} \mathbf{B}^T\mathbf{D}\mathbf{B}t dA \tag{3.37}$$ {eq:stiffness

The global equilibrium equations for the mesh are obtained by setting that the nodes are in equilibrium, i.e. the external points loads are balanced by the sum of all equilibrating nodal forces. The summation of equilibrium for all nodes is presented in equation 3.38.

$$\sum_e q_i^{(e)} = p_j \quad , \quad j = 1, N \tag{3.38}$$ {eq:global_eq

where $p_j = [P_{x_j}, P_{y_j}]^T$ is the point loads in the global node j and the sum of q_i refers to the nodal forces of all elements sharing the node global numbering j. The nodal point loads q_i can be expressed using equation 3.35 and the final global equilibrium equation is presented in matrix form in equation 3.39.

$$\mathbf{K}\mathbf{a} = \mathbf{f} \tag{3.39}$$ {eq:global_eq

where **K** and **f** are the global stiffness matrix and the global equivalent nodal force vector for the entire mesh.

The global **K** and **f** can be assembled from the element stiffness matrix and force vector in a repetitive procedure. A schematic of the assembly procedure is presented in figure 3.3 adapted from [6].

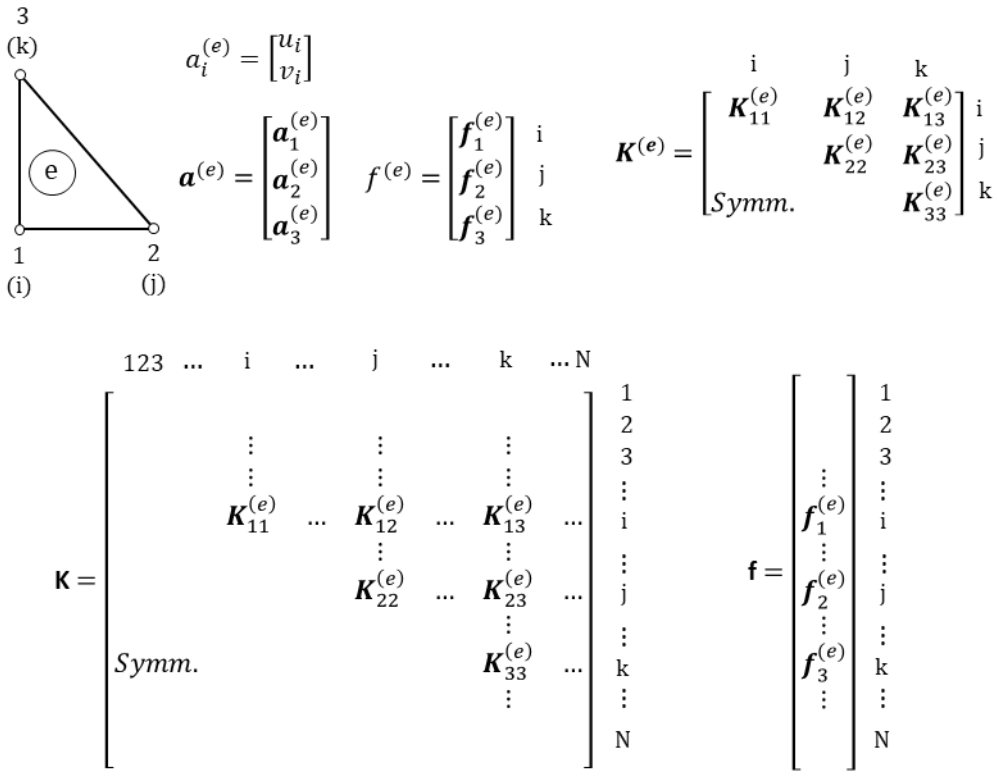


Figure 3.3: Assembly of stiffness matrix and equivalent nodal forces {fig:assembly

3.5.3 Stiffness matrix and equivalent nodal force vectors - 3-noded triangular element

Element stiffness matrix

The element stiffness matrix for the 3-noded triangular element can be written substituting the equations 3.25 and 2.15 in the equation 3.37. The final equation is presented in 3.40 as the i,j element in the matrix. Note that $K_{ij}^{(e)}$ is always symmetrical as $d_{12} = d_{21}$.

$$\mathbf{K}_{ij}^{(e)} = \iint_{A^{(e)}} \frac{1}{2A^{(e)}} \begin{bmatrix} b_i & 0 & c_i \\ 0 & c_i & b_i \end{bmatrix} \begin{bmatrix} d_{11} & d_{12} & 0 \\ d_{21} & d_{22} & 0 \\ 0 & 0 & d_{33} \end{bmatrix} \frac{1}{2A^{(e)}} \begin{bmatrix} b_j & 0 \\ 0 & c_j \\ c_j & b_j \end{bmatrix} tdA \tag{3.40}$$ {eq:stiffness

Equivalent nodal forces

The equivalent nodal forces for body forces and surface tractions can be easily derived from the equation 3.35.

Body forces are computed in the discretized equilibrium equations as the equation 3.41 and, in terms of nodal contributions, the resulting equation is 3.42. If the forces are uniformly distributed over the element, the nodal contribution can be computed as 3.43. An example of a body force is the self-weight due to gravity, which acts only in one direction, supposes y. In this scenario, $b_x = 0$ and $b_y = -\rho g$, where ρ is the material density and g is the gravity acceleration.

$$\mathbf{f}_b^{(e)} = \iint_{A^{(e)}} \mathbf{N}^T \mathbf{b} t dA = \iint_{A^{(e)}} \begin{bmatrix} \mathbf{N}_1^T \mathbf{b} \\ \mathbf{N}_2^T \mathbf{b} \\ \mathbf{N}_3^T \mathbf{b} \end{bmatrix} tdA \tag{3.41}$$ {eq:body_forc

$$\mathbf{f}_{b_i}^{(e)} = \iint_{A^{(e)}} \mathbf{N}_i^T \mathbf{b} t dA = \iint_{A^{(e)}} \begin{bmatrix} N_i b_x \\ N_i b_y \end{bmatrix} tdA \tag{3.42}$$ {eq:body_forc

$$\mathbf{f}_{b_i}^{(e)} = \frac{(At)^{(e)}}{3} \begin{bmatrix} b_x \\ b_y \end{bmatrix} \tag{3.43}$$ {eq:body_forc

Surface tractions are computed in the discretized equilibrium equations as the equation 3.44, only nodes "i" belonging to a loaded external boundary are taken to account. The vector for a loaded element side of nodes 1-2 for a uniformly distributed traction $[t_x, t_y]$ is presented in equation 3.45.

$$\mathbf{f}_{t_i}^{(e)} = \oint_{l^{(e)}} \mathbf{N}_i^T \mathbf{t} t ds = \oint_{l^{(e)}} \begin{bmatrix} N_i t_x \\ N_i t_y \end{bmatrix} tds \tag{3.44}$$ {eq:traction_

$$f_t^{(e)} = \frac{(l_{12} t)^{(e)}}{2} [t_x, t_y, t_x, t_y, 0, 0]^T \tag{3.45}$$ {eq:traction_

where $l_{12}^{(e)}$ is the side length.

FEM code

{chapter:fem_

The implementation of the FEM was done using Python with supporting material from [7] and chapter 11 [6], which developed the code in Matlab. The modules that needed to be developed were the input of information, which included the geometry, the discretized mesh and forces applied; the building of stiffness matrix and equivalent forces; and the solution and presentations of results.

4.1 Input data

{sec:input_da

The first step in the program was to read data about the problem: nodal coordinates, element node identification, boundary conditions, material properties, and loadings. The nodal coordinates and element node identification were generated in the preprocessing of the geometry.

An isotropic linear elastic material was assumed for the whole geometry, simplifying the problem and code. The properties of the material passed were the density, Young modulus, and Poisson's ratio. A flag indicating which kind of problem was being treated was created as "$flag\_planeStressorStrain$", for which a value 1 indicates a plane stress problem; and 0, is a plane strain problem. The unity thickness for the plane strain problem was assigned to the *thick* variable in the program.

The discretization was defined by 2 arrays. The number of elements and nodes are named Nelem, and Nodes, respectively and the shape of the arrays is indicated below between brackets. The arrays were:

- *triang*: (Nelem, 3) array, each line contained the global numbering nodal connectivity for each element.
- *coord*: (Nodes, 2) array, each line contained each node's global coordinates.

Three arrays defined the boundary conditions. The number of fixed nodes, loads, and side loads are named Nfixed, Nload, and NSload, respectively and the shape of the arrays is indicated below between brackets. The arrays were:

- $fixnodes$: (Nrestriction, 3) array, identified the restrictions in node displacement. Each line contained the global node identification that's being prescribed in the first column, the second column identified if the prescription was in the x (1) or y (2) direction and the last column was the prescribed value;
- *pointload*: (Nloads, 3) array, identified the loads applied in the geometry. Each line contained the global node identification that the load acted in the first columns, the second columns identified if the load was applied in x (1) or y (2) direction, and the third and last column identified the magnitude of it;
- *sideload*: (NSloads, 4) array, identified the loads uniformly applied in the geometry sides. Each line contained the two global node identification of the side that the load acted in the first and second columns. The third and fourth columns were the magnitude of the load (per unit length) acting in the x and y direction, respectively.

4.2 Stiffness matrix and equivalent nodal force

{sec:stiffnes

The next steps to set the problem were the definition of the element and global stiffness matrix and the global equivalent nodal forces. The geometry material was homogeneous, so the constitutive matrix was the same for all elements and constant.

The definition of the constitutive matrix, *dmat*, was dependent on the type of problem that was being solved (plane stress or plane strain) and the material properties. The final matrix for each case is presented in equations 2.16 and 2.17.

The B strain matrix for each element was built as the equation 3.25 and the global stiffness matrix was built from the element's stiffness matrix computed as 3.40 using the procedures described in 3.3.

The self-weight nodal forces were built as the equations 3.43 from Onate [6]. The prescribed point loads were applied directly in the RHS of the equations and the forced vector of the prescribed displacements was built using the equation 4.1. The fixed equations index was stored so only the unknown nodal displacements were solved for the system.

$$\mathbf{f}_{final} = \mathbf{f} - \mathbf{K}\mathbf{u}_{prescribed} \tag{4.1}$$ {eq:prescribe

where $\mathbf{u}$ contained only the prescribed displacement values.

Finally, the solution of the linear system was done using a solver for the unknown displacements and the reactions in the prescribed nodes were computed as equation 4.2.

$$\mathbf{r} = \mathbf{K}\mathbf{a} - \mathbf{f} \tag{4.2}$$ {eq:reactions

The element stresses are computed using equation 3.31 with the nodal displacements solution found and the already matrix B computed in the program for each element. The stresses at the nodes were computed by nodal averaging as the method described in section 11.10.2 in [6].

4.3 Discretization of geometry

{discretizati

The discretization of the geometry was done based on Delaunay triangulation and it was adapted from the code shared by [8]. The algorithm generates uniform triangles for a rectangular geometry.

The boundary conditions generated were the prescription of zero displacements in the x and y directions to the x equal 0 coordinates. The imposed load in the geometry was set in the middle of the tip of the geometry in the y direction. Sideloads setting was not implemented in this project, as it was not needed.

4.4 Solution of the global system

{sec:solution

The solution of the system was done using the library scipy, which enabled the pre-factorization of the stiffness matrix making the LU decomposition so when only the forces were changed (RHS of the equation), it was possible to compute the displacement solution in a fast manner. Using the numpy solver was slower and computing the inverse of the stiffness matrix and keeping it was not a good idea as explained in the article of [2].

Artificial Neural Networks

{chapter:ANN}

Artificial neural networks (NN) are powerful modeling technique that uses the so-called neurons to perform a series of different learning tasks. Nowadays, it's vastly used in problems in that no explicit rules can be programmed, for example, image recognition.

The general construction of an NN is inspired by the structure of the brain neural networks and it was proposed back in 1940 - 50, but reached practical applications in the 80 - 90s. The big boom of the uses came in 2010 with NNs as deep architectures (many hidden layers) reaching high performance in complex learning tasks ([12]).

The NN has a series of interconnections (links) between neurons, which normally are present in large numbers. These connections are normally represented as lines and the neurons as nodes in the graphs. Neurons have a basic function in transforming the input data into an output one via a transfer function. Each interconnection has a weighting factor that is applied before the output information of one neuron enters the next one that is connected by the edge. The input data of the neurons are the summed information that arrives through the edges.

In the hypotheses class in the learning context, it is possible to define the neural network predictors sharing the same underlying graph structure but differing in the weights at the edges. The training process of NN is the minimization of a defined loss function in the hypotheses class of different parameters, normally weights and bias.

A feedforward NN is presented in figure 5.1. This representation is used to explain the NNs because it doesn't contain cycles, as a recurrent NNs, for example, so it is easier to understand. The model is generally structured in layers, input, hidden, and output. The artificial neurons are fully connected and information travel from the input layer to the output layer. The details of the feedforward NNs are presented in section 5.1.

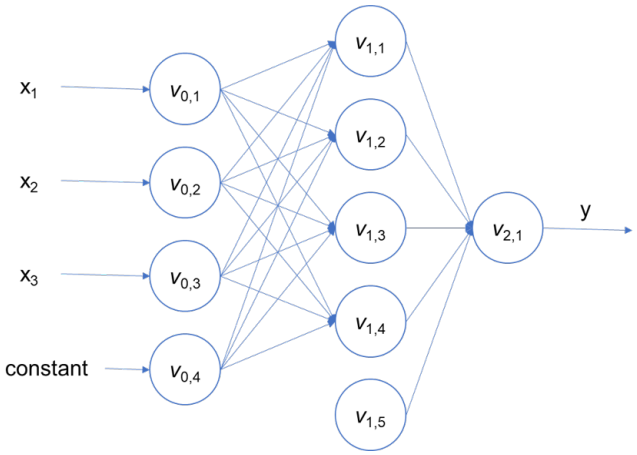


Figure 5.1: Feedforward Neural Network

{fig:NN_feedf

The training process of an NN is a computationally hard task, which is normally solved following the framework of Stochastic Gradient Descendent (SGD) in the learning process. The most complicated part of the computation in SGD is the calculation of the gradient of the loss function, which is being optimized,

with respect to the parameters of the network. This is overcome by the algorithm of backpropagation which efficiently computes the gradient in the Network. All definitions and results from this chapter were extracted or adapted from the book of [9].

5.1 Feedforward

{sec:ANN_feed

The feedforward NN is represented by a directed acyclic graph. Nodes in the graph are neurons and the edges are the interconnection between them.

Every single neuron is a scalar function, transforming the input data in the output one, $\sigma > \mathbb{R} \to \mathbb{R}$ and it generally acts as a threshold function or, as it is called, activation function. There is a large number of possible functions, the most used one is the sigmoid presented in equation 5.1. Three other ones that may be used are the sign, threshold, and relu function, presented in 5.2, 5.3 and 5.4, respectively

$$\sigma(a) = \frac{1}{1 + exp(-a)} \tag{5.1}$$ {eq:sigmoid_f

$$\sigma(a) = sign(a) \tag{5.2}$$ {eq:sign_func

$$\sigma(a) = \infty_{[a>0]} \tag{5.3}$$ {eq:thres_fun

$$\sigma(a) = max(0, a) \tag{5.4}$$ {eq:relu_func

To better understand an NN, the structure can be divided into layers (V). T is the number of layers, which are a disjoint subset of neurons from the NN. The nodes in one layer are connected to the next one, so every layer connects a node in V_{t-1} to nodes in V_t, where t ϵ [T]. The connection is represented by an edge (E) and each one of it has a weight function $w : E \to \mathbb{R}$.

The first layer, V_0, is the input layer and it contains n+1 neurons. The n is the size of the input space and the added neuron is a constant bias value that always has the value 1.

The $v_{t,i}$ represents the ith neuron in the tth layer and the output of this neuron for the input x is denoted by $o_{t,i}(x)$. The calculation proceeds layer by layer, i.e. the outputs of the layers t is the inputs for the layer t+1. It's interesting to note the difference in cyclic NNs, which have outputs of subsequent layers as inputs of the previous one, i.e. some outputs of layer t+1 enter as input in the layer t.

A final expression of the input in the t+1 layer can be constructed using the outputs of the t layer and the weights. Given a j neuron in the t+1 layer, $v_{t+1,j} \epsilon V_{t+1}$, the input $a_{t+1,j}(x)$ when the NN is fed with the input vector x is presented in equation 5.5. The output of the j neuron is presented in equation 5.6

$$a_{t+1,j}(x) = \sum_{r:(v_{t,r}, v_{t+1,j} \epsilon E)} w((v_{t,r}, v_{t+1,j}))o_{t,r}(x) \tag{5.5}$$ {eq:input_tra

$$o_{t+1,j}(x) = \sigma(a_{t+1,j}(x)) \tag{5.6}$$ {eq:output_ne

The input to the neuron $v_{t+1,j}$ is the sum of outputs of the neurons ϵ V_t that are connected to it. Besides the summation, the outputs of the neurons in V_t are weighted by the weight function w assigned over the connecting edge to the neuron $v_{t+1,j}$. The final output of this neuron is only the application of the activation function σ in its input.

The layers $V_1, ..., V_{T-1}$ are normally called hidden layers and the final one, V_T, is the output layer. It can have a single or more neurons depending on the type of target that the model is predicting. The hidden layers t+1 can have neurons that are not connected to the previous layers outputting a constant.

Some parameters to describe a feedforward NN are:

- 1) The value T is generally called the depth of the network (excluding the input layer V_0);
- 2) The size of the network, $|V|$, representing the number of neurons in it;
- 3) The width of the network, $max_t|V_t|$, representing the maximum number of neurons in a layer.

The feedforward NN's structure of depth 2, size 10 and width 5 is presented in figure 5.1.

5.2 NN Learning

{sec:NN_learn

The NN once specified by the parameters (V, E, σ, w) is a function $h_{V,E,\sigma,w} : \mathbb{R}^{|V_0-1|} \to \mathbb{R}^{V_T}$, i.e. a function that with an input returns an output. Any one of these functions can be a hypothesis class for learning.

The hypothesis class of neural network predictors usually used are all functions $h_{V,E,\sigma,w}$ in which the triplet (V, E, σ) are fixed. This triplet is normally called the architecture of the network and the weights over the edges are the parameter defining a hypothesis in the class. The hypotheses class is presented in equation 5.7.

$$\mathcal{H}_{V,E,\sigma} = h_{V,E,\sigma,w} : w \text{ is a mapping from E to } \mathbb{R} \tag{5.7}$$

{eq:hyp_class

In order to understand the learning process of the NN, some basic definitions need to be explained. The learner in the basic statistical learning setting has access to the following data:

- Domain set ($\mathcal{X}$): an arbitrary set of objects that we may wish to label;
- Label set ($\mathcal{Y}$): the possible labels of the objects, for example, 0 or 1;
- Training data (S): The sequence of pairs $\mathcal{X} \times \mathcal{Y}$ that the learner has access. Each input ($\mathcal{X}$) has a known label ($\mathcal{Y}$), often called a target, and it serves as the training set of the problem;
- Learner's output: h, a predictor.

Using this given data, the learner is expected to output a prediction rule, or hypothesis, $h : \mathcal{X} \to \mathcal{Y}$. This hypothesis can be used to predict the label of new objects in the domain set. The notation A(S) identifies the hypothesis that the learning algorithm A returns with the training set S.

The learning algorithm learns through the minimization of the training error that it has access to. This error is called empirical error or empirical risk, so the whole process of outputting a prediction rule h that minimizes the error is called Empirical Risk Minimization (ERM). The equation for the computed empirical risk function for a certain prediction rule h in a training set of m samples is presented in equation 5.8. The equation 5.9 presents the best output of a learner.

$$L_s(h) := \frac{|i\epsilon[m] : h(x_i) \neq y_i|}{m} \tag{5.8}$$

{eq:empirical

$$\hat{h} := \underset{h\epsilon\mathcal{H}}{argmin} L_S(h) \tag{5.9}$$

{eq:ERM}

The learning problem is the problem to minimize the risk function 5.10. This risk is called true risk because it's computed in the true distribution of the process. The empirical risk is generally used as an estimate to minimize the true risk.

$$L_{\mathcal{D}}(\mathbf{w}) = \underset{z\sim\mathcal{D}}{\mathbb{E}}[l(\mathbf{w}, z)] \tag{5.10}$$

{eq:risk_func

where l is the loss function, which computes the error between the predictions and the true labels.

There are many different learning tasks that NN can tackle, here two are described with their suitable loss function to evaluate the quality of the predictor. The way to evaluate a predictor will change according to the objective of the learning and the format of the loss function used is as important as the definition of the hypothesis class.

In Multiclass Classification, for example, the objective is to classify objects based on the training set of correctly classified ones. A predictor that identifies the color (label) (e.g. red, blue, yellow, green) is a Multiclass Classification problem and the evaluation of this predictor could be based on the probability of getting a wrong classification result.

In Regression, the objective is to find a pattern between $\mathcal{X}$ and $\mathcal{Y}$ values, i.e. a function to predict the target $\mathcal{Y}$ using the domain set $\mathcal{X}$ as input. For example, a predictor that identifies the maximum stress ($\mathbb{R}$) of a structure based on the displacement in a point ($\mathbb{R}^2$) is a regression problem and the evaluation of this predictor could be done by the sum of the squared differences between the true target and the predicted values.

Formally, the loss functions (l) are defined as any function such that given a set of hypothesis $\mathcal{H}$ and some domain $\mathcal{Z}$, it maps $\mathcal{H} \times \mathcal{Z}$ to nonnegative real numbers, $l : \mathcal{H} \times \mathcal{Z} \to \mathbb{R}_b$. The domain Z in the prediction problem is the product of the domain set and the labels, i.e. an element in the domain Z is the tuple z = (x, y) where $x \, \epsilon \mathcal{X}$ and $y \, \epsilon \, \mathcal{Y}$.

The empirical risk is the expected loss of a sample S = $(z_1, ..., z_m) \, \epsilon \, \mathcal{Z}^m$ and its equation is presented in 5.11

$$L_S(h) := \frac{1}{m} \sum_{i=1}^{m} l(h, z_i) \tag{5.11}$$ {eq:expected_

The typical loss functions used for classification and regression tasks are:

- 0-1 loss: normally used for classification problems:

$$l_{0-1}(h, (x, y)) := \begin{cases} 0 \text{ if } h(x) = y \\ 1 \text{ if } h(x) \neq y \end{cases} \tag{5.12}$$

- Square loss: normally used for regression problems:

$$l_{sq}(h, (x, y)) := (h(x) - y)^2 \tag{5.13}$$

The solution difficulty of optimizing the empirical risk is directly related to the hypothesis class of the model. An example of an analytical solution for this minimization is for linear regression predictors, which can be solved by deriving the empirical risk expression. Because of its complicated structure, the learning process of the hypothesis class of neural network predictors $\mathcal{H}_{V,E,\sigma}$ is computationally hard and the commonly used method to overcome this is the Stochastic Gradient Descendent and Backpropagation algorithms.

5.3 SGD and Backpropagation

{sec:sgd_back

The Stochastic Gradient Descent algorithm (SGD) is a heuristic applied in the training of NNs. It is used in the minimization of the risk function $L_{\mathcal{D}}(\mathbf{w})$ to find the hypothesis in $\mathcal{H}_{V,E,\sigma}$ tuning the weights over the edges. The minima search in this section will be described using the sigmoid function, equation 5.1, and the loss function as the squared loss, but any derivation using a differentiable scalar function as activation and a differentiable function for loss similarly holds.

Suppose an NN has n inputs and k outputs and given E is a finite set, i.e. a finite number of edges, it's possible to represent the weights as a vector $\mathbf{w} \, \epsilon \, \mathbb{R}^{|E|}$. This NN can be denoted as $h_{\mathbf{w}} : \mathbb{R}^n \to \mathbb{R}^k$ and given a target $\mathbf{y} \, \epsilon \mathcal{Y}$ the loss of the prediction is denoted as $\Delta(h_{\mathbf{w}}(\mathbf{x}), \mathbf{y})$. As mentioned for simplicity, the loss function assumed for the derivations is the squared loss as presented in equation 5.14 and the risk function of the network over the examples domain, $\mathbb{R}^n \times \mathbb{R}^k$, is presented in equation 5.15

$$\Delta(h_{\mathbf{w}}(\mathbf{x}), \mathbf{y}) = \frac{1}{2} \|h_{\mathbf{w}}(\mathbf{x}) - \mathbf{y}\|^2 \tag{5.14}$$ {eq:sgd_loss}

$$L_{\mathcal{D}}(\mathbf{w}) = \mathop{\mathbb{E}}_{(\mathbf{x},\mathbf{y}) \sim \mathcal{D}} [\Delta(h_{\mathbf{w}}(\mathbf{x}), \mathbf{y})] \tag{5.15}$$ {eq:risk_netw

The gradient descendent (GD) procedure is applied in SGD. GD is an iterative optimization procedure in which each step of the solution is improved by taking a step in the direction of the negative gradient from the current point. The SGD is the modification of the gradient computation and the step is done in a random direction that is the expected value of the negative gradient direction. This makes the algorithm less computationally intense and less prone to getting stuck in local minima.

The gradient descendent (GD) approach is normally used in the problem to minimize a differentiable function $f(\mathbf{w})$. Given a differentiable function $f : \mathbb{R}^d \to \mathbb{R}$ at the point $\mathbf{w}$, the gradient $\boldsymbol{\nabla} f(\mathbf{w})$ is computed as the vector of partial derivatives of f as in equation 5.16.

$$\boldsymbol{\nabla} f(\mathbf{w}) = \left(\frac{\partial f(\mathbf{w})}{\partial w_1}, ..., \frac{\partial f(\mathbf{w})}{\partial w_d} \right) \tag{5.16}$$ {eq:gradient_

The GD is an iterative algorithm in that the value of $\mathbf{w}$ is updated in every step according to a step parameter η, the current point, and the gradient computed on it. The update step equation is presented in 5.17. The algorithm starts from an initial value of $\mathbf{w}$, for example, $\mathbf{w}^1 = 0$.

$$\mathbf{w}^{t+1} = \mathbf{w}^t - \eta \boldsymbol{\nabla} f(\mathbf{w}^t) \tag{5.17}$$ {eq:gd_update

This updated value of $\mathbf{w}$ decreases the value of the function going in the opposite direction of the maximum increase of f around $\mathbf{w}^t$. The final output of the algorithm after T steps could be the three options presented in 5.18. The chosen option depends on the use of the results and the methods applied.

$$\hat{w} = \begin{cases} \mathbf{w}^T \\ argmin_{t \,\epsilon\, [T]} f(\mathbf{w}^t) \\ \frac{1}{T} \sum_{t=1}^{T} \mathbf{w}^t \end{cases} \tag{5.18}$$ {eq:gradient_

The requirement of differentiable function can be generalized and the GD algorithm can be applied to nondifferentiable functions using subgradients of $f(\mathbf{w})$ at $\mathbf{w}^{(t)}$. The definition of the subgradients is based on the property of convex functions, which is the existence of a tangent line that lies below f, expressed in lemma 1. The definition of subgradients uses the lemma inequality as presented in 1

{lemma:convex

Lemma 1 *Let S be an open convex set. A function $f : S \to \mathbb{R}$ is convex iff for every $\mathbf{w} \,\epsilon\, S$ there exists $\mathbf{v}$ such that*

$$\forall \mathbf{u} \,\epsilon\, S,\ f(\mathbf{u}) \geq f(\mathbf{w}) + < \mathbf{u} - \mathbf{w}, \mathbf{v} >$$

{def:subgradi

Definition 1 *A subgradient of f at $\mathbf{w}$ is a vector $\mathbf{v}$ that satisfies the inequality in Lemma 1. The differential set denoted by $\partial f(\mathbf{w})$ is the set of subgradients of f at $\mathbf{w}$.*

The difference in the SGD algorithm is that the updating direction may not be exactly the opposite of the gradient of the function. In the iterations, it's allowed that the direction assumes a random vector, but the expected value at each iteration is required to be equal to the gradient direction. Based on previous generalizing definitions, the requirement is that the expected value of the random vector is a subgradient of the function at the current vector.

The SGD algorithm iterative steps are presented below. The algorithm is repeated for T steps and the direction of updating is regulated by the parameter η.

Stochastic Gradient Descent (SGD) for minimizing function f(w)
parameters: Scalar $\eta > 0$, integer T > 0
initialize: $\mathbf{w}^{(1)} = \mathbf{0}$
for t from 1 to T:

choose $\mathbf{v}_t$ at random such that $\mathbb{E}[\mathbf{v}_t |\ \mathbf{w}^{(t)}] \,\epsilon\, \partial f(\mathbf{w}^{(t)})$
update $\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)}$ - $\eta \mathbf{v}_t$

output $\bar{w} = \frac{1}{T} \sum_{t=1}^{T} \mathbf{w}^{(t)}$

For learning problems, the construction of the random vector that satisfies the requirement of the SGD is easy and comes directly from the loss function. The procedure is modified as described below.

Stochastic Gradient Descent (SGD) for minimizing $L_{\mathcal{D}}(\mathbf{w})$
parameters: Scalar $\eta > 0$, integer T > 0
initialize: $\mathbf{w}^{(1)} = \mathbf{0}$
for t from 1 to T:

sample z $\sim \mathcal{D}$
pick $\mathbf{v}_t \, \epsilon \, \partial l(\mathbf{w}^{(t)}, z)$
update $\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)}$ - $\eta \mathbf{v}_t$

output $\bar{w} = \frac{1}{T} \sum_{t=1}^{T} \mathbf{w}^{(t)}$

The SGD for the neural network learning process is slightly modified compared to the previous steps presented. The minimization of $L_{\mathcal{D}}$ has not a closed-form solution, the computation of its gradient is a computationally intense procedure and the function $L_{\mathcal{D}}$ is nonconvex and typically presents many local minima with a not smooth graph.

There are many details for the neural network SGD:

- The initialization of the weights is a vector with random values close to zero, this is because a zero vector would lead the learning process of a fully layered network to have the same weights in all hidden neurons. This procedure helps to escape from bad local minima when the SGD is repeated many times as well.
- The step size, η, that regulates the updating of the weights needs to be variable, η_t, because of the nonconvexity of the loss function. The variation of the step size is tuned by trial and error.
- The choice of the weights is based on the best-performing vector on a validation set.
- Possible addition of regularization, which reduces the space of feasible weights combination by modifying the function optimized. One possibility is to add an L2 regularization with parameter λ as in equation 5.19.

$$L_{\mathcal{D}}(\mathbf{w}) + \frac{\lambda}{2}||\mathbf{w}||^2 \tag{5.19}$$ {eq:regulariz

Stochastic Gradient Descent (SGD) for Neural Networks
parameters: Number of iterations τ; step size sequence $\eta_1, \eta_2, ..., \eta_\tau$; regularization parameter $\lambda > 0$
input: $\mathcal{H}_{V,E,\sigma}$
initialize: choose $\mathbf{w}^{(1)} \, \epsilon \, \mathbb{R}^{|E|}$ at random s.t. $\mathbf{w}^{(1)}$ is close enough to $\mathbf{0}$
for i from 1 to τ:

sample ($\mathbf{x}$, $\mathbf{y}$) $\sim \mathcal{D}$
calculate $\mathbf{v}_i$ = backpropagation($\mathbf{x}$, $\mathbf{y}$, $\mathbf{w}$, $\mathcal{H}_{V,E,\sigma}$)
update $\mathbf{w}^{(i+1)} = \mathbf{w}^{(i)}$ - $\eta_i(\mathbf{v}_i + \lambda \mathbf{w}^{(i)})$

output $\bar{w}$ = best performing $\mathbf{w}^{(i)}$ on validation set

Backpropagation is the algorithm that computes the gradient of the loss function in the neural network, exploiting the construction of the model in layers. In the next brief explanation, the mechanism of computing the gradient of the loss function on an example ($\mathbf{x}$,$\mathbf{y}$) w.r.t. the vector $\mathbf{w}$ is presented. An important definition for understanding the backpropagation algorithm is the Jacobian of $\mathbf{f}$ at $\mathbf{w} \, \epsilon \, \mathbb{R}^n$, given f is a function with multiple m outputs $\mathbf{f}$: $\mathbb{R}^n \to \mathbb{R}^m$. The Jacobian definition is presented in definition 2 for this previously described function.

{def:Jacobian

Definition 2 *The Jacobian of* $\boldsymbol{f}$ *at* $\boldsymbol{w} \epsilon \mathbb{R}^n$*, denoted as* $J_{\boldsymbol{w}}(\boldsymbol{f})$*, is an* $m \times n$ *matrix of partial derivatives in which the ith row is related to the ith output variable and the jth column is related to the jth variable at* $\boldsymbol{w}$*. Given* f_i*:* $\mathbb{R}^n \to \mathbb{R}$ *the ith output function, the element at i,j of the Jacobian matrix is presented in 5.20.*

$$J_{i,j} = \frac{\partial f_i}{\partial w_j} \tag{5.20}$$ {eq:jacobian}

The chain rule propriety for the Jacobian given two functions $\mathbf{f}$: $\mathbb{R}^n \to \mathbb{R}^m$ and $\mathbf{g}$: $\mathbb{R}^k \to \mathbb{R}^n$ and their composition (f ∘ g): $\mathbb{R}^k \to \mathbb{R}^m$ is presented in equation 5.21.

$$J_{\mathbf{w}}(f \circ g) = J_{g(\mathbf{w})}(f) J_{\mathbf{w}}(g) \tag{5.21}$$ {eq:jacobian_

The backpropagation exploits the decomposition of the neural network in layers to compute the gradient. For every layer t in the NN denote as $V_t = v_{t,1}, ..., v_{t,k_t}$ the set of $k_t = |V_t|$ neurons and as $W_t \epsilon \mathbb{R}^{k_{t+1}, k_t}$ the weight matrix, which gives the values of the connections of layer V_t to V_{t+1}. If the edge E exists the $W_{t,i,j}$ is the weight of the edge $(v_{t,j}, v_{t+1,i})$, and when the edge is not present, the weight is set to 0.

Computing the partial derivatives w.r.t. edges from V_{t-1} to V_t is computing them w.r.t. the elements in W_{t-1}. All other weights are fixed, so the outputs of the neurons in the layer V_{t-1} are fixed numbers denoted by the vector $\mathbf{o}_{t-1}$. A loss function for the subnetwork of layers V_t to V_T can be defined based on the output of the neurons in V_t. This loss function is denoted by $l_t : \mathbb{R}^{k_t} \to \mathbb{R}$ and can be written as in equation 5.22.

$$g_t(W_{t-1}) = l_t(\mathbf{o}_t) = l_t(\sigma(\mathbf{a}_t)) = l_t(\sigma(W_{t-1}\mathbf{o}_{t-1})) \tag{5.22}$$ {eq:nn_loss_f

For convenience, the previous results are written in a transpose form reducing the weight matrix to a vector. Let $\mathbf{w}_{t-1} \epsilon \mathbb{R}^{k_{t-1}k_t}$ be the reduced row vector by concatenating the rows of W_{t-1}. The output vector is transformed to an $k_t \times (k_{t-1}k_t)$ matrix O_{t-1}, which values in the diagonal are the transposed $\mathbf{o}_{t-1}$ vector. This rearrangement modifies the input of the neurons in the layer V_t as $W_{t-1}\mathbf{o}_{t-1} = O_{t-1}\mathbf{w}_{t-1}$ and the loss function of the layer becomes equation 5.23.

$$g_t(\mathbf{w}_{t-1}) = l_t(\sigma(O_{t-1}\mathbf{w}_{t-1})) \tag{5.23}$$ {eq:loss_laye

Besides the chain rule in terms of Jacobian, two results are used to compute the Jacobian of the loss function:

- Given $\mathbf{f}(\mathbf{w}) = \mathrm{A}\mathbf{w}$ for $\mathrm{A} \epsilon \mathbb{R}^{m,n}$. $J_{\mathbf{w}}(\mathbf{f}) = \mathrm{A}$
- For every n, given the notation σ to the function $\mathbb{R}^n$ to $\mathbb{R}^n$ that applies the sigmoid function element-wise. Therefore denoting $\alpha = \sigma(\theta)$, for every i $\alpha_i = \sigma(\theta_i) = \frac{1}{1+exp(-\theta_i)}$. The Jacobian $J_\theta(\sigma)$ is a diagonal matrix with entry $\sigma'(\theta_i)$. The derivation of the scalar sigmoid function is presented in equation 5.24.

$$\sigma'(\theta_i) = \frac{1}{(1 + exp(\theta_i))(1 + exp(-\theta_i))} \tag{5.24}$$ {eq:sigmoid_f

The Jacobian of the loss function in the layer V_t is computed using the chain rule and it is presented in equation 5.25.

$$J_{\mathbf{w}_{t-1}}(g_t) = J_{\sigma(O_{t-1}\mathbf{w}_{t-1})}(l_t) diag(\sigma'(O_{t-1}\mathbf{w}_{t-1}))O_{t-1} \tag{5.25}$$ {eq:jacobian_

In terms of output and input values of the layer, the equation is simplified to 5.26.

$$J_{\mathbf{w}_{t-1}}(g_t) = J_{\mathbf{o}_t}(l_t) diag(\sigma'(\mathbf{a}_t))O_{t-1} \tag{5.26}$$ {eq:jacobian_

Denoting $\delta_t = J_{\mathbf{o}_t}(l_t)$, the Jacobian can be rewritten as equation 5.27.

$$J_{\mathbf{w}_{t-1}}(g_t) = (\delta_{t,1}\sigma'(a_{t,1})\mathbf{o}_{t-1}^T, ..., \delta_{t,k_t}\sigma'(a_{t,k_t})\mathbf{o}_{t-1}^T) \tag{5.27}$$ {eq:jacobian_

The gradient of l_t at $\mathbf{o}_t$, δ_t, is calculated in a recursive manner. Starting from the last layer, V_T, the loss is computed using the NN output $\mathbf{u}$ and the true labels $\mathbf{y}$ as $l_T(\mathbf{u}) = \Delta(\mathbf{u}, \mathbf{y})$. Assuming the loss function is $\Delta(\mathbf{u}, \mathbf{y}) = \frac{1}{2}||\mathbf{u} - \mathbf{y}||^2$, $J_{\mathbf{u}}(l_T) = (\mathbf{u} - \mathbf{y})$. In the last layer, $\delta_T = J_{\mathbf{o}_T}(l_T) = (\mathbf{o}_T$ - $\mathbf{y})$. Noting equation 5.28 and using the chain rule, the Jacobian of the loss function on the layer V_t can be written using the loss function on the layer V_{t+1}, the final result is expressed in equation 5.29.

$$l_t(\mathbf{u}) = l_{t+1}(\sigma(\mathbf{W}_t u)) \tag{5.28}$$ {eq:sublosses

$$\delta_t = \delta_{t+1} diag(\sigma'(\mathbf{a}_{t+1}))W_t \tag{5.29}$$ {eq:backward_

This result for the Jacobian and the definition of the computation of the vectors a_t, o_t in the NN fully define how to compute the elements of equation 5.27. Therefore, the Backpropagation algorithm pseudo code is the following, which can be divided into two macro steps: forward and backward. The first one is the computation of all inputs and outputs of neurons run from the input to the output layer; the second one, is the computation of all Jacobians run from the output to the input layer.

Backpropagation for NNs
input: an example ($\mathbf{x}$, $\mathbf{y}$), weight vector $\mathbf{w}$, $\mathcal{H}_{V,E,\sigma}$
forward:

set $\mathbf{o}_0 = \mathbf{x}$
for t from 1 to T:
for i from 1 to k_t
set $a_{t,i} = \sum_{j=1}^{kt-1} W_{t-1,i,j} o_{t-1,j}$
set $o_{t,i} = \sigma a_{t,i}$
backward:
set $\delta_T = \mathbf{o}_T$ - $\mathbf{y}$
for t = T-1, T-2, ..., 1
for i = 1, ..., k_t
$\delta_{t,i} = \sum_{j=1}^{k_{t+1}} W_{t,j,i} \delta_{t+1,j} \sigma'(a_{t+1,j})$
output each edge partial derivative of ($v_{t-1,j}$,$v_{t,i}$) is set to $\delta_{t,i}\sigma'(a_{t,i})o_{t-1,j}$

Commonly in NN modeling, the so-called SGD performs a weight parameters update, using the algorithm described, for each example in the training set, which presents a high variance and can be unstable besides complicating the convergence to the minimum. The GD performs the parameter update using the entire training dataset, which can be very costly and slow in the context of NNs being trained with large datasets. An intermediate approach is using batches and updating the weight parameters using n training examples, a method called Mini-batch GD, which leads to lower variance in the updates and more stable convergence. Deep learning libraries such as TensorFlow let the user specify the number of samples that the model has to take into account to update its weights. The common terminology in the libraries is "Batch size" for the number n of training examples presented before updating the weights and Epoch for the number of times that the training set is presented to the NN until the end of the learning process (optimization).

5.4 Overfitting, underfitting and data transformation

{overfitting_

It's possible to see that the number of parameters in the neural network predictors class can be huge. This can lead to overfitting in the learning process and it's a challenge that needs to be addressed in this type of predictor.

Overfitting it's a natural path that the learning may lead and can be described as a learner reaching a predictor that has an excellent performance in the training set, but a very bad one in the true "world". This happens when the hypothesis chosen by the learner in the hypothesis class is the one that fits the training set very well and can not generalize to new samples. The validation set is used to identify the scenario of overfitting and escape from it.

It's obvious that one predictor overfitted to the training set is probably not useful, because it may not predict well outside the already known domain. Choosing a more restricted hypothesis class protects against overfitting, but this causes a stronger inductive bias.

Overfitting is detected in the training by monitoring the loss in a validation set. The validation set is composed of samples that the model has no access to during the training and is used to estimate the true loss of the model. Many different strategies can be applied to segregate the data collected in a training and validation set. Overfitting starts when the training error of the NN keeps declining in the learning process, but the validation error starts increasing.

There are some computationally cheap methods to avoid overfitting:

- The dropout method, which consists in a method that randomly "removes" neurons during the training from the network, and at each step the network is trained with a subset of neurons, which reduces the number of weights and avoid the output depending "too much" on a single neuron. The probability of dropout is uniform and independent between neurons and is a hyper-parameter that needs to be tuned.
- Regularizing the weights, which consists in adding a term in the loss function as a penalization for having a high sum of the weights. An L1 or L2 term of the weights can be used and the hyper-parameter λ can be tuned to determine how big is the penalty.
- Monitoring the training and validation loss is valuable to identify the right moment to stop the learning process. This method to avoid overfitting is called early stopping and it uses the validation error to decide to stop the training.

The opposite of overfitting is underfitting, which happens when the hypothesis class is not good for fitting the problem or the learning algorithm was not completed successfully. The underfitting because of the low hypothesis class complexity in NN can be addressed by increasing the number of neurons or layers in the network topology. Selecting the best hypothesis class between the many topologies trained is common when building NN models. The selection is based on the validation error and, normally, there is a test set to estimate the final true risk of the fully defined NN model.

Another practical point about Neural Networks is that the input and output data are generally standardized when building them, which may enhance stability depending on the data because weights and gradients would be lower and more uniform. For example, large inputs may lead to large weights and a problem of scale between inputs may be present. Outputs with large spread may cause problems in gradient computation as well and the range of the output has to match the range of the activation function in the output layer.

In this report, the MinMaxScaler transformation was applied to the input and output vectors to limit the range of the values between 0 and 1, translating and scaling each vector, when suitable. The transformation equation is presented in 5.30 and the inverse transformation is presented in 5.31, where X is the variable that's being standardized. In the modeling, the scaler is fitted in the train samples to set the min and max values.

$$X_{standard} = \frac{(X - min(X))}{(max(X) - min(X))} \tag{5.30}$$ {eq:MinMaxSca

$$X_{scaled} = X_{standard} * (max - min) + min \tag{5.31}$$ {eq:inv_MinMa

5.5 Neural network code

{sec:neural_n

The Neural Network in this report was built using the deep learning library TensorFlow with the support of Keras API. TensorFlow is a library that enables the construction and computation of Neural Networks using tensors. The Keras API makes TensorFlow syntax more friendly and straightforward to construct the Neural Networks and it presents many callbacks that make it possible to easily control the learning process.

Four callbacks from Keras were used in the training of the NNs to help avoid overfitting and understand the model training. The list of callbacks applied were:

- ReduceLRONPlateau: reduces the learning rate in the optimization if the monitor variable is not improving after a specified number of epochs;
- EarlyStopping: stops the training process when the variable that's being monitored is not improved after a specific number of epochs;
- ModelCheckpoint: saves the best model, in terms of the monitor variable, during the training;
- CSVLogger: saves the log of the training, i.e. register the metrics and losses for each epoch.

Different Neural Network structures were tested with varying different parameters for each learning problem in this report. The list of parameters that was chosen to define the model were:

- Hidden layers: the number T of hidden layers in the problem
- Number of neurons: the number k_t of neurons in each hidden layer
- Dropout: the dropout probability in the hidden layers
- Learning rate: the factor that controls the updating of the weights η
- Epochs: the number of times all samples were presented for the model in the training
- Batch size: the number of samples that was taken each time to update the weights
- Activation function: the function of the neurons in the hidden layers

The topology and learning configuration that best suited each learning problem was chosen by comparing the performance of the Neural Networks built using cross-validation. The process of cross-validation consists in keeping part of the training data that is not presented to the model in the training process to evaluate the error.

Normally, the training part reserved for validation is changed many times so the choice of the best model is evaluated without biasing to one specific validation set. A common method of cross-validation is called k-fold, which split the data in k folds, keeping one for validation. For example, cross-validation with 5 folds initially split the data into 5 equal parts and in 5 sessions of the training presents 4 folds to the neural network and validates with the 5th one. The validation set changes each session so all folds of the data are tested.

The performance was evaluated using the k-fold and comparing the mean and variance of the square loss of the validation folds. The best significant mean was chosen between the different candidate models. The final model was labeled then with the error computed using a test set of data that was kept from the beginning.

Results

{chapter:resu

This section presents the results of this project that included the FEM simulations and the modeling of some problems using Neural Networks. The machine learning problems were based on the simulated structure results. The first step in generating the results was validating if the code solution was good.

6.1 Validation of FEM code

{sec:validati

The validation of the code was performed by comparing the results with the one presented in section 4.5 of [6]. The geometry is a cantilever beam under endpoint load and two points are evaluated for displacement and stress in the x direction. The geometry is presented in figure 6.1 with all other relevant data for the problem as the indication of points A and B where the results are evaluated. All units are in SI.

The exactness of the solution and convergence was checked by running the implemented code with 8 different numbers of elements. The results are presented in figure 6.2b for displacement and 6.2a for stress. The exact solution for the displacement at point A is 0.18 and the σ_x at point B is 2.7E+05. It is possible to see that the results from the code developed are good and the convergence to the exact solution occurs with the increase of elements in the mesh. The number of elements around 5000 was taken to build all simulations of this report.

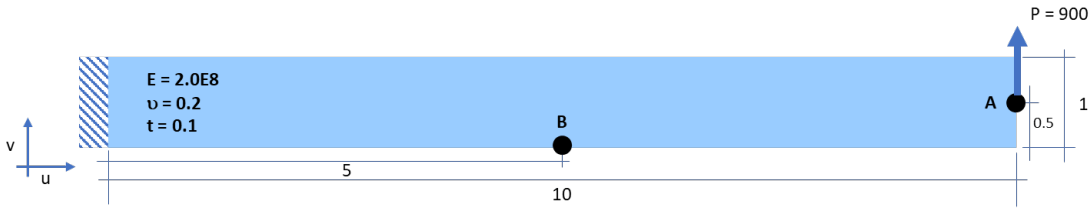
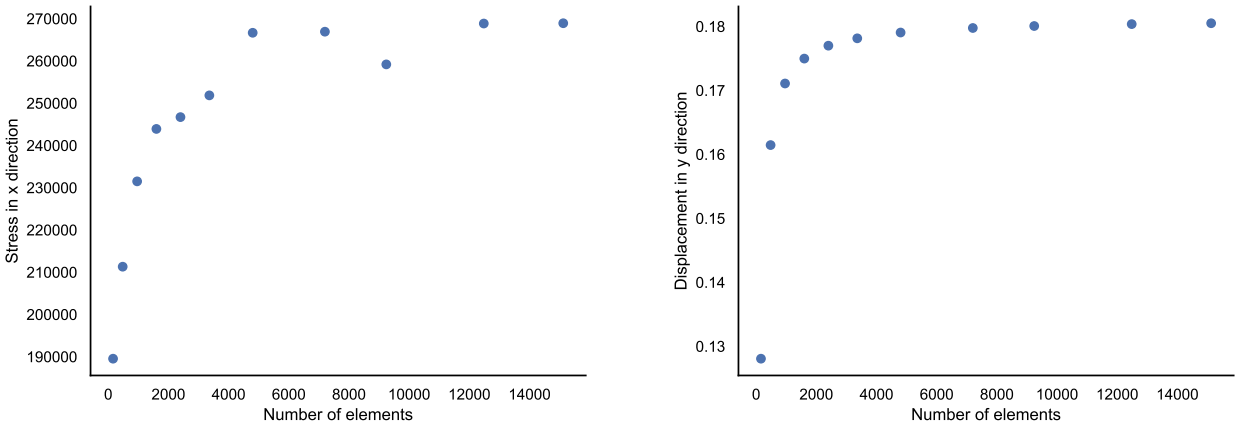


Figure 6.1: Validation problem adapted from [6]

{fig:validato

The results that the FEM code outputted for the maximum number of elements in the validation problem are presented in the figures 6.3, 6.4, which presents the final deformed geometry with discretization and final x, y, xy and von Mises stresses, respectively. All units are in SI and the figures with color mapping present the value of the variables in the nodes with smoothing from the plotting library.

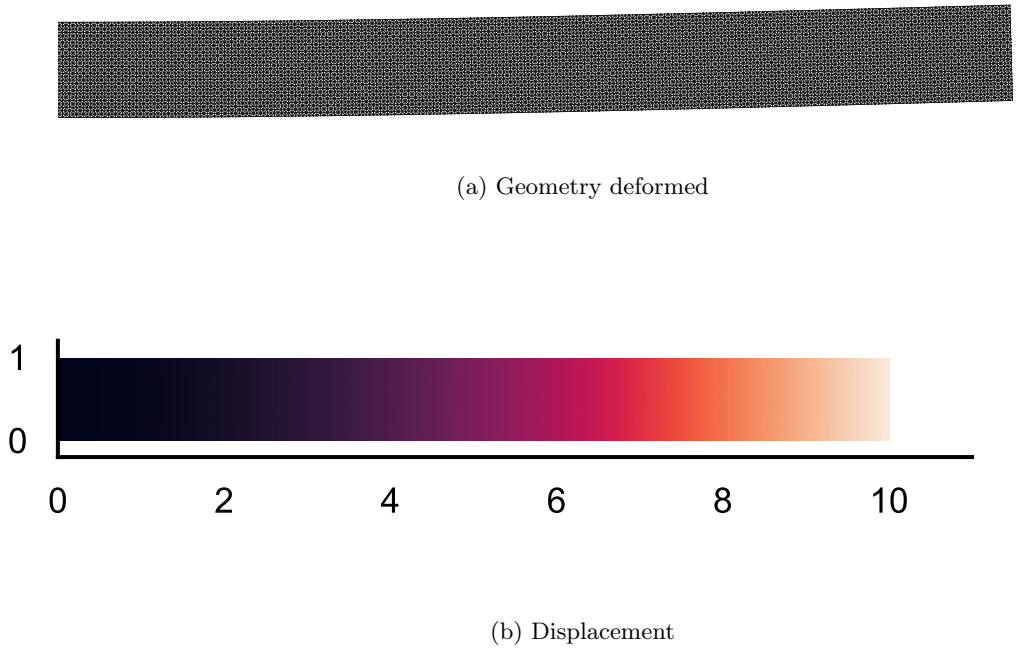
It's important to notice from the figures that the stress field is more complicated than the displacement in the geometry, this is especially observed in the fixed nodes and at the point where the force is applied. Stress naturally has higher gradients than displacement and presents many discontinuities in the approximation used. This complicated profile of stress in the places mentioned needs more elements to converge or higher order shape functions, this is discussed in the latter results which compare the results of stress and displacement convergence with the increase of the number of elements for a different structure.



(a) Stress in point B

(b) Displacement in point A

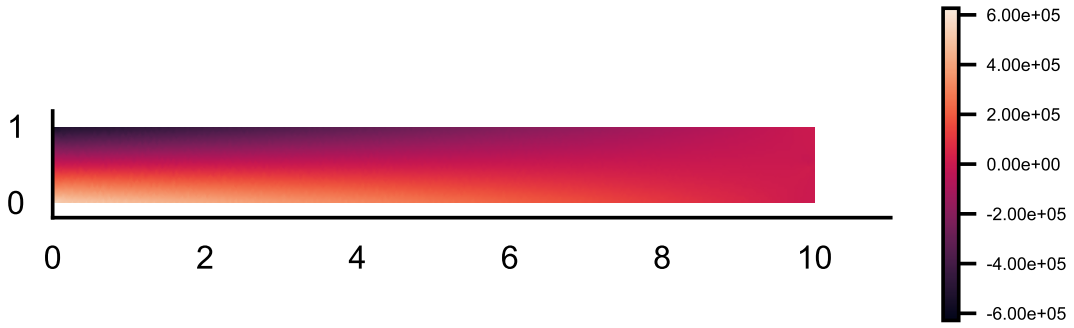
Figure 6.2: Stress and Displacement convergence with the increase of the numbers of elements



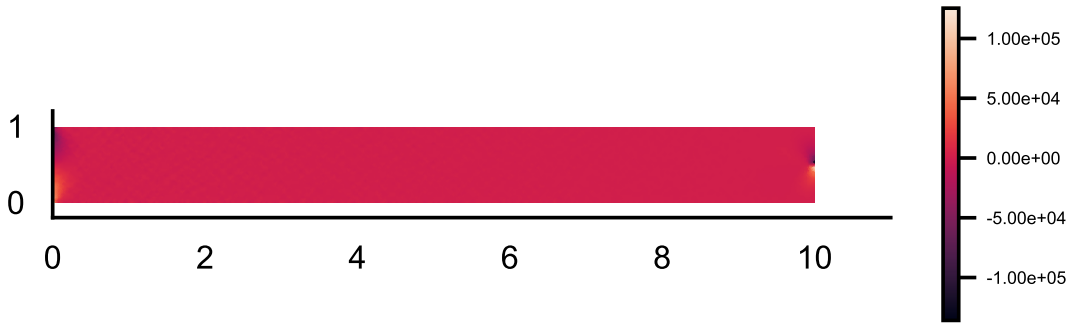
(a) Geometry deformed

(b) Displacement

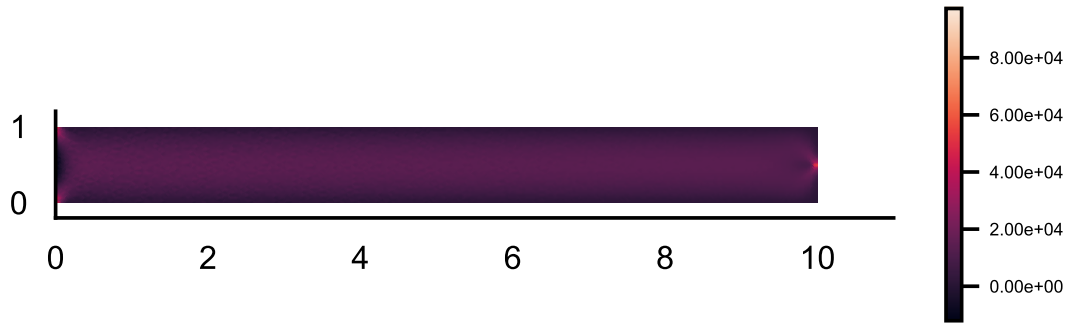
Figure 6.3: Validation geometry deformed and displacement color mapping



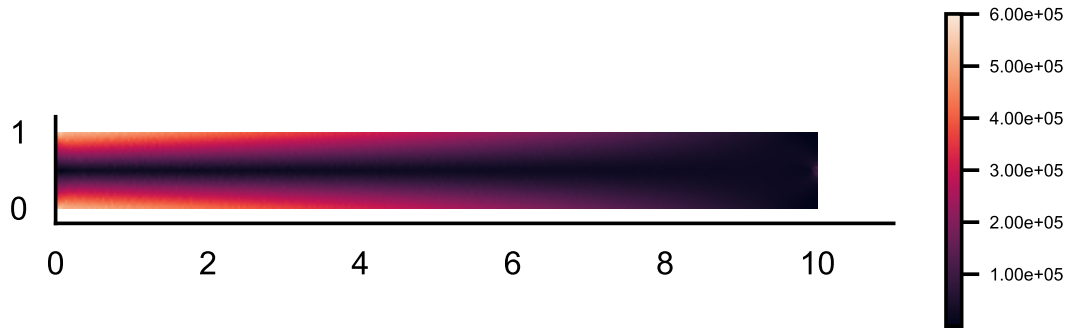
(a) Stress x direction



(b) Stress y direction



(c) Stress xy direction



(d) von Mises stress

Figure 6.4: Nodal stresses color mapping in the validation geometry

6.2 Data generation - FEM results

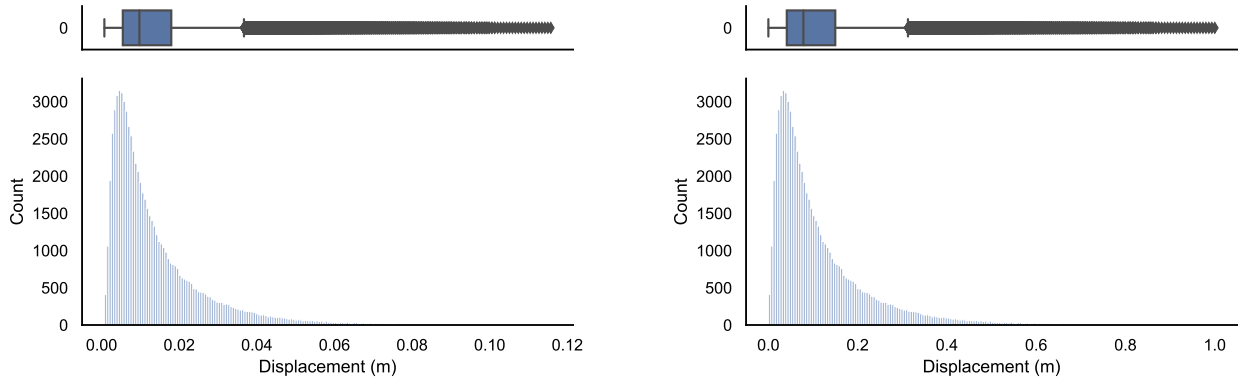
{sec:fem_resu

Using the validated code many different conditions of load, geometry, and materials were simulated to build samples to be used in the training and testing of the Neural Networks. The table 6.1 presents the intervals of parameters used in the simulation runs and the number of different values that were used in the simulations, all units are in SI. The total number of simulated cases was 113.190 and the results of displacement and stresses were stored. The young modulus simulated were generated between the ones from Aluminium (7e10 Pa) and Steel (2e11 Pa) and all loads applied were in the y direction at the middle tip of the beam as in the validation problem.

The relevant maximum, minimum and mean of the outputs for the FEM simulations are presented in table 6.2 and 6.3. The variables presented are the maximum or minimum outputs of each simulated structure, e.g. the maximum stress in the x-direction between all nodal stresses in the x-direction for a given simulated structure. The maximum and minimum for stresses were compiled because the maximum represents the maximum stress of traction in the structure; and, the minimum, the maximum stress of compression.

It's important to notice that although the linear assumption was considered for all simulations, this could be not a good approximation of displacement in the cantilever beam for some cases. The rules of thumb presented in the article of [4] are not satisfied in some simulations, but they are sufficient for the purpose of this present work. The problem could be modeled by updating the stiffness matrix to consider the change in the geometry (nonlinear geometry) to evaluate the impact of the linear assumptions in all simulations, but this is computationally hard and it is out of the scope of this report. It is favorable to notice that using Neural Network models to simulate structures on a nonlinear geometry assumption could return a higher economy of computational effort than in linear case simulations.

The distribution profiles are presented in figures 6.5 and 6.6 with the data after and before standardization for the maximum displacement in the y-direction and the maximum von Mises stress, respectively. The figures present the data distribution in histogram and boxplot forms and are possible to see that the standardization maintained the general properties of the distributions.



(a) Displacement

{fig:distribution_maxuy}

(b) Standardized displacement

{fig:distribu

Figure 6.5: Distributions of target: maximum displacement in y direction

fig:distribution_displacement

Table 6.1: Values used in the FEM simulation.

tab:simulation_variables

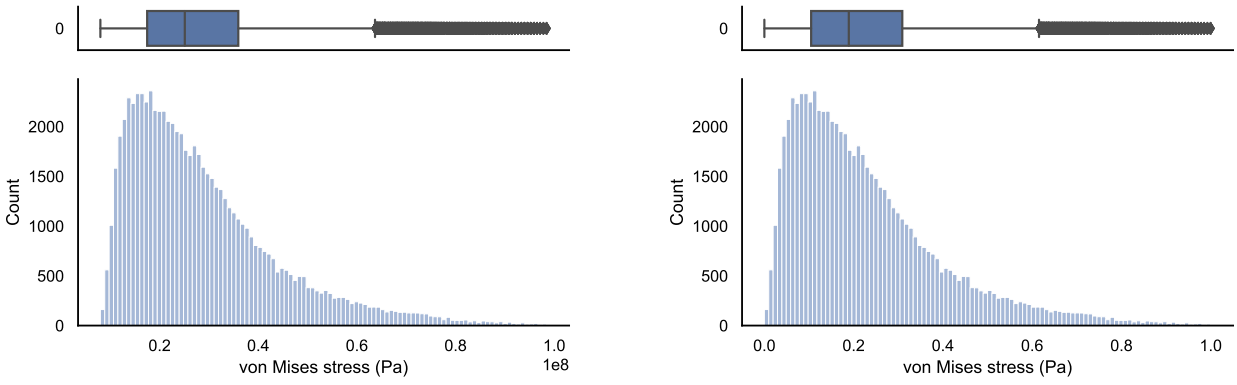
Parameter	Values	Number values
Geometry length	8 to 10	7
Geometry height	0.8 to 2	7
Geometry thickness	0.1 to 0.25	7
Material young modulus	7e10 to 2e11	11
Load magnitude	90000 to 110000	30

Table 6.2: Interval of stress variables for all cases simulated | tab:simulation_output_displacement

Output variable	Minimum	Maximum	Mean
Maximum displacement x direction	1.1e−4	6.91e−3	1.20e−3
Maximum displacement y direction	7.13e−4	1.16e−1	1.39e−2

Table 6.3: Interval of stress variables for all cases simulated | tab:simulation_output_stress

Output variable	Minimum ($\times 10^6$)	Maximum ($\times 10^6$)	Mean ($\times 10^6$)
Minimum stress in x direction	−101	−8.28	−29.3
Maximum stress in x direction	7.75	102	29.9
Minimum stress in y direction	−30.4	−2.33	−8.28
Maximum stress in y direction	2.47	30.7	8.85
Minimum stress in xy direction	−21.8	−0.103	−0.947
Maximum stress in xy direction	1.82	17.6	5.55
Maximum von Mises stress	7.98	98.48	28.83



(a) von Mises (b) Standardized von Mises

{fig:distribution_vonMises} fig:distribution_vonMises

Figure 6.6: Distributions of target: maximum von Mises stress

6.3 FEM results exploration

The maximum displacement in the y-direction of the nodes was compared using the different results generated with respect to the change in load, young modulus, and geometry dimensions. Figures 6.7 and 6.8 present the results of the mean of the maximum displacement in the y directions with respect to the load magnitude for aluminum and steel, respectively. Figures 6.9 and 6.10 present the same results but with respect to the geometry dimensions and, the figure 6.11, with respect to young modulus. In all figures, the filled area around the mean points represents the variance of data and the graph on the right side presents the results with all other variables fixed but the one on the x-axis. The geometry selected for the young and force figures was the one with 7 m of length, 0.8 m of height, and 0.2 m of thickness. The geometry figures used the same dimensions but varied the one in the x-axis, and the load magnitude of 91.896N.

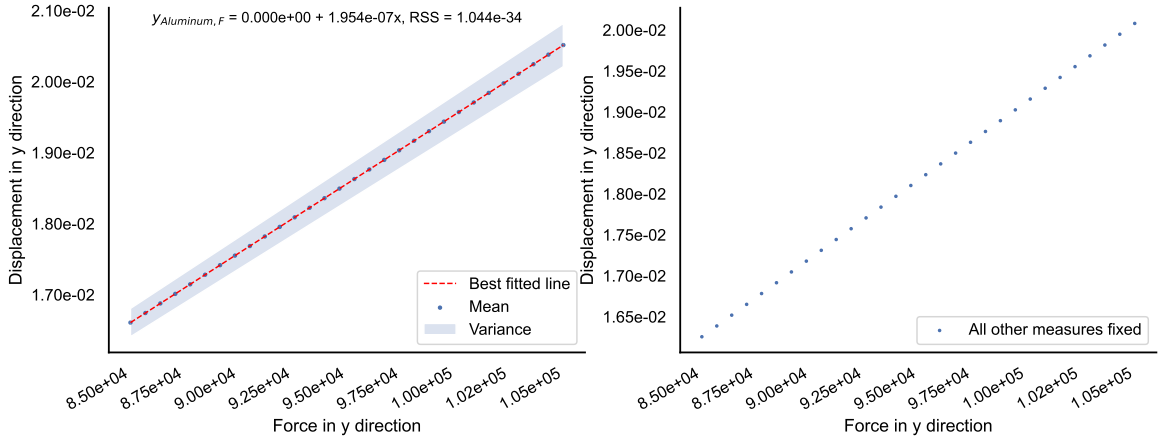


Figure 6.7: Maximum displacement in y-direction by load magnitude for aluminum {fig:meanvard

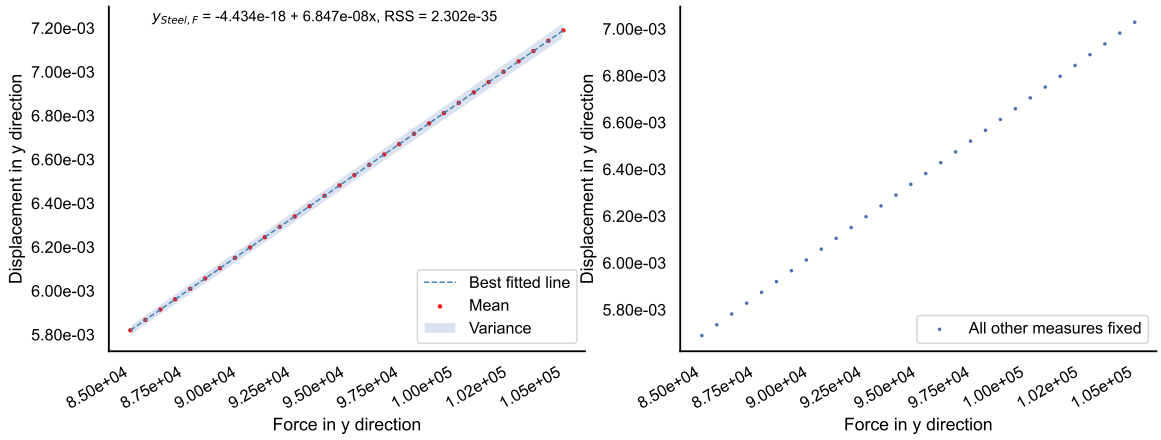


Figure 6.8: Maximum displacement in y-direction by load magnitude for steel {fig:meanvard

From the figures 6.7 and 6.8, it´s possible to see that the relation of load magnitude and the maximum displacement is linear and the variance appears to be linear with the force as well. The angular coefficient of the fitted line for aluminum is higher than the steel as expected from the higher young modulus of the latter one. Equally, the variance in the displacement of aluminum geometries is higher than from the steel ones.

It can be seen that the relation between maximum displacement and length/height is not linear, as expected from the way the stiffness matrix is resolved. The explanation of the inverse relation of maximum

displacement to thickness is trivial when we look at equation 3.40 as thickness doesn't change the shape functions.

The same analysis was done for the maximum nodal stresses (x, y and xy direction and von Mises) by plotting against the change in load magnitude, geometry dimensions, and young modulus. Figures 6.12 and 6.13 present the plotting for load magnitude for aluminum and steel, respectively; figures 6.14 to 6.16, for geometry dimensions; and figure 6.17, for young modulus. The variance filled area is not presented in the graphs, because it had values too high to be informative, and from figures 6.12 and 6.13, it's possible to check that the stresses are not dependent on Young modulus of the material so all plots by geometry dimensions were only based on aluminum.

From figure 6.17, it is possible to see that the variation in the stresses by the Young modulus arises only because of truncation or approximation errors, and the stresses are actually independent of this variable.

6.3.1 Convergence of stress

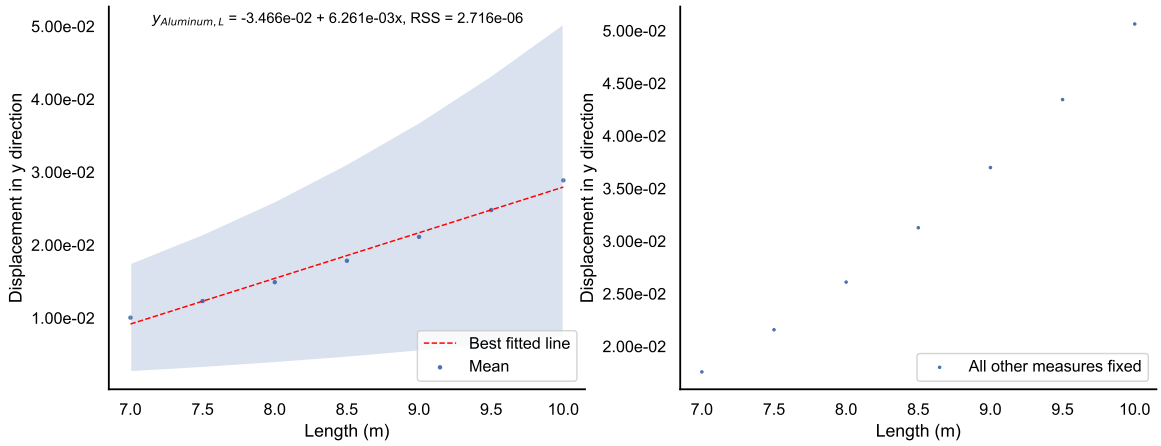
{sec:converge

There were problems in the computation of stress in the structures indicating that the number of elements used for them was not sufficient to converge the value of nodal stresses. The structures where the problem of convergence appears are the ones with the highest height, length, and young modulus as seen in figures 6.15b, 6.14a and 6.17a. The problem can be noticed by the abrupt change of tendency of the stress in these conditions, which might indicate the converging issue when using only 4.200 for these structures. New simulations were performed for different numbers of elements for three structures with the highest length (10), highest height (1.5), and highest young modulus to test this hypothesis. All the simulations presented problems in the convergence of the stress and figure 6.18 presents the convergence of maximum displacement, maximum stresses in x, y, xy direction, and von Mises for an aluminum structure of length of 10 m, height of 0.8 m, thickness of 0.2 m.

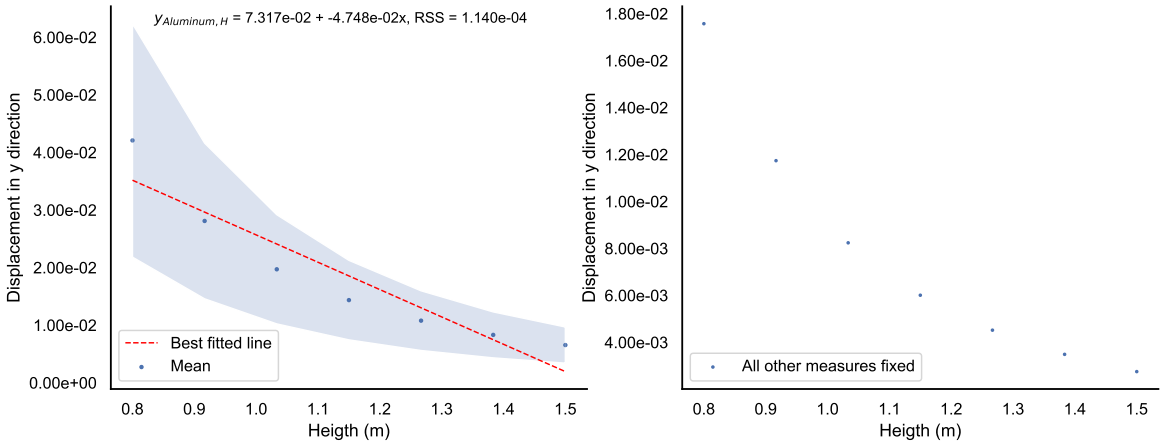
It's possible to check from figure 6.18 that the maximum stress in x, y and xy directions convergence are not monotonic and stable with the increasing number of elements as the maximum displacement, presenting fluctuations not present in the latter. This would be explained by the fact that the nodal stresses are obtained by the derivation of displacement, which reduces the order of the approximation, but in this report, the stresses were computed in the central Gauss point of the triangles. Therefore, the nodal stresses computation comes with an approximation error equal to the displacement but presents discontinuities. The problem in the oscillation of the maximum stresses in the structures is explained by the more complicated behavior of the stress field compared to the displacement one, presenting high gradients that are poorly approximated by the linear shape functions and which suffer from the quality of discretization in regions which the high gradients are present.

In this report, the shape functions are linear, so each element's stress field is constant, resulting in a stress field with large discontinuities. The averaging of stress used in this report is a way to smooth the discontinuity to obtain values in the nodes. Different stress smoothing techniques are present in section 9.8 of [6]; and better points to compute stress and strain (Gauss points), are in section 6.7. If the nodal stress needs to be more accurate and continuous to an application or to build models using it, the number of elements can be increased, but other strategies to increase the accuracy may be implemented, for example, refining the mesh only in the regions were the stress gradients are higher or using higher order shape functions. These strategies and new simulations of more elements were out of the scope of this report because no models were built using nodal stress results in the end.

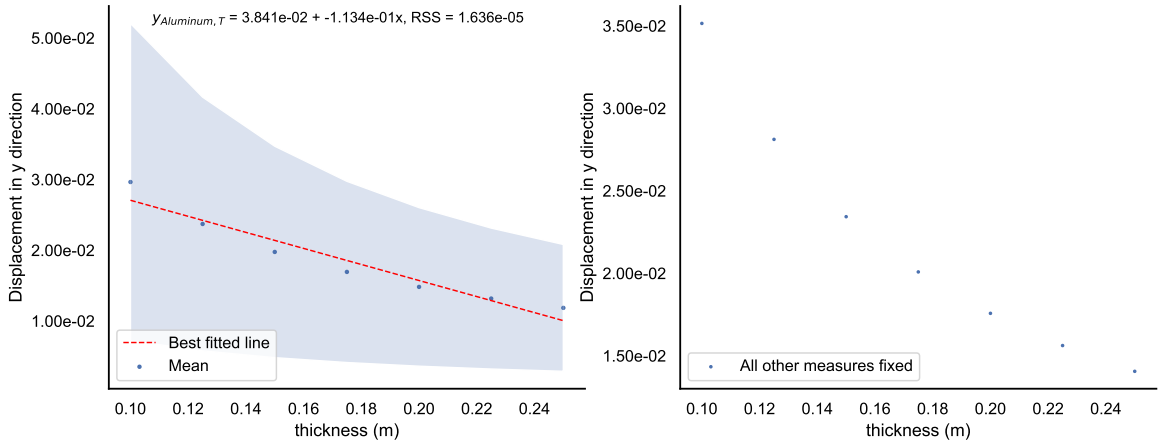
The values of maximum von Mises stress were better in converging in the simulations and were used to build a NN model to predict it. This better behavior comes from the fact that the computation of the von Mises stress probably dilutes the effect of ill maximums in the stresses.



(a) By length

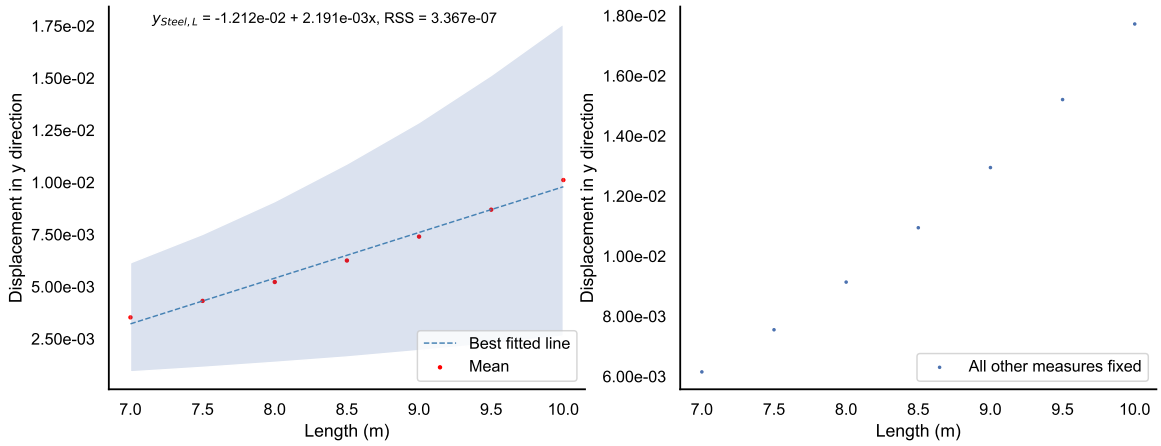


(b) By height

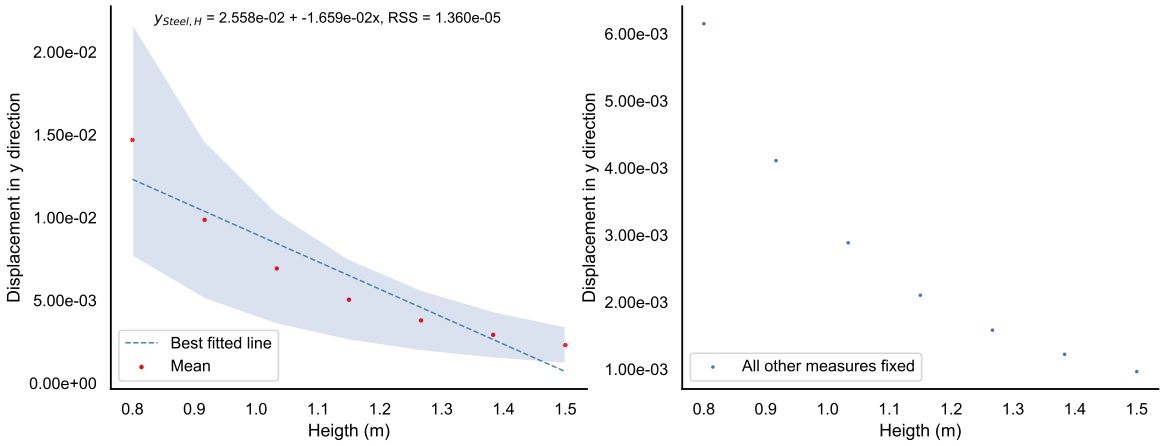


(c) By thickness

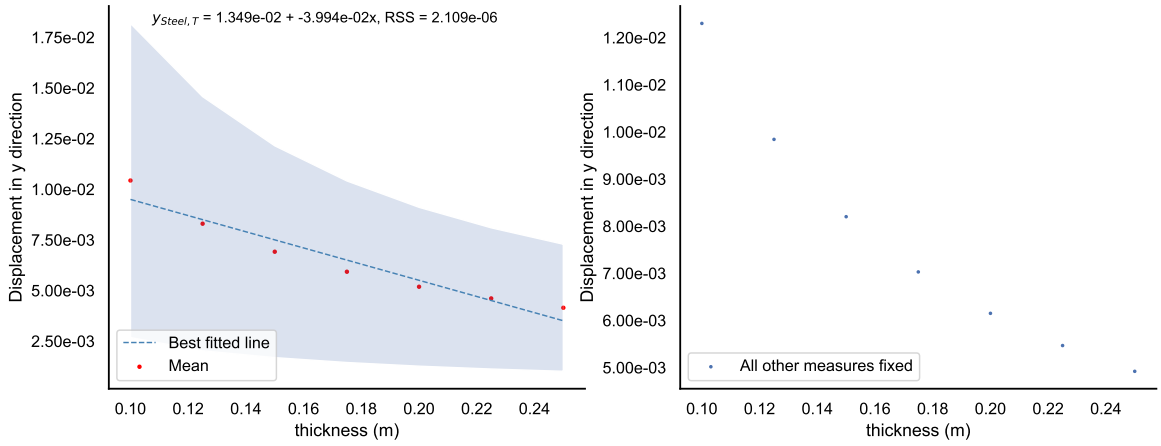
Figure 6.9: Maximum displacement in y direction by geometry dimensions for aluminum



(a) By length



(b) By height



(c) By thickness

Figure 6.10: Maximum displacement in y direction by geometry dimensions for steel

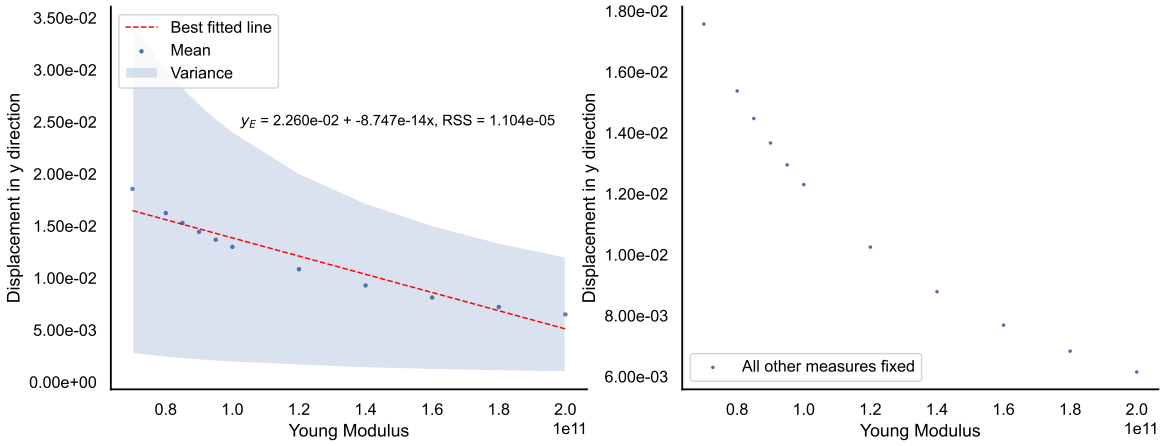
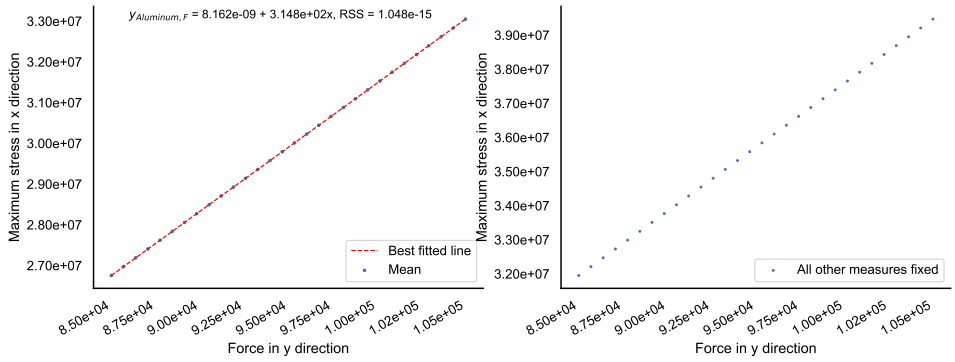
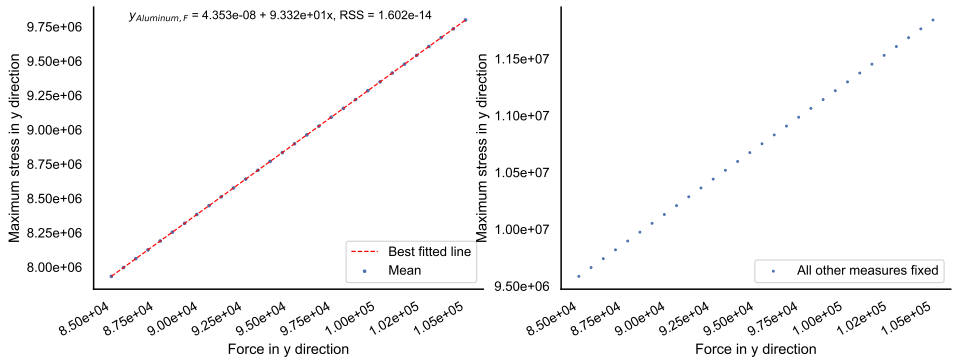


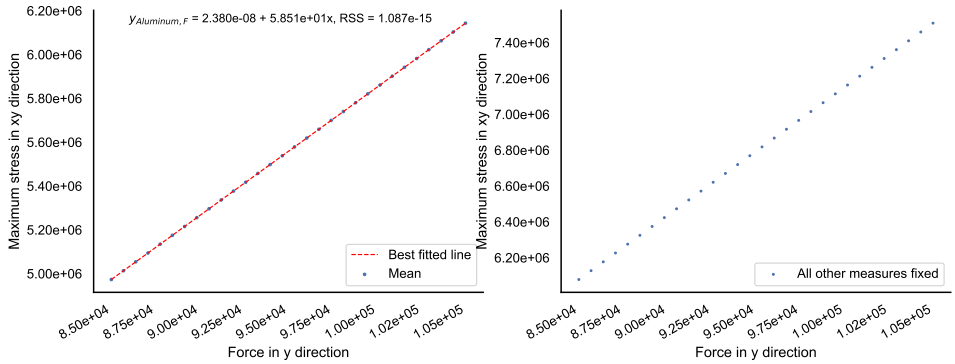
Figure 6.11: Maximum displacement in y-direction by young modulus {fig:meanvard



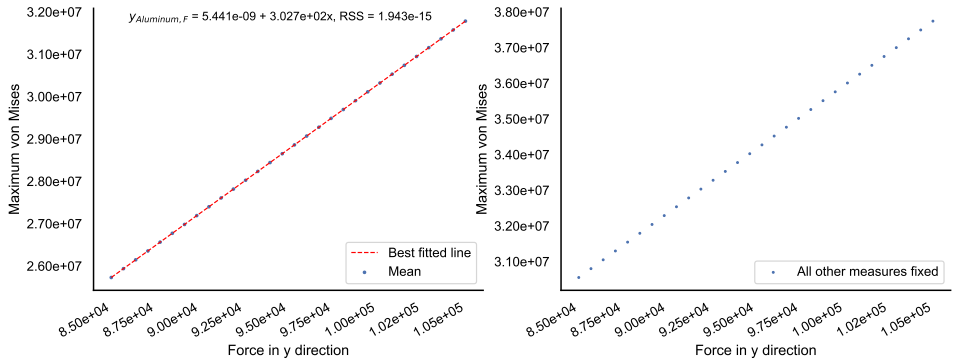
(a) x direction



(b) y direction

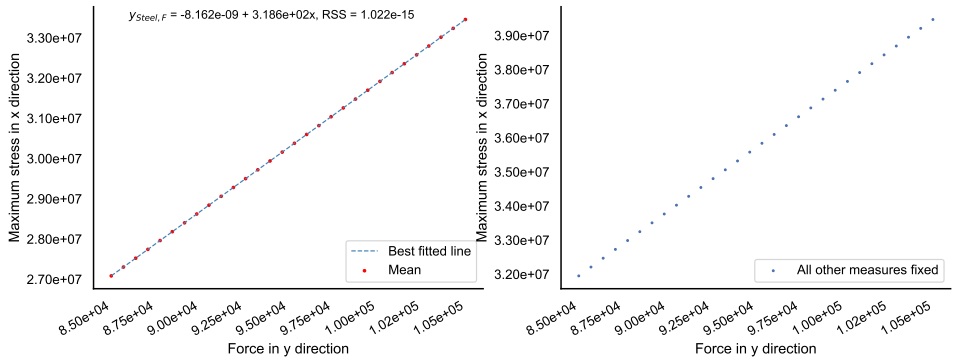


(c) xy direction

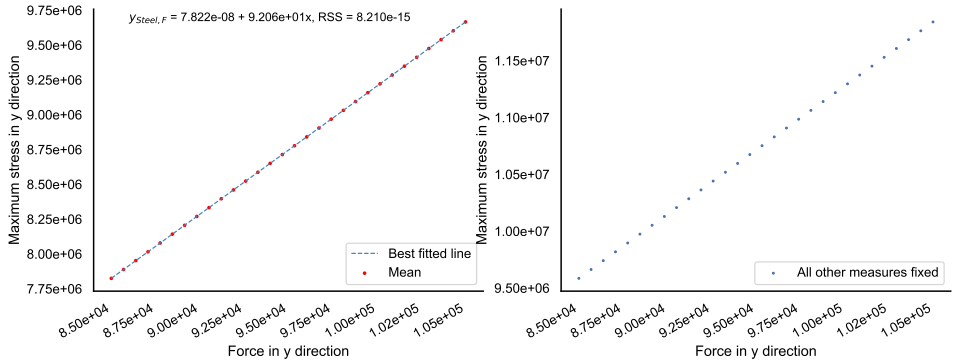


(d) von Mises

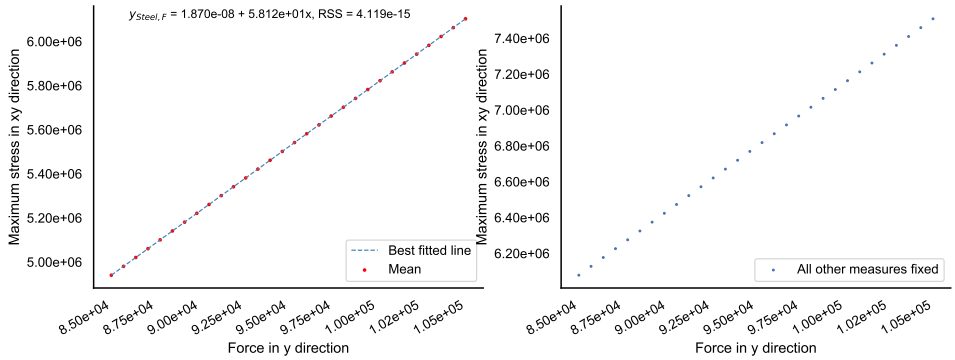
Figure 6.12: Maximum stresses by load magnitude for aluminum



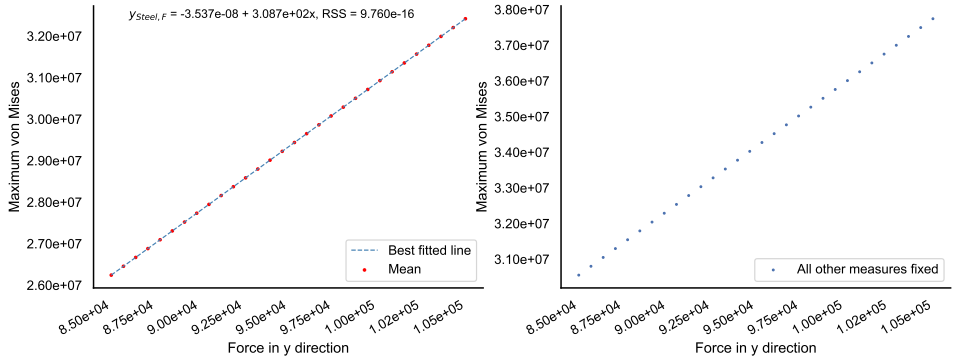
(a) x direction



(b) y direction

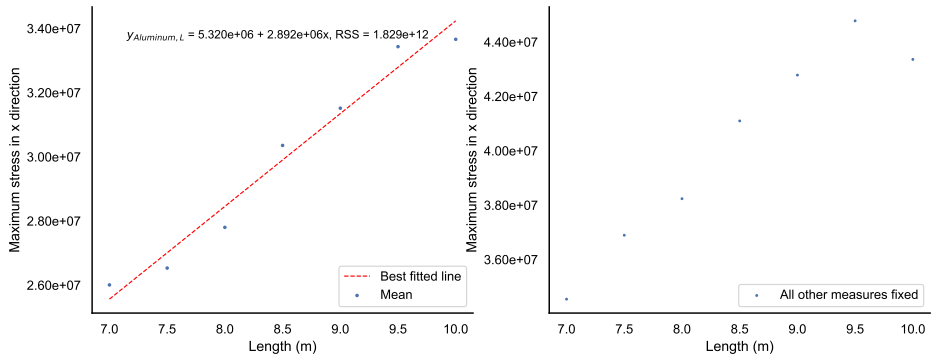


(c) xy direction

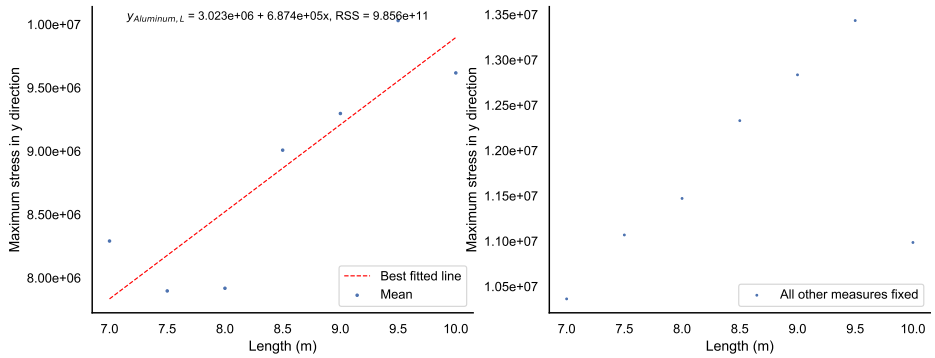


(d) von Mises

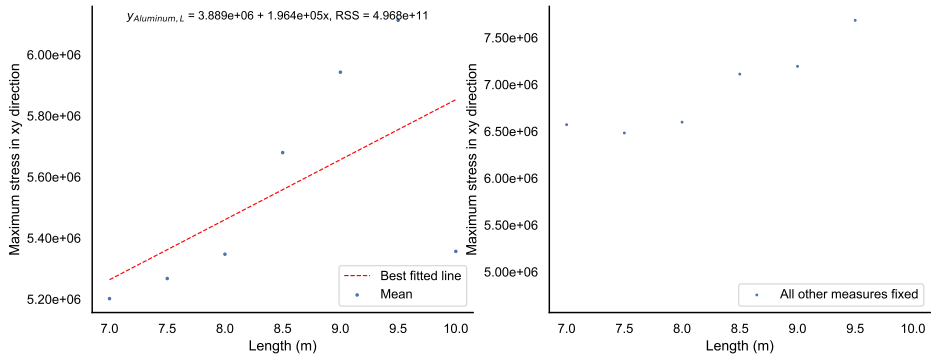
Figure 6.13: Maximum stresses by load magnitude for steel



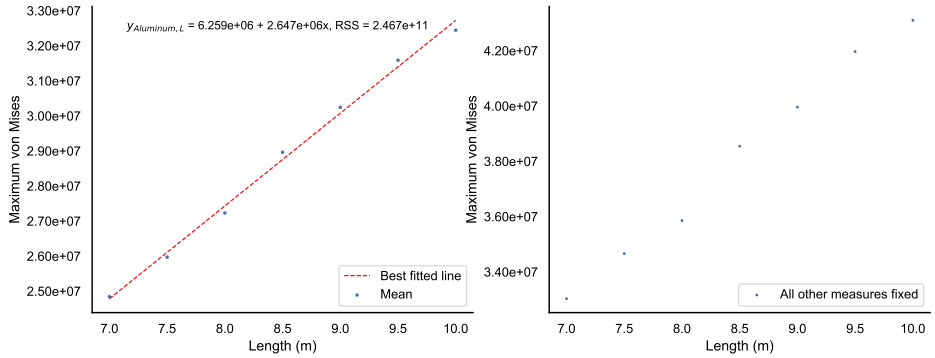
(a) x direction



(b) y direction

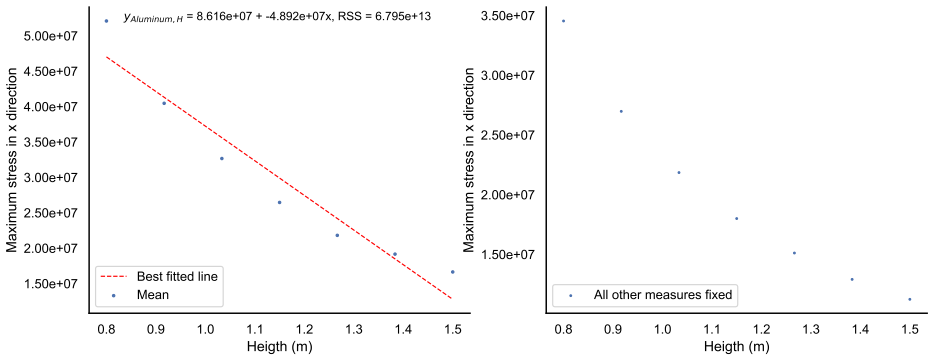


(c) xy direction

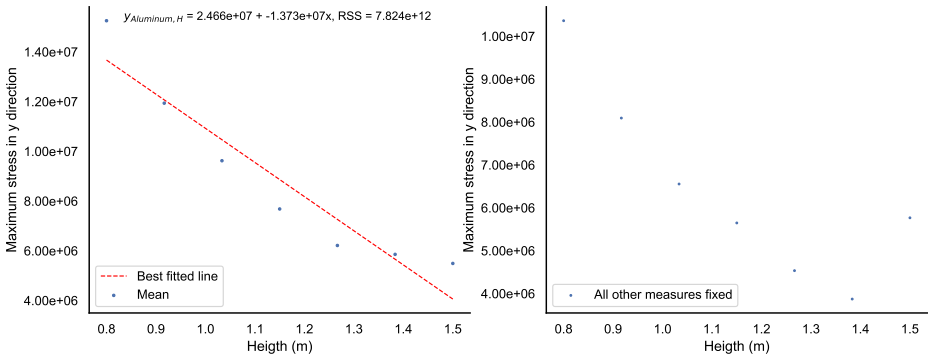


(d) von Mises

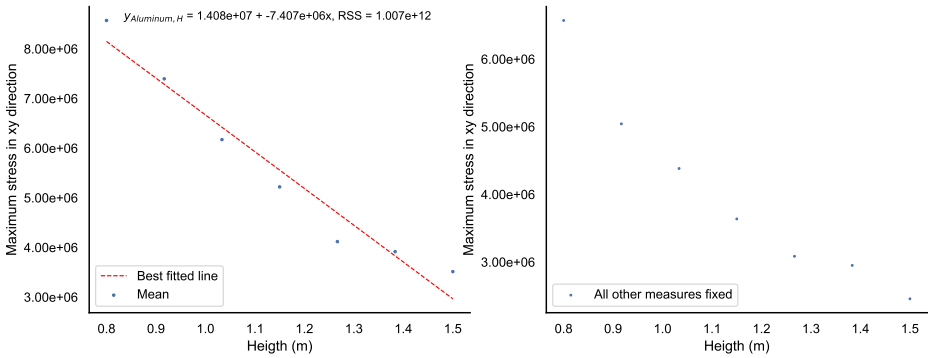
Figure 6.14: Maximum stresses by geometry length for aluminum



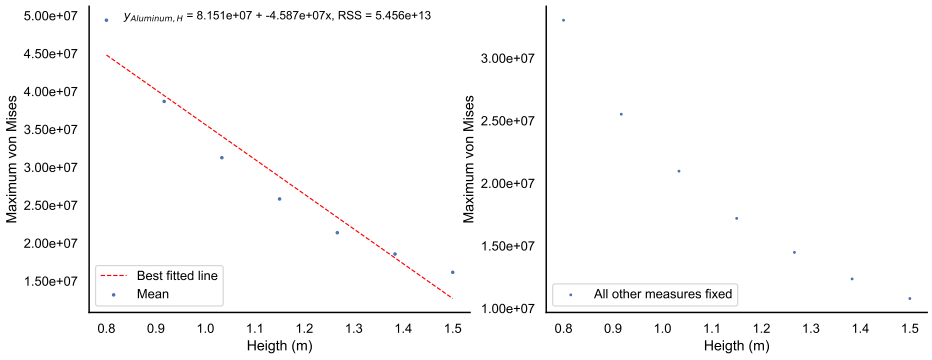
(a) x direction



(b) y direction

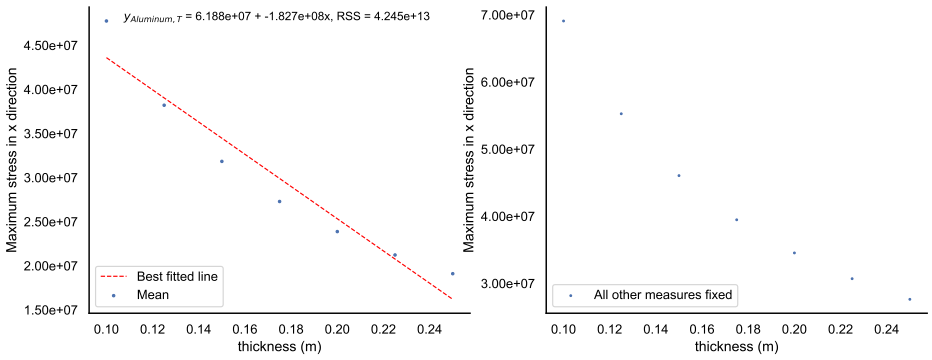


(c) xy direction

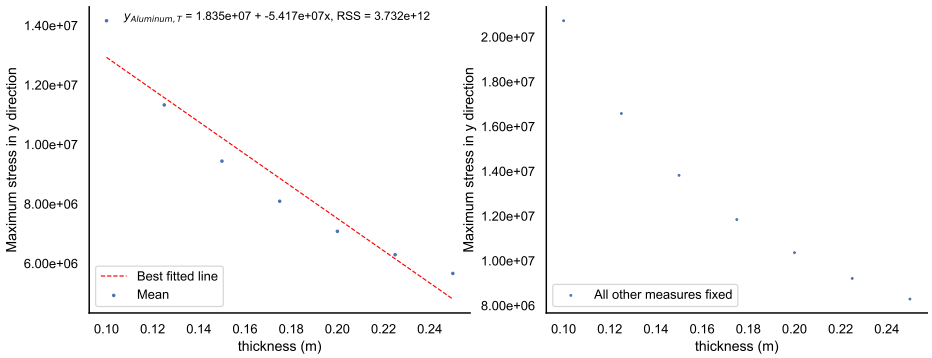


(d) von Mises

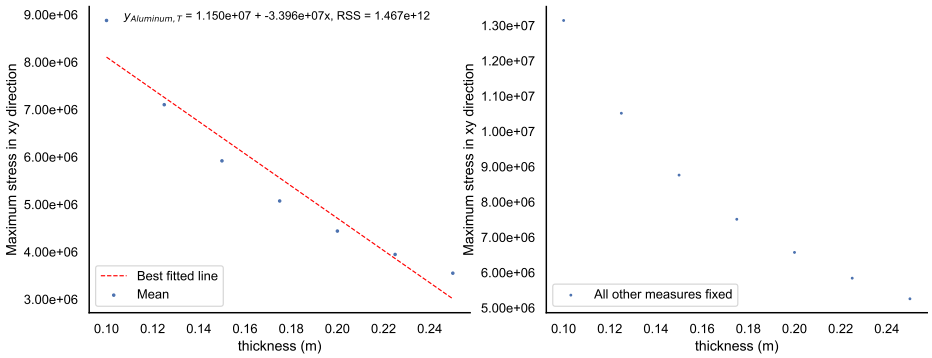
Figure 6.15: Maximum stresses by geometry height for aluminum



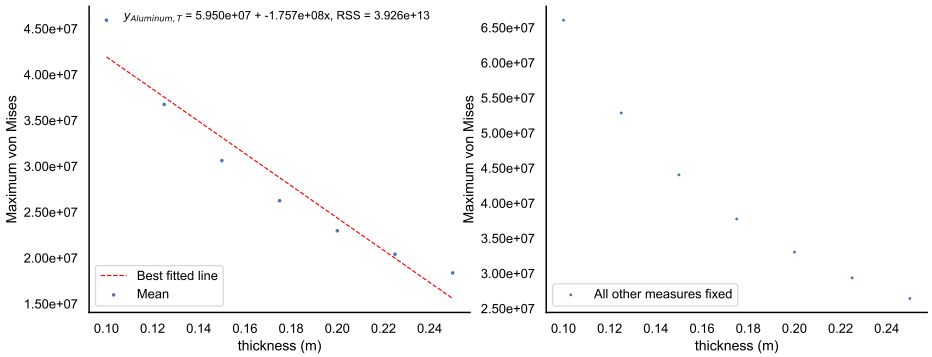
(a) x direction



(b) y direction

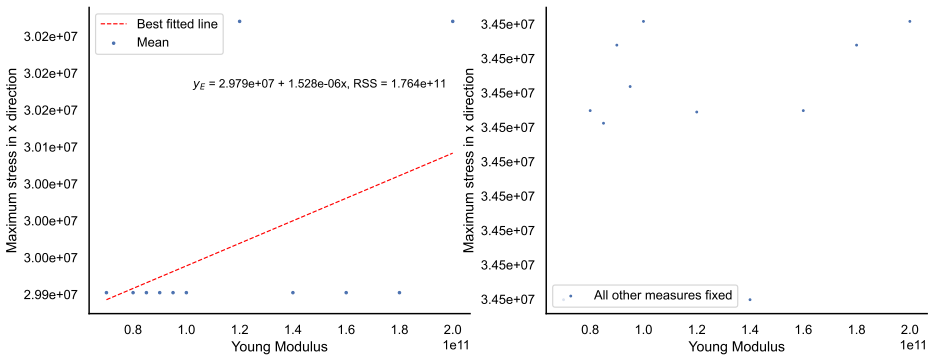


(c) xy direction

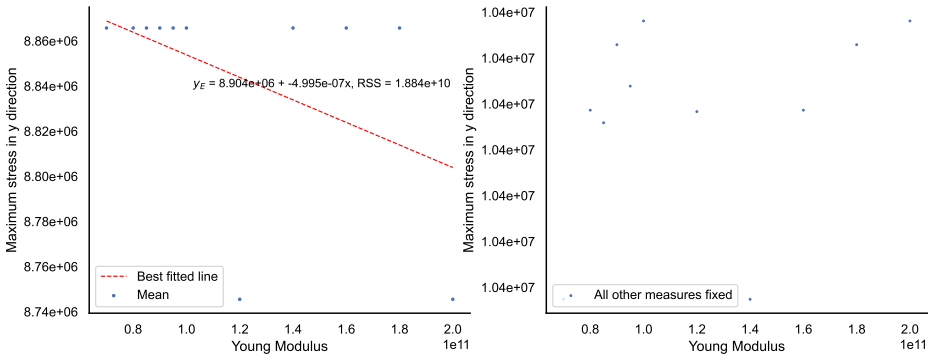


(d) von Mises

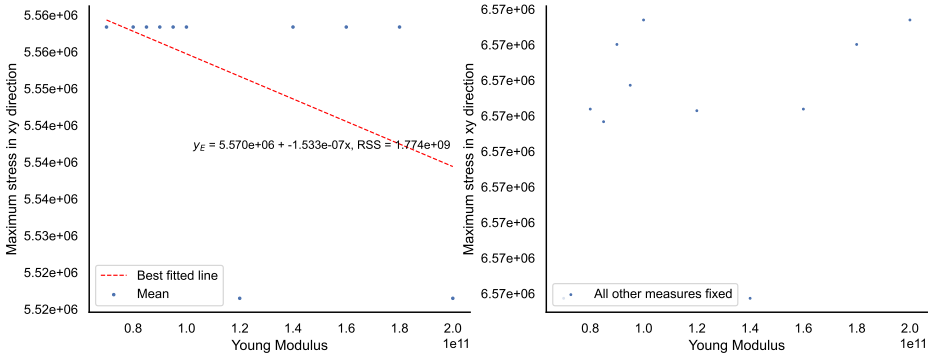
Figure 6.16: Maximum stresses by geometry thickness for aluminum



(a) x direction

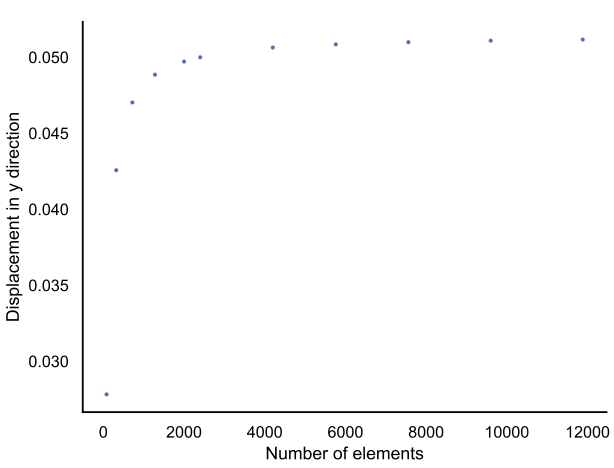


(b) y direction

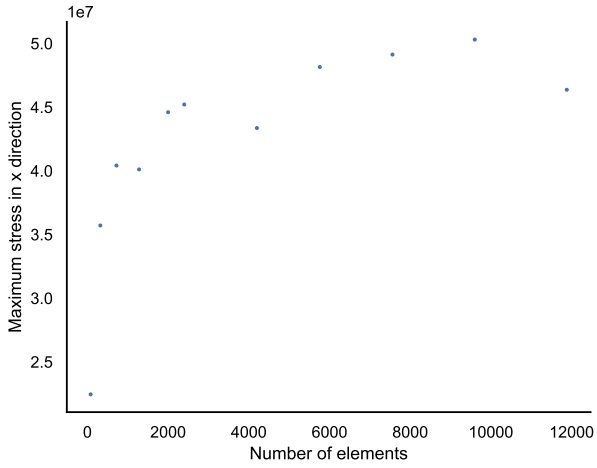


(c) xy direction

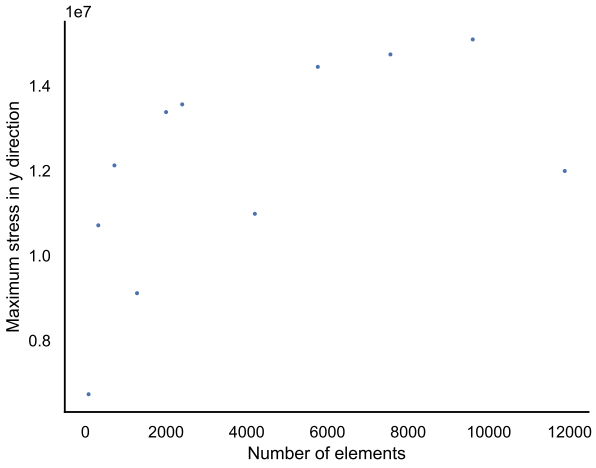
Figure 6.17: Maximum stresses by Young modulus



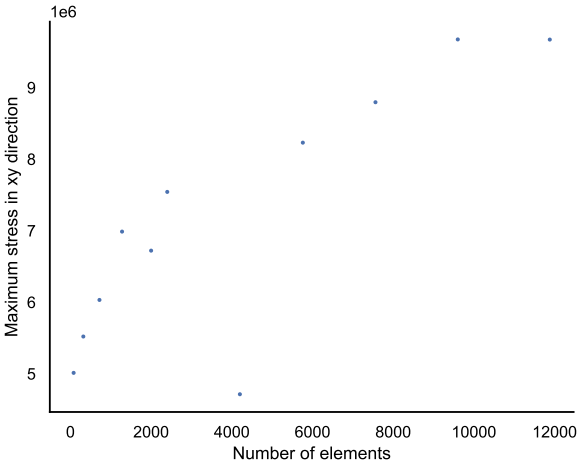
(a) Displacement



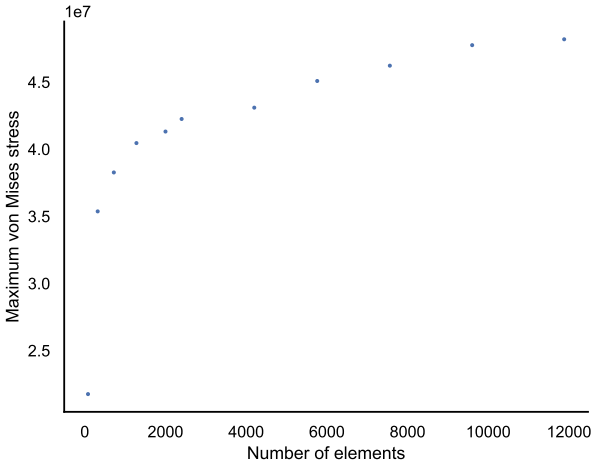
(b) x direction



(c) y direction



(d) xy direction



(e) von Mises

Figure 6.18: Convergence of maximum displacement and stresses by increasing the number of elements

6.4 Trained models

Two different Neural Networks were built and trained to give results as the FEM simulations, the main targets were: to predict the maximum displacement in y-direction and the maximum von Mises stress. The table 6.4 summarizes the outputs for each NN built. These two neural networks could be useful to evaluate the failure criteria of materials and the maximum deflection of the tip without necessarily simulating the structures. Because of the problems with the stress accuracy in x, y and xy directions, it was chosen to build Neural Networks based only on the displacement data and the von Mises stress. Further work needs to be developed to guarantee the quality of stress data, which is out of the scope of this report.

The table 6.5 presents the parameters varied in the grid search to build the NNs. Each NN built was trained and validated five times by cross-validation and the validation mean squared error was stored for each NN and CV group. The learning rate had a start value of 0.01 for the training but was adapted during it reducing it by a factor of 0.1 to the minimum limit of 0.0001. The activation functions were the same for all models and layers, relu was used, and the loss used was the mean squared error (mse). Only one hidden layer was tested for the problems in this work because the relations between the features and targets were apparently simple so just one hidden layer would be sufficient, and dropout was only applied in the hidden layer.

The next sections present the results of the two NNs and how the best model was chosen analyzing the mean and variance of validation error for each candidate model. The final training of the models used all data in the training set and the final error presented for the chosen model was computed using 10 percent of the data that was hold-off in the beginning, so the models never had access to this data neither in the train set nor in the validation set.

6.4.1 Displacement predictions

{sec:NN_displ

The displacement predictions were a regression problem based on the inputs of geometry dimensions (length, height, and thickness), material property (Young modulus), and load magnitude. The grid search was run and the results of the 48 NNs are presented in the boxplot 6.19a for mse and boxplot 6.19b for the coefficient of determination. The plotting for the coefficient of determination only used the models that reached R^2 greater than 0 for at least one CV group.

The boxplot is useful to analyze the generalization properties of each model built, so a model that was good for all cross-validation groups could be chosen instead of one model that achieved the best minimum in just one CV group for example. The two best models for mse (mean squared error) were identified as 26 and 27, but the latter had a better performance in terms of coefficient of determination so it was chosen as the best model. It can be confirmed by the boxplot of the coefficient of determination only for these two models presented in figure 6.20

The final performance of model 27 is presented in figure 6.21, which presents the actual values by the predicted ones for the test set. It's possible to see that the predictions are better for lower values of displacement, probably caused by the higher number of data in those points in the training.

The residuals were analyzed to understand more about the model, figure 6.22a presents the distribution of the residuals, and figure 6.22b presents the residuals by the predicted value. It's possible to see from figure 6.22b that for higher displacement values the errors increased while from the distribution figure 6.22a it's possible to understand that the model systematically predicted lower values for the displacement, but maintained a good general performance with an almost normal residual curve with mean close to zero.

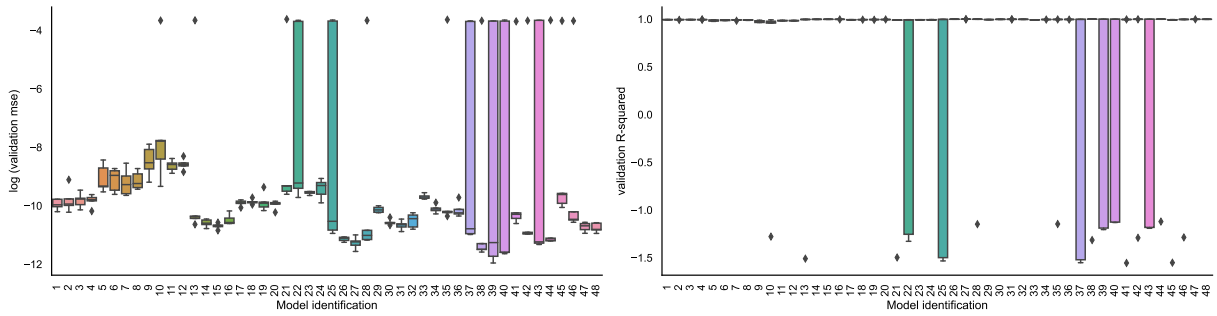
The residual analysis points out that there is room for improvement in the model, one action to enhance the performance could be generating more high displacement examples or reducing the ones from lower displacement in the training data. Without generating more data, it's possible to investigate losses and data

Table 6.4: Main purposes of the Neural Networks built | tab:NN_built

Purpose	Target
Predict maximum nodal displacement	Max(u_y)
Inverse problem: Predict geometry thickness	Max(von Mises)

Table 6.5: Parameters in the grid search {tab:NN_gridsearch

Parameter	Values
Neurons in hidden layer	16, 32, 64, 128
Dropout	0.1, 0.2, 0.3
Batch size	16, 32, 64, 128



(a) Validation mse

(b) Validation coefficient of determination

{sub

{sub

Figure 6.19: Validation error for displacement predictions of candidate NNs fig:validation_error_NN_displa

transformations that could remove this bias of underestimating the displacement from the model.

All these hypotheses to enhance the model are out of the scope of this project and the final model selected is the 27 with 64 neurons, 0.1 dropout, and 64 as batch size. This model has a final mean absolute error of 2.96e−4, the relative error of 2.43% and R^2 of 99.88% in the test data.

6.4.2 von Mises predictions

{sec:NN_vonmi

The von Mises stress predictions were a regression problem based on the inputs of geometry dimensions (length, height, and thickness), material property (Young modulus), and load magnitude. The grid search was run and the results of the 48 NNs are presented in the boxplot 6.23a for mse and boxplot 6.23b for the coefficient of determination.

The same analysis to select the best model was done as in the displacement model. The only good model from the grid search for this problem was the number 2 with 16 neurons, 0.1 dropout and 32 as batch size. It's a smaller Neural Network than the displacement with a lower number of weights to be tuned. One probable explanation for this simpler NN is related to the fact that the stresses are independent of the material Young, which reduces the number of features that the model needs to take into account to predict the target.

The final performance of model 2 is presented in figure 6.24, which presents the actual values by the predicted ones for the test set. The same residuals analysis of the displacement model was performed for the von Mises one and it's presented in figures 6.25a and 6.25b.

The residual analysis of the distribution shows that the model has a normally distributed error, but looking at the residual vs predicted it is possible to see the same challenges from the displacement: underestimation and higher errors for higher values. Some higher errors occur in middle values for von Mises, probably related to the stress convergence issue. This model has a final mean absolute error of 6.72e5, a relative error of 2.33%, and R^2 of 99.50% in the test data.

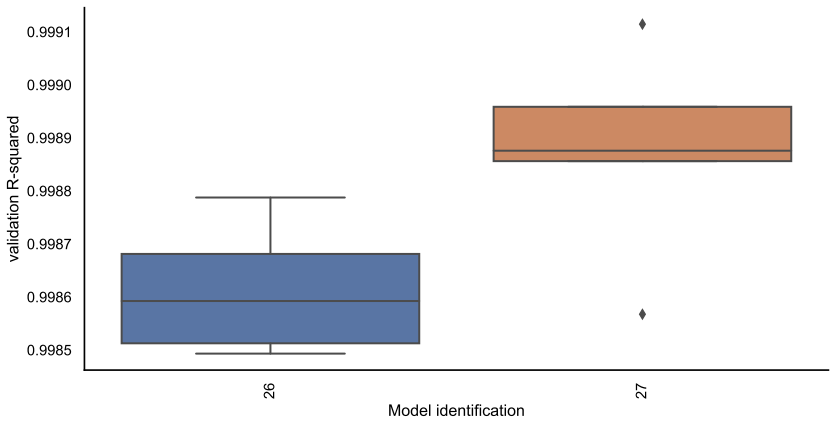


Figure 6.20: Coefficient of determination for the two best models in displacement prediction {fig:coeff_de

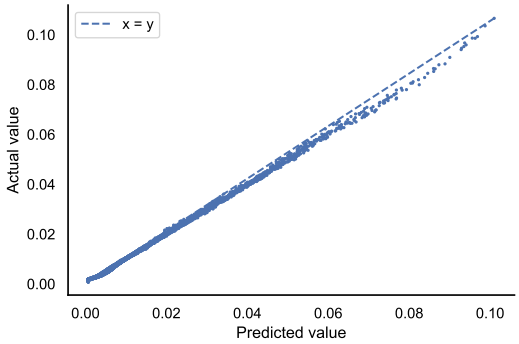
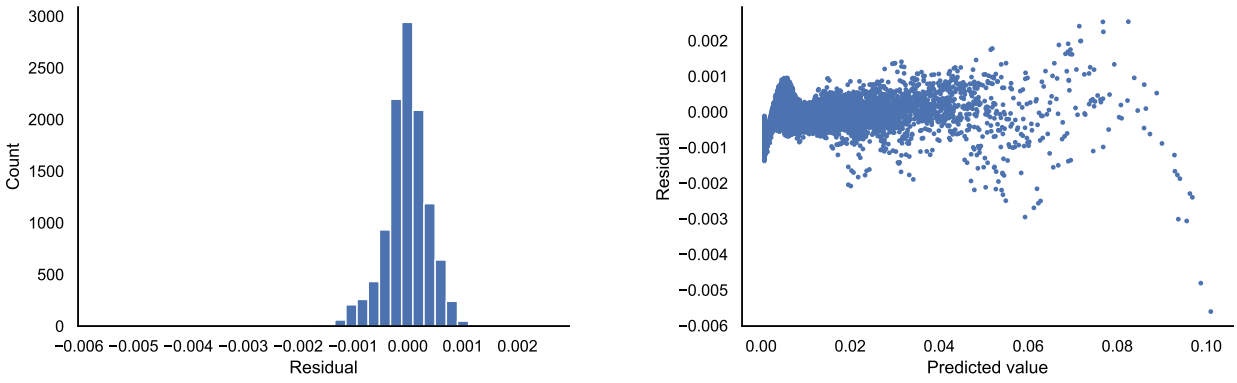


Figure 6.21: Actual vs predict values for displacement {fig:predicti

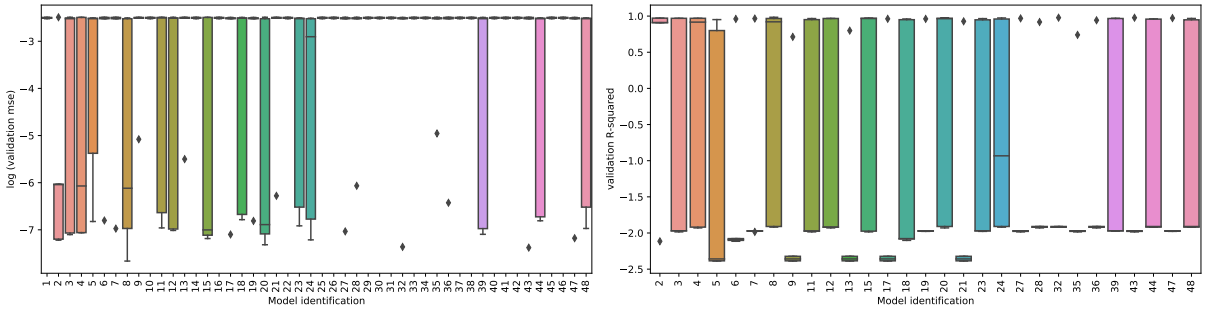


(a) Distribution (b) By predicted values

{fig:residuals_distribution} {fig:residual

fig:residuals_analysis

Figure 6.22: Residuals analysis



(a) Validation mse

(b) Validation coefficient of determination

Figure 6.23: Validation error for von Mises predictions of candidate NNs

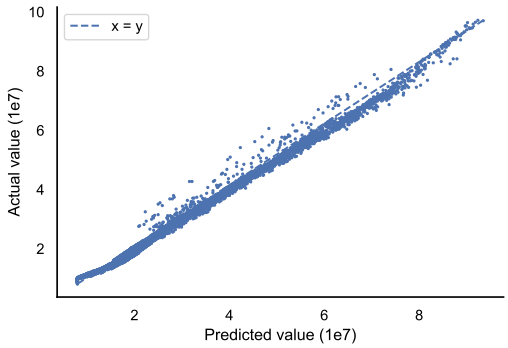
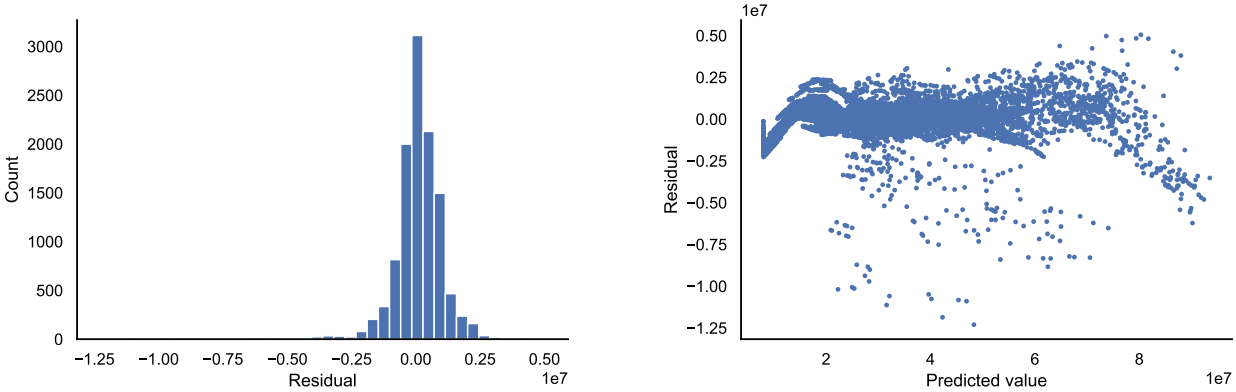


Figure 6.24: Actual vs predict values for von Mises



(a) Distribution

(b) By predicted values

Figure 6.25: Residuals analysis

Conclusion

The final objective of constructing an introductory FEM program to simulate a 2D structure was achieved in this work as well as the construction of the machine learning models using Neural Networks with the generated results by the program. The code was validated using a problem present in the literature, which was a fundamental step to understand if the FEM program was working properly for the problem intended. The final NNs built and chosen for the prediction of the maximum displacement and von Mises stress achieved a good performance, a relative error of 2.43% for the first and 2.33% for the latter. This performance was expected after the analyses of the relations between variables in the FEM simulations, which presented simple relations to the outputs and therefore could be modeled by a NN.

Further developments

This work can be extended with further analysis and more complicated scenarios, for example, a more complicated geometry structure or sets of loads. Different machine learning models could be constructed as well, for example, an inverse problem of defining maximum von Mises stress based on the displacement of points in the material. All codes to generate this work are available in the GitHub of the author, [11], and may be used and altered to suit other applications.

The Neural Networks built in this project can be further analyzed and enhanced, for example, analyze the effect of the batch size, activation function, and losses in the prediction performance. The NNs built were the simplest ones and further methods of regularization, topologies, and losses can be studied to deal with problems in the FEM universe.

Bibliography

[1] Jason Brownlee. *How to use Data Scaling Improve Deep Learning Model Stability and Performance*. https://machinelearningmastery.com/how-to-improve-neural-network-stability-and-modeling-performance-with-data-scaling/. 2019.

[2] John D. Cook. *Don't invert that matrix*. https://www.johndcook.com/blog/2010/01/19/dont-invert-that-matrix. 2010.

[3] Marco Favretti. *Lecture notes: Models of mathematical physics*. 2020.

[4] Robert Hurlston. *What Is Nonlinear Geometry In FEA? And When Should You Use It?* https://www.fidelisfea.com/post/what-is-nonlinear-geometry-in-fea-and-when-should-you-use-it. 2021.

[5] Ethan Irby. *Gradient Descent vs Stochastic GD vs Mini-Batch SGD*. https://medium.com/analytics-vidhya/gradient-descent-vs-stochastic-gd-vs-mini-batch-sgd-fbd3a2cb4ba4. 2021.

[6] Eugenio Oñate. *Structural analysis with the finite element method: linear statistics*. Lecture notes on numerical methods in engineering and sciences. 2010, pp. xxiv + 472. ISBN: 1-4020-8733-0. DOI: https://doi.org/10.1007/978-1-4020-8733-2.

[7] Mario Putti. *Notes on Numerical Methods for Differential Equations*. 2021.

[8] C Rusu. *Simple Mesh Project*. https://github.com/CyprienRusu/Feaforall/blob/master/Simple_Mesh/Simple_mesh.ipynb. 2021.

[9] Shai Shalev-Shwartz and Shai Ben-David. *Understanding Machine Learning - From Theory to Algorithms*. Cambridge University Press, 2014, pp. I–XVI, 1–397. ISBN: 978-1-10-705713-5.

[10] Henrik Sönnerlind. *What Is the Difference Between Plane Stress and Plane Strain?* https://www.comsol.com/blogs/what-is-the-difference-between-plane-stress-and-plane-strain. 2021.

[11] Kenji Urazaki. *FEM2D-Solidmech*. https://github.com/kjrurazaki/FEM2D-Solidmech. 2023.

[12] Wikipedia. *Artificial neural network — Wikipedia, The Free Encyclopedia*. http://en.wikipedia.org/w/index.php?title=Artificial\%20neural\%20network&oldid=1109022706. [Online; accessed 12-September-2022]. 2022.