Lecture Notes: Introduction to the Finite Element Methods

Juan David Gómez Cataño
jgomezc1@eafit.edu.co
Nicolás Guarín-Zapata
nicoguarin@gmail.com

Grupo de Investigación en Mecánica Aplicada
Civil Engineering Department
School of Engineering
Universidad EAFIT
Medellín, Colombia
2019

Contents

Summary

Class Notes for the Course Introduction to the Finite Element Methods

Keywords: Scientific Computing, Computational Mechanics, Finite Element Methods, Numerical Methods.

Chapter 1

Introduction

This set of lecture notes for the course **Introduction to the Finite Element Method** is part of the learning resources used in the flipped classroom[1] version of the subject adopted at Universidad EAFIT. In the flipped class approach, the students cover most of the theoretical aspects through independent home study, while the class time is invested in various learning activities conducted with support from the instructor and strongly based on peer-to-peer interaction. In this context, by collecting and summarizing fundamental numerical, theoretical and computational aspects required in the formulation of finite element algorithms, the lecture notes are intended to serve as a self-study guide. The notes are not as rigorous as material existing in published literature and are certainly not close in quality to excellent textbooks available in the subject.

This introductory version of the course covers fundamental theoretical and computational aspects required in the formulation of finite element methods as a numerical solution technique of boundary value problems. Although the studied algorithms are general the course evolves around the model of the linearized theory of elasticity. The fundamental theoretical framework is mainly covered in the lecture notes and some referenced complementary material. The various theoretical topics are then associated with in-class

[1] "Inverting the classroom means that events that have traditionally taken place inside the classroom now take place outside the classroom and vice versa." [1].

learning activities involving some degree of computational work. Most of these activities are given in the form of Jupyter Notebooks.[2] In addition to the lecture notes and notebooks, the authors have also developed **SolidsPy**[3] which is a complete Python-based finite element code to conduct stress analysis over arbitrary two-dimensional elastic domains. The code, which follows a modular structure facilitates the realization of learning activities in the form of (i) full stress analysis simulations (ii) implementation of intermediate steps aimed at covering the various steps in the FE algorithm and (iii) extensions to incorporate additional kinematic models. In the rest of this introductory chapter, we present a simple problem of a mechanical spring-mass system that resembles the final form of a finite element algorithm and which serves to justify why we cover the material in the specific order proposed here. Then we describe the topics covered in the notes and in the final part we indicate how to use the material.

Introductory problem

A simple discrete system

The simple case of a mechanical spring-mass system considered next resembles most of the algorithmic aspects of a finite element code. This system serves as a nice motivational example since the problem is already discrete thus avoiding the discussion of mathematical complexities. As will be presented throughout the course one of the goals of the finite element algorithm, when applied to a continuous system represented in terms of a boundary value problem, is to reformulate it as a discrete system fully analogous to the simple problem.

The system consists of an assemblage of masses joined by different springs submitted to time varying loads. For later purposes it is convenient to associate a spring with the concept of a finite element and a mass with a nodal

[2] Jupyter notebooks are open-source web applications used to create and share documents that contain live code, equations, visualizations, and narrative text.

[3] Check the documentation in `https://solidspy.readthedocs.io` [2].

point in a finite element algorithm.

The system may be like the one shown in fig. 1.1 where the masses are represented by "cars" connected by springs (or finite elements) of different stiffness coefficients.

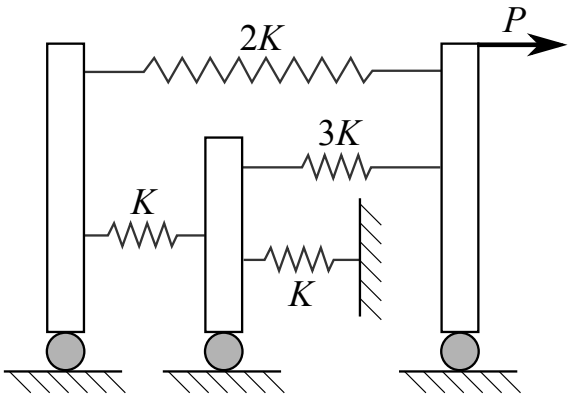


Figure 1.1. Typical assemblage of springs and masses.

Considering a typical spring (or finite element), fig. 1.2

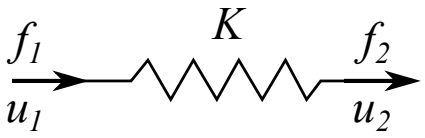


Figure 1.2. Typical spring element.

one finds that under a relative displacement $\delta u = u_1 - u_2$ the spring develops a force :

$$f_1 = K(u_1 - u_2)$$

while equilibrium of the spring requires

$$f_1 + f_2 = 0.$$

The force-displacement and equilibrium equations can be combined into:

$$\begin{Bmatrix} f_1 \\ f_2 \end{Bmatrix} = K \begin{bmatrix} 1.0 & -1.0 \\ -1.0 & 1.0 \end{bmatrix} \begin{Bmatrix} u_1 \\ u_2 \end{Bmatrix}. \tag{1.1}$$

Equation (1.1) resembles a first fundamental aspect displacement based finite element methods which is the fact that the problem is described in terms of displacements of specific points (or nodes) and where internal element forces are found from these nodal displacements via a constitutive relationship. In the case of a continuous problem an analogous force-displacement relationship is established using interpolation theory together with equilibrium statements.

On the other hand, the equilibrium equation for a typical mass m_j with displacement u_j (see fig. 1.3) and assumed to be attached to springs i and $i+1$ reads

$$f_2^i + f_1^{i+1} + m_j \frac{\mathrm{d}V_j}{\mathrm{d}t} = P_j. \tag{1.2}$$

This equation can be written in terms of displacements after expressing the involved forces f_2^i and f_1^{i+1} using terms from equations like 1.1

$$(K^i + K^{i+1})u_j - K^i u_{j-1} - K^{i+1} u_{j+1} + m_j \frac{\mathrm{d}V_j}{\mathrm{d}t} = P_j.$$

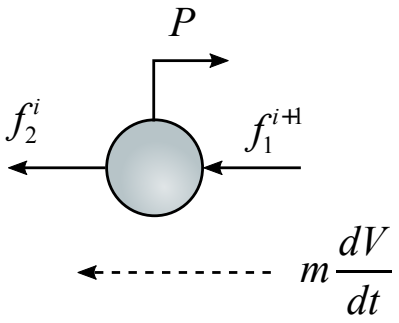


Figure 1.3. Free body diagram for a typical mass connected to springs i and $i+1$.

Writing the elemental equilibrium equations for the springs in terms of the corresponding mass displacements u_{j-1}, u_j and u_{j+1} and generalizing the coefficients notation we get:

$$\left\{ \begin{array}{c} f_1^i \\ f_2^i \end{array} \right\} = \left[\begin{array}{cc} k_{11}^i & k_{12}^i \\ k_{21}^i & k_{22}^i \end{array} \right] \left\{ \begin{array}{c} u_{j-1} \\ u_j \end{array} \right\}$$

and

$$\left\{ \begin{array}{c} f_1^{i+1} \\ f_2^{i+1} \end{array} \right\} = \left[\begin{array}{cc} k_{11}^{i+1} & k_{12}^{i+1} \\ k_{21}^{i+1} & k_{22}^{i+1} \end{array} \right] \left\{ \begin{array}{c} u_j \\ u_{j+1} \end{array} \right\}$$

which gives for the equilibrium equation of the m_j mass:

$$k_{21}^i u_{j-1} + (k_{22}^i + k_{11}^{i+1}) u_j + k_{12}^{i+1} u_{j+1} + m_j \frac{dV_j}{dt} = P_j.$$

If one the other hand we also consider the contributions from the springs K^i and K^{i+1} to the equilibrium of masses m_{j-1} and m_{j+1} respectively we have the following matrix block:

$$\left[\begin{array}{ccc} & & \\ k_{11}^i & k_{12}^i & \\ k_{21}^i & k_{22}^i + k_{11}^{i+1} & k_{12}^{i+1} \\ & k_{21}^{i+1} & k_{22}^{i+1} \end{array} \right].$$

Consideration of the complete system of masses leads to a system of linear equations of the general form

Considering now the complete system of masses and springs leads to a system of linear equations of the form

$$[K_G] \{U_G\} + [M] \{A_G\} = \{F_G\}. \tag{1.3}$$

where each equation represents the equilibrium of a given mass.

The process of forming these global coefficient matrices by adding the contribution from different elements (springs) to the equilibrium equations of the different masses is known as element assembly and this can be achieved in a systematic way by establishing the connection between the global and local degrees of freedom. This can be accomplished through an operator storing in each row the global degrees of freedom corresponding to each element. For instance, fig. 1.4 shows elements K^i and K^{i+1} and the global degrees of freedom corresponding to masses m_{j-1}, m_j and m_{j+1}. The corresponding entries of the DME operator for these elements are given by:

$$DME = \begin{bmatrix} j-1 & j \\ j & j+1 \end{bmatrix}$$

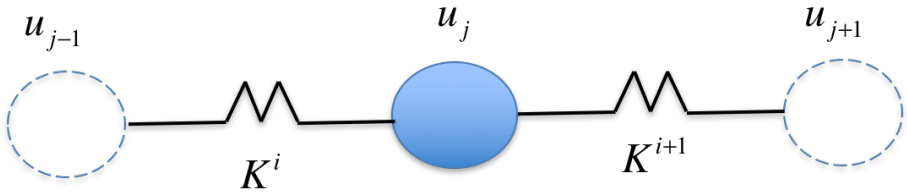


Figure 1.4. Global degrees of freedom connected to the spring elements K^i and K^{i+1} respectively.

and the assembly process from the contribution of these elements to the global coefficient matrix for elements i and $i+1$ proceeds as:

$$\begin{aligned} K_{j-1,j-1} &\leftarrow K_{j-1,j-1} + k^i_{11} \\ K_{j-1,j} &\leftarrow K_{j-1,j} + k^i_{12} \\ K_{j,j-1} &\leftarrow K_{j,j-1} + k^i_{21} \\ K_{j,j} &\leftarrow K_{j,j} + k^i_{22} \end{aligned}$$

and

$$
\begin{aligned}
K_{j,j} &\leftarrow K_{j,j} + k_{11}^{i+1} \\
K_{j,j+1} &\leftarrow K_{j,j+1} + k_{12}^{i+1} \\
K_{j+1,j} &\leftarrow K_{j+1,j} + k_{21}^{i+1} \\
K_{j+1,j+1} &\leftarrow K_{j+1,j+1} + k_{22}^{i+1}
\end{aligned}
$$

The system given by eq. (1.3) and assembled with the aid of the DME operator can be solved for the global displacements U_G and later use these displacements to compute the element forces.

This algorithmic strategy of assembling the contribution from all the elements to formulate the global equilibrium equations of the system is typical of finite element methods. However in the case of a BVP the continuous system must be converted first into a discrete system through different numerical and mathematical methods. A broad description of the method is described in algorithm 1.

Algorithm 1: Springs Algorithm.

Data: Problem parameters; NUMNP, NUMEL, NMATP
Result: Displacements and spring forces
Create DME operator;
Assemble K^G, F^G;
while $j \leq 1, NUMEL$ **do**
  $K^G \leftarrow K^G + K^i$
  $F^G \leftarrow F^G + F^i$
end
Solve $[K^G]U = F^G$
Find internal forces

The full implementation of the mass-springs system is given in an accompanying notebook in the course REPO.

Contents of the course

The course is divided in three parts as follows:

- Part 1 covers classical numerical methods such as interpolation theory and numerical integration within the context of finite element methods. These methods are shown to be fundamental in the conversion of the continuous system into a discrete system analogous to the discussed mass-springs system. The numerical methods are studied from its theoretical and computational aspects and the suggested learning activities are described in notebooks 1 through 6.
- Part 2 presents the boundary value problem corresponding to the model of the linearized theory of elasticity and in particular its description through formulations which are suitable for a solution in terms of finite elements algorithms. The boundary value problem is covered in notebook 7.
- Part 3 concentrates on the formulation of the elasticity boundary value problem in a finite element algorithm. The algorithm and several related aspects are covered in notebooks 8 through 11, while notebook 12 contains a brief reference to the course finite element code SolidsPy.

The main set of accompanying notebooks also includes NB-0 which makes an introduction to the use of notebooks and a quick reference to data flow structures in Python. The set of lecture notes also include an appendix section presenting some more mathematical aspects of the method and additional tools that may result useful depending on the student's abilities.

How to follow the course?[4] The course and its different resources have been created to be used in a flipped class environment or as a self study

[4]This set of lecture notes and additional course material has been developed as part of the sabatical period of the first author and with the collaboration of the second author Nicolas Guarin-Zapata. The development of the notebooks and design of the learning activities in these notebooks has been developed thanks to the advisory of Camilo Vieira.

material. In the formal course, as thought in the graduate program at Universidad EAFIT, it is recommended to follow the proposed sequence of activities starting with the section covering numerical methods as described previously.This same sequence is also recommended for independent learners with no previous knowledge on fundamental numerical methods like interpolation theory and numerical integration. More advanced students, with previous backgrounds on mathematical analysis and numerical methods can test their actual abilities by developing the activities proposed on NB-4 and NB-6 and then moving directly into Chapter 4 discussing the boundary value problem.

Chapter 2

Interpolation in the Finite Element Method

Preliminary

In a broad sense the finite element method is nothing else than an approximation technique for solutions to boundary and initial value problems. In this section we will discuss some fundamental and basic aspects related to the approximation of functions through interpolation. In the finite element method, such approximation takes place for the geometry of the computational domain and for the field variable of the problem at hand. As it will be seen, the concept of finite element it self corresponds to a local domain or sub-domain in which the solution function is approximated via interpolation techniques. We will cover this subject from a materialistic point of view, as required on the implementation of a first finite element algorithm, with mathematical rigor left to the excellent texts on the subject. The set of notes starts with the problem definition and its solution in terms of the Lagrange interpolation theorem. From that point the notes discuss practical applications including its implementation in Python scripts built for finite element analysis.

At the end of this chapter[1] the student should be able to:

- Formulate appropriate interpolation schemes to generate approximate functions out of sets of discrete values representing the behavior of the unknown functions over one-dimensional and two-dimensional domains.
- Recognize the differences in terms of advantages and limitations between global and local interpolation schemes. This includes the pathologies associated with each method.
- Recognize the concept **finite element** like a family of locally defined interpolation schemes valid over canonical domains.
- Develop efficient and effective Python implementations of interpolation schemes within the context of the finite element method for one-dimensional and two-dimensional applications.

2.1 Statement of the problem

Let $f(x)$ be an unknown function, whose values, however, are known at n discrete points $x_1, x_2, ..., x_n$. We want to know (interpolate) the value of $f(x)$ at an arbitrary point $x \in [x_1, x_n]$ and different to one of the n points.

The problem of interpolation is precisely that of finding the unknown value of $f(x)$ using the known values $\{f^1, f^2, ..., f^n\}$. As schematically described in fig. 2.1 it involves two steps:

i Fitting an approximate function, known as the interpolating polynomial, to the known data points.

ii Evaluating the function at the point of interest where the function is unknown.

[1]This chapter, together with theoretical and computational learning activities is complemented by Jupyter Notebooks 1 through 4 available at the course's repository.

We can (i) follow a global approach using all the known n-data points and fit an $(n-1)$-th order polynomial to these data points[2](fig. 2.1) or (ii) use a local approach where one splits the domain into sub-intervals and fits lower order polynomials to the data points within each sub-interval(fig. 2.2).

Local interpolation uses a finite number of nearest-neighbors and generates interpolated versions of $f(x)$ that do not in general have continuous first or higher derivatives. The advantage of this local approach lies in the fact that independent of the function or the number of data points, the interpolation operation always uses the same polynomial. For instance in fig. 2.2 only first-order polynomials are being used in each local sub-domain: as a result a computer implementation of this scheme would only need to store the local polynomial once. In the jargon of the finite element method the interpolation functions or its resulting local polynomials are termed a finite element.

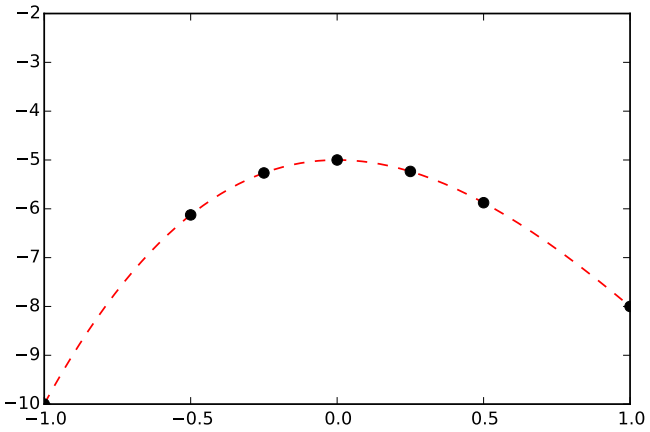


Figure 2.1. The black points represent known values of an otherwise unknown function. The dashed line represents an approximation to the unknown function in terms of a polyonmial of order $n-1$.

[2]As this approach is problem-dependent it is also cumbersome and difficult to code and therefore not amenable to be used in finite elements.

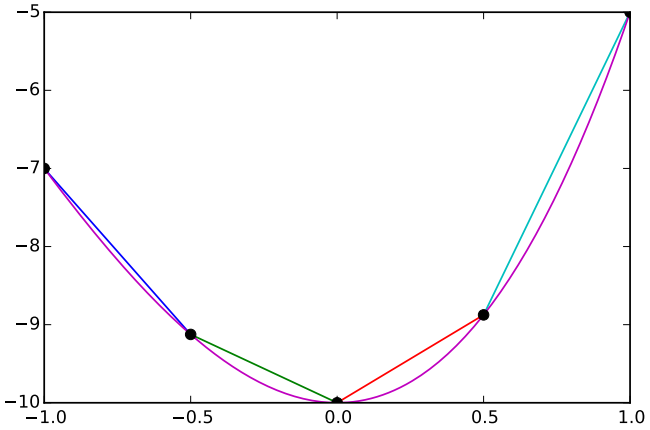


Figure 2.2. Interpolation takes place inside each sub-interval producing a piecewise interpolation approximation to the unknown function

2.2 One-dimensional scalar functions

2.2.1 Lagrange interpolation theorem

Given a set of n-points $\{(x^1, y^1), \cdots, (x^n, y^n)\}$ where $y^n \equiv f(x^n)$ then: "there exists a unique polynomial $p(x)$ of order at most $(n-1)$ such $p(x^I) = f(x^I)$ for $I = 1, 2, \cdots, n$". The polynomial is given by;

$$p(x) = L^1(x)f(x^1) + L^2(x)f(x^2) + \ldots + L^n(x)f(x^n) \tag{2.1}$$

and the term $L^I(x)$ is computed as;

$$L^I(x) = \prod_{\substack{J=1 \\ I \neq J}}^{n} \frac{(x - x^J)}{(x^I - x^J)}. \tag{2.2}$$

The approximate function $p(x)$ of order $n-1$ is termed **the interpolating polynomial** such

$$f(x) \simeq p(x)$$

while each one of the terms $L^I(x)$, also of order $n-1$, are termed **the interpolation functions**. In the context of the finite element method these are also called **shape functions**.

The use of index notation, and particularly the summation convention, can be extended to represent the linear superposition of functions given by eq. (2.1). We use capital super-scripts to denote interpolation in such a way that a capital superscript denotes a data point in the interpolation scheme. Accordingly eq. (2.1) can be equivalently written like:

$$p(x^I) = L^I(x) f(x^I) \tag{2.3}$$

where the fact that $I = 1, 2, ..., n$ is implicit in the notation.

Recall that the interpolating polynomial $p(x)$ is just an approximation to the actual function $f(x)$. However, the approximation should at least be such that $p(x^I) = f(x^I)$ at the n nodal points. To satisfy this condition the interpolation polynomials must be such that:

$$L^I(x^J) = \delta^{IJ}$$

and where δ^{IJ} is the delta function extended to the interpolation polynomials.

The approximate function $p(x)$ resulting from the interpolation process is called the **interpolating polynomial** while the Lagrange polynomials $L^I(x)$ are called interpolation functions. In the language of the finite element methods these are simply called **shape functions**.

A note regarding superscripts in indicial notation: In this Class Notes superscripts associated to symbols like in the expression x^4 are frequently used to describe a variable associated to a nodal point: for instance, in this context the expression x^4 represents the x-coordinate of nodal point 4. However, in some other cases this same expression might appear, for example, in the definition of a function like in $f(x) = x^4 + 4x^3$. In both cases the specific meaning should be clear by the context in which it appears.
Python function **LagrangPoly()** in NB-1 generates Lagrange interpolation polynomials of different order and over a varying range.

Example: Interpolation of a function using 3 data points Use a Lagrange interpolation scheme to find an interpolating polynomial that approximates the function

$$f(x) = x^3 + 4x^2 - 10$$

at the sampling (or also nodal) points $x^1 = -1.0$, $x^2 = +1.0$ and $x^3 = 0.0$ over the interval $[-1, 1]$ together with its first order derivative at $x = 0.7$. Assume that the values of the first derivative at the nodal points are unknown.

Table 3.1 contains the exact values for the function

x	$f(x)$
-1.0	-7.000
0.00	-10.00
1.00	-5.000

Table 2.1. Known values of the function $f(x) = x^3 + 4x^2 - 10$ over the interval $[-1, 1]$

The formulation of the interpolation scheme consists in finding the interpolation polynomials $L^I(x)$ and the interpolating function $p(x)$. From

eq. (2.2) the interpolation functions, shown in fig. 2.3, are:

$$L^1(x) = \frac{(x-x^2)(x-x^3)}{(x^1-x^2)(x^1-x^3)} \equiv -\frac{1}{2}(1-x)x$$
$$L^2(x) = \frac{(x-x^1)(x-x^3)}{(x^2-x^1)(x^2-x^3)} \equiv +\frac{1}{2}(1+x)x$$
$$L^3(x) = \frac{(x-x^1)(x-x^2)}{(x^3-x^1)(x^3-x^2)} \equiv +(1-x^2).$$

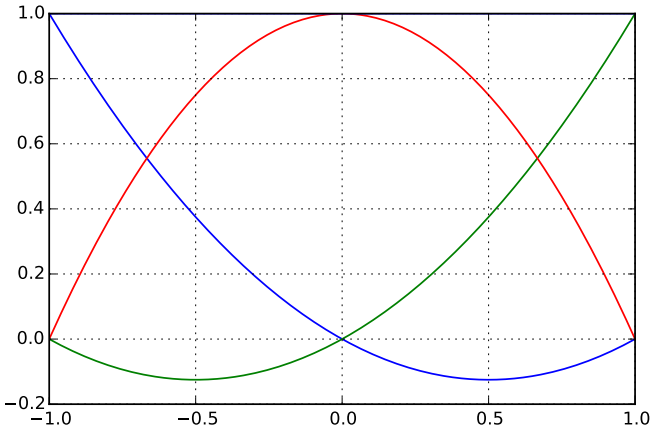


Figure 2.3. (top) Interpolation polynomials $L^1(x)$, $L^2(x)$ and $L^3(x)$ as per eq. (2.2) computed for a second order scheme with 3 nodal points.

The interpolating polynomial approximating the function is obtained using eq. (2.3) which in this case is given by

$$p(x) = 10x^2 + \frac{7}{2}(1-x)x - \frac{5}{2}(1+x)x - 10.$$

The approximate and actual function are compared in fig. 2.4. Clearly $p(x)$, being a second order function differs with $f(x)$, which is a third order function. However, as stated in the interpolation theorem both functions

coincide at the nodal points where the known values of the function are available.

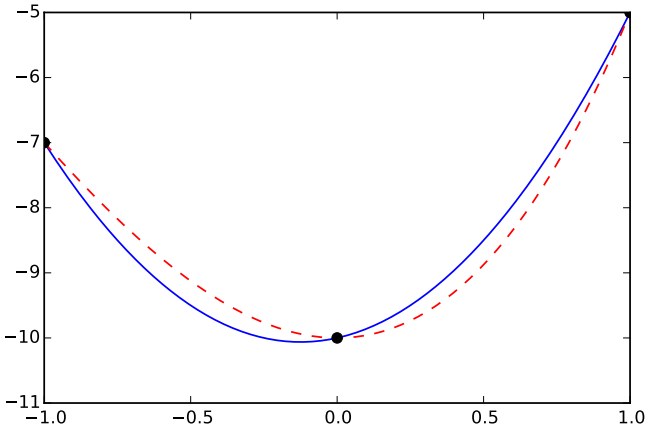


Figure 2.4. Resulting interpolating function $p(x)$ as per eq. (2.3) compared with the exact function $f(x) = x^3 + 4x^2 - 10$. The Python code used for the interpolation is available in NB-1

In finding an approximations to the first derivative recall that at the nodal points the values of these first derivatives are unknown. Thus the only available choice is to operate directly on $p(x)$ and use it to approximate also the first derivative like:

$$\frac{dp(x)}{dx} = \frac{dL^1(x)}{dx} f^1 + \frac{dL^2(x)}{dx}(x) f^2 + \frac{dL^3(x)}{dx} f^3 \tag{2.4}$$

It is evident that the approximation takes the form of a lineal superposition of products between interpolation functions (which in this case are derivatives of the L^I) and known values of the function. The first derivative obtained in this form differs from the actual derivative of $f(x)$ since this approach uses the values of f and not those of $\frac{df(x)}{dx}$ and $\frac{dL^I(x)}{dx}$ instead of L^I and these functions do not satisfy the condition $\frac{dL^I(x^J)}{dx} = \delta^{IJ}$.

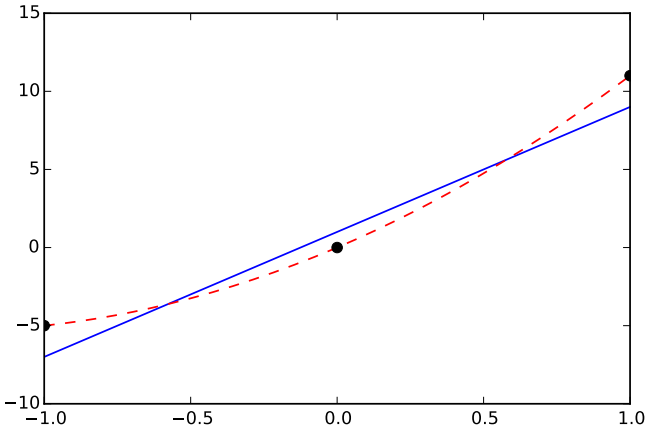


Figure 2.5. Comparison between the first order derivative of $p(x)$ with the closed-form result of $\frac{df(x)}{dx}$

Figure 2.6 shows the approximation to $f'(x)$ when $p(x)$ is computed using 4th-order polynomials.

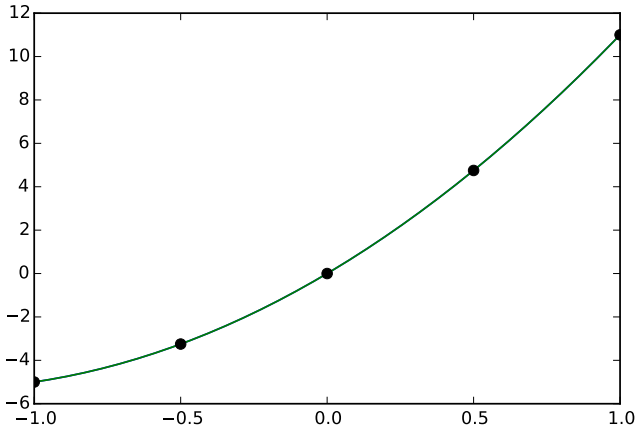
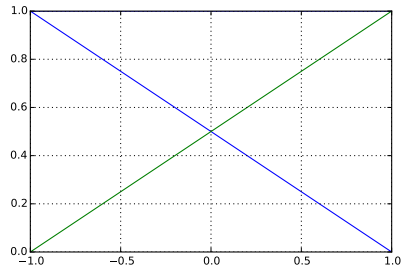
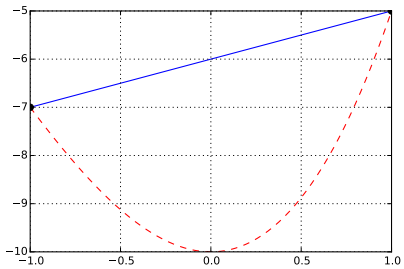


Figure 2.6. Comparison between the first order derivative of $p(x)$ computed using 4th-order polynomials with the closed-form result of $\frac{df(x)}{dx}$

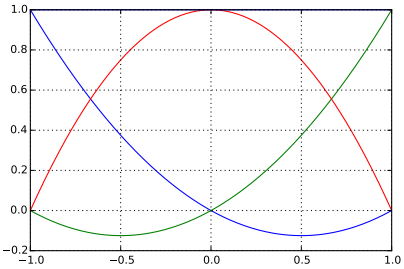
To identify the variation in the solution with different interpolation schemes fig. 2.7 compares the exact and interpolated solution for polynomials of order 1, 2 and 4 respectively. The left column displays the interpolation polynomials $L^I(x)$.



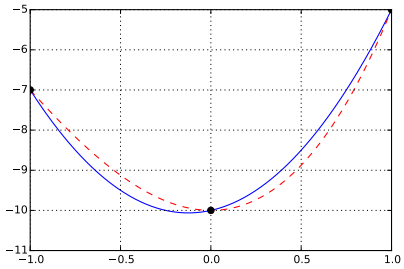
(a) First order.



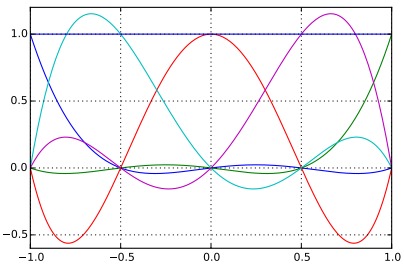
(b) Actual and interpolated function.



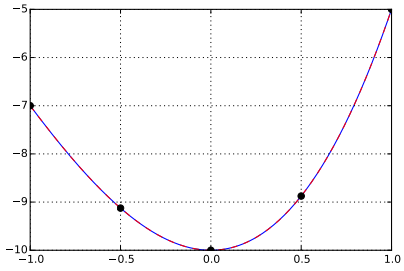
(c) Second order.



(d) Actual and interpolated function.



(e) Fourth order.



(f) Actual and interpolated function.

Figure 2.7. Interpolation of the function $f(x) = x^3 + 4x^2 - 10$ using Lagrange polynomials of increasing order.

2.2.2 Local interpolation using a piece-wise continuous function

An alternative to the non-uniform nodal distribution approach used in the previous section to improve the interpolation scheme is based on splitting the solution interval $[x_1, x_n]$ in smaller sub-domains where the interpolation is performed locally. For instance table 2.2 shows such a partition for the interval $[-1.0, 1.0]$ where each sub-domain is conformed by a pair of consecutive nodes. The table shows values of the function $f(x) = x^3 + 4x^2 - 10$ at the edges of the sub-domains. In this particular case, considering each sub-domain to be conformed by a pair of points, the lineal interpolation scheme reduces to finding the straight line (first order polynomial) that passes along the pair of nodes. Higher order local schemes are possible if additional points are added to the sub-domains.

Subdomain	Range	Values for $f(x)$
1	$[-1.0, -0.5]$	$[-7.000, -9.125]$
2	$[-0.5, 0.00]$	$[-9.125, -10.00]$
3	$[+0.0, +0.5]$	$[-10.00, -8.875]$
4	$[+0.5, +1.0]$	$[-8.875, -5.000]$

Table 2.2. Partition of the interval $[-1.0, 1.0]$ into subdomains

Using piecewise interpolation over constant size intervals implies also the use of constant polynomials. In the finite element method each sub-domain is mapped into a constant size interval facilitating systematization of the interpolation process. Notebook 2 in the REPO shows the Python implementation of piecewise interpolation.

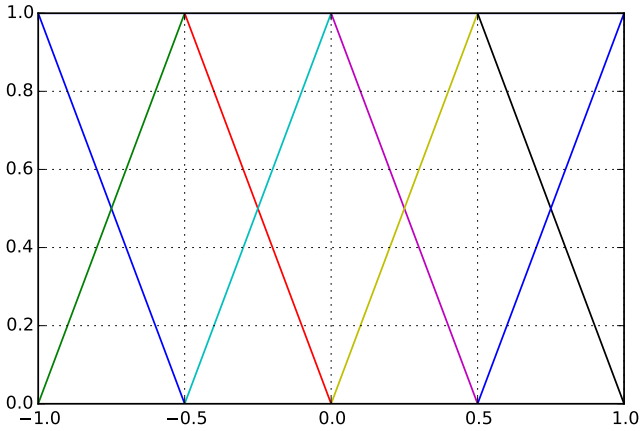
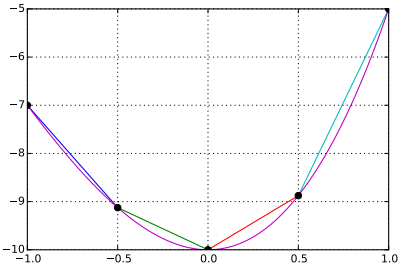
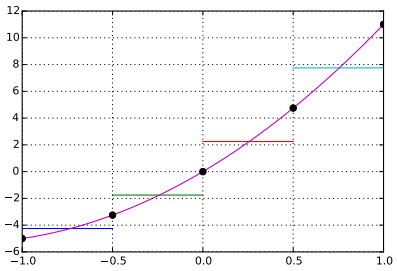


Figure 2.8. Local interpolating polynomials.

This approach results in an interpolating polynomial as shown in fig. 2.9 and where the approximated function is piece-wise continuous. As a result the local based technique the first derivative of the function is now discontinuous at the boundaries of the subdomains. However this local schema is advantageous since the local polynomials are unique and the scheme can be used for an arbitrary number of nodal points facilitating computer implementation.



(a) Approximation to the function using first order interpolation polynomials.



(b) Variation of the first order derivative.

Figure 2.9. Locally based interpolation scheme for the function $f(x) = x^3 + 4x^2 - 10$.

2.2.3 Distribution of the sampling (or nodal) points

The simple problems considered so far have used a small number of sample points and a constant separation distance. This approach worked nicely considering the smooth functions involved. However if the function to be interpolated exhibits strong gradients, as might be the case in problems of wave propagation, the approximation with a small number of data points is very likely insufficient. The natural solution seems to be the addition of data points and thus an increase in the order of the interpolating polynomial. Unfortunately, as we will show here this approach does not always works in the desired direction.

Consider the function:

$$f(x) = \frac{1}{1 + 25x^2}$$

shown in fig. 2.10.

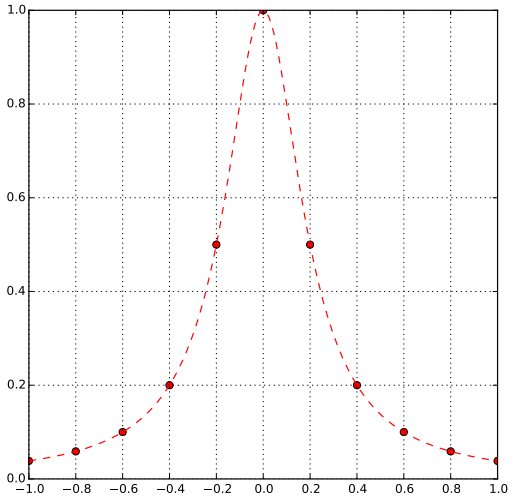
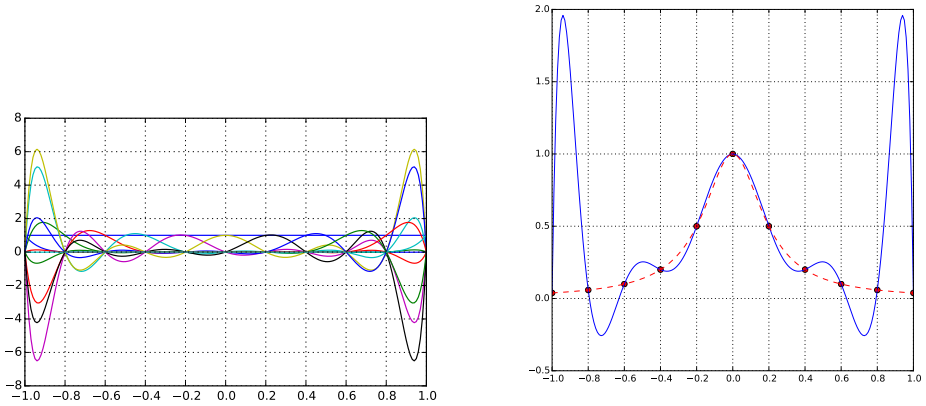


Figure 2.10. Runge function $f(x) = \frac{1}{1+25x^2}$.

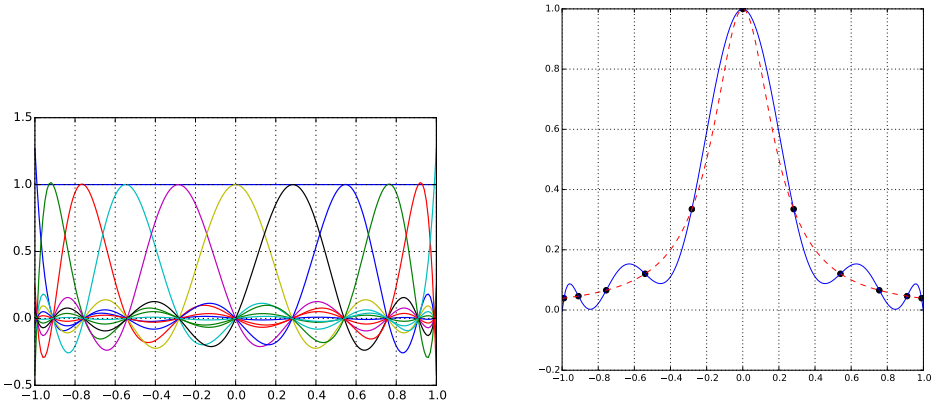
This function shows a strong spatial variation requiring an interpolating polynomial of larger order and as a result a larger number of nodal points. The 11 black dots shown in the figure represent nodal points equally spaced at $\Delta x = 0.2$ and where the function is assumed to be known. We wish to approximate this function using these 11 points and an order 10 interpolating Lagrange polynomial.

Figure 2.11 shows the 11 order-10 Lagrange interpolation polynomials for the equidistant nodal distribution and the resulting interpolating polynomial $p(x)$. Clearly the approximation is highly inaccurate, specially near the edges of the interval where it exhibits strong oscillations. This spurious result along the edges is introduced by the equidistant separation of the sampling points. The interpolation scheme can be improved using a non-uniform nodal spacing. The resulting alternative scheme is shown in fig. 2.12 where there is a large concentration of nodal points along the edges.



(a) **(b)**

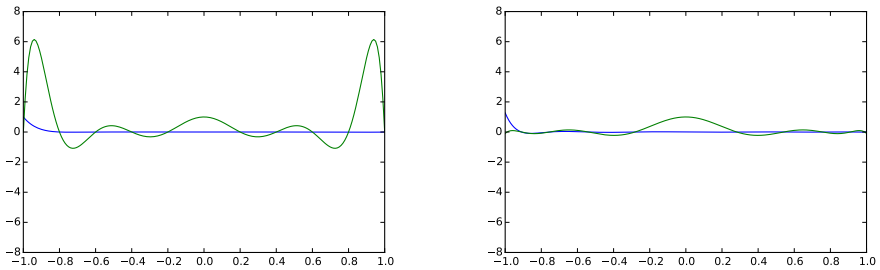
Figure 2.11. (a) Lagrange interpolation polynomials of order 10 associated to the 11 sampling points for fig. 2.10. (b) Interpolating polynomial to approximate the Runge function with the order-10 polynomials from part (a).



(a) **(b)**

Figure 2.12. (a) Order 10 Lagrange interpolation polynomials associated to non-equidistant sampling points. (b) Interpolating polynomial to approximate a Runge function built with the polynomials derived in part (a).

To understand this numerical pathology, related with the distribution of the nodal points, consider the interpolation polynomials corresponding to the central point and to the edge point of the equidistant distribution (fig. 2.11) as shown in part(a) of fig. 2.13. The green line corresponds to the polynomial associated to the central node, while the blue line is that of the edge nodal point. Clearly, the central-point polynomial introduces a strong variation along the edges of the sampling interval, while the edge-point polynomial exhibits a rather smooth variation. Similarly, part(b) in the same figure shows once again the central-point and the edge-point polynomials associated to non-uniform nodal distribution. It can be observed how in this last case both polynomials exhibit a smooth variation over the interval eliminating the strong oscillation towards the edges.

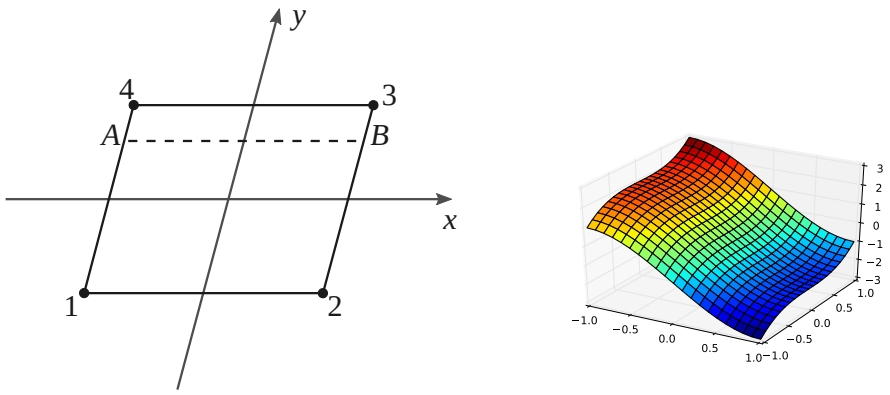


(a) **(b)**

Figure 2.13. (a) Equidistant nodal points (b) Non-uniform nodal points.

2.3 Extension to two-dimensional domains

Assume we are now interested in conducting interpolation of a function over a spatial 2-dimensional domain where every point is specified by a position vector of the form $\overrightarrow{x} = x\hat{\imath} + y\hat{\jmath}$. We want to know, via interpolation, the value of a function $f(\overrightarrow{x})$ at an arbitrary point $\overrightarrow{x}$ provided we know the set of n-pairs of the form $\{(\overrightarrow{x}^1, f^1), \cdots, (\overrightarrow{x}^n, f^n)\}$. The domain and the visualization of the function are shown in fig. 2.14.



(a) Square two-dimensional domain **(b)** Interpolated function $f(x, y)$.

Figure 2.14. Function $f(x, y)$ over a square two-dimensional domain with nodal points labeled 1 , 2 , 3 , 4.

Since now the function f depends on the 2D space coordinates (x, y) it is natural to expect that interpolation functions depend also on (x, y). Using this condition on the function approximation we have:

$$f(x, y) = N^1(x, y)f^1 + N^2(x, y)f^2 + N^3(x, y)f^3 + N^4(x, y)f^4$$

where now $N^Q(x, y)$ is the 2D interpolation (or shape) function associated to the sampling point Q. Using indicial summation convention, we can write the interpolated function as:

$$f(x, y) = N^Q(x, y)f^Q$$

where now $Q = 1, ..., N$.

The method to find the required 2D shape functions $N^Q(x, y)$ consists in the recursive (or iterated) application of the Lagrange one-dimensional scheme discussed previously in terms of interpolating polynomials $L^Q(\eta)$ and where now η is a dummy variable that can assume the role of x or y.

In the domain shown in fig. 2.14 assume that we wish to interpolate the value of the function along the 1-4 direction. Note that along this line x is constant and then the function depends only on y. Fixing $x = x^A$ it is possible to conduct 1-dimensional interpolation along the y direction as shown in fig. 2.15. In this case η assumes the role of y and we have:

$$f(x^A, y) = L^1(y)f^1 + L^4(y)f^4.$$

Since the interpolation scheme is taking place along the 1-4 direction in terms of the 2 nodal values f^1 and f^4, the functions L^1 and L^4 in this case are the first order Lagrange polynomials associated to the points 1 and 4 respectively, and obtained with the already known product formula given in eq. (2.2).

Clearly, the above scheme provides the value of the function for an arbitrary point A along the 1-4 line. Proceeding similarly along the 2-3 direction, that is setting $x = x^B$ and interpolating once again along the y direction we have:

$$f(x^B, y) = L^2(y)f^2 + L^3(y)f^3.$$

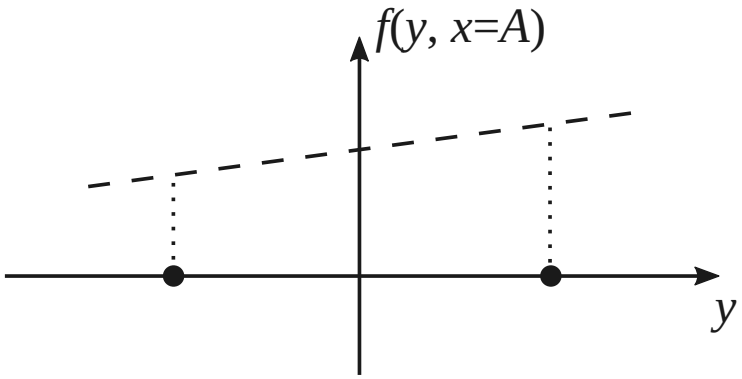


Figure 2.15. Interpolation along the y-direction

So far, we have captured only the dependence on y since x was assumed constant as indicated by $f(x^A, y)$ and $f(x^B, y)$. The dependence on x is

now captured proceeding similarly along the arbitrary line $A - B$ using the functions $f(x^A, y)$ and $f(x^B, y)$ respectively as follows;

$$f(x,y) = L^A(x)f(x^A,y) + L^B(x)f(x^B,y)$$

which after substituting with the found expressions for $f(x^A, y)$ and $f(x^B, y)$ becomes

$$f(x,y) = L^A(x)\{L^1(y)f^1 + L^4(y)f^4\} + L^B(x)\{L^2(y)f^2 + L^3(y)f^3\}$$
$$f(x,y) = L^A(x)L^1(y)f^1 + L^A(x)L^4(y)f^4 + L^B(x)L^2(y)f^2 + L^B(x)L^3(y)f^3 \ .$$

Note that strictly speaking there are only 2 interpolation functions of the form $L^Q(\eta)$ since one-dimensional interpolation is taking place. Thus the interpolating polynomials satisfy the following equivalences:

where

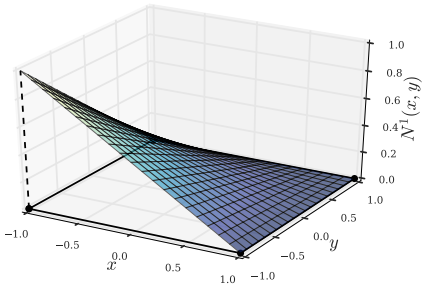
$$\begin{aligned} L^A(x) &\equiv L^1(x) \\ L^B(x) &\equiv L^2(x) \\ L^1(y) &\equiv L^1(y) \\ L^2(y) &\equiv L^1(y) \\ L^3(y) &\equiv L^2(y) \\ L^4(y) &\equiv L^2(x) \ . \end{aligned}$$

The resulting two-variable shape functions $N^Q(x, y)$ follow from the product of one-dimensional interpolation functions like:

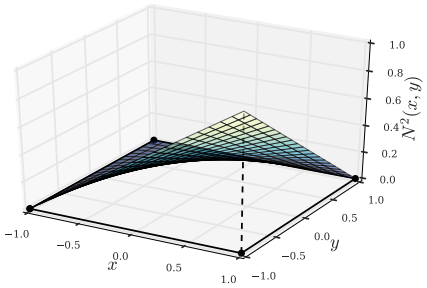
$$\begin{aligned} N^1(x,y) &= L^1(x)L^1(y) \\ N^2(x,y) &= L^2(x)L^1(y) \\ N^3(x,y) &= L^2(x)L^2(y) \\ N^4(x,y) &= L^1(x)L^2(y) \ . \end{aligned}$$

In the actual computer implementation of the discussed interpolation scheme it is desirable to have the actual functions embedded into the code instead of having the computer finding the corresponding Lagrange polynomials each time the size of the square domain changes. In practice, the functions $N^Q(x, y)$ are coded for a canonic square of general side h. This resulting canonic domain can be referred as a finite element.

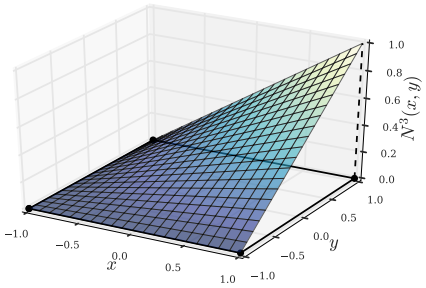
A 2D finite element From the geometric point of view a **finite element** is a canonical interpolation domain together with a set of shape functions and its derivatives. Figure 2.16 shows the shape functions for a so-called bi-linear element of side $h = 1.0$. The element is called bi-linear as linear (or first order) interpolation is used along the x and y directions. Elements of higher order result after adding nodal points and the required corrections to the 2D-shape functions $N^Q(x, y)$.



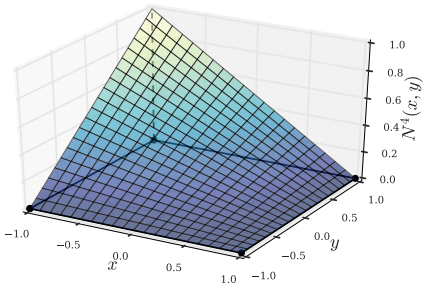
(a) Shape function $N^1(x,y) = \frac{1}{4}(1-x)(1-y)$.



(b) Shape function $N^2(x,y) = \frac{1}{4}(1+x)(1-y)$.



(c) Shape function $N^3(x,y) = \frac{1}{4}(1+x)(1+y)$.



(d) Shape function $N^4(x,y) = \frac{1}{4}(1-x)(1+y)$.

Figure 2.16. Shape functions for a 4-nodes element.

See Notebooks 3 and 4 for an easy to follow Python implementation of interpolation resembling the finite element method.

Finite element mesh Spatial-discretization of the computational domain is in the core of finite element analysis. This corresponds to the partition of the whole domain into **finite elements**. The complete set of finite elements, and its defining attributes, corresponding to a particular domain is termed a **mesh**. If the geometry is irregular the mesh would contain mostly distorted elements with respect to the canonical shape. In the finite element method this is nicely solved using space transformations between the distorted and the canonical shape.

2.4 Interpolation over distorted domains

So far all the 2D interpolation operations have taken place over perfectly squared domains of side h where the interpolation functions are known at least in terms of the side parameter h. In many cases and particularly in finite element methods it is common to find distorted interpolation domains (see fig. 2.17) which difficult the interpolation operation as the interpolation polynomials would be element-dependent and the problem would become impossible to code in a systematic way.

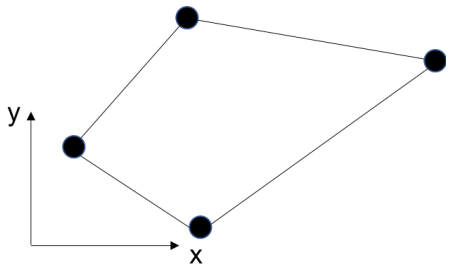


Figure 2.17. Distorted quadrilateral interpolation domain.

In these cases the solution approach is based upon the continuous mapping of the distorted domain and the functions defined in this space into a constant or canonical element where the interpolation functions are always the same. Moreover, the mapping between both spaces is conducted also

using interpolation theory. This idea is explained next with reference to fig. 2.18.

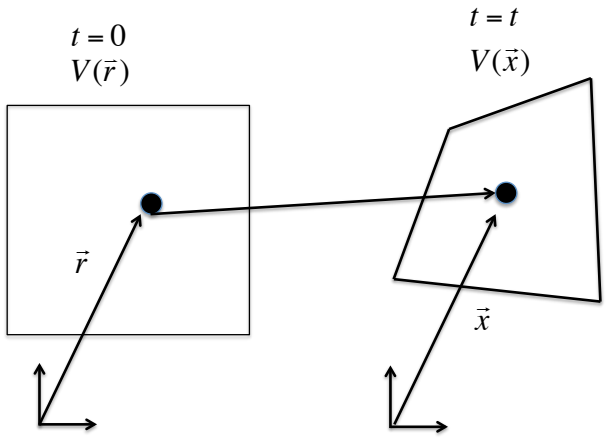


Figure 2.18. Schematic representation of the one-to-one mapping between points in the physical space (left) and the natural or canonical space (right).

In the figure the space of the distorted domain, and with position vectors $\overrightarrow{x}$, is termed the physical space as this corresponds to the space of interest in a particular problem. The figure also shows a perfectly squared domain where we denote space coordinates by a position vector $\overrightarrow{r}$: we refer to this perfectly squared element as the canonical element and to its mathematical space as the natural space. For reasons that will be explained later, it is convenient to have canonical elements of side $h = 2.0$. Note also that since the canonical element is a constant square of side h the corresponding Lagrange interpolation functions are always the same. This implies that if the interpolation process were to be conducted over the natural space it could be easily coded as it would amount to coding the interpolation functions per se.

In mathematical language the connection between both spaces is written in terms of general functional relations as:

$$\begin{aligned} x_i &= x_i(\overrightarrow{r}) \\ r_I &= r_I(\overrightarrow{x}). \end{aligned} \tag{2.5}$$

The first of these functional relationships maps every point $\overrightarrow{r}$ from the canonical space into a point $\overrightarrow{x}$ of the physical space. In particular the relationship 2.5(a) can be written using:

$$x_i = N^Q(\overrightarrow{r})x^Q. \tag{2.6}$$

where the physical space has been represented as an interpolated approximation using as the exact functions the coordinates of the nodal points of the physical space. In particular, the first expression provides the position vector $\overrightarrow{x}$ in the physical space for a point that in the canonical space occupies the position vector $\overrightarrow{r}$. Similarly, the inverse relation gives the position vector $\overrightarrow{r}$ in the canonical space for a point that occupies the position vector $\overrightarrow{x}$ in the physical space. Note that since in 2.5(a) we have used the interpolation functions formulated for the perfectly squared canonical space to approximate the geometry or actual physical space this same transformation can be used to transform functions as explained next.

Assume that $f = f(\overrightarrow{x})$ is a function that describes the space variation of a quantity of interest over a given domain. Using the mapping 2.5 it is also possible to obtain the variation of the quantity in the canonical space after writing:

$$f = f(\overrightarrow{x}) \equiv f[x_i(\overrightarrow{r})] \equiv F(\overrightarrow{r}).$$

In the above expression $F = F(\overrightarrow{r})$ represents the same physical variable but expressed in terms of the position vector in the canonical space. Being able to represent functions in the physical and the canonical space is an important result since it is now possible to conduct interpolation of functions over constant domains as:

$$F(\overrightarrow{r}) = N^Q(\overrightarrow{r})f^Q. \tag{2.7}$$

Note that as a result of the space transformation 2.5, finding the physical function at a point r_I is equivalent to finding the function at an associated

point x_i in the physical domain.

Notebooks 3 and 4 in the REPO make use of Pyhton functions to perform interpolation over two-dimensional domains. In particular, NB4 applies interpolation over arbitrary domains to visualize closed-form solutions.

Proposed problems

1. For the computational domain $x \in [-1.0\ ,\ 1.0]$ and 3 nodal points corresponding to $x^1 = -1.0$, $x^2 = +1.0$ and $x^3 = 0.0$ find the Lagrange polynomials $L^1(x)$, $L^2(x)$ and $L^3(x)$.

2. Verify that the polynomials $L^1(x)$, $L^2(x)$ and $L^3(x)$ satisfy the property $L^I(x^J) = \delta^{IJ}$.

3. Implement a Python script that uses the vector of known values of a function $[f^1, f^2, f^3]$ and the polynomials from problem 1 and compute the interpolating polynomial $p(x)$.

4. Using $p(x)$ from problem 3 compute and plot the first order derivative of $f(x)$ in the interval $[-1, 1]$.

5. For the function $f(x) = x^3 + 4x^2 - 10$ for x in the range $[-1.0, 1.0]$ find values at nodal points that result from splitting the complete interval into 4 sub-domains each one with 3 nodal points. Using these values implement a local interpolation scheme using 2-nd order local interpolation polynomials. Plot the interpolation polynomial in each sub-domain and the corresponding interpolating function $p(x)$. In the same plot compare $p(x)$ and $f(x)$. Additionally, plot the first derivative of the function obtained from $p(x)$ and $f(x)$.

6. For the Runge function defined by:

$$f(x) = \frac{1}{1 + 25x^2}$$

implement an interpolation scheme using local 1st-order Lagrange polynomials. Use (i) sub-domains of constant size $\Delta x = 0.2$ and (ii) sub-domains whose size decreases towards the edges of the interval.

7. Using an independent script (or a notebook) implement a local interpolation scheme using a canonical element of size 2.0 and use it to approximate the Runge function.

8. Assume that at the 4 nodal points of a 2D square domain of side $2h$ we know the vector field

$$\overrightarrow{u} = u(x,y)\widehat{i} + v(x,y)\widehat{j}$$

and where $u(x,y)$ and $v(x,y)$ are the scalar rectangular components along the x and y direction of a cartesian coordinate system.

   - Implement an interpolation scheme to compute the vector field $\overrightarrow{u} = \overrightarrow{u}(x,y)$ at an arbitrary point (x,y).
   - Use the interpolation scheme to compute $\varepsilon_{xx} = \frac{\partial u}{\partial x}$ and $\varepsilon_{yy} = \frac{\partial v}{\partial y}$
   - Implement a Python script to visualize the vector field and the scalars ε_{xx} and ε_{yy}.

9. Implement a Python script to visualize analytic (or numerical) solutions available at a set of nodes. Use the following steps;

   - Use external software[3] to define and mesh and arbitrary solution domain.
   - Evaluate the solution at the nodal points of the mesh and store the results into arrays.
   - Use Python triangularization objects together with matplotlib routines to visualize the solution.

[3]Gmsh is an open source software for pre and post processing of complex 1D, 2D and 3D domains. Meshio is a set of Python scripts to read and write Gmsh readable files using dictionaries.

Chapter 3

Quadratures: numerical integration

Preliminary

The formulation of finite element algorithms is strongly based on the weak formulation of the boundary value problems. In loose terms, in typical finite element algorithms the boundary value problem originally written as a set of governing differential equations and properly specified boundary conditions, is re-formulated in the form of integral representations. For instance, this is the case in the boundary value problem of linearized theory of elasticity where the differential equations (corresponding to the equilibrium of a material point) and the tractions (and/or displacement) boundary conditions are shown to be equivalent to the the integral form representation of the principle of virtual displacements. In this case a finite element algorithm results from the discretization through interpolation schemes of the integrand appearing in the principle. However to generate an efficient method, valid over arbitrary domains it is necessary to implement effective numerical integration algorithms. To illustrate the need for numerical integration in the formulation of finite element methods consider a typical term resulting from the discretization of the internal virtual energy in the principle of virtual

displacements such as the stiffness matrix given by:

$$K^{QP} = \int_V H_{ij}^Q C_{ijkl} H_{kl}^P \, \mathrm{d}\, V. \tag{3.1}$$

The accurate computation of these integrals is important to the formulation of the finite element algorithm.

Note that the integration given by (3.1) is conducted over a domain V which is typically a finite element of arbitrary shape (e.g., a distorted quadrilateral element) therefore difficulting the computation of K^{QP}. This chapter discusses the most relevant details required in the numerical computation of integrals like the one appearing in (3.1) which are typical in finite element algorithms. In the first part of the chapter we define a general formula for numerical integration. For completeness we derive integration formulas based on Lagrange interpolation although emphasis will be placed on the more efficient Gaussian integration formulas.

At the end of this chapter[1] the student should be able to:

- Recognize the difference between explicit and numerical computation of integrals.
- Recognize the advantages and disadvantages of different numerical integration schemes.
- Propose integration schemes for specific finite elements.
- Implement efficient Python subroutines required in the integration of functions over specific finite elements.

[1] **This chapter, together with theoretical and computational learning activities is complemented by Jupyter Notebooks 5 and 6 available at the course REPO. Notebook 5 covers numerical integration as used in the finite element method, while notebook 6 combines interpolation theory with numerical integration in the calculation of the stiffness matrix for a finite element.**

3.1 Statement of the problem

A note on notation: Recall that in our indicial notation we are using subscripts to refer to the scalar components of a vector or a tensor function, while superscripts are reserved to represent elements of the interpolating polynomial. Thus for scalar components of a vector we use:

$$V_i \equiv \begin{bmatrix} V_x & V_y & V_z \end{bmatrix}.$$

For interpolation of a scalar function we use:

$$p(x) = L^Q(x) f^Q.$$

And for interpolation of a vector function we use:

$$u_i(x) = L_i^Q(x) u^Q$$

In the most general case we are interested in numerically computing integrals of the general form

$$I = \iiint f(x, y, z) dV \tag{3.2}$$

where the triple integral represents an integration over a given volume. As in the case of interpolation theory, the problem of integration can also be solved from the fundamental problem of integrating a one-dimensional function like

$$\int_a^b f(x) dx \approx \sum_{I=1}^{N} w^I f(x^I). \tag{3.3}$$

In this fundamental one-dimensional problem the integral of a function $f(x)$ from $x = a$ to $x = b$ is approximated by a weighted summation of the

values of the function at a set of N-points. In (3.3) w^I represents a weighting factor associated to the value of the function $f(x^I)$ at the point I.

This numerical approximation of the integral in terms of a weighted summation is called a quadrature formula and the derivation of a specific quadrature corresponds to prescribing the required number of points N, the corresponding weighting factors and the location of the N sampling or integration points.

Example Using the following set of quadrature points and weighting factors

x^I	w^I
-0.86113	0.34785
-0.33998	0.65214
$+0.33998$	0.65214
$+0.86113$	0.34785

evaluate the integral

$$I = \int_{-1}^{+1} (x^3 + 4x^2 - 10)dx. \tag{3.4}$$

The numerical evaluation of the integral in (3.4) just reduces to the computation of the following weighted summation:

$$\int_{-1}^{+1} (x^3 + 4x^2 - 10)dx \approx 0.34785 \cdot f(-0.86113) + 0.65214 \cdot f(-0.33998)$$
$$+0.34785 \cdot f(0.86113) + 0.65214 \cdot f(0.33998) = -17.3333$$

3.2 Numerical integration using interpolation polynomials

A simple quadrature formula can be easily obtained if one represents the actual function $f(x)$ through a Lagrange based interpolation polynomial $p(x)$:

$$\int_a^b f(x)dx \approx \int_a^b p(x)dx. \tag{3.5}$$

where

$$p(x) = L^I(x)f(x^I) \tag{3.6}$$

and $L^I(x)$ is the Lagrange interpolation polynomial associated to point x^I.

Substituting eq. (3.6) in eq. (3.5) yields

$$\int_a^b f(x)\ dx \approx \int_a^b L^I(x)f(x^I)dx \equiv f(x^I)\int_a^b L^I(x)dx$$

which can be written like

$$\int_a^b f(x)dx \approx \sum_{I=1}^{N} w^I f(x^I) \tag{3.7}$$

after noticing that

$$w^I = \int_a^b L^I(x)\, dx. \tag{3.8}$$

Integration schemes are classified in Newton-Cotes and Gaussian quadratures methods. In the first case the range of integration is divided into $N-1$ subintervals of constant size $\frac{b-a}{N-1}$ while in the second group one searches for the optimum location of the integration points inside the interval in order to obtain maximum accuracy.

Example: Extended trapezoidal rule Consider the particular case in which $N = 2$ (i.e., 1 integration interval). Clearly, in this case the size of the interval is $h = b - a$ and the interpolating polynomial is given by:

$$p(x) = L^1(x)f^1 + L^2(x)f^2 \equiv L^1(x)f(a) + L^2(x)f(b)$$

with interpolation polynomials corresponding to:

$$L^1(x) = \frac{(x - x^2)}{(x^1 - x^2)} \equiv -\frac{1}{h}(x - b)$$

$$L^2(x) = \frac{(x - x^1)}{(x^2 - x^1)} \equiv \frac{1}{h}(x - a).$$

Substitution in (3.8) yields:

$$w^1 = -\frac{1}{h}\int_a^b (x - b)dx \equiv \frac{h}{2}$$

$$w^2 = +\frac{1}{h}\int_a^b (x-a)dx \equiv \frac{h}{2}$$

giving the final quadrature

$$I = w^1 f^1 + w^2 f^2 \equiv \frac{h}{2}\left[f(a) + f(b)\right]$$

or equivalently:

$$\int_a^b f(x)dx = h\left[\frac{1}{2}f(a) + \frac{1}{2}f(b)\right]. \tag{3.9}$$

Example: Computation of a definite integral Use the trapezoidal rule to compute the integral

$$I = \int_{-1}^{+1} (x^3 + 4x^2 - 10)dx.$$

In this case h=2.0, therefore:

$$I = f(-1) + f(+1) \equiv -7 - 5 = -12$$

Example: Integration over two-dimensional domains. Figure 3.1 shows an schematic description of a two-dimensional domain (continuous black line) denoted by R. We wish to compute the integral

$$I = \iint_R f(x,y)dA.$$

To proceed with the computation, the domain has been divided in N rectagular subdomains (black dashed lines) in such a way that a typical subdomain has dimensions $\Delta x_i \times \Delta y_i$ as shown in the auxiliary figure.

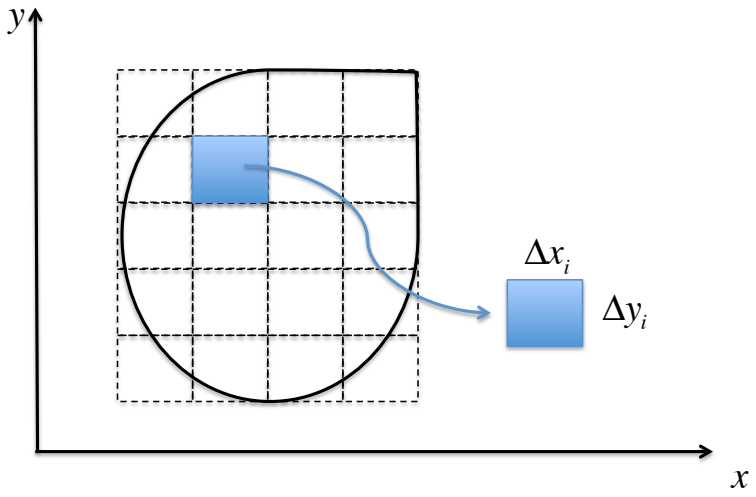


Figure 3.1. Riemman partition for a two-dimensional domain.

Defining

$$|p| = \max|\Delta x_i| \vee \max|\Delta y_i|$$

as the norm of the partition, we have, according to the definition of an integral as a Riemman sum that:

$$I = \iint_R f(x,y)dA \equiv \lim_{|p|\to 0} \sum_{j=1}^{N} f(x_j, y_j)\Delta x_j \Delta y_j.$$

Taking each one of the limits independently allows to identify 2 integration process such that the integral over R reduces to the double integral given by:

$$I = \int \int f(x, y) dx dy.$$

To identify the integration limits consider fig. 3.2 showing a rectangular integration domain with largest side parallel to the x direction and with mid height corresponding to a y constant value. The small sides of the rectangle have abscissas $x_1(y)$ and $x_2(y)$ respectively.

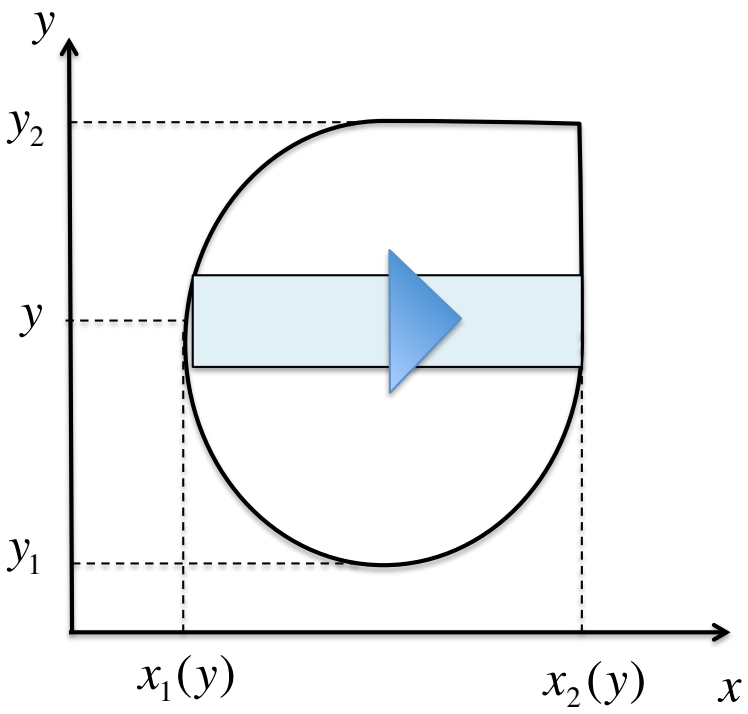


Figure 3.2. Integration along the x direction.

Considering once again the definition of an integral as the limit of a Riemman sum we then have that for constant y values the contribution to

the integral over region R of the rectangle bounded by $x_1(y)$ and $x_2(y)$ is given by:

$$\int_{x_1(y)}^{x_2(y)} f(x,y)dx$$

in such a way that the computation of the integral over the full region R is completed after repeating the process for constant values of y varying between y_1 and y_2 giving for the total integral:

$$I = \int_{y_1}^{y_2} \left\{ \int_{x_1(y)}^{x_2(y)} f(x,y)dx \right\} dy \tag{3.10}$$

To clarify eq. (3.10), note that the internal integral can be written as a function of y

$$F(y) = \int_{x_1(y)}^{x_2(y)} f(x,y)dx$$

and the external integral like:

$$I = \int_{y_1}^{y_2} F(y)dy.$$

Alternatively (see fig. 3.3) it is possible to define:

$$H(x) = \int_{y_1(x)}^{y_2(x)} f(x,y)dy$$

in such a way that the full integral I is defined by:

$$I = \int_{x_1}^{x_2} H(x)dx.$$

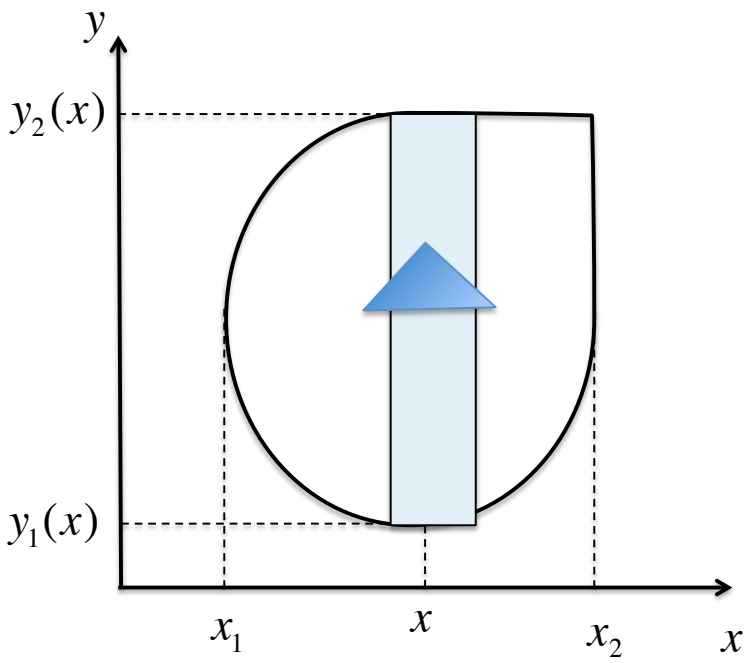


Figure 3.3. Integration along the y direction.

Let us apply the previous ideas to compute the integral

$$I = \int\int xy^2 dA$$

over the region shown in fig. 3.4.

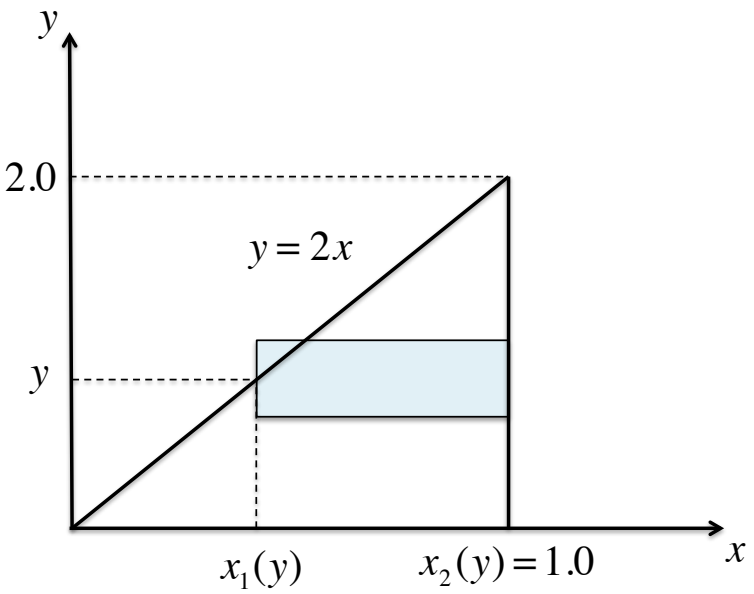


Figure 3.4. Integration along the x direction over the triangular region R.

Identifying the lower and upper integration limits along the y direction as $y_1 = 0$ and $y_2 = 2$ respectively, and the functions $x_1(y)$ and $x_2(y)$ like:

$$x_1(y) = \frac{y}{2}$$

and

$$x_2(y) = 1$$

we have that:

$$F(y) = \int_{x_1(y)}^{1.0} xy^2 dx \equiv \int_{y/2}^{1.0} xy^2 dx$$

then

$$F(y) = \frac{1}{2}x^2y^2\Big|_{y/2}^{1.0} \equiv \frac{1}{2}y^2 - \frac{1}{8}y^4$$

using this function to integrate in y one finally gets that:

$$I = \int_0^{2.0} F(y)dy \equiv \int_0^{2.0} (\frac{1}{2}y^2 - \frac{1}{8}y^4)dy \equiv \frac{8}{15}.$$

Proceeding alternatively (see fig. 3.5) it is possible to write:

$$H(x) = \int_{y_1(x)}^{y_2(x)} xy^2dy$$

and

$$I = \int_{x_1}^{x_2} H(x)dx$$

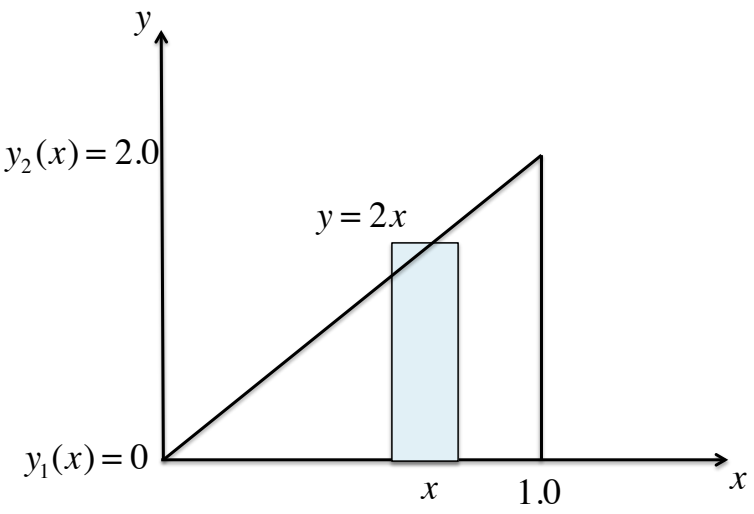


Figure 3.5. Integration along the y direction over the triangular region R.

where:

$$y_1(x) = 0$$
$$y_2(x) = 2x$$

then

$$H(x) = \int_0^{2x} xy^2 dy \equiv \frac{1}{3}xy^3\Big|_0^{2x} \equiv \frac{8}{3}x^4$$

so the integral reduces to:

$$I = \int_0^{1.0} \frac{8}{3}x^4 dx \equiv \frac{8}{15}.$$

3.3 Gaussian quadratures

In the numerical quadrature corresponding to the extended trapezoidal rule written in the form

$$\int_a^b f(x)dx \approx \sum_{I=1}^{npts} w^I f(x^I) \tag{3.11}$$

the integration points are equidistantly spaced. In a Gaussian quadrature in addition to adjusting the N weighting factors w^I one also leaves as adjustable parameters the location of the N integration points. As a result, there are now $2N$ parameters to adjust in the derivation of an algorithm to numerically approximate the integral of $f(x)$ between $x = a$ and $x = b$ with the maximum accuracy and the minimum number of operations. This class of quadratures provide better precision than those based on Newton-Cotes techniques (such as the trapezoidal rule) when the function to integrate can be appropriately represented by a polynomial.

In general, different Gaussian quadratures are found in the literature reported in terms of tables providing the locations of integration (or Gauss) points and the corresponding weighting factors w^I. As an example table 3.1 gives abscissas and weighting factors for a 4-point Gaussian quadrature.

x^I	w^I
-0.86113	0.34785
-0.33998	0.65214
$+0.33998$	0.65214
$+0.86113$	0.34785

Table 3.1. Abscissas and weighting factors to compute $\int_{-1}^{+1} f(x)dx$

To facilitate coding of these quadratures and allow for approximation of general integrals, it is common to consider a primitive range of integration

$[-1.0, +1.0]$ which requires transforming the original integral (including the function and its integration limits)to this primitive integral as discussed in section 3.4. Figure 3.6 schematizes the primitive integration range and the corresponding Gauss points denoted by the black xs. Transformation of a given integral to the primitive space is discussed at a later section.

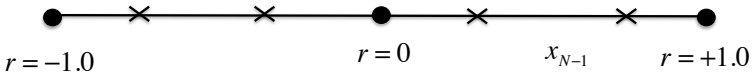


Figure 3.6. Schematic reperesentation of a Gaussian quadrature in the primitive range $[-1.0, 1.0]$.

Example:Derivation of a Gaussian quadrature Let $n = 2$ and the integration interval $[a, b] = [-1, +1]$. Find w^1, w^2 and x^1, x^2 such the quadrature

$$I = \int_{-1}^{+1} f(x)dx \approx w^1 f(x^1) + w^2 f(x^2)$$

integrated exactly the function $f(x)$ corresponding to a third order polynomial like:

$$f(x) = a_0 + a_1 x + a_2 x^2 + a_3 x^3.$$

Using $f(x)$ in I and stating the integral for each term we have:

$$I = \int_{-1}^{+1} a_0 dx + \int_{-1}^{+1} a_1 x dx + \int_{-1}^{+1} a_2 x^2 dx + \int_{-1}^{+1} a_3 x^3 dx$$

where:

$$\int_{-1}^{+1} dx = 2 = w^1 \cdot 1 + w^2 \cdot 1$$

$$\int_{-1}^{+1} x dx = 0 = w^1 \cdot x^1 + w^2 \cdot x^2$$

$$\int_{-1}^{+1} x^2 dx = \frac{2}{3} = w^1 \cdot (x^1)^2 + w^2 \cdot (x^2)^2$$

$$\int_{-1}^{+1} x^3 dx = 0 = w^1 \cdot (x^1)^3 + w^2 \cdot (x^2)^3.$$

The resulting system of equations is solved in order to determine the 4 quadrature parameters, namely w^1, w^2 and x^1, x^2 giving $w^1 = 1$, $w^2 = 1$, $x^1 = -\sqrt{3}/3$ and $x^1 = +\sqrt{3}/3$ which allows us to write the quadrature in the general form:

$$I = \int_{-1}^{+1} f(x)dx \approx 1.0 \cdot f(-\sqrt{3}/3) + 1.0 \cdot f(+\sqrt{3}/3)$$

which is exact for polynomial functions of order at most 3.

The idea behind Gaussian quadratures can be extended to the integration of higher order polynomials, however its derivation requires an effective method to determine the weighting factors and the abscissas of the Gauss points. The next section discusses a method which is applicable to $2n$-order polynomials, in which advantage is taken from the property of orthogonality existing in certain special polynomials.

3.4 Numerical integration in the finite element method

3.4.1 One-dimensional domains

The systematization and construction of tables with coordinates and weighting factors for different quadratures is useful if these are specified for a fixed range. For mathematical convenience it is common to use as base interval $[-1, +1]$. However considering that we are interested in integrating a function $f(x)$ in the general range with limits $x = a$ and $x = b$ it is required that we re-write the integral like:

$$\int_a^b f(x)dx \equiv \int_{-1}^{+1} F(r)dr. \tag{3.12}$$

The mapping indicated in eq. (3.12) is described in fig. 3.7, in which the space represented by the independent variable x and contained between $x = a$ and $x = b$ is mapped to a fictitious "natural" space described by a new independent variable r and enclosed in $r = -1$ y $r = 1$.

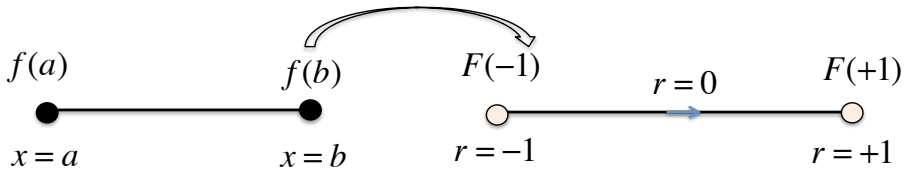


Figure 3.7. Mapping between the physical space $[a, b]$ and the primitive or natural space $[-1.0, +1.0]$.

This is exactly the same transformation used when approximating unknown functions using interpolation theory over arbitrary distorted domains. Repeating for convenience, the transformation between both spaces can be written like:

$$x = x(r)$$
$$r = r(x)$$

and where $x(r)$ and $r(x)$ represent functional relationships between both spaces. For instance, in reference to fig. 3.7, it is evident that independent of the functional relationship this must satisfy the condition $x(-1.0) = a$ and $x(+1) = b$. Therefore, a valid relationship can be derived after assuming that both spaces are related through a Lagrange interpolating polynomials as follows:

$$x(r) = L^1(r)x(r^1) + L^2(r)x(r^2)$$

and where the interpolation polynomials are given by:

$$L^1(r) = \frac{(r - r^2)}{(r^1 - r^2)} \equiv \frac{1}{2}(1 - r)$$

$$L^2(r) = \frac{(r - r^1)}{(r^2 - r^1)} \equiv \frac{1}{2}(1 + r)$$

which results in:

$$x(r) = \frac{1}{2}(a + b) + \frac{r}{2}(b - a).$$

It is also necessary to re-write $f(x)$ using as independent variable r instead of x using the functional relationship $x = x(r)$ like:

$$f = f(x) \equiv f[x(r)] = \hat{F}(r)$$

where $\hat{F}(r)$ represents the same function but now written in terms of r.

Finally, to complete the transformation it is necessary to transform physical differential space elements, that is dx. Proceeding directly from the mapping via the Lagrange interpolation polynomials we have:

$$\frac{dx}{dr} = \frac{dL^1(r)}{dr}x(r^1) + \frac{dL^2(r)}{dr}x(r^2) \equiv \frac{1}{2}(b-a)$$

and therefore

$$\frac{dx}{dr} = \frac{1}{2}(b-a)$$

which allows us to finally write the integral in the fictitious domain enclosed in $[-1,+1]$ according to:

$$\int_a^b f(x)dx \equiv \int_{-1}^{+1} \hat{F}(r)\frac{\ell}{2}dr \equiv \int_{-1}^{+1} F(r)dr$$

and where $\ell = \frac{1}{2}(b-a)$.

The module **Gaussutil()** in the finite element code **SolidsPy** contains several one-dimensional and two-dimensional Gauss quadratures.

Example Use a 2 point Gaussian quadrature (see table 3.2) to evaluate the integral:

$$I = \int_0^3 (2^x - x)dx.$$

x^I	w^I
-0.577350269189626	1.000000
$+0.577350269189626$	1.000000

Table 3.2. Abscissas and weighting factors to compute $\int_{-1}^{+1} f(r)dr$

To perform the numerical integration using the 2-point Gaussian quadrature given in table 3.2 it is necessary to transform the integration range and the integrand of the function to the range corresponding to $[-1.0, +1.0]$. The transformation is given by:

$$x(r) = \frac{3}{2} + \frac{3}{2}r$$

while the differential elements satisfy

$$dx = \frac{3}{2}dr.$$

To transform the function we use:

$$\hat{f}(r) = f[x(r)] \equiv f\left(\frac{3}{2} + \frac{3}{2}r\right)$$

from which:

$$I = \int_0^3 (2^x - x)dx \equiv \int_{-1.0}^{+1.0} \left[2^{\frac{3}{2}(1+r)} - \frac{3}{2}(1+r)\right]\frac{3}{2}dr$$

and evaluating:

$$I = \sum_{I=1}^{2} w^I \left[2^{\frac{3}{2}(1+r^I)} - \frac{3}{2}(1+r^I) \right] \frac{3}{2} = 1.0 \cdot (1.37678967978) + 1.0 \cdot (4.18374583924) \equiv 5.56053551$$

3.4.2 Two-dimensional domains

In the finite element method there is interest in computing integrals like:

$$I = \int_{V(\overrightarrow{x})} f(\overrightarrow{x}) \, \mathrm{d}\, V(\overrightarrow{x}) \tag{3.13}$$

where $V(\overrightarrow{x})$ is the domain of a typical finite element in a reference system with position vector $\overrightarrow{x}$. In one-dimensional problems $V(\overrightarrow{x}) \equiv [x_a, x_b]$; in two-dimensional problems $V(\overrightarrow{x})$ is a plane surface; and in three-dimensional problems $V(\overrightarrow{x})$ is a volume. Moreover, previously we have defined a finite element like a local interpolation space where values of a function are known at specific points (nodes). For instance, in two-dimensional space we had a quadrilateral bi-lineal element as the one shown in fig. 3.8.

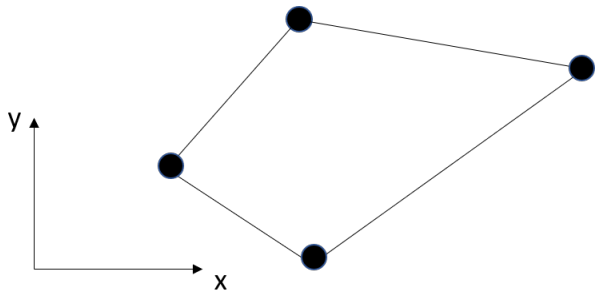


Figure 3.8. Arbitrary bi-lineal element defined in the physical space

As in the one-dimensional case discussed in the previous section, to conduct numerical integration in a systematic way, we actually need to transform generalized finite elements into canonical interpolation spaces (see fig. 3.9).

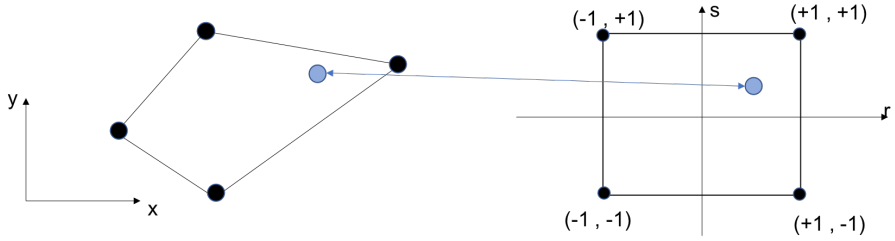


Figure 3.9. Transformation from a general distorted element in the physical space to a perfect square in the natural space

Repeating the transformation for convenience, we have:

$$\begin{aligned} x_i &= x_i(\overrightarrow{r}) \\ r_I &= r_I(\overrightarrow{x}) \end{aligned} \tag{3.14}$$

where $\overrightarrow{x}$ and $\overrightarrow{r}$ denote position vectors in the physical and canonical space respectively. Using the above to transform functions we can write:

$$f = f(\overrightarrow{x}) \equiv f[x_i(\overrightarrow{r})] \equiv F(\overrightarrow{r}). \tag{3.15}$$

Equation (3.15) above indicates that passing $f(\overrightarrow{x})$ from the physical to the natural space representation corresponds to a change of variables. In the particular case of classical finite element methods the change of variables is conducted after approximating the physical geometry using interpolation. particularly, using the same set of shape functions as in the approximation of the primary fields we can write:

$$x_i(\overrightarrow{r}) = N_i^Q(\overrightarrow{r})x^Q \tag{3.16}$$

where x^Q denotes the spatial coordinates of nodal point Q and $= N_i^Q(\overrightarrow{r})$ is the shape function associated to the nodal point Q. According to this expression in the finite element method the geometry is interpolated in terms similar to the ones used for the primary field variable.

To transform the domain of integration we start once again from the general functional relationship:

$$x_i = x_i(\overrightarrow{r})$$

and use it to stablish the relationship between differential lengths in both spaces as:

$$dx_i = \frac{\partial x_i}{\partial r_J} dr_j. \tag{3.17}$$

The second order tensor $\frac{\partial x_i}{\partial r_J}$ appearing in 3.17 is the Jacobian of the transformation J_{iJ} explicitly defined by;

$$J_{iJ} = \frac{\partial x_i}{\partial r_J}$$

and this tensor contains all the information regarding the geometric changes between both spaces. In terms of the shape functions it follows that:

$$J_{iJ} = \frac{\partial x_i}{\partial r_J} \equiv \frac{\partial N_i^Q}{\partial r_J} x^Q. \tag{3.18}$$

To complete the transformation we make use of Nanson's formula from continuum mechanics, from which:

$$dV(\overrightarrow{x}) = |J|\, dV(\overrightarrow{r})$$

where $|J|$ is the determinant of the Jacobian tensor. We can write for I in both spaces:

$$I = \int_{V(\overrightarrow{x})} f(\overrightarrow{x})\, \mathrm{d}\, V(\overrightarrow{x}) \equiv \int_{V(\overrightarrow{r})} F(\overrightarrow{r})\, |J(\overrightarrow{r})|\, \mathrm{d}\, V(\overrightarrow{r}) \tag{3.19}$$

Consider now the particular case of a two-dimensional bi-lineal finite element discussed previously and its transformation into the natural space shown in fig. 3.10. Notice that this canonical element is a perfect square of element side 2.0 contained between $x \in [-1.0\ ,\ +1.0]$ and $y \in [-1.0\ ,\ +1.0]$ which is the same range of the fundamental quadrature studied in the one-dimensional context.

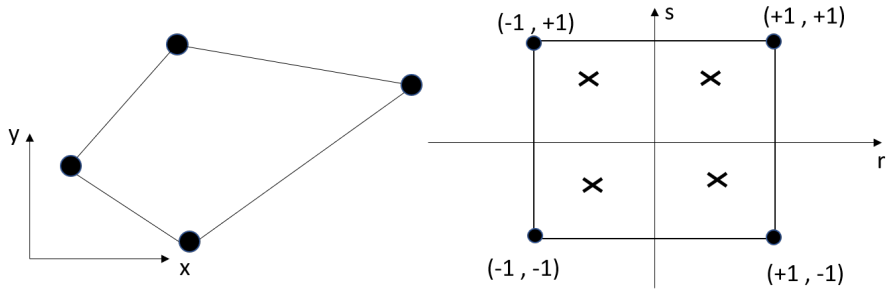


Figure 3.10. Transformation from a general distorted element in the physical space to a perfect square in the natural space

In the particular case of the transformation described in fig. 3.10 we have that

$$dV(\overrightarrow{r}) = drds$$

therefore

$$dV(\overrightarrow{x}) = |J|\, drds$$

and eq. (3.19) takes the form:

$$I = \int_{S(\overrightarrow{x})} f(\overrightarrow{x})\,\mathrm{d}\,S(\overrightarrow{x}) \equiv \int_{r=-1}^{r=+1}\int_{s=-1}^{s=+1} F(r,s)\,|J(r,s)|\,\mathrm{d}\,r ds. \tag{3.20}$$

To integrate 3.20 using a quadrature of $Ngpts$ integration points we use:

$$I = \int_{r=-1}^{r=+1}\int_{s=-1}^{s=+1} F(r,s)\,|J(r,s)|\,\mathrm{d}\,r ds \approx \sum_{i=1}^{Ngpts}\sum_{j=1}^{Ngpts} F(r_i,s_j)\,|J(r_i,s_j)|\,w_i w_j \tag{3.21}$$

which can be simplified into

$$I = \int_{r=-1}^{r=+1}\int_{s=-1}^{s=+1} F(r,s)\,|J(r,s)|\,\mathrm{d}\,r ds \approx \sum_{k=1}^{Ngpts*Ngpts} F(r_k,s_k)\,|J(r_k,s_k)|\,\alpha_k \tag{3.22}$$

which the one-dimensional version resulting from the iterated summation. The points marked with a cross in fig. 3.10 represent integration points.

> Notebook 5 in the REPO uses transformation or mapping from the physical to the canonical space, together with numerical integration, to compute the area of a distorted quadrilateral finite element.

3.4.3 Matrix formulation

For the computer implementation of the different methods discussed so far it is practical to use a combined notation in terms of index and matrix representation of variables. Consider the interpolated version of a vector field, for instance the displacement vector in elasticity:

$$u_i(\overrightarrow{r}) = N_i^Q(\overrightarrow{r})u^Q \tag{3.23}$$

in which:

- Susbcript i refers to the normal components of vectors in the physical space
- Superscript Q refers to the contribution from nodal point Q to the approximated field (in this case $u_i(\overrightarrow{r})$)
- $\overrightarrow{r}$ position vector of a point in the natural space. In index notation we refer to the scalar components of this vector using capitalized subscripts as $\overrightarrow{r} = r_I$
- $N_i^Q(\overrightarrow{r})$ shape function associated to the nodal point Q evaluated at the point $\overrightarrow{r}$.

The contribution to the Q nodal point implicit in eq. (3.23) can be written in explicit expanded form as:

$$\begin{Bmatrix} u \\ v \end{Bmatrix} = \begin{bmatrix} \cdots & N^Q(\overrightarrow{r}) & 0 & \cdots \\ & 0 & N^Q(\overrightarrow{r}) & \end{bmatrix} \begin{Bmatrix} \vdots \\ u^Q \\ v^Q \\ \vdots \end{Bmatrix}.$$

To compute the Jacobian tensor at point $\overrightarrow{r}$ assume that the nodal coordinates are stored in a matrix

$$coord = \begin{bmatrix} & \vdots & \\ x^Q & & y^Q \\ & \vdots & \end{bmatrix}$$

then we can further express:

$$\begin{bmatrix} \frac{\partial x}{\partial r} & \frac{\partial y}{\partial r} \\ \frac{\partial x}{\partial s} & \frac{\partial y}{\partial s} \end{bmatrix} = \begin{bmatrix} \cdots & \frac{\partial N^Q}{\partial r} & \cdots \\ & \frac{\partial N^Q}{\partial s} & \end{bmatrix} \begin{bmatrix} & \vdots & \\ x^Q & & y^Q \\ & \vdots & \end{bmatrix} \tag{3.24}$$

Having found the Jacobian of the transformation the remaining step consists in transforming the integrand $f(\overrightarrow{x})$. This step however is not constructed explicitly but it depends on the structure of $f(\overrightarrow{x})$. In most finite element algorithms the problem formulation involves spatial derivatives of the primary function rather than the primary function itself. To consider these terms in the transformed version of I, it becomes necessary to relate spatial differentiation in both spaces. To clarify this aspect of the formulation assume that I is of the form:

$$I = \int_{V(\overrightarrow{x})} \frac{\partial f}{\partial \overrightarrow{x}} \, \mathrm{d}\, V(\overrightarrow{x}).$$

We already found that

$$\mathrm{d}\, V(\overrightarrow{x}) = |J| \, \mathrm{d}\, V(\overrightarrow{r})$$

and to transform the integrand

$$\frac{\partial f}{\partial \overrightarrow{x}}$$

we recall that in the finite element method the interpolation of the primary variable is conducted directly in the natural space. This means that we already have $F(\overrightarrow{r})$ or more explicitly that:

$$F(\overrightarrow{r}) = N^Q(\overrightarrow{r}) F^Q.$$

However to capture correctly the physics of the problem we are interested in finding

$$\frac{\partial f}{\partial \overrightarrow{x}}.$$

Using implicit differentiation:

$$\frac{\partial f}{\partial x_i} = \frac{\partial F}{\partial r_J}\frac{\partial r_J}{\partial x_i}$$

and re-arranging for convenience we write

$$\frac{\partial f}{\partial x_i} = \frac{\partial r_J}{\partial x_i}\frac{\partial F}{\partial r_J}. \tag{3.25}$$

Notice that the first factor is the Jacobian inverse given by;

$$\frac{\partial r_J}{\partial x_i} = (J_{iJ})^{-1} \equiv \left(\frac{\partial x_i}{\partial r_J}\right)^{-1}$$

while the second factor reads:

$$\frac{\partial F}{\partial r_J} = \frac{\partial N^Q}{\partial r_J}F^Q$$

which allows us to write:

$$I = \int_{V(\overrightarrow{x})} \frac{\partial f}{\partial \overrightarrow{x}}\,\mathrm{d}\,V(\overrightarrow{x}) \equiv \int_{V(\overrightarrow{r})} J^{-1}_{iJ}\frac{\partial N^Q}{\partial r_J}\,|J(\overrightarrow{r})|\,\mathrm{d}\,V(\overrightarrow{r}).$$

As an in-class activity Notebook 6 in the REPO requires the computation of the stiffness matrix for a plane strain solid element.

Proposed problems

1. Compute the integral of the function

$$f(x, y) = 4x^2 + 3xy + y^2$$

over the two-dimensional domains shown in the figure

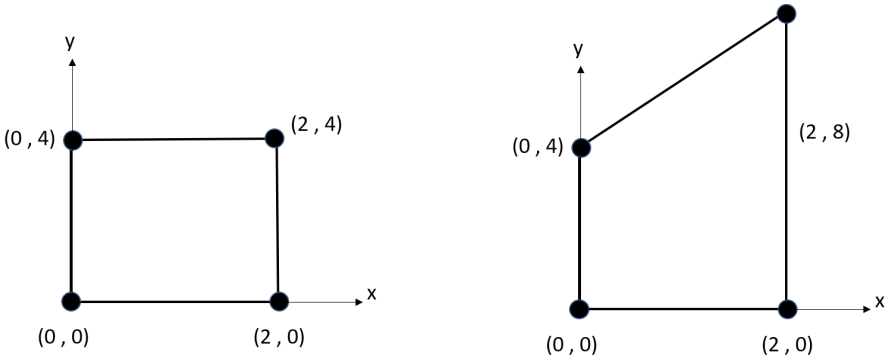


Figure 3.11. Integration domains for problem 1

2. Compute the jacobian of the transformation of the domains shown in the figure

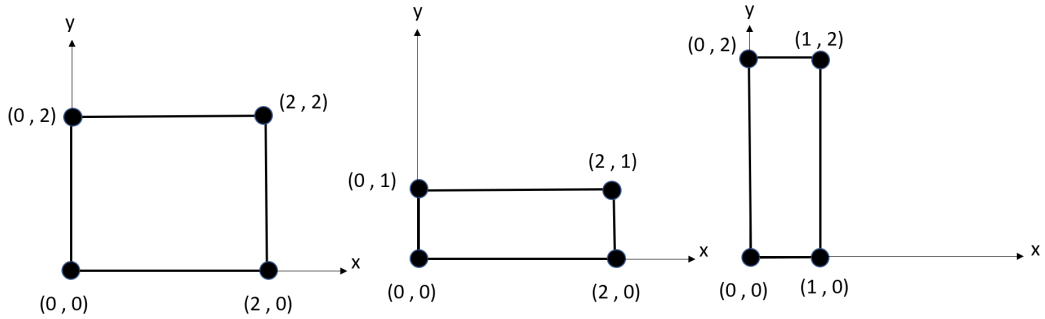


Figure 3.12. Integration domains for problem 1

to a perfectly square canonical element of side 2.0.

3. Compute the integral of the function

$$f(r, s) = rs$$

over the triangular domain shown in the figure

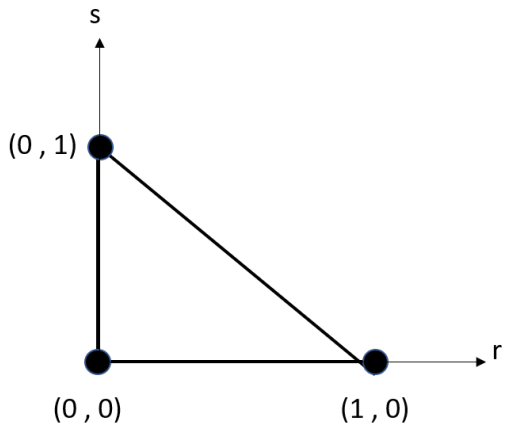


Figure 3.13. Integration domains for problem 1

Chapter 4

The Boundary Value Problem

Preliminary

In this section we review, in terms of governing equations and boundary conditions, the boundary value problem corresponding to the model of theory of elasticity. The first part of the chapter presents the stress and displacements formulation of the problem leading to the strong form. In the second part the problem is re-written in a weak sense leading to a form suitable for finite element algorithms. The chapter concludes with a third equivalent formulation, namely the principle of virtual work, which is obtained after minimization of the total potential energy functional. In a sense, this last statement of equilibrium is also a weak formulation useful for finite element implementation.

At the end of this chapter[1] the student should be able to:

- Understand the fundamental hypothesis leading to the model of linearized theory of elasticity.

[1] **This chapter, together with theoretical and computational learning activities is complemented by Jupyter Notebook 7.**

- Understand the fundamental elements, including those of stress and strain, conforming the model of linearized theory of elasticity.
- Understand the problem of elasticity as a boundary value problem which can be equivalently formulated in differential and integral forms.
- Recognize valid solutions of the BVP specified over a particular domain and set of boundary conditions.

4.1 Brief review of the linearized theory of elasticity model

4.1.1 Stress formulation

Here we present a brief description of the boundary value problem governing the mechanical response of an elastic body. For a full discussion of the model and its mathematical aspects the reader is referred to [3]. The equations are defined for a material point defined by its position $\mathbf{x}$ inside a body occupying a volume V and bounded by the surface S as shown in fig. 4.1. In a mathematical sense V is the problem domain and the surface S is the boundary where the solution is expected to be given in terms of prescribed (boundary) conditions (BCs).

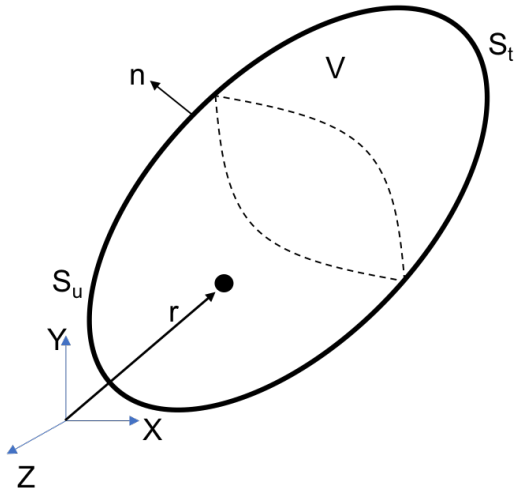


Figure 4.1. Problem domain.

The governing equations (in terms of stresses) stem from application of the principle of conservation of linear momentum and conservation of moment of linear momentum to an arbitrary portion of the domain. Conservation of linear momentum leads to a set of 3 partial differential equations in the components of the stress tensor σ_{ij} while conservation of moment of linear momentum leads to the symmetric character of this stress tensor. The resulting equations are given by:

$$\begin{aligned} &\sigma_{ij,j} + f_i = \rho \ddot{u}_i \quad \forall\, \mathbf{x} \in V,\, t \in \mathbb{R}^+ \\ &\sigma_{ij} = \sigma_{ji}. \end{aligned} \tag{4.1}$$

In 4.1 f_i is the vector of body forces and u_i is the displacements vector where each dot represents a time differentiation. The set of governing equations in this stress formulation is complemented by boundary conditions specified in terms of the tractions vector $t_i^{\hat{n}}$ associated with a surface S with normal direction $\hat{n}_j$. At any given surface this tractions vector is related to the stress tensor by:

$$t_i^{\hat{n}} = \sigma_{ij}\hat{n}_j \quad \forall \in \mathbf{x} \in S.$$

It must be noted that eq. (4.1) are a system of 6 equations in 12 unknowns (i.e., the 9 components of the stress tensor and the 3 components of the displacements vector) and therefore it is undetermined. To eliminate such indetermination we must relate the stresses to the changes in configuration. In the linearized model of elasticity the local changes in configuration are described in terms of changes of magnitude and internal distortions among material fibbers emanating from the material point. This kinematic information is described in terms of the strain tensor defined in terms of components of spatial gradients of the displacement vector as follows:

$$\varepsilon_{ij} = \frac{1}{2}(u_{i,j} + u_{j,i}). \tag{4.2}$$

Note that the term ϵ_{ij} is the symmetric component of the displacements gradient tensor. The components of the strain tensor describe the distortions and changes in magnitude (volumetric changes) of the material point in the continuum model. To complete the problem formulation the local changes in configuration must be related to the stress tensor at the material point through a constitutive law. The simplest stress-strain (constitutive) relationship is given by Hooke's law[2] as follows:

$$\sigma_{ij} = 2\mu\varepsilon_{ij} + \lambda\varepsilon_{kk}\delta_{ij} \quad . \tag{4.3}$$

and where μ and λ are material constants.

Note that eq. (4.1) to eq. (4.3) involves now a total of 18 equations in 18 unknowns and mathematically it is now a solvable system. However to find unique solutions the elastic solid must be subjected to properly specified boundary conditions.

[2]Despite the name of *law* used, this relation is not always valid, but is a good approximation for small strains.

4.1.2 Displacement formulation

The equilibrium equations in terms of stresses given by eq. (4.1) can be simplified after substituting eq. (4.2) in eq. (4.3) and the result in eq. (4.1) yielding after some manipulation:

$$(\lambda + \mu)u_{j,ij} + \mu u_{i,jj} + f_i = \rho\ddot{u}_i \quad \forall \mathbf{x} \in V,\, t \in \mathbb{R}^+. \tag{4.4}$$

Note that now eq. (4.4) simultaneously satisfies equilibrium, kinematic relations and the constitutive law and they constitute a system of 3 partial second order differential equations in the 3 components of the displacement vector. For a unique solution these governing equations must be complemented with prescribed boundary values of the displacement or its first order derivatives as given by:

$$\begin{aligned} t_i^{\hat{n}} &= \sigma_{ij}\hat{n}_{ij} \quad \forall\ \mathbf{x} \in S_t \\ u_i &= \bar{u}_i \quad \forall \mathbf{x} \in S_u \end{aligned} \tag{4.5}$$

and where $S_t \cup S_u = S$ and $S_t \cap S_u = \emptyset$.

Notice that strictly speaking the problem involves time derivatives through the term $\rho\ddot{u}_i$ thus requiring also the specification of initial conditions in $t = 0$ for u_i and $\dot{u}_i$. Here we will drop the inertial term and assume a static idealization and the problem becomes a boundary value problem (BVP) summarized next:

$$\begin{aligned} &(\lambda + \mu)\, u_{j,ij} + \mu u_{i,jj} + f_i = 0 \quad \forall \mathbf{x} \in V \\ &t_i^{\hat{n}} = \sigma_{ij}\hat{n}_{ij} \quad \forall\ \mathbf{x} \in S_t \\ &u_i = \bar{u}_i \quad \forall \mathbf{x} \in S_u \end{aligned} \tag{4.6}$$

In 4.6 the boundary conditions specified by the traction vector $t_i^{\hat{n}}$

$$t_i^{\hat{n}} = \mu(u_{i,j} + u_{j,i})\hat{n}_j + \lambda u_{k,k}\delta_{ij}\hat{n}_j \ ,$$

correspond to the natural boundary conditions. In fact these actually involve first order displacements derivative and as such they are Neumann boundary condition on u_i while those specified in terms of the displacements vector $\bar{u}_i$ represent the essential boundary conditions.

> The classical finite element algorithm to solve problems of elasticity interpolates nodal displacements and then computes derivatives to find strain and stress distributions as secondary variables, thus it is a displacement based solution analogous to Navier equations.

Example: Two-dimensional idealization: Simple wedge under self-equilibrated loads The general equations governing the elasticity BVP can be represented by simplified two-dimensional versions under certain conditions satisfied by the geometry of the problem domain, the distribution of the external loads or specific values of some field components. In the most general cases these correspond to plane strain, plane stress and axy-symmetric problems. In the following example we present the special case of a two-dimensional idealization in a plane strain problem in which $\epsilon_{zz} = 0$.

Consider the double wedge of side ℓ and internal angle 2ϕ shown in fig. 4.2. It is assumed to be contained in the $X - Y$ plane, with loading conditions satisfying a plane strain idealization. The material is elastic and described by Lame constants λ and μ. The wedge is loaded by uniform tractions of intensity S applied over its four faces in such a way that the wedge is self-equilibrated. We wish to find the elasticity solution for the stress, strain and displacement fields throughout the problem domain.

Under two-dimensional plane strain conditions the general three-dimensional

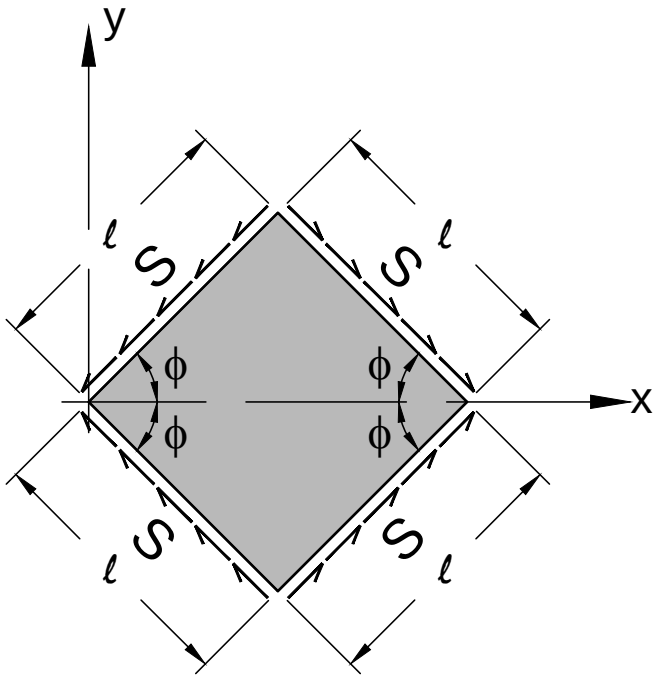


Figure 4.2. 2D Self-equilibrated wedge.

stress equilibrium equations 4.1 reduce to:

$$\begin{aligned} \frac{\partial \sigma_{xx}}{\partial x} + \frac{\partial \tau_{xy}}{\partial y} &= 0 \\ \frac{\partial \tau_{xy}}{\partial x} + \frac{\partial \sigma_{yy}}{\partial y} &= 0 \end{aligned} \tag{4.7}$$

while the kinematic relation 4.2 reads

$$\begin{aligned} \epsilon_{xx} &= \frac{\partial u}{\partial x} \\ \epsilon_{yy} &= \frac{\partial v}{\partial y} \\ \gamma_{xy} &= \frac{\partial u}{\partial y} + \frac{\partial v}{\partial x} \end{aligned} \tag{4.8}$$

where u and v are respectively the horizontal and vertical components of the displacement vector.

Stress field

The stress field can be obtained by simple inspection from the traction boundary conditions prescribed over the inclined surfaces. Expressing the surface forces in terms of stresses gives:

$$\sum F_x = 0 \longrightarrow -\ell S\cos(\phi) + \sigma_{xx}\ell\sin(\phi) = 0$$
$$\sum F_y = 0 \longrightarrow -\ell S\sin(\phi) - \sigma_{yy}\ell\cos(\phi) = 0$$

producing the following stress solution:

$$\begin{aligned} \sigma_{xx} &= S\cot(\phi) \\ \sigma_{yy} &= -S\tan(\phi) \\ \tau_{xy} &= 0\,. \end{aligned} \tag{4.9}$$

In eq. (4.9) the condition $\tau_{xy} = 0$ has been assumed as suggested by the symmetries in the problem. Clearly the above stress field satisfies the local equilibrium equations, however for it to be the actual solution to the BVP it must also satisfy the boundary conditions.

Traction boundary conditions

Let us verify that the above stress solution satisfies the traction BCs using the expression:

$$t_i^{\hat{n}} = \sigma_{ij}\hat{n}_{ij}.$$

Denoting the outward normals to the inclined surfaces of the wedge by

$\hat{n}^1$, $\hat{n}^2$, $\hat{n}^3$, $\hat{n}^4$ these are given by:

$$\begin{aligned}
\hat{n}^1 &= -\sin(\phi)\hat{e}_x + \cos(\phi)\hat{e}_y \\
\hat{n}^2 &= -\sin(\phi)\hat{e}_x - \cos(\phi)\hat{e}_y \\
\hat{n}^3 &= +\sin(\phi)\hat{e}_x + \cos(\phi)\hat{e}_y \\
\hat{n}^4 &= +\sin(\phi)\hat{e}_x - \cos(\phi)\hat{e}_y \ ,
\end{aligned}$$

where $\hat{e}_x$ and $\hat{e}_y$ are the reference unit vectors. Now, the components of the traction vector follow directly.

For the face with normal $\hat{n}^1$ we have

$$\begin{aligned}
t_x &= -S\cos(\phi) \\
t_y &= -S\sin(\phi)
\end{aligned}$$

similarly, over the face with normal $\hat{n}^2$ the traction reads

$$\begin{aligned}
t_x &= -S\cos(\phi) \\
t_y &= +S\sin(\phi)
\end{aligned}$$

over the face with normal $\hat{n}^3$

$$\begin{aligned}
t_x &= +S\cos(\phi) \\
t_y &= -S\sin(\phi)
\end{aligned}$$

and finally, over the face with normal $\hat{n}^4$;

$$\begin{aligned}
t_x &= +S\cos(\phi) \\
t_y &= +S\sin(\phi) \ .
\end{aligned}$$

Note that the above set of tractions found from the relation:

$$t_i^{\hat{n}} = \sigma_{ij}\hat{n}_{ij}$$

and after using the assumed stress field satisfies the problem boundary condition. Thus the stress field given by eq. (4.9) is the unique solution to the BVP for the self equilibrated wedge.

Strain field

The strain field can be obtained after using the stress solution found in eq. (4.9) together with the constitutive law given by eq. (4.3) which for a plane strain idealization takes the form:

$$\begin{aligned} \epsilon_{xx} &= \frac{1}{E}(\sigma_{xx} - \nu\sigma_{yy}) \\ \epsilon_{yy} &= \frac{1}{E}(\sigma_{yy} - \nu\sigma_{xx}) \\ \gamma_{xy} &= \frac{\tau_{xy}}{\mu}. \end{aligned} \tag{4.10}$$

Particularizing we have:

$$\begin{aligned} \epsilon_{xx} &= +\frac{S}{E}\left[\cot(\phi) + \nu\tan(\phi)\right] = +\frac{S}{E}K_1(\nu,\phi) \\ \epsilon_{yy} &= -\frac{S}{E}\left[\tan(\phi) + \nu\cot(\phi)\right] = -\frac{S}{E}K_2(\nu,\phi) \\ \gamma_{xy} &= 0. \end{aligned} \tag{4.11}$$

Displacement field

The displacement field is obtained after direct integration of the strains after using the fact that

$$\mathrm{d}u_i = \epsilon_{ij}\,\mathrm{d}x_j + \omega_{ij}\,\mathrm{d}x_j$$

and the condition $\omega_{xy} = 0$ (as a result of $\gamma_{xy} = 0$ which gives:

$$u = +\frac{S}{E}K_1(\nu,\phi)x + A$$
$$v = -\frac{S}{E}K_2(\nu,\phi)y + B$$

and where A and B are integration constants while $K_1(\nu,\phi)$ and $K_2(\nu,\phi)$ are also constants that depend on Poisson's ratio and the wedge internal angle.

From the condition $u = 0$ at $x = \ell\cos(\phi)$ we have that

$$A = -\frac{S}{E}K_1(\nu,\phi)\ell\cos(\phi)$$

then it follows that

$$u = \frac{S}{E}K_1(\nu,\phi)(x - \ell\cos(\phi)).$$

Similarly, from the condition $v = 0$ at $y = 0$ we have that $B = 0$ from which

$$v = -\frac{S}{E}K_2(\nu,\phi)y$$

Notebook 7 in the REPO summarizes the BVP and uses interpolation to visualize the elastic field in the wedge solution.

4.2 Strong and weak forms of the BVP

In this section we will present the elasticity BVP in an equivalent form suitable for finite element formulation. For that purpose, and just for completeness, we will repeat the differential formulation recently described and will immediately re-write it in an equivalent integral representation.

4.2.1 Strong form

The so-called strong form of the boundary value problem corresponds to the differential formulation just described, involving the governing equations and boundary conditions for a well posed problem. In the mathematical literature it is customary to denote the strong form as $\{S\}$ and express it like:

Given f_i, $t_i^{\hat{n}}$ and $\bar{u}_i$ find $u_i : V \to \mathbb{R}$ such:

$$\begin{aligned} &(\lambda+\mu)u_{j,ij} + \mu u_{i,jj} + f_i = 0 \quad \forall \mathbf{x} \in V \\ &t_i^{\hat{n}} = \sigma_{ij}\hat{n}_{ij} \quad \forall \mathbf{x} \in S_t \\ &u_i = \bar{u}_i \quad \forall \mathbf{x} \in S_u \end{aligned} \tag{4.12}$$

4.2.2 Weak form

A note on solution functions.

- We are interested in developing methods to obtain approximate solutions to $\{S\}$.
- The FEM is formulated starting from a statement equivalent to $\{S\}$ in which we use trial functions until certain prescribed conditions are met.

- We will look for solutions u_i subject to the following conditions:

$$u_i = \bar{u}_i \qquad \text{in} \qquad S_u$$
$$\int_S \left(\frac{\partial u_i}{\partial x_j}\right)^2 \mathrm{d}S < \infty$$

The first condition corresponds to the satisfaction of the essential boundary condition, while the second corresponds to the functions being square integrable. The space of functions satisfying the above two conditions is denoted by ς and formally defined like

$$\varsigma = \{u_i \,|u_i \in H, u_i = \bar{u}_i \in S_u\} \quad .$$

On the other hand, in order to validate the introduced trial functions we also need testing functions w_i also called in the FEM literature weighting or distribution functions. These functions are arbitrary apart from having to satisfy the following conditions:

$$w_i = 0 \quad in \quad S_u$$
$$\int_S \left(\frac{\partial w_i}{\partial x_j}\right)^2 \mathrm{d}S < \infty$$

In what follows we formally denote the space of these functions by V and define it like

$$V = \{w_i \,|w_i \in H, u_i = w_i = 0 \in S_u\} \quad .$$

Here we will show that the equilibrium statement represented in the differential formulation can be described in alternative forms. In such description the continuity requirement for the trial functions is weaker than in the strong form leading to the term "weak" statement. Here this alternative representation will be denoted like $\{W\}$ and it reads;

Given f_i, $t_i^{\hat{n}}$ and $\bar{u}_i$ find $u_i : V \to \mathbb{R}$ and $\forall w_i \in V$ such:

$$\int_V \sigma_{ij} w_{i,j} \, \mathrm{d}V - \int_V f_i w_i \, dV - \int_{S_t} t_i^{\hat{n}} w_i \, \mathrm{d}S = 0$$

Proof 1:

Let $u_i \in \varsigma$ be a solution to $\{S\}$ and let $w_i \in V$. Forming the inner product of the equilibrium statement given in eq. (4.1) with w_i and forcing the integral over the domain to be zero we have

$$\int_V (\sigma_{ij,j} + f_i) w_i \, dV = 0 \ ,$$

expanding the terms in the integrand and integrating by parts the first term on the left we have

$$\int_V \sigma_{ij,j} w_i \ \mathrm{d}V + \int_V f_i w_i \ \mathrm{d}V = 0 \, ,$$
$$- \int_V w_{i,j} \sigma_{ij} \ \mathrm{d}V + \int_S \sigma_{ij} \hat{n}_j w_i \ \mathrm{d}S + \int_V w_i f_i \ \mathrm{d}V = 0 \, ,$$

since $w_i \in V$ it follows that $w_i = 0$ in S_u from which

$$\int_V \sigma_{ij} w_{i,j} \ \mathrm{d}V - \int_V f_i w_i \ \mathrm{d}V - \int_{S_t} t_i^{\hat{n}} w_i \ \mathrm{d}S = 0 \, . \tag{4.13}$$

Now, considering that u_i is solution of the strong form $\{S\}$ it must satisfy $u_i = \bar{u}_i \quad \in \quad S_u$ and as a result $u_i \in \varsigma$. On the other hand, since u_i satisfies eq. (4.13) $\forall w_i \in V$ we have that u_i satisfies the definition of weak solution specified in $\{W\}$.

Proof 2:

Let u_i be a solution of $\{W\}$ and thus $u_i \in \varsigma$ which means that

$$u_i = \bar{u}_i \quad \in \quad S_u$$

and that it satisfies

$$\int_V \sigma_{ij} w_{i,j} \ \mathrm{d}V - \int_V f_i w_i \ \mathrm{d}V - \int_{S_t} t_i^n w_i \ \mathrm{d}S = 0 \ ,$$

integrating by parts,

$$-\int_V \sigma_{ij,j} w_i \,\mathrm{d}V + \int_S \sigma_{ij} n_j w_i \,\mathrm{d}S - \int_V f_i w_i \,\mathrm{d}V - \int_{S_t} t_i^n w_i \,\mathrm{d}S = 0$$

Since $w_i \in V$ we have that $w_i = 0$ in S_u and therefore

$$\int_V w_i(\sigma_{ij,j} + f_i) \,\mathrm{d}V + \int_{S_t} w_i(\sigma_{ij} n_j - t_i^n) \,\mathrm{d}S = 0$$

from which

$$\begin{aligned} &\sigma_{ij,j} + f_i = 0 \quad \mathbf{x} \in V \\ &t_i^n = \sigma_{ij} n_j \quad \forall \mathbf{x} \in S_t \\ &u_i = \bar{u}_i \quad \forall \mathbf{x} \in S_u \end{aligned} \tag{4.14}$$

which is once again the strong form of the problem given in eq. (4.1).

4.3 Variational formulation

In this section we formulate the boundary value problem using the approach of the calculus of variations in which the governing PDEs and boundary conditions are obtained after finding the minimum (or maximum) of a functional according to a variational principle[3].

[3] According to Wikipedia [4]

> A variational principle is a scientific principle used within the calculus of variations, which develops general methods for finding functions which minimize or maximize the value of quantities that depend upon those functions. For example, to answer this question: "What is the shape of a chain suspended at both ends?" we can use the variational principle that the shape must minimize the gravitational potential energy.
>
> According to Cornelius Lanczos, any physical law which can be expressed

We will see that the weak form (and therefore also the strong form) can be obtained alternatively through the process of finding extreme values for a functional. We will illustrate this idea for the general case of theory of elasticity and then we will present particular examples.

Some vague definitions in the calculus of variations

In variational calculus a **functional** can be understood as a "function" having as independent variables or arguments a space of vector functions and producing as a result (or dependent variable) a scalar. For instance, in the particular case of the theory of elasticity such a "function" corresponds to the total potential energy functional Π given by;

$$\Pi(u_i) = \frac{1}{2}\int_V \sigma_{ij}\varepsilon_{ij}\,\mathrm{d}V - \int_V f_i\,u_i\,\mathrm{d}V - \int_S t_i^{(n)}u_i\,\mathrm{d}S \tag{4.15}$$

and where the first term in the left hand side corresponds to the internal strain energy, while the last two terms are the work done by the external body and traction forces. The above functional has as independent variables the displacement vector and its spatial derivatives. This is indicated by the presence of the displacement vector u_i in the expression $\Pi(u_i)$.

In variational calculus we are interested in finding a function u_i that renders the functional Π a maximum or a minimum. In loose terms, the analogous to the differential operator in calculus of functions is now termed the variational operator δ (i.e., δ is analogous to $\frac{\partial}{\partial x_i}$). As such $\delta\Pi$ acts over the function u_i and its derivatives as follows

$$\delta\Pi = \frac{\partial\Pi}{\partial u_i}\,\delta u_i + \frac{\partial\Pi}{\partial\left(\frac{\partial u_i}{\partial x_j}\right)}\delta\left(\frac{\partial u_i}{\partial x_j}\right) + \cdots + \frac{\partial\Pi}{\partial\left(\frac{\partial^n u_i}{\partial x_j\cdots\partial x_k}\right)}\delta\left(\frac{\partial^n u_i}{\partial x_j\cdots\partial x_k}\right)\ .$$

The following rules apply to the variational operator δ:

as a variational principle describes an expression which is self-adjoint. These expressions are also called Hermitian. Such an expression describes an invariant under a Hermitian transformation.

- For functionals Π and Φ it follows that

$$\delta(\Pi + \Phi) = \delta\Pi + \delta\Phi$$

- For functionals Π and Φ it follows that

$$\delta(\Pi\Phi) = \delta\Pi\,\Phi + \Pi\,\delta\Phi$$

- For a functional Π and an integer n it follows that

$$\delta(\Pi^n) = n(\Pi^{n-1})\,\delta\Pi$$

- For a functional Π it follows that

$$\delta \int \Pi \,\mathrm{d}x = \int \delta\Pi \,\mathrm{d}x$$

If the variational operator is applied to the functional $\Pi(u_i)$ it produces functions or variations in u_i which are arbitrary and such $\delta u_i \in V$ and $\delta u_i = 0$ in S_u.

To find an extreme function in the calculus of variations we proceed like in differential calculus. Here we compute the first variation of the functional $\delta\Pi$ and solve the variational equation

$$\delta\Pi = 0 \tag{4.16}$$

in the unknown function u_i.

In the particular case of the total potential energy functional Π this yields

$$\int_V \sigma_{ij}\,\delta\varepsilon_{ij}\,\mathrm{d}V - \int_V f_i\,\delta u_i\,\mathrm{d}V - \int_{S_t} t_i^{(n)}\,\delta u_i\,\mathrm{d}S = 0 \tag{4.17}$$

where we recognize the weak form of the BVP stated previously thus it is equivalent to the strong form:

$$\begin{aligned} &\sigma_{ij,j} + f_i = 0 \quad \mathbf{x} \in V \\ &t_i^n = \sigma_{ij} n_j \quad \forall \mathbf{x} \in S_t \\ &u_i = \bar{u}_i \quad \forall \mathbf{x} \in S_u \end{aligned} \tag{4.18}$$

It becomes evident that the functions δu_i in eq. (4.17) play the role of the test functions w_i introduced in the weak form. On the other hand, since we have already shown that the weak and strong forms are equivalent we conclude that having the functional and the essential boundary conditions is equivalent to having the strong form of the problem. This result can be formalized in the following principle.

4.3.1 Principle of minimum potential energy

In the theory of elasticity the total potential energy Π is the result of adding the elastic strain energy which is stored in the body upon deformation and the potential energy (work) imparted to the body by the applied forces. The principle of virtual work states that the body is in equilibrium when this total potential energy reaches a minimum. This is equivalent to stating that an equilibrium configuration is attained when an infinitesimal variation from the position of minimum potential energy involves null changes in energy. This implies the variational condition:

$$\delta\Pi = 0. \tag{4.19}$$

The above principle leads to the so-called principle of virtual displacements stated as follows[4]:

"The equilibrium of the body requires that for any compatible small virtual displacements satisfying the condition of being zero at S_u, imposed on the body in its state of equilibrium, the total internal virtual work is equal to the total external virtual work"

$$\int_V \sigma_{ij}\,\delta\varepsilon_{ij}\,\mathrm{d}V - \int_V f_i\delta u_i\,\mathrm{d}V - \int_{S_t} t_i^{(n)}\,\delta u_i\,\mathrm{d}S = 0$$

where δu_i are the virtual displacements and $\delta\varepsilon_{ij}$ are the corresponding virtual strains.

[4]See Bathe pp 156.

Comparing the virtual work principle with the weak formulation given in eq. (4.13) we identify δu_i with the test functions w_i. As such the PVW takes the form of a powerful tool to test if a body is in equilibrium for a given solution (represented by the trial functions). In what follows we illustrate the use of the principle through some examples corresponding to problems in Bathe's textbook.

Problem[5]

Establish the differential equation of equilibrium of the problem shown and the boundary conditions. Determine whether the differential operator of the problem is symmetric and positive definite and prove your answer.[5]

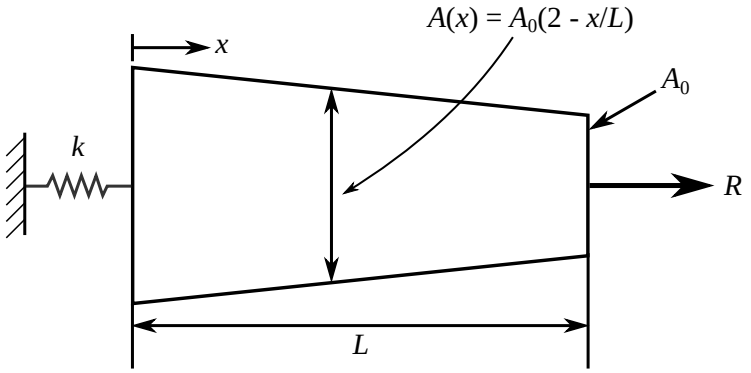


Figure 4.3. Rod with varying cross-sectional area. The Young's modulus is E.

$$\Pi = \frac{1}{2}\int_0^L \sigma_{xx}\varepsilon_{xx}A(x)\,\mathrm{d}x + \frac{1}{2}ku_0^2 - Ru_L$$
$$= \frac{1}{2}\int_0^L EA(x)\left(\frac{\mathrm{d}u}{\mathrm{d}x}\right)^2 \mathrm{d}x + \frac{1}{2}ku_0^2 - Ru_L \ .$$

[5] 3.15 from Finite Element Procedures

The first variation for this functional is

$$\delta\Pi = \int_0^L EA(x)\frac{\mathrm{d}u}{\mathrm{d}x}\frac{\mathrm{d}\,\delta u}{\mathrm{d}x}\,\mathrm{d}x + ku_0\,\delta u_0 - R\,\delta u_L \quad .$$

If we integrate by parts, we obtain

$$\delta\Pi = -\int_0^L \frac{\mathrm{d}}{\mathrm{d}x}\left[EA(x)\frac{\mathrm{d}u}{\mathrm{d}x}\right]\delta u\,\mathrm{d}x + \left.EA(x)\frac{\mathrm{d}u}{\mathrm{d}x}\,\delta u\right|_0^L + ku_0\,\delta u_0 - R\,\delta u_L$$

from which

$$\begin{aligned}
&\frac{\mathrm{d}}{\mathrm{d}x}\left[EA(x)\frac{\mathrm{d}u}{\mathrm{d}x}\right] = 0\\
&\left.EA(x)\frac{\mathrm{d}u}{\mathrm{d}x}\right|_{x=0} = ku_0\\
&\left.EA(x)\frac{\mathrm{d}u}{\mathrm{d}x}\right|_{x=L} = R
\end{aligned}$$

The statements:

$$\int_V \sigma_{ij}\,\delta\varepsilon_{ij}\,\mathrm{d}V - \int_V f_i\,\delta u_i\,\mathrm{d}V - \int_{S_t} t_i^{(n)}\,\delta u_i\,\mathrm{d}S = 0$$

complemented by the condition $u_i = \overline{u}_i \quad \forall \mathbf{x} \in S_u$
and

$$\begin{aligned}
&\sigma_{ij,j} + f_i = 0 \quad \mathbf{x} \in V\\
&t_i^n = \sigma_{ij}n_j \quad \forall \mathbf{x} \in S_t\\
&u_i = \bar{u}_i \quad \forall \mathbf{x} \in S_u
\end{aligned}$$

are equivalent.

The Hu-Washizu Variational Principle-Alternative formulation of the variational statement[6] Consider the Hu-Washizu functional:

$$\Pi^* = \Pi - \int_V \lambda_{ij}^{\varepsilon}(\varepsilon_{ij} - L_{ijk}u_k)\,\mathrm{d}V - \int_{S_u} \lambda_i^u(u_i^{S_u} - \bar{u}_i)\,\mathrm{d}S \tag{4.20}$$

where

- Π: is the potential energy functional.
- L_{ijk} is a differential operator such $\varepsilon_{ij} = L_{ijk}u_k$.
- S_u surface where essential boundary conditions are prescribed.
- $\lambda_{ij}^{\varepsilon}$ and λ_i^u are Lagrange multipliers.

Using the condition $\delta\Pi^* = 0$ derive for the interior of the body the equilibrium equations

$$\sigma_{ij,j} + f_i = 0$$

the strain-displacement relationship

$$\varepsilon_{ij} = L_{ijk}u_k$$

and the constitutive equation

$$\sigma_{ij} = C_{ijkl}\varepsilon_{kl}$$

and at the surface of the body the relation between the stress tensor and the applied tractions vector at S_t

$$t_i = \sigma_{ij}n_j$$

the relation between the stress tensor and the unknown tractions vector (or reactions) at S_u

$$t_i = \tilde{\sigma_{ij}}n_j$$

and the essential boundary condition at S_u

$$u_i = \tilde{u}_i$$

[6] 4.35 from Finite Element Procedures

We want to determine the Euler equations resulting from the condition $\delta\Pi^* = 0$. Applying the variational operator we have

$$\begin{aligned}\delta\Pi^* = \delta\Pi &- \int_V \delta\lambda_{ij}^{\varepsilon}\,(\varepsilon_{ij} - L_{ijk}u_k)\,\mathrm{d}V - \int_V \lambda_{ij}^{\varepsilon}(\delta\varepsilon_{ij} - L_{ijk}\,\delta u_k)\,\mathrm{d}V \\ &- \int_V \delta\lambda_i^u\,(u_i^{S_u} - \bar{u}_i)\,\mathrm{d}S - \int_V \lambda_i^u\,\delta u_i^{S_u}\,\mathrm{d}S\end{aligned} \tag{4.21}$$

and

$$\begin{aligned}\delta\Pi^* = &\int_V C_{ijkl}\varepsilon_{kl}\,\delta\varepsilon_{ij}\,\mathrm{d}V - \int_{S_t} t_i\,\delta u_i\,\mathrm{d}S - \int_V f_i\,\delta u_i\,\mathrm{d}V \\ &- \int_V \lambda_{ij}^{\varepsilon}\,\delta\varepsilon_{ij}\,\mathrm{d}V + \int_V \lambda_{ij}^{\varepsilon}L_{ijk}\,\delta u_k\,\mathrm{d}V - \int_V \delta\lambda_{ij}^{\varepsilon}(\varepsilon_{ij} \\ &- L_{ijk}u_k)\,\mathrm{d}V - \int_S \delta\lambda_i^u\,(u_i^{S_u} - \bar{u}_i)\,\mathrm{d}S - \int_{S_u} \lambda_i^u\,\delta u_i\,\mathrm{d}S = 0\end{aligned} \tag{4.22}$$

using

$$\int_V (\lambda_{ij}^{\varepsilon}\,\delta u_i)_{,j}\,\mathrm{d}V = \int_V \lambda_{ij}^{\varepsilon}\,\delta u_{i,j}\,\mathrm{d}V + \int_V \lambda_{ij,j}^{\varepsilon}\,\delta u_i\,\mathrm{d}V$$

In the above we can write

$$\begin{aligned}\int_V \lambda_{ij}^{\varepsilon}L_{ijk}\,\delta u_k\,\mathrm{d}V &= \int_V (\lambda_{ij}^{\varepsilon}\,\delta u_i)_{,j}\,\mathrm{d}V - \int_V \lambda_{ij,j}^{\varepsilon}\delta u_i\,\mathrm{d}V \\ &= \int_{S_t} \lambda_{ij}^{\varepsilon}\,\delta u_i\,\hat{n}_j\,\mathrm{d}S - \int_V \lambda_{ij,j}^{\varepsilon}\,\delta u_i\,\mathrm{d}V\end{aligned}$$

therefore

$$\begin{aligned}\delta\Pi^* = &\int_V (C_{ijkl}\varepsilon_{kl} - \lambda^\varepsilon_{ij})\,\delta\varepsilon_{ij}\,\mathrm{d}V - \int_{S_t} t_i\,\delta u_i\,\mathrm{d}S - \int_V f_i\,\delta u_i\,\mathrm{d}V \\ &+ \int_{S_t} \lambda^\varepsilon_{ij}\,\delta u_i\,\hat{n}_j\,\mathrm{d}S - \int_V \lambda^\varepsilon_{ij,j}\delta u_i\,\mathrm{d}V - \int_V \delta\lambda^\varepsilon_{ij}(\varepsilon_{ij} \\ &- L_{ijk}u_k)\,\mathrm{d}V - \int_{S_u} \delta\lambda^u_i\,(u_i^{S_u} - \bar{u}_i)\,\mathrm{d}S - \int_{S_u} \lambda^u_i\,\delta u_i\,\mathrm{d}S = 0 \\ = &\int_V (C_{ijkl}\varepsilon_{kl} - \lambda^\varepsilon_{ij})\,\delta\varepsilon_{ij}\,\mathrm{d}V + \int_{S_t} (\lambda^\varepsilon_{ij}\hat{n}_j - t_i)\delta u_i\,\mathrm{d}S \\ &- \int_V (\lambda^\varepsilon_{ij,j} + f_i)\,\delta u_i\,\mathrm{d}V - \int_V (\varepsilon_{ij} - L_{ijk}u_k)\,\delta\lambda^\varepsilon_{ij}\,\mathrm{d}V \\ &- \int_{S_u} (u_i^{S_u} - \bar{u}_i)\,\delta\lambda^u_i\,\mathrm{d}S - \cancel{\int_{S_u} \lambda^u_i\,\delta u_i^{S_u}\,\mathrm{d}S} = 0\end{aligned}$$

Now, imposing the conditions $\delta\varepsilon_{ij} \neq 0$, $\delta\lambda^\varepsilon_{ij} \neq 0$, $varu_i \neq 0$ in S_t, $\delta u_i \neq 0$ in V and $\delta\lambda^u_i \neq 0$ in S_u we have

$$\lambda^\varepsilon_{ij} = C_{ijkl}\varepsilon_{kl} \tag{4.23}$$

$$\varepsilon_{ij} = L_{ijk}u_k \tag{4.24}$$

$$t_i = \lambda^\varepsilon_{ij}\hat{n}_j \tag{4.25}$$

$$\lambda^\varepsilon_{ij,j} + f_i = 0 \tag{4.26}$$

$$u_i^{S_u} = \bar{u}_i \ . \tag{4.27}$$

Proposed problems

1. For the following displacements field:

$$\mathbf{u} = a(x^2 - 5y^2)\hat{\mathbf{i}} + (2axy)\hat{\mathbf{j}}$$

   (a) Find the strain tensor.

(b) Find the principal strains.

(c) Find the principal stresses.

2. Figure 4.4 shows a cantilever beam of unitary cross section with a distributed load applied at the free edge and following a parabolic distribution given by:

$$\mathbf{t} = \frac{Pc^2}{2I}\left(1 - \frac{y^2}{c^2}\right)\hat{\mathbf{j}}$$

.

The stress solution is given by:

$$\begin{aligned}\sigma_{xx} &= -\frac{P}{I}xy \ , \\ \sigma_{yy} &= 0 \ , \\ \sigma_{xy} &= -\frac{P}{2I}(c^2 - y^2) \ .\end{aligned}$$

while the displacement field reads:

$$\begin{aligned}u_x &= \left(\frac{1}{G} - \frac{\nu}{E}\right)\frac{Py^3}{6I} + \left(\frac{l^2}{E} - \frac{x^2}{E} - \frac{c^2}{G}\right)\frac{Py}{2I} \ , \\ u_y &= \frac{\nu Pxy^2}{2EI} + \frac{Px^3}{6EI} - \frac{Pl^2x}{2EI} + \frac{Pl^3}{3EI} \ ,\end{aligned}$$

where $G = E/(2 + 2\nu)$ is the shear modulus and I is the moment of inertia of the cross section.

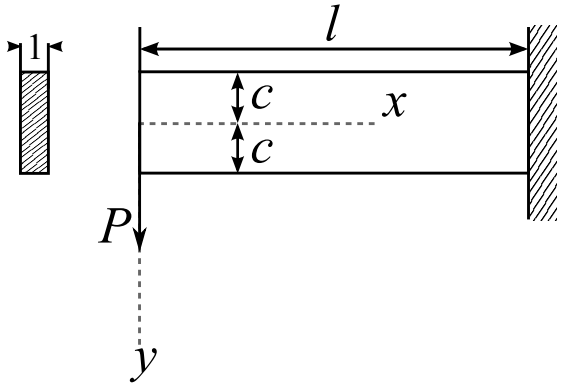


Figure 4.4. Cantilever beam with a distributed load at the end.

(a) Identify the displacement and traction boundary conditions of the problem.

(b) Verify that the stress and displacement solutions satisfy the proper boundary conditions.

(c) Find the region of the beam under tension and compression and thus its neutral axis. Find the equation corresponding to the displacement of the neutral axis and compare it with beam theory.

Chapter 5

FEM formulation of the elasticity BVP

Preliminary

The previous chapter covered the boundary value problem for the model of theory of elasticity. It was shown that the model could equivalently be written in strong form (i.e., governing equations and boundary conditions) or in weak form. It was also shown that a particular interpretation of the weak form was that of the principle of virtual displacements. In this chapter we start from this principle and use interpolation theory to approximate both physical and virtual functions. As a result, this weak form of the BVP becomes a system of linear equations in the interpolation parameters that artificially enforce the principle of virtual displacements. The resulting finite element equations are shown to be equivalent to those governing the mechanical equilibrium of a discrete system of particles. To facilitate conceptual understanding we find first the equations for a single element and then proceed to consider the full mesh after invoking equilibrium and displacement compatibility conditions along element interfaces.

At the end of this chapter[1] the student should be able to:

- Use interpolation theory to write the finite element based discrete version of the principle of virtual work.
- Recognize the finite element equilibrium equations of the elasticity BVP as physically analogous to those of a discrete system of particles.
- Understand the computational steps required for the implementation of the finite element. equilibrium equations in a computer language like Python.
- Understand the approximate nature of finite element solutions.

5.1 FE algorithm starting from the weak formulation of the BVP

In the previous chapter it was shown that the elasticity BVP could equivalently be formulated in differential and integral form in a so-called weak formulation. In this section we show that a simple finite element algorithm can be obtained if one introduces discretization, through the use of interpolation theory, into this weak form. Discretization appears in two forms: first, when one assumes that a primary variable is known at discrete (nodal) points and second when the interpolation functions exist over finite sub-domains or elements. For completeness we start by recalling the weak formulation introduced previously and then use our combined index notation for scalar components of tensorial functions and for componentes of nodal variables. The weak form is then given as follows:

Given the body forces f_i, the surface tractions $t_i^{\hat{n}}$ prescribed over the part of the boundary S_t and the surface displacements $\bar{u}_i$ prescribed over the part

[1] **This chapter, together with theoretical and computational learning activities is complemented by Jupyter Notebooks 8 through 12 available at the course REPO.**

of the boundary S_u find the displacement field $u_i : V \to \mathbb{R}$ and such $\forall w_i \in V$ it satisfies:

$$\int_V \sigma_{ij} w_{i,j} \, \mathrm{d}V - \int_V f_i w_i \, dV - \int_{S_t} t_i^{\hat{n}} w_i \, \mathrm{d}S = 0.$$

In the expression above V represents the complete problem domain while S_t is that part of the boundary over which tractions are prescribed. Assume now that the full domain V is divided into $NUMEL$ non-overlapping sub-domains (or finite elements) as shown schematically in fig. 5.1

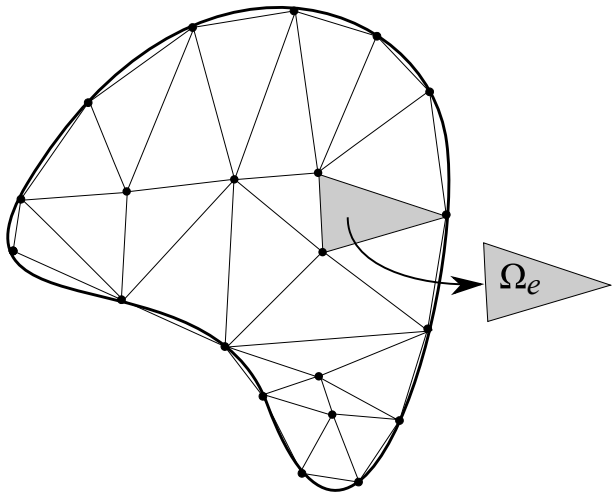


Figure 5.1. Meshed blow

and in such a way that mathematically this can be expressed like:

$$V = \bigcup \Omega_e$$

and where Ω_e represents the sub-domain occupied by one such an element.

In the original strong form of the BVP the partial differential equations result from equilibrium arguments valid over infinitesimal material points occupying the domain V. In the FEM the operation of dividing the full domain V into **NUMEL** subdomains is equivalent to rendering the continuous problem with infinite material points into a discrete problem with a finite number of elements. In conclusion, a finite element is the discrete approximation of an infinitesimal material point.

Note that each element is defined by an ordered set of nodal points establishing a prescribed interpolation space and used to conduct function approximations. In this element partition adjacent elements are connected through the nodal points in such a way that interpolation functions exist only within its element but are continuous through element interfaces. Using this partition in the weak form of equilibrium and using a summation symbol yields;

$$\sum_e \int_{\Omega_e} \sigma_{ij} w_{i,j} \, \mathrm{d}\,\Omega_e - \sum_e \int_{\Omega_e} f_i w_i \, \mathrm{d}\,\Omega_e - \sum_e \int_{S_e} t_i w_i \, \mathrm{d}\,S_e = 0$$

which is still the same form of the weak equilibrium expression.

In a second discretization operation we use interpolation methods to approximate both, the displacement field u_i and the weighting function w_i over each element as discussed previously. In particular assume that inside every element the fields can be approximated like:

$$u_i(\overrightarrow{x}) = N_i^Q(\overrightarrow{x})\widehat{u}^Q$$
$$w_i(\overrightarrow{x}) = N_i^Q(\overrightarrow{x})\widehat{w}^Q$$

and where u^Q represent the scalar components of the displacement vector at the nodal point Q, $N_i^Q(\overrightarrow{x})$ is the shape function for the nodal point Q evaluated at the field point $\overrightarrow{x}$ and $u_i(\overrightarrow{x})$ is the interpolated approximation to the displacement vector at the field point $\overrightarrow{x}$.

Recall that in expressions like

$$u_i(\overrightarrow{x}) = N_i^Q(\overrightarrow{x})\widehat{u}^Q$$

the repeated super-script Q implies summation over $Q = 1, .., Nnodes$ where $Nnodes$ is the number of nodes of the element.

Considering the displacement field as the primary variable we can write in terms of the elastic constitutive tensor:

$$\sigma_{ij} = C_{ijkl}\varepsilon_{kl} \equiv C_{ijkl}B_{kl}^Q(\overrightarrow{x})\widehat{u}^Q$$

after using

$$\varepsilon_{ij}(\overrightarrow{x}) = B_{ij}^Q(\overrightarrow{x})\widehat{u}^Q$$
$$w_{(i,j)}(\overrightarrow{x}) = B_{ij}^Q(\overrightarrow{x})\widehat{w}^Q$$

and where

$$B_{ij}^Q(\overrightarrow{x}) = \frac{1}{2}\left[\frac{\partial N_i^Q}{\partial x_j} + \frac{\partial N_j^Q}{\partial x_i}\right].$$

Substitution of the above in the discretized weak form gives us for an arbitrary e-element of domain Ω_e

$$\widehat{w}^P \int_{\Omega_e} B_{ij}^P C_{ijkl} B_{kl}^Q \,\mathrm{d}\,\Omega_e \widehat{u}^Q - \widehat{w}^P \int_{\Omega_e} N_i^P f_i \,\mathrm{d}\,\Omega_e - \widehat{w}^P \int_{S_e} N_i^P t_i^{(n)} \,\mathrm{d}\,s_e = 0.$$

Note that w_i in the above expression is an arbitrary function and it is convenient to assign values like

$$\widehat{w}^T = \begin{bmatrix} 0 \cdots & 1 \cdots & 0 \end{bmatrix}$$

in turn for every node in order to cancel the common term $\widehat{w}^P$ in such a way that we can write for an arbitrary P degree of freedom:

$$\int_{\Omega_e} B_{ij}^P C_{ijkl} B_{kl}^Q d\Omega_e \widehat{u}^Q = \int_{\Omega_e} N_i^P f_i d\Omega_e + \int_{S_e} N_i^P t_i dS_e.$$

In this expression the dummy superscripts imply a summation over the number of nodes of the element in such a way that the only free index is P which in this compact notation implies also that we have a system of equations corresponding to $P = 1, .., Nnodes$, which is the number of nodes of the element. Note also that the term K_e^{PQ} defined by

$$K_e^{PQ} = \int_{\Omega_e} B_{ij}^P C_{ijkl} B_{kl}^Q \, \mathrm{d}\, \Omega_e$$

represents the force along degree of freedom P due to a unit displacement along degree of freedom Q. In this sense we can define the following set of nodal forces along the P-th degree of freedom:

$$\begin{aligned} f_\sigma^P &= K_e^{PQ} \widehat{u}^Q \\ f_B^P &= \int_{\Omega_e} N_i^P f_i \, \mathrm{d}\, \Omega_e \\ f_c^P &= \int_{S_e} N_i^P t_i^{(n)} \, \mathrm{d}\, S_e \end{aligned}$$

satisfying

$$f_\sigma^P = f_B^P + f_C^P \tag{5.1}$$

and where f_σ^P, f_B^P and f_C^P are forces due to the internal element stresses, body forces and surface tractions respectively.

Equation (5.1) which can also be written like:

$$K_e^{PQ}\widehat{u}^Q = f_B^P + f_C^P$$

is an equation in the nodal displacement u^Q . However it must be observed that this equation has been derived for a single element Ω_e and the possible contribution in terms of forces and stiffness terms from additional elements is yet to be considered. This final step, leading to the assembly of the global system of equilibrium equations (to be discussed later) gives:

$$\left[K^G\right]\left\{U^G\right\} = \left\{RHS^G\right\} \tag{5.2}$$

and where the term RHS^G is a vector of assembled global forces of the type $f_B^P + f_C^P$ storing the contribution from body forces and external surface tractions.

The details of the Python implementation of the stiffness matrix:

$$K_e^{PQ} = \int_{\Omega_e} B_{ij}^P C_{ijkl} B_{kl}^Q \,\mathrm{d}\,\Omega_e$$

are presented as an in-class activity in Notebook 6. This activity requires prior knowledge of interpolation (see Notebook 3) and numerical integration.

5.2 FE algorithm starting from the principle of virtual work

As shown in the previous chapter a valid solution $(u_i, \sigma_{ij}, \epsilon_{ij})$ to the well-posed boundary value problem in theory of elasticity and where $u_i = \overline{u}_i \quad \text{in} \quad S_u$

satisfies

$$\int_V \sigma_{ij}\, \delta\varepsilon_{ij}\, \mathrm{d}V - \int_V f_i \delta u_i\, \mathrm{d}V - \int_{S_t} t_i^{(n)}\, \delta u_i\, \mathrm{d}S = 0 \tag{5.3}$$

for arbitrary functions δu_i such that $\delta u_i = 0 \quad \text{in} \quad S_u$ and where:

$$\delta\varepsilon_{ij} = \frac{1}{2}\left(\frac{\partial \delta u_i}{\partial x_j} + \frac{\partial \delta u_j}{\partial x_i}\right).$$

After using the same arguments and interpolation scheme as in the weak form discretization eq. (5.3) leads to:

$$\int_{\Omega_e} B_{ij}^P C_{ijkl} B_{kl}^Q d\Omega_e \widehat{u}^Q = \int_{\Omega_e} N_i^P f_i d\Omega_e + \int_{S_e} N_i^P t_i dS_e$$

which is the same equation resulting from the discretization of the weak form.

Note that the PVW statement is an scalar (energy) equation relating the strain energy,

$$\delta E_s = \int_{\Omega_e} \sigma_{ij} \delta\varepsilon_{ij}\, \mathrm{d}\,\Omega_e$$

accumulated in the elastic solid as a result of the imposed virtual strain field $\delta\varepsilon_{ij}$ to the work of the external loads

$$\delta W = \int_{\Omega_e} f_i \delta u_i \, \mathrm{d}\,\Omega_e + \int_{S_e} t_i^{(n)} \delta u_i \, \mathrm{d}\, S_e$$

due to the imposition of the virtual displacement δu_i.

Using the same notation as in the previous algorithm the discrete version can also be written like:

$$\delta\widehat{u}^P f_\sigma^P - \delta\widehat{u}^P f_B^P - \delta\widehat{u}^P f_c^P = 0$$

and after using the arbitrariness in δu_i

$$f_\sigma^P - f_B^P - f_c^P = 0. \tag{5.4}$$

In a sense the FEM scheme makes use of interpolation theory to convert the continuous BVP governed by a set of partial differential equations into a discrete problem governed by a set of algebraic equations. Within this context eq. (5.4) can be interpreted as the force equilibrium equation governing the response of a particle P subject to the action of:

- External nodal forces f_B^P consistent with body forces f_i.
- External nodal forces f_c^P consistent with surface tractions $t_i^{(n)}$
- Internal forces f_σ^P consistent with element stresses and equilibrating the external forces.

On the other hand, recall that the PVW statement only holds if the set $(u_i, \sigma_{ij}, \epsilon_{ij})$ is the actual solution to the BVP. This fact implies that in the FE formulation where we only have an approximate solution $u_i(\overrightarrow{x}) = N_i^Q(\overrightarrow{x})\widehat{u}^Q$

the principle does not hold as this is not the actual solution. However, the nodal displacements found after solving eq. (5.4) enforces such condition.

Since the finite element equation written in the form the term:

$$f_\sigma^P = K_e^{PQ}\widehat{u}^Q$$

represents nodal forces equivalent to element stresses the coefficients K_e^{PQ} represent the nodal force along degree of freedom P due to a unit displacement along degree of freedom Q.

5.3 Global assembly

The fundamental equation

$$K_e^{PQ}\widehat{u}^Q = f_B^P + f_c^P \tag{5.5}$$

resulting from the discretization of the PVW through the introduction of interpolation functions was derived over a single element of domain ω_e. In order to consider the contribution from all the elements in a finite element discretized version of a solid we use Newton's third law of motion (action and reaction principle). For that purpose recall that the term f_C^P in eq. (5.5) corresponds to the contact forces resulting from surface tractions and it is computed according to:

$$f_c^P = \int_{S_t} N_i^P t_i^n \,\mathrm{d}\, S \tag{5.6}$$

where it is evident that they depend on the external outward normal vector $\widehat{n}$ over S_t. Clearly, the mechanical interaction between two elements in contact takes place through these contact forces. To explain this interaction consider the schematic mesh shown in fig. 5.2 below:

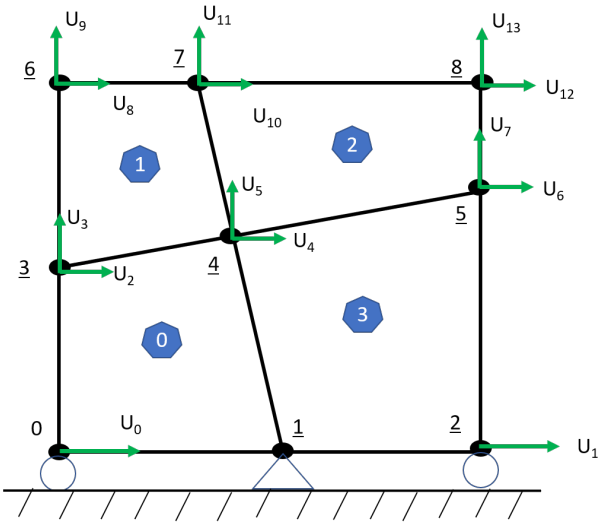


Figure 5.2. 4-elements mesh

Focusing attention in elements 1 and 2 let the interacting surface be labelled as S_b and extend this notation to the degrees of freedom and forces in such a way that the degrees of freedom of nodal points along S_b would be named U_b while those at other parts of the element would be named U_c. Accordingly the finite element equilibrium equations for each element can be written like:

$$\begin{Bmatrix} F_a \\ F_b(\widehat{n}) \end{Bmatrix} = \begin{bmatrix} K^1_{aa} & K^1_{ab} \\ K^1_{ba} & K^1_{bb} \end{bmatrix} \begin{Bmatrix} U_a \\ U_b \end{Bmatrix} \tag{5.7}$$

and

$$\begin{Bmatrix} -F_b(\widehat{n}*) \\ F_C \end{Bmatrix} = \begin{bmatrix} K^2_{bb} & K^2_{bc} \\ K^2_{cb} & K^2_{cc} \end{bmatrix} \begin{Bmatrix} U_b \\ U_c \end{Bmatrix} \tag{5.8}$$

The dependency of the contact forces along the common interface S_b upon the normal outward vector $\widehat{n}$ has been made explicit in the equations.

In particular this vector reads $\widehat{n}$ for element 1 and $\widehat{n}^*$ for element 2 and they satisfy:

$$\widehat{n} = -\widehat{n}^*.$$

Using this relation between the normal vectors at the contact interface we can write for the nodal forces:

$$F_b(\widehat{n}) + F_b(\widehat{n}^*) = 0$$

or equivalently

$$F_b(\widehat{n}) - F_b(\widehat{n}) = 0. \tag{5.9}$$

This condition is shown for completeness in fig. 5.3

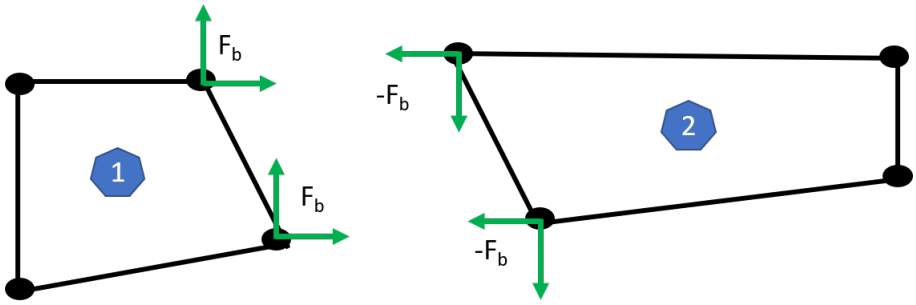


Figure 5.3. Equilibrated nodal forces along the contact interface S_b of elements 1 and 2.

where it is evident that the contact nodal forces along element interfaces are equal and opposite in accordance with Newton's third law. On the other hand, from displacements continuity we have the condition:

$$U_B = U_B^1 = U_B^2. \tag{5.10}$$

Using eq. (5.9) and eq. (5.10) in the equilibrium relationships yields the partial assemblage of the global stiffness matrix resulting after considering the interaction between elements 1 and 2:

$$\begin{bmatrix} K_{aa}^1 & K_{ab}^1 & 0 \\ K_{ba}^1 & K_{bb}^1 + K_{bb}^2 & K_{bc}^2 \\ 0 & K_{cb}^2 & K_{cc}^2 \end{bmatrix} \begin{Bmatrix} U_a \\ U_b \\ U_c \end{Bmatrix} = \begin{Bmatrix} F_a \\ 0 \\ F_c \end{Bmatrix}. \tag{5.11}$$

Note that in the right hand side vector from the above assembled global equilibrium equations only contact forces have been included. This process of adding elemental matrices through contact nodal forces is termed **element assembly** and it is symbolically written like:

$$K^G = \bigwedge_{i=1}^{Numel} k^i \tag{5.12}$$

and where K^G is the global coefficient matrix; k^i is the local elemental matrix for the i-th element; $\bigwedge_{i=1}^{Numel}$ is the assembly operator and i is an element index ranging between 1 and the total number of elements $Numel$ that conform the finite element model. Notice that the assembly operator is analogous to a summation operator commonly used in the representation of series of finite or infinite number of terms, however in the finite element algorithm the assembly operator contains information indicating the position of each single term from the element coefficient matrix within the global system.

In most finite element codes the assembly operator is an array of integer values indicating the position (row and column) from each coefficient of the stiffness matrix of a given element. In **SolidsPy** this operator is termed the **DME()** operator and it is computed in the assembly module **ASSEMUTIL**.

5.4 The assembly algorithm

In this section we will discuss the fundamental algorithmic steps required for building the system of algebraic equations through the addition or assembly of elemental coefficient matrices as described by eq. (5.12). This process of assembly of the elemental coefficient matrices into the global system involves:

- (i) The identification of the active degrees of freedom (or equation numbers) assigned to each node in the mesh.
- (ii) The identification of the relationship between the elemental degrees of freedom and the global degrees of freedom.
- (iii) The computation of the coefficient matrix for each element in the model.

> In the finite element jargon the word **elemental** commonly refers to variable or computations performed at the element level.

Step (i) is easily accomplished by assigning a boundary condition flag to the degrees of freedom existing at each node indicating if the degree of freedom is active, in which case it contributes with an equation, or if it is prescribed, in which case it contributes with a specified displacement boundary condition. In our notation we use a 0 value to indicate an active degree of freedom and a -1 value to indicate a prescribed degree of freedom. In the Python based code **SolidsPy** this information is stored in a boundary conditions array termed IBC and with dimensions $nn \times MDIM$, where nn corresponds to the total number of nodal points and $MDIM$ represents the problem dimensionality. The IBC array is first given to the program through an input data file and later translated from 0s and -1s to equation numbers. Thus during this process the boundary condition flag is read and equations are counted and numbers assigned according to the result. If the boundary condition flag is equal to 0 the program assigns an equation number while it retains a -1 value for a -1 flag. In summary, the IBC array exists in two instances. In its first instance it just contains integer flags indicating active

or restrained degrees of freedom at each node while in a second instance it indicates the actual active equations existing at each node.

In parallel and also with data given through an input file, the nodes conforming each element are stored in a **connectivity** array termed $IELCON$ and of dimension $Numel \times MxNNel$ where $Numel$ is the number of elements in the model and $MxNNel$ is the maximum number of nodes in a given element. Thus each row in the $IELCON$ array stores the nodal point data for the element. Each entry in the $IELCON$ array can now be directly translated into equation numbers using the processed version of the IBC array. This process results in the discrete version of the assembly operator $\bigwedge_{i=1}^{Numel}$ also called the assembly list or the DME operator in **SolidsPy**.

In the final step the mesh is looped one element at a time and each elemental coefficient matrix is assembled into the global matrix using the information (equation numbers) stored in the DME operator. The actual computation of the elemental matrix is conducted by an element based subroutine, called UEL in **SolidsPy**. These subroutines may be different for each element in the mesh according to different kinematic o material assumptions. The assembly process is further illustrated by the following sample problem.

Example For the mesh shown in fig. 5.4 where the elements are numbered from left to right and bottom to top write:

- (i) The boundary conditions array **IBC()** in its two instances.
- (ii) The connectivities array for the mesh.
- (iii) The assembly **DME()** operator.

(i) In a first instance the boundary conditions array **IBC()** is filled with data assigned by the user in the input file. Since all the nodes in the bottom and top faces of the mesh are restrained in the horizontal and vertical directions they are assigned the flag -1 in each column. Similarly, node 4 is restrained only in the horizontal direction so it is assigned a -1 in the first

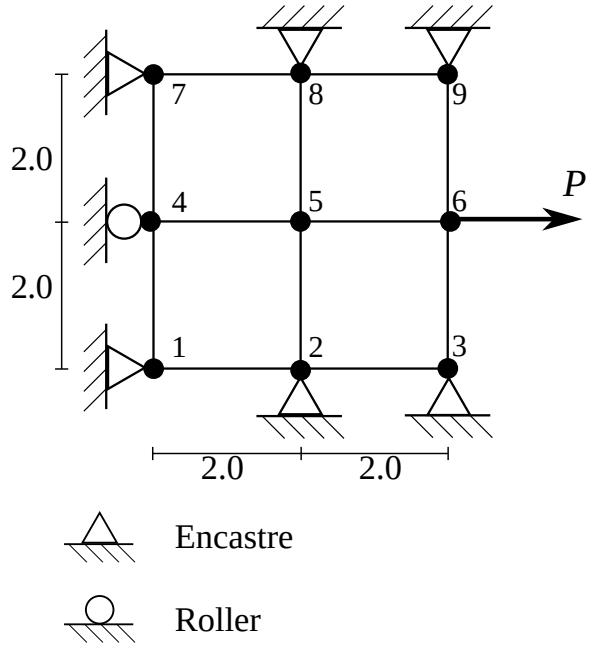


Figure 5.4. Mesh of 4 bi-linear finite elements.

position and a 0 in the second position, while nodes 5 and 6 are completely free along both directions and they are assigned values of 0. The resulting array is given by:

$$IBC_1 = \begin{bmatrix} -1 & -1 \\ -1 & -1 \\ -1 & -1 \\ -1 & 0 \\ 0 & 0 \\ 0 & 0 \\ -1 & -1 \\ -1 & -1 \\ -1 & -1 \end{bmatrix}$$

In a second instance the boundary conditions array **IBC()** is modified by the program which reads every value retaining those corresponding to -1

and assigning sequential numbers to those corresponding to 0. Accordingly the first equation identified as U_0 corresponds to the vertical displacement of nodal point 4 while the remaining 4 equations associated to degrees of freedom U_1, U_2, U_3, U_4 are the horizontal and vertical displacements of nodes 5 and 6. The final form of the array is given like:

$$IBC_2 = \begin{bmatrix} -1 & -1 \\ -1 & -1 \\ -1 & -1 \\ -1 & 0 \\ 1 & 2 \\ 3 & 4 \\ -1 & -1 \\ -1 & -1 \\ -1 & -1 \end{bmatrix}$$

(ii) The next step corresponds to the creation of the connectivities array **IELCON()** storing the nodal points defining each element. In this case each element is defined by a list of 4 nodal point numbers. Recall that for the computation of the elemental stiffness matrix each element in the physical mesh is mapped to the canonical element in the natural space. In the natural space the shape functions are associated to a fixed node numbering and element orientation. If one follows a counter-clockwise orientation the node numbering must follow the same counter-clockwise orientation in the physical space. However the selection of the first node in the sequence is arbitrary and the mapping into the natural space will take care of the corresponding rotation. In this case a valid connectivities array is that given by:

$$IELCON = \begin{bmatrix} 0 & 1 & 4 & 3 \\ 1 & 2 & 5 & 4 \\ 3 & 4 & 7 & 6 \\ 4 & 5 & 8 & 7 \end{bmatrix}$$

(iii) The final ingredient required for the assembly of the global system of equations is the so-called **DME()** operator. This is actually the same **IELCON()** array but translated into equation numbers assigned at each element.

Computationally this involves a subroutine that runs over the **IELCON()** array, reads the value of the nodal identifier and then extracts the assigned equation numbers from the boundary conditions array **IBC()**. In this example the assembly operator consistent with the definition of the connectivities array corresponds to:

$$DME = \begin{bmatrix} -1 & -1 & -1 & -1 & 1 & 2 & -1 & 0 \\ -1 & -1 & -1 & -1 & \mathbf{3} & 4 & 1 & 2 \\ -1 & 0 & 1 & 2 & -1 & -1 & -1 & -1 \\ 1 & 2 & 3 & 4 & -1 & -1 & -1 & -1 \end{bmatrix}$$

A note on the assembly operator: The definition of the element connectivities and its subsequent translation into degrees of freedom is carried out according to the local or canonical element definition in the natural space(fig. 5.5)

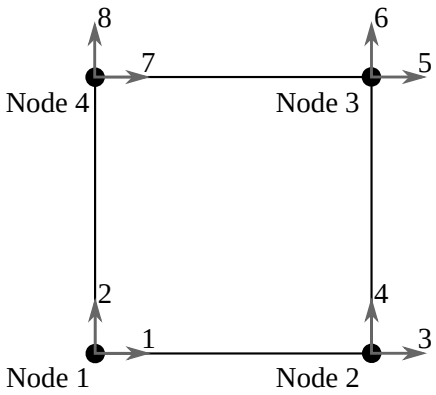


Figure 5.5. Local definition for a 4-noded element.

As a result the (i, j) entry in the DME array corresponds to the global equation number associated with the local degree of freedom j of the i element. For instance the value of 3 stored at position $(1, 4)$ in the current **DME()** operator indicates that the global equation 3 corresponds to the local equation 4 (column index) in element 1 (row index).

The assembly process is then conducted by identifying the relation between the entries in each row of the **DME()** operator and the list of local degrees of freedom for the reference element shown in fig. 5.5. Accordingly, for element 1 it follows that the assembly of row 4 of the local stiffness matrix proceeds as follows:

$$K^G_{3,3} \leftarrow K^G_{3,3} + k^1_{4,4}$$
$$K^G_{3,4} \leftarrow K^G_{3,4} + k^1_{4,5}$$
$$K^G_{3,1} \leftarrow K^G_{3,1} + k^1_{4,6}$$

5.5 Summary: The finite element algorithm

At this point it becomes evident that the algorithm for the displacements based finite element method reduces to the solution of a linear system of algebraic equations in the unknown nodal displacements U^G resulting after assembling the contribution to the stiffness matrix K^G and loads vector RHS^G from all the elements in the mesh. This process can be summarized in the 4-steps pseudo-code described in algorithm 2:

Algorithm 2: Global steps involved in the finite element algorithm

Data: Finite element model
Result: Field function

PREPROCESSING (Reads the model and computes **IBC()**, **IELCON()** and **DME()** arrays);
ASSEMBLY (Forms the system of equations $\left[K^G\right]\left\{U^G\right\} = \left\{RHS^G\right\}$);
SOLUTION (Finds U^G);
POSTPROCESSING (Scatter the global solution to the elements and computes element forces, strains, and stresses);

In the pre-processing stage the code reads the input file and forms the arrays **IBC()**, **IELCON()** and **DME()** which are required for the assembly process. Once the global system of equations is assembled and solved, the values of the nodal displacements are distributed to the corresponding elements in a process commonly referred to like scattering. This process is

somehow inverse to the assembly operation. With the nodal displacements for each element at hand it is now possible to compute nodal forces together with strain and stress distributions inside the element. The assembly and solution process is given in algorithm 3. In this algorithm it must be noticed that the local stiffness matrix for each i element is obtained by the call to the local element subroutine **UEL()**.

Algorithm 3: Details of the assembly and system solution process

Data: Finite element model
Result: Field function

READ **IBC()** and **IELCON()** arrays from input files ;
COMPUTE modified **IBC()** array ;
COMPUTE **DME()** operator (using **IBC()** and **IELCON()** arrays);
INITIALIZE Global stiffness matrix $K^G \leftarrow 0.0$;
(Start assembly)

for $i \leftarrow 1$ *to* $Numel$ **do**
    Call UEL(element parameters;k^i) ;
    $K^G \leftarrow K^G +^i$ (Assemble each k^i into K^G according to the **DME()** operator);
    $RHS^G \leftarrow RHS^G + rhs^i$ (Assemble each rhs^i into RHS^G);
end

SOLVE $K^G U^G = RHS^G$;
Scatter nodal displacements solution to the elements

The details of the elemental subroutine **UEL()** are given in algorithm 4. This process basically involves looping through the Gauss points and computing the required terms. The computation of the the strain-displacements matrix $B(r_j, s_j)$ and Jacobian determinant $\|J_j\|$ at each Gauss point is conducted by additional subroutines.

Algorithm 4: Details of the elemental **UEL** subroutine

Data: Material parameters, nodal coordinates, applied loads
Result: k^l and rhs^l

INITIALIZE $k^l \leftarrow 0.0$;
COMPUTE constitutive tensor C_{ijkl} ;
FORM Gauss quadrature arrays $w()$ and $r(), s()$;
(Start loop through the Gauss points)

for $j \leftarrow 1$ *to* $Ngpts$ **do**
  RETRIEVE r_j , s_j , w_j ;
  COMPUTE $\|J_j\|$, $B(r_j, s_j)$;
  $k^l \leftarrow k^l + B^T C B \, \|J_j\| \, \|J_j\|$;
end

Proposed problems

1. In the bi-linear element shown in fig. 5.6 the nodal displacements resulting from a finite element solution are indicated by the blue arrows. For the material parameters given in the figure find the nodal forces consistent with the element stresses. Assume plane strain behaviour.

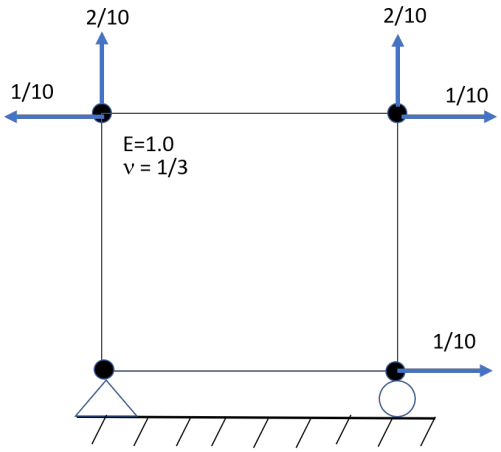


Figure 5.6. One-element mesh for problem 1.

2. For the mesh shown in the figure, with internal surfaces between elements 1-3 and 3-2 labelled S_b and S_c respectively, write the form of the global stiffness matrix resulting from the physical assembly. Explicitly formulate the force and displacement compatibility equations along both boundaries

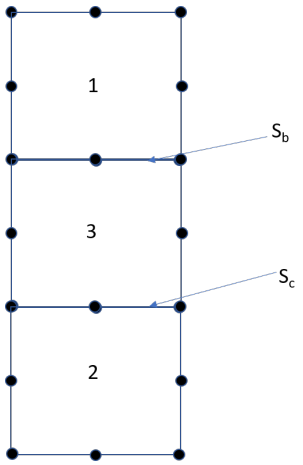


Figure 5.7. Long mesh for problem 2.

3. For the mesh shown in the figure propose different node numbering schemes and identify the resulting changes in the size of the half-band in the stiffness matrix. Assume that each element subroutine is full of 1s.

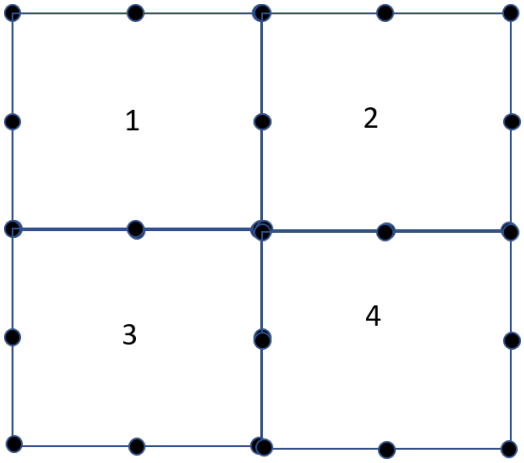


Figure 5.8. Long mesh for problem 3.

4. A finite element model is conformed by an assemblage of three elements of the type shown in fig. 5.9

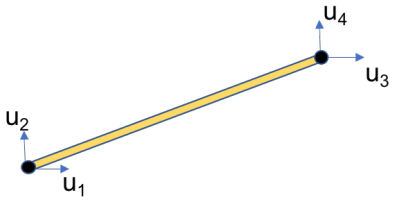


Figure 5.9. Fundamental element

and with a stiffness matrix (in the global coordinate system) given by:

$$k = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1. \end{bmatrix}$$

On the other hand, it is known that global stiffness matrix has been assemble using the following operator

$$DME = \begin{bmatrix} -1 & -1 & 2 & 3 \\ -1 & -1 & 1 & -1 \\ 1 & -1 & 2 & 3 \end{bmatrix}$$

in which a value of -1 refers to a restrained degree of freedom and thus equal to zero. It is required to:

- Draw the complete model including its displacement boundary conditions.
- Indicate the order of the global stiffness matrix for the complete structure.
- Find the global stiffness matrix for the structure.

5. Using SolidsPy compute the stress concentration factor defined according to:

$$SCF = \frac{\max \sigma_{yy}}{\text{promedio}\, \sigma_{yy}}$$

and introduced after drilling a hole in the following plates:

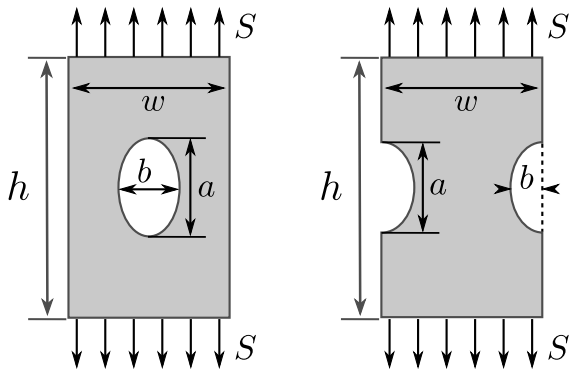


Figure 5.10. Plates with holes.

In both cases the material parameters are defined in table 5.1

Parameter	Value
Young's modulus (Pa)	$E = 200 \times 10^9$
Poisson's ratio	$\nu = 0.285$
Load (N/m)	$S = 10^8$

Table 5.1. Material parameters for the analysis defined in problem 2.

while the geometric parameters are those of table 5.2

a	b	h	w
20	40	50	100

Table 5.2. Geometric parameters for the analysis defined in problem 2.

Bibliography

[1] Lage, Maureen J., Glenn J. Platt, and Michael Treglia: *Inverting the classroom: A gateway to creating an inclusive learning environment*. The Journal of Economic Education, 31(1):30–43, 2000.

[2] Gómez, Juan and Nicolás Guarín-Zapata: *Solidspy: 2d-finite element analysis with python*, 2018. `https://github.com/AppliedMechanics-EAFIT/SolidsPy`.

[3] Shames, Irving H: *Elastic and inelastic stress analysis*. CRC Press, 1997.

[4] Wikipedia: *Variational principle — wikipedia, the free encyclopedia*, 2015. `http://en.wikipedia.org/w/index.php?title=Variational_principle&oldid=646823082`, [Online; accessed 30-May-2015].

[5] Bathe, Klaus Jurgen: *Finite Element Procedures*. Prentice Hall, 2nd edition, June 1995.

[6] Geuzaine, Christophe and Jean François Remacle: *Gmsh: A 3-d finite element mesh generator with built-in pre-and post-processing facilities*. International Journal for Numerical Methods in Engineering, 79(11):1309–1331, 2009. `http://geuz.org/gmsh/doc/preprints/gmsh_paper_preprint.pdf`.

[7] Timoshenko, Stephen and J. N. Goodier: *Theory of Elasticity*. McGraw-Hill, 3rd edition, 1970.

Appendices

Appendix A

Combined index notation for finite element analysis

A.1 Combined index notation for finite element analysis

In the formulation of finite element methods it is customary to start from expressions written in index notations like:

$$\delta W = \int_V \sigma_{ij} \delta u_{i,j} \, \mathrm{d}V$$

and then proceed to introduce discretization or approximations via interpolation theory. In this case it is useful to combine index notation to describe the physical tensorial fields and at the same time the superposition implicit in interpolation schemes. In this appendix we clarify the use of such combined index notation.

A.1.1 Index notation of Cartesian tensor fields

In index notation vector and tensor fields are represented by a letter defining the name of the field and a set of different subscripts (or index). The number of different indices associated to the letter indicates whether the field is a vector (1 index), a second order tensor (2 indices), a third order tensor (3 indices) as in:

$$u_i, \sigma_{ij}, C_{ijk}$$

and further clarified next.

Consider a vector $\vec{u}$ represented in a Cartesian reference system in terms of its scalar components u_x, u_y, u_z. The following representations of the vector are equivalent:

$$u_i \leftrightarrow \begin{bmatrix} u_x & u_y & u_z \end{bmatrix} \leftrightarrow \vec{u} = u_x\widehat{i} + u_y\widehat{j} + u_z\widehat{k}.$$

In the first case, the vector is denoted by the letter u and by the subscript i which represents in condensed form the scalar components $\begin{bmatrix} u_x & u_y & u_z \end{bmatrix}$ of the vector.

Similarly, consider a second order tensor $\overset{\rightarrow}{\sigma}^{(2)}$ represented in a cartesian reference system in terms of its scalar components $\sigma_{xx}, \sigma_{xy}, \sigma_{xz}, \sigma_{yx}, \sigma_{yy}, \sigma_{yz}, \sigma_{zx}, \sigma_{zy}, \sigma_{zz}$. The following representations of the tensor are equivalent:

$$\sigma_{ij} \leftrightarrow \begin{bmatrix} \sigma_{xx} & \sigma_{xx} & \sigma_{xx} \\ \sigma_{xx} & \sigma_{xx} & \sigma_{xx} \\ \sigma_{xx} & \sigma_{xx} & \sigma_{xx} \end{bmatrix} \leftrightarrow \overset{\rightarrow}{\sigma}^{(2)}.$$

Note that each free or different subscript associated to a letter is a condensed representation of normal scalar components along a basis $\widehat{i} - \widehat{j} - \widehat{k}$. Accordingly, in the case of the second order tensor the subscript i represents variation through $x - y - z$ and subscript j represents variation through $x - y - z$.

A.1.2 The summation convention

In the context of indicial notation subscripts which appear repeated (forming pairs) are used to represent summation. The most simple example is the representation of the dot product between two vectors which can be represented in the following alternative forms:

$$\overrightarrow{u} \cdot \overrightarrow{w} \leftrightarrow u_x w_x + u_y w_y + u_z w_z \leftrightarrow \sum_{i=1}^{3} u_i w_i \leftrightarrow u_i w_i.$$

Accordingly repeated indices represent summation over the range of variation of the index.

A.1.3 Indicial notation in interpolation

From interpolation theory we know that a function $f(x)$ can be approximated in terms of n known values of the solution $\begin{bmatrix} f^1 & f^2 & ... f^n \end{bmatrix}$ using a superposition like

$$f(x) = L^1(x)f^1 + L^2(x)f^2 + ... + L^n(x)f^n$$

and where the terms $L^k s$ are n interpolation functions of order $n-1$. This approximated function can also be represented in terms of indicial notation where we now use capitalized superscripts to refer to the components of the interpolation set as follows:

$$f(x) = L^Q(x)f^Q.$$

In the expression above the approximated function represents a scalar quantity varying over a one-dimensional domain with independent variable x.. In the case of approximation via interpolation theory of vector or tensor

valued functions we will have subscripts indicating the scalar components of the vector or tensor field and superscripts representing the interpolation polynomials. Using explicit notation a vector function $u_i(x, y, z)$ varying over a three-dimensional space with independent variables x, y, z would be represented like:

$$\begin{aligned} u_x(x,y,z) &= N^1(x,y,z)u_x^1 + N^2(x,y,z)u_x^2 + ... + N^n(x,y,z)u_x^n \\ u_y(x,y,z) &= N^1(x,y,z)u_y^1 + N^2(x,y,z)u_y^2 + ... + N^n(x,y,z)u_y^n \\ u_z(x,y,z) &= N^1(x,y,z)u_z^1 + N^2(x,y,z)u_z^2 + ... + N^n(x,y,z)u_z^n. \end{aligned}$$

In this representation it has been assumed that each component u_x, u_y and u_z has been approximated using the same interpolation space. In the generalization of the indicial notation to the interpolation of the vector valued function we will associate the subscript, representing the scalar components of the field to the interpolation polynomials. Thus the above expression would be written like:

$$u_i\ (x,y,z)\ =\ N_i^Q(x,y,z)u^Q$$

or

$$u_i(\overrightarrow{x})\ = N_i^Q(\overrightarrow{x})u^Q$$

after considering $\overrightarrow{x} \equiv x, y, z$.

The subscript i, representing the physical components of the vector field u_i has been carried out as a subscript to the shape function N^Q for the nodal point Q, while the term u^Q refers to the scalar components of the field u_i at the nodal point Q. The main advantage in this notation is the possibility of conducting further operations, as required in the derivation of very general finite element algorithms, while combining physical and discrete information.

Example In the expression

$$\delta W = \int_V \sigma_{ij} \delta u_{i,j} \, \mathrm{d}V$$

assume that the primary variable is the displacement field u_i in such a way that at the nodal point Q the displacement vector u^Q is known. Write the interpolated version of δW.

First, let us write the interpolated approximation to the displacement field δu_i carrying the subscript to the shape functions:

$$\delta u_i(\overrightarrow{x}) \; = N_i^Q(\overrightarrow{x}) \delta u^Q.$$

To write the interpolated version of $\delta u_{i,j}$ we note that in the expression above the spatial variation of the field has been assumed by the shape function so we just extend the derivative with respect to x_j to these functions to obtain the following interpolated version of the second order tensor field $\delta u_{i,j}$:

$$\delta u_{i,j}(\overrightarrow{x}) \; = N_{i,j}^Q(\overrightarrow{x}) \delta u^Q.$$

Making

$$B_{ij}^Q(\overrightarrow{x}) \equiv N_{i,j}^Q(\overrightarrow{x})$$

the above can be written like:

$$\delta u_{i,j}(\overrightarrow{x}) \; = B_{ij}^Q(\overrightarrow{x}) \delta u^Q. \tag{A.1}$$

In eq. (A.1), the term $B_{ij}^{Q}(\overrightarrow{x})$ is an interpolation function (which is indicated by the superscript Q) associated to a second order tensor (which is indicated by the subscripts ij). It must be recognized that $B_{ij}^{Q}(\overrightarrow{x})$ are not independent shape functions but just derivatives of the primary interpolation polynomials $N^{Q}(x, y)$.

Consider now the stress-strain relationship in theory of elasticity relating the stress tensor σ_{ij} to the strain tensor ϵ_{ij} through the elastic constitutive tensor C_{ijkl}as:

$$\sigma_{ij} = C_{ijkl}\epsilon_{kl}.$$

In the above the strain tensor ϵ_{ij} is given by a combination of space derivatives of the displacement field u_i like

$$\varepsilon_{ij} = \frac{1}{2}(u_{i,j} + u_{j,i})$$

which can be written like

$$\epsilon_{ij}(\overrightarrow{x}) \ = \frac{1}{2}[B_{ij}^{Q}(\overrightarrow{x}) + B_{ji}^{Q}(\overrightarrow{x})]u^{Q} \equiv H(\overrightarrow{x})_{ij}^{Q}u^{Q}$$

Using the above set of results in δW gives:

$$\delta W = \delta u^{Q} \int_{V} H_{ij}^{Q} C_{ijkl} H_{kl}^{P} \, \mathrm{d}\, V u^{P} \tag{A.2}$$

which is the final discrete version of δW.

In the indicial representation of a tensor quantity an index which does not repeat itself in the expression is termed a **free index** and the number of non-repeated free indices appearing in the expression gives the expression order. By contrast, repeated indices appearing in the expression are termed **dummy indices** and they imply summation.

Problem: Discretization of the principle of virtual displacements from theory of Elasticity The principle of virtual displacements in the linearized theory of elasticity is given by:

$$\int_V \sigma_{ij}\delta u_{i,j}\,\mathrm{d}V - \int_V f_i\delta u_i\,\mathrm{d}V - \int_{S_t} t_i^n\delta u_i\,\mathrm{d}S = 0. \tag{A.3}$$

and where u_i is the displacement field; ϵ_{ij} is the strain field and σ_{ij} is the stress field satisfying the following relations

$$\varepsilon_{ij} = \frac{1}{2}(u_{i,j} + u_{j,i})$$

and

$$\sigma_{ij} = C_{ijkl}\epsilon_{kl}$$

where C_{ijkl} is a fourth-order tensor whose terms are material constants. Also f_i and t_i^n are the body forces and the surface traction vectors.

Assuming that in a finite element method the displacement field u_i is approximated via interpolation like:

$$u_i(\overrightarrow{x}) = N_i^Q(\overrightarrow{x})u^Q.$$

- Write the discrete version of eq. (A.3)
- Write the term $K^{QP} = \int_V H_{ij}^Q C_{ijkl} H_{kl}^P \, \mathrm{d}\, V$ in matrix form and implement it in a python script.

Appendix B

Generalized boundary value problems

In this section we use as problem to be solved via the finite element algorithm the case of general initial boundary value problem (I-BVP). In the first part of this appendix we will introduce the differential formulation given in terms of a set of governing equations and properly specified boundary conditions. The resulting equations are obtained after using a generalized balance law. Following this classical and well known approach we formally re-state these equations in the so-called strong form. Subsequently we re-write and prove an equivalent form of the balance law in the form of an integral representation highly friendly for a numerical solution. Since in the integral description of the problem the order of the derivatives in the field functions decreases by one, the resulting statement is called a weak formulation.

B.1 Governing equations

Let $\mathrm{d}S$ be a differential surface element, $\mathrm{d}V$ a differential volume element, $u(\mathbf{x}, t)$ a scalar (or vector) function of space and time. The flux or rate of

flow of the quantity $u(\mathbf{x}, t)$ through $\mathrm{d}S$ at time t is defined like

$$p(\mathbf{x})\boldsymbol{\nabla} u \cdot \hat{n}\, \mathrm{d}S \quad ,$$

where $p(\mathbf{x})$ is a positive function, assumed known and time independent. Similarly, the time rate of change of $u(\mathbf{x}, t)$ in an element $\mathrm{d}V$ is given by

$$\rho(\mathbf{x})\frac{\partial u}{\partial t}\, \mathrm{d}V \quad ,$$

where once again $\rho(\mathbf{x})$ is a known, given, time independent positive function. Additional effects occurring in the element $\mathrm{d}V$ at the time t can be expressed like

$$H(\mathbf{x}, t)\, \mathrm{d}V \equiv -q(\mathbf{x})u(\mathbf{x}, t) + \hat{F}(\mathbf{x}, t)$$

where $\hat{F}(\mathbf{x}, t) = \rho(\mathbf{x})F(\mathbf{x}, t)$. In the above the term qu represents internal effects due to changes proportional to u while $\hat{F}(\mathbf{x}, t)$ are other external influences in the medium.

Balancing the internal and external changes yields

$$\int_V \rho(\mathbf{x})\frac{\partial u}{\partial t}\, \mathrm{d}V = \int_S p(\mathbf{x})\boldsymbol{\nabla} u \cdot \hat{n}\, \mathrm{d}S + \int_V H(\mathbf{x}, t)\, \mathrm{d}V \quad ,$$

or equivalently

$$\int_V \rho(\mathbf{x})\frac{\partial u}{\partial t}\, \mathrm{d}V = \int_S p(\mathbf{x})\boldsymbol{\nabla} u \cdot \hat{n}\, \mathrm{d}S - \int_V q(\mathbf{x})u(\mathbf{x}, t)\, \mathrm{d}V + \int_V \rho(\mathbf{x})F(\mathbf{x}, t)\, \mathrm{d}V \quad .$$

Using the divergence theorem as

$$\int_S p(\mathbf{x})\boldsymbol{\nabla} u \cdot \hat{n}\, \mathrm{d}S = \int_V \boldsymbol{\nabla}\cdot(p\boldsymbol{\nabla} u)\, \mathrm{d}V$$

yields after substitution

$$\int_V \left[\rho(\mathbf{x})\frac{\partial u}{\partial t} - \boldsymbol{\nabla} \cdot (p(\mathbf{x})\boldsymbol{\nabla} u) + q(\mathbf{x})u(\mathbf{x}, t) - \rho(\mathbf{x})F(\mathbf{x}, t)\right] \mathrm{d}V = 0$$

Assuming a continuous integrand, the arbitrariness of V implies

$$\rho(\mathbf{x})\frac{\partial u}{\partial t} - \boldsymbol{\nabla}\cdot(p(\mathbf{x})\boldsymbol{\nabla}u) + q(\mathbf{x})u(\mathbf{x},t) - \rho(\mathbf{x})F(\mathbf{x},t) = 0$$

Letting

$$\mathcal{L} \equiv -\boldsymbol{\nabla}\cdot p(\mathbf{x})\boldsymbol{\nabla} + q(\mathbf{x})$$

allows us to write the generalized set of partial differential equations like:

$$\rho(\mathbf{x})\frac{\partial u(\mathbf{x},t)}{\partial t} + \mathcal{L}u(\mathbf{x},t) = \rho(\mathbf{x})F(\mathbf{x},t) \ . \tag{B.1}$$

These are categorized as:

- Hyperbolic;
$$\rho(\mathbf{x})\frac{\partial^2 u(\mathbf{x},t)}{\partial t^2} + \mathcal{L}u(\mathbf{x},t) = 0$$
- Parabolic;
$$\rho(\mathbf{x})\frac{\partial u(\mathbf{x},t)}{\partial t} + \mathcal{L}u(\mathbf{x},t) = 0$$
- Elliptic
$$\mathcal{L}u(\mathbf{x},t) = \rho(\mathbf{x})F(\mathbf{x},t) \ .$$

It is convenient to show that $\mathcal{L}$ satisfies the *symmetry* condition

$$\int_V \mathcal{L}(u)v\,\mathrm{d}V = \int_V \mathcal{L}(v)u\,\mathrm{d}V$$

and that the operator is positive definite, that is

$$\int_V \mathcal{L}(u)u\,\mathrm{d}V > 0, \quad \forall u \ .$$

B.1.1 Strong form

The strong form for the generalized boundary value problem formulated above can now be explicitly written as follows.

Given $\rho(\mathbf{x})$, $q(\mathbf{x})$, $p(\mathbf{x})$, $F(\mathbf{x},t)$ and $\bar{u}(\mathbf{x},t)$ find $u(\mathbf{x},t) : V \to \mathbb{R}$ such:

$$\rho(\mathbf{x})\frac{\partial u}{\partial t} - \boldsymbol{\nabla}\cdot[p(\mathbf{x})\boldsymbol{\nabla}u] + q(\mathbf{x})u(\mathbf{x},t) - \rho(\mathbf{x})F(\mathbf{x},t) = 0 \quad \forall \mathbf{x} \in V$$

and

$$\begin{aligned} &u = \bar{u} \quad \forall \mathbf{x} \in S_u \\ &p(\mathbf{x})u_{,i}\hat{n}_i = B(\mathbf{x},t) \quad \forall \mathbf{x} \in S_t \ . \end{aligned}$$

In the FEM we will look for approximate solutions to u subject to the following conditions:

$$u = \bar{u} \quad \forall \mathbf{x} \in S_u \quad \text{(Essential boundary conditions)}$$

and

$$\int_S \left(\frac{\partial u}{\partial x_j}\right)^2 \mathrm{d}S < \infty \ ,$$

which corresponds to the functions being square integrable. We will denote this space by $\mathbb{H}$.

The space of functions satisfying the above two conditions will be denoted by ζ and termed the space of trial functions, formally defined like:

$$\zeta = \{u \mid u \in \mathbb{H}, u = \bar{u} \quad \forall \mathbf{x} \in S_u\}$$

On the other hand, to validate (or test) the correctness of the approximated or proposed trial functions u it is also necessary to introduce test functions w which are arbitrary except that they satisfy the following conditions:

$$w = 0 \quad \forall \mathbf{x} \in S_u$$

and

$$\int_S \left(\frac{\partial w}{\partial x_j}\right)^2 \mathrm{d}S < \infty \ ,$$

which corresponds to the functions being square integrable. The space of functions satisfying the above two conditions will be denoted by $\pounds$ and termed the space of test functions, formally defined like:

$$\pounds = \{w \mid w \in \mathbb{H}, w = 0 \quad \forall \mathbf{x} \in S_u\}$$

B.1.2 Weak form

Given $\rho(\mathbf{x})$, $q(\mathbf{x})$, $p(\mathbf{x})$, $F(\mathbf{x},t)$ and $\bar{u}(\mathbf{x},t)$ find $u(\mathbf{x},t) : V \to \mathbb{R}$ and $\forall w \in \pounds$ such:

$$\int_V p(\mathbf{x}) u_{,i}\, w_{,i}\, \mathrm{d}V - \int_{S_t} B(\mathbf{x},t) w\, \mathrm{d}S + \int_V q(\mathbf{x}) u(\mathbf{x},t) w\, \mathrm{d}V + \int_V \rho(\mathbf{x}) \frac{\partial u}{\partial t} w\, \mathrm{d}V - \int_V \rho(\mathbf{x}) F(\mathbf{x},t) w\, \mathrm{d}V = 0$$

and

$$u = \bar{u} \quad \forall \mathbf{x} \in S_u \ .$$

B.1.3 Equivalence between the strong and weak forms

$$- \int_V [p(\mathbf{x}) u_{,i}]_{,i} w\, \mathrm{d}V + \int_{S_t} [p(\mathbf{x}) u_{,i}] \hat{n}_i w\, \mathrm{d}S - \int_{S_t} B(\mathbf{x},t) w\, \mathrm{d}S + \int_V q(\mathbf{x}) u(\mathbf{x},t) w\, \mathrm{d}V + \int_V \rho(\vec{x}) \frac{\partial u}{\partial t} w\, \mathrm{d}V - \int_V \rho(\mathbf{x}) F(\mathbf{x},t) w\, \mathrm{d}V = 0 \quad \text{(B.2)}$$

Grouping together common terms yields

$$\int_V \left\{ \rho(\mathbf{x}) \frac{\partial u}{\partial t} - [p(\mathbf{x}) u_{,i}]_{,i} + q(\mathbf{x}) u(\mathbf{x}, t) - \rho(\mathbf{x}) F(\mathbf{x}, t) \right\} w \, \mathrm{d}V$$
$$+ \int_{S_t} \{ [p(\mathbf{x}) u_{,i}] \hat{n}_i - B(\mathbf{x}, t) \} \, w \, \mathrm{d}S = 0$$

from which

$$\rho(\mathbf{x}) \frac{\partial u}{\partial t} - [p(\mathbf{x}) u_{,i}]_{,i} + q(\mathbf{x}) u(\mathbf{x}, t) - \rho(\mathbf{x}) F(\mathbf{x}, t) = 0$$

and

$$p(\mathbf{x}) u_{,i} \hat{n}_i = B(\mathbf{x}, t) \quad \forall \mathbf{x} \in S_t \ .$$

B.2 Weighted residual methods

This section introduces the concept of residual or difference from zero in a differential equation once its solution is approximated. For that purpose we will take as prototype equation the one obtained as our general model of BVP (see eq. (B.1)) and recalled here for completeness

$$\rho(\mathbf{x}) \frac{\partial u(\mathbf{x}, t)}{\partial t} + \mathcal{L} u(\mathbf{x}, t) = \rho(\mathbf{x}) F(\mathbf{x}, t) \ . \tag{B.3}$$

We will assume that the actual solution to the generalized BVP given by eq. (B.3) is approximated by $\tilde{u}(\mathbf{x})$ through a superposition like

$$\tilde{u}(\mathbf{x}) = N^I(\mathbf{x}) u^I \tag{B.4}$$

where $N^I(\mathbf{x})$ are interpolating functions and I denotes a superposition index varying like $I = 1, 2, ..., K$ with K being the number of points where the solution is known. In what follows we will use $u(\mathbf{x})$ instead of $\tilde{u}(\mathbf{x})$ but will keep in mind that we are actually using the approximation given by eq. (B.4). Similarly, in order to keep the discussion simple for the time being we will drop the time effects reducing the generalized PDE to the simple form:

$$\mathcal{L} u(\mathbf{x}) = \rho(\mathbf{x}) F(\mathbf{x}) \ . \tag{B.5}$$

Now, since we are using the approximation given by eq. (B.4) this equation is not strictly satisifed but instead we will have the following "unbalanced" condition

$$\mathcal{L}u(\mathbf{x}) - \rho(\mathbf{x})F(\mathbf{x}) \equiv R \neq 0$$

where the term R corresponds to a residual error which is to be distributed throughout the solution domain. The so-called weighted residual methods differ in the form in which they distribute the residual between the different K points conforming the computational domain.

Using eq. (B.4) in eq. (B.3) and the linearity in the differential operator yields

$$R = \mathcal{L}(N^P)u^P - \rho F\,.$$

We can see that the residual R is a function defined over the domain of interest. The residual would be exactly zero for the solution of the differential equation, but it will not be zero in general. Thus, we want to make the function R as close to zero as possible. To make R as small as possible we need a function (a functional) where we can compare different approximation functions. After getting this functional we can minimize its value. For this minimization we could use the norm of the function, another option is to compute a weighted *average* of the function over the domain. This is what we call a weighted residual

$$\Pi[u, w] = \int_V wR(u)\,\mathrm{d}V\ ,$$

and we want to minimize it by making

$$\delta\Pi\,[u, w] = 0\,,$$

In what follows we will consider different strategies to distribute or weight the residual R over the computational domain.

B.2.1 Galerkin method

In the Galerkin scheme the interpolation functions are used also as weighting functions leading to:

$$\int_V N^Q R \, \mathrm{d}V = 0$$

or explicitly

$$\int_V N^Q \mathcal{L}(N^P) \, \mathrm{d}V \, u^P = \int_V N^Q \rho F \, \mathrm{d}V \; . \tag{B.6}$$

Imposing eq. (B.6) in the K points conforming the computational domain or equivalently ranging Q from 1 to K leads to the following system of algebraic equation

$$K^{QP} U^P = f^Q \tag{B.7}$$

where U^P is a vector that stores the point values of the function u along te K points of the computational domain, while f^Q stores the corresponding point excitations.

B.2.2 Least squares method

In this method the integral of the square of the residual is minimized with respect to the K point parameters or nodal values of the function. Accordingly,

$$\frac{\partial}{\partial u^I} \int_V R^2 \, \mathrm{d}V = 0$$

$$\int_V R \frac{\partial R}{\partial u^I} \, \mathrm{d}V = 0 \, ,$$

The least squares method is a special case of the weighted residual method for the weight functions

$$w^I = \frac{\partial R}{\partial u^I} \, .$$

Expanding the residual, and considering the operator $\mathcal{L}$ as linear, we obtain

$$\frac{\partial}{\partial u^I}\int_V [\mathcal{L}(N^P u^P) - \rho F]^2 \,\mathrm{d}V = 0$$
$$\int_V [\mathcal{L}N^P u^P - \rho F]\mathcal{L}(N^I)\,\mathrm{d}V = 0$$
$$\int_V \mathcal{L}(N^I)\mathcal{L}(N^P)\,\mathrm{d}V\, u^P - \rho\int_V \mathcal{L}(N^I)F\,\mathrm{d}V = 0$$

which can be written like

$$K^{IP}U^P = f^I \tag{B.8}$$

B.2.3 Collocation method

In the collocation method the coefficients of the approximation are determined by forcing the residual to be exactly zero at K points over the computational domain, i.e.,

$$\mathcal{L}(N^I)u^I - \rho F = 0\,,$$

or

$$\mathcal{L}[N^I(x^J)]u^I - \rho F(x^J) = 0\,,$$

where J ranges between 1 and K. This equation can be rewritten as a weighted-residual if we consider the residual to be $\delta(x - x^I)$, the Dirac delta function over the selected points

The resulting system of algebraic equation can be written as

$$K^{IP}U^P = f^I\,. \tag{B.9}$$

B.2.4 Subdomain method

The zero value of the residual is imposed upon K subdomains

$$\int_{V^I} \mathcal{L}(N^P)\,\mathrm{d}V^I\,u^P - \rho \int_{V^I} F\,\mathrm{d}V^I = 0 \qquad I = 1, \cdots, K\,.$$

For instance, for the N-th element it follows that

$$\int_{V^N} \mathcal{L}(N^P)\,\mathrm{d}V^N\,u^P - \rho \int_{V^N} F\,\mathrm{d}V^N = 0 \qquad P = 1, \cdots, K\,.$$

Applying the equation over the K subdomains leads to the discrete system;

$$K^{IP}U^P = f^I \qquad I = 1, \cdots, K\,. \tag{B.10}$$

B.2.5 Ritz method

It operates directly upon the variational statement of the problem. For a given functional

$$\Pi = \Pi(N^Q u^Q)$$

the variational equation reads

$$\delta\Pi \equiv \frac{\partial \Pi}{\partial u^Q}\,\delta u^Q = 0$$

from which

$$\frac{\partial \Pi}{\partial u^Q} = 0\,.$$

Example: The generalized parabolic equation Let us consider the case of the generalized parabolic equation and its discretization following the Galerkin method

$$\rho(\mathbf{x})\frac{\partial u(\mathbf{x},t)}{\partial t} + \mathcal{L}u(\mathbf{x},t) = 0$$

which can also be written using index notation

$$\frac{\partial}{\partial x_i}\left[p(x)\frac{\partial u}{\partial x_i}\right] + q(x)u + \rho\frac{\partial u}{\partial t} = \rho F\,,.$$

Assuming that $p(x) = 1$ yields

$$\begin{aligned}
&-\int_V N^P N^Q_{,ii}\,\mathrm{d}V\,u^Q + \int_V qN^P N^Q\,\mathrm{d}V\,u^Q + \rho\int_V N^P N^Q\,\mathrm{d}V\,v^Q \\
&-\rho\int_V N^P F\,\mathrm{d}V = 0 \\
&\int_V N^P_{,i} N^Q_{,i}\,\mathrm{d}V\,u^Q - \int_S N^P N^Q_{,i}\hat{n}_i\,\mathrm{d}S\,u^Q + \int_V qN^P N^Q\,\mathrm{d}V\,u^Q \\
&+\rho\int_V N^P N^Q\,\mathrm{d}V\,v^Q - \rho\int_V N^P F\,\mathrm{d}V = 0 \\
&\int_V \left(N^P_{,i} N^Q_{,i} + qN^P N^Q\right)\mathrm{d}V\,u^Q + \rho\int_V N^P N^Q\,\mathrm{d}V\,V^Q = \\
&\int_S N^P N^Q_{,i}\hat{n}_i\,\mathrm{d}S\,u^Q + \rho\int_V N^P F\,\mathrm{d}V
\end{aligned}$$

which can be written in discrete form as

$$K^{PQ}U^Q + C^{PQ}V^Q = f^p\,.$$

Example: The wave equation. In this case the differential equation reads

$$\boldsymbol{\nabla}\cdot\left[\frac{1}{\rho}\boldsymbol{\nabla}p(\mathbf{x})\right] - \frac{\partial}{\partial t}\left(\frac{1}{\lambda}\frac{\partial\rho}{\partial t}\right) - q(\mathbf{x}) = 0$$

where we recognize that

$$\mathcal{L} \equiv \boldsymbol{\nabla}\cdot\left(\frac{1}{\rho}\boldsymbol{\nabla}\right) - \frac{\partial}{\partial t}\left(\frac{1}{\lambda}\frac{\partial}{\partial t}\right)\,.$$

Let

$$p(x) = N^K p^K$$

then

$$\mathcal{L}(p) \equiv \vec{\nabla} \cdot \left(\frac{1}{\rho} \vec{\nabla} N^K p^K \right) - \frac{\partial}{\partial t} \left(\frac{1}{\lambda} \frac{\partial N^K p^K}{\partial t} \right)$$

or in index notation

$$\mathcal{L}(p) \equiv \left(\frac{1}{\rho} N^K_{,i} \right)_{,i} p^K - \frac{\partial}{\partial t} \left(\frac{1}{\lambda} \frac{\partial N^K}{\partial t} \right) p^K$$

which is equivalent to

$$\mathcal{L}(p) \equiv \mathcal{L}(N^K) p^K$$

using the trial functions as weighting function and recalling the definition of the residual which in this case reads

$$R = \mathcal{L}(N^K) p^K - q \,,$$

and yields

$$\int_V N^J R dV = 0 \qquad J = 1, 2, ..., K$$

$$\int_V N^J L(N^K) dV p^K - \int_V N^J q dV = 0$$

$$\int_V N^J \left(\frac{1}{\rho} N^K_{,i} \right)_{,i} dV p^K - \int_V N^J \frac{\partial}{\partial t} \left(\frac{1}{\lambda} \frac{\partial N^K}{\partial t} \right) dV p^K - \int_V N^J q dV = 0$$

Integrating by parts the first term on the right-hand-side gives us

$$-\int_V N^J_{,i} \frac{1}{\rho} N^K_{,i} dV p^K + \int_S N^J \frac{1}{\rho} N^K_{,i} \hat{n}_i dS p^K = \int_V N^J \frac{\partial}{\partial t} \left(\frac{1}{\lambda} \frac{\partial N^K}{\partial t} \right) dV p^K + \int_V N^J q dV$$

$$\int_V N^J_{,i} \frac{1}{\rho} N^K_{,i} dV p^K + \int_V N^J \frac{1}{\lambda} N^K dV \ddot{p}^K = \int_S N^J \frac{1}{\rho} N^K_{,i} \hat{n}_i dS p^K + \int_V N^J q dV$$

$$K^{JK}P^K + M^{JK}\ddot{P}^K + f^J = 0$$

Example: The Navier equations. The stress equilibrium equations when written in terms of displacements after using the constitutive tensor and the proper kinematic description take the form:

$$(\lambda + \mu)u_{j,ij} + \mu u_{i,jj} + f_i = 0$$

which are known as the Navier equations and where the differential operator reads

$$L_{ij} \equiv (\lambda + \mu)\frac{\partial^2}{\partial x_i \partial x_j} + \mu \frac{\partial^2}{\partial x_k \partial x_k}\delta_{ij}.$$

Applying this operator to an interpolated version of the displacement field $u_i = N_i^Q u^Q$ gives:

$$L_{ij}(u_j) \equiv R_i(u_j)$$

thus

$$\begin{aligned}
\mathcal{L}_{ij}(u_j) &\equiv (\lambda + \mu)(N_j^Q u^Q)_{,ij} + \mu(N_j^Q u^Q)_{,kk}\delta_{ij} \\
\mathcal{L}_{ij}(u_j) &\equiv (\lambda + \mu)N_{j,ij}^Q u^Q + \mu N_{i,kk}^Q u^Q \\
\mathcal{L}_{ij}(u_j) &\equiv \mathcal{L}_{ij}(N_j^Q)u^Q \quad .
\end{aligned}$$

In the Galerkin scheme we use the trial function as weighting function.

$$R_i \equiv L_{ij}(N_j^Q)u^Q + f_i$$

and we state

$$\int_V N_i^P R_i \, \mathrm{d}V = 0 \qquad P = 1, 2, \cdots N \, .$$

Thus

$$\int_V N_i^P \mathcal{L}_{ij}(N_j^K)\,\mathrm{d}V\,u^K + \int_V N_i^P f_i\,\mathrm{d}V = 0$$

$$(\lambda+\mu)\int_V N_i^P N_{j,ij}^K\,\mathrm{d}V\,u^K + \mu\int_V N_i^P N_{i,kk}^K\,\mathrm{d}V\,u^K + \int_V N_i^P f_i\,\mathrm{d}V = 0$$

integrating by parts

$$-(\lambda+\mu)\int_V N_{i,j}^P N_{j,i}^K\,\mathrm{d}V\,u^K + (\lambda+\mu)\int_S N_i^P N_{j,i}^K \hat{n}_j\,\mathrm{d}S\,u^K - \mu\int_V N_{i,k}^P N_{i,k}^K\,\mathrm{d}V\,u^K$$
$$+\mu\int_S N_i^P N_{i,k}^K \hat{n}_k\,\mathrm{d}S\,u^K + \int_V N_i^P f_i\,\mathrm{d}V = 0$$

which can be written like

$$K^{PQ}U^Q = F^P$$

where

$$K^{PQ} = (\lambda+\mu)\int_V N_{i,j}^P N_{j,i}^Q\,\mathrm{d}V + \mu\int_V N_{i,k}^P N_{i,k}^Q\,\mathrm{d}V$$

and

$$F^P = \int_S N_i^P t_i^{(\hat{n})}\,\mathrm{d}S + \int_V N_i^P f_i\,\mathrm{d}V = 0\,.$$

Appendix C

Convergence analysis

In this section we address the problem of convergence of analysis results. We will approach the problem in a loose way proceeding from an engineering point of view. For a thorough discussion the reader is referred to textbooks of numerical analysis, see for instance [**?**]. Particularly, we will review the fundamental aspects that must be satisfied by a finite element solution. In the first part we address the problem from the element point of view, while in the final part we study the convergence of particular problem in terms of several self-contained meshes.

C.1 ¿What is meant by convergence?

Mathematical convergence of order p and rate c for a series of numerically computed values $\vec{u}_k$ and for a problem with exact solution $\vec{u}$ is defined like:

$$\lim_{k\to\infty} \frac{\|\vec{u}_{k+1} - \vec{u}\|}{\|\vec{u}_k - \vec{u}\|^p} = c.$$

A practical definition of convergence in finite element analysis is given as

follows. Let us denote by Π and Π_{FE} the potential energy functionals corresponding to the exact mathematical model and to the finite element solution respectively, where the functional corresponding to a given discretization can be computed as:

$$\Pi_{FE} = -\frac{1}{2}U^T KU$$

where K and U are the global stiffness matrix and the global nodal displacements vector. If k represents the number of finite elements in a given discretization then we define convergency as the condition that:

$$\lim_{k\to\infty} \Pi_{FE} \to \Pi \tag{C.1}$$

In order to guarantee that a finite element solution convergences to the exact (unknown) solution of a problem certain conditions must be met by both, the single elements and the whole assembled finite element mesh. The analysis of the element is conducted when the element is formulated for the first time while the analysis of the mesh is problem dependent. In the following sections we will address both problems.

C.2 Conditions on a single element

From a physical point of view we may expect the following behaviour from the individual elements in a given discretization:

- Under a rigid body compatible displacement field the element must predicts a zero strain field $\varepsilon_{ij} = 0$. This condition is required in order to maintain actual regions of the domain which are submitted to rigid body modes in a stress free condition.

- The element must be able to predict constant strain states as its size decreases. This condition guarantees that as the element size decreases it also approaches the condition of an actual material point.
- The work from the surface tractions along the element interfaces must vanish. This is nothing but Newton's third law in terms of surface tractions. The fact that the first order derivatives of the shape functions (equivalent to surface tractions) are discontinuous along the element boundaries results in finite jumps in the boundary tractions. The element must be such that these jumps vanish as the element size decreases.

In terms of the shape functions these three conditions are equivalent to:

(i) All the element shape functions must be selected in such a way that the element predicts $\varepsilon_{ij} = 0$ under rigid body compatible nodal displacements.

(ii) All the element shape functions must be selected in such a way that if the nodal displacements are compatible with a constant strain state that state is actually obtained.

(iii) All the element shape functions must be selected in such a way that the strains over the element interfaces are finite and the displacements along these boundaries are continous.

Conditions (i) and (ii) are known as the **completeness condition** while condition (iii) is referred to as the **compatibility** condition.

In order to determine if a specific element is complete we find the eigenvalues of the stiffness matrix and study the resulting eigenvectors which corresponde to the deformation modes of the element. The eigenvalue problem reads:

$$[K - \lambda I]\,\phi = 0 \tag{C.2}$$

whose solution gives the rigid body modes and straining modes that can be reproduced by the specific element. In eq. (C.2) λ corresponds to the eigenvalues and the vector ϕ stores the corresponding eigenmodes.

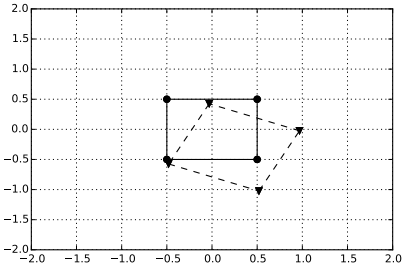
As an example of the single element analysis we show the solution for a single bi-linear perfectly square element of characteristic size $2h = 1.0$ and material properties given by $E = 1.0$ and $\nu = 0.30$ (see [**?**]). The eigenvalue problem is solved with the script **modes.py** listed in the last section. Solution of eq. (C.2) predicts the following set of eigenvalues:

$$\lambda = [0(3), 0.495(2), 0.769(2), 1.43].$$

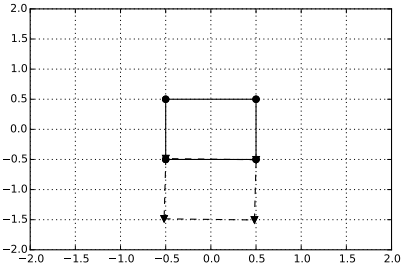
From these the first three zero-valued eigenvalues can be shown to describe the possible rigid body motions, namely one rotation and two translations along the horizontal and vertical directions respectively. The next two eigenvalues corresponding to 0.495 represent flexure modes. Similarly, the repeated values corresponding to 0.769 are associated to shear modes while the last eigenvalue corresponds to a uniform extension mode. The number of different modes satisfy the following condition:

$$N_S = N_{DOF} - N_{RB}$$

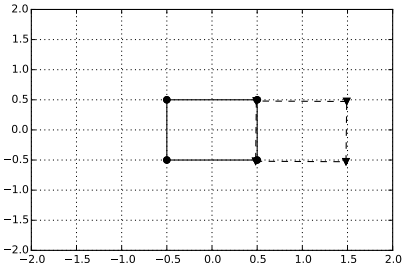
where N_S, N_{DOF} and N_{RB} are the number of straining modes, number of degrees of freedom and number of rigid body modes. The original and deformed element shapes are shown in fig. C.1 which is obtained after one combines the eigenvalues properly. The last row of the figure (obtained with the script **strfield.py**) shows the zero-valued strain field associated to the first rigid body mode.



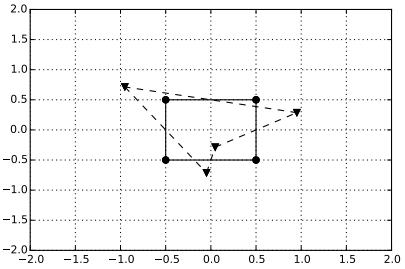
(a) $\lambda_1 = 0$.



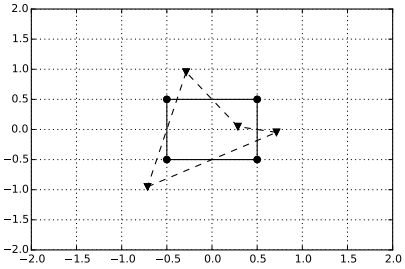
(b) $\lambda_2 = 0$.



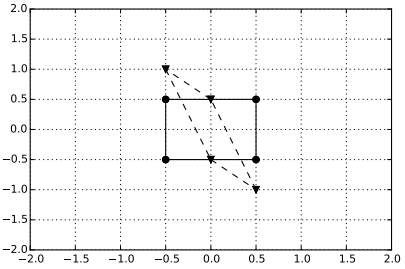
(c) $\lambda_3 = 0$.



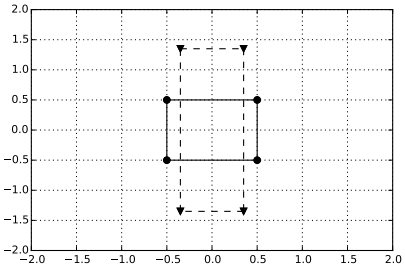
(d) $\lambda_4 = 0.495$.



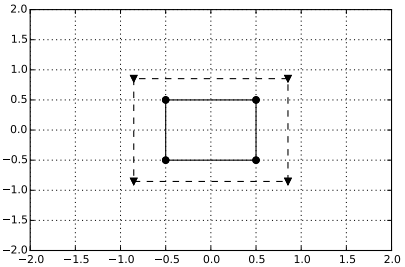
(e) $\lambda_5 = 0.495$.



(f) $\lambda_6 = 0.769$.

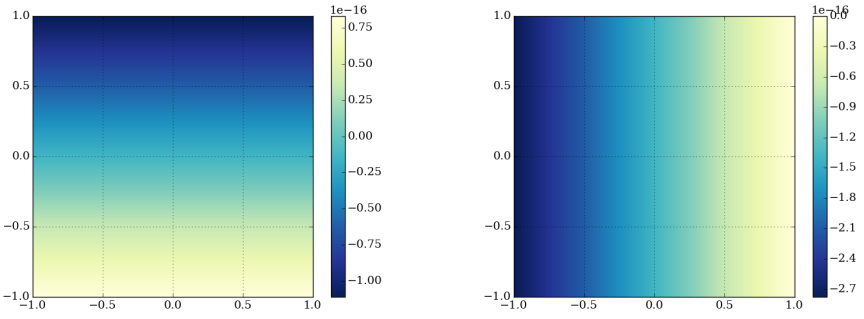


(g) $\lambda_7 = 0.769$.



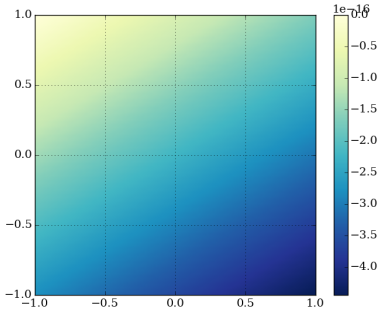
(h) $\lambda_8 = 1.43$.

Figure C.1. Deformation modes of a bi-linear element.



(a) ϵ_{xx}.

(b) ϵ_{yy}.



(c) γ_{xy}

Figure C.2. Deformation modes of a bi-linear element.

C.3 Analysis of the mesh results

Consider the square energy norm of the error $\|\vec{e}_h\|_E^2$. This error satisfies

$$\|\vec{e}_h\|_E^2 > 0$$

and will be used as an error estimate of the accuraccy of the finite element solution. Particularly we will use the following relationship (see [**?**])

$$\|\vec{e}_h\| \leq \alpha h^k \tag{C.3}$$

from which we can write:

$$\log \|\vec{e}_h\| \approx \log \alpha + k \log h \tag{C.4}$$

In eq. (C.4) k is the order of the complete interpolation polynomial present in the mesh and gives a measure of the order of convergence in the finite element solution, while the rate of convergency is given by α.

In order to conduct the convergency study we perform a series of finite element analysis. For each mesh we compute $\|\vec{u} - \vec{u}_h\|$ which is equivalent to $\vec{e}_h$ and where $\vec{u}$ is the exact solution. In order to find the exact solution we assume that the most refined results have functionals corresponding to Π_{n-2}, Π_{n-1} and Π_n from which:

$$\Pi_{Exa} = \frac{\Pi_{n-1}^2 - \Pi_n \Pi_{n-2}}{(2\Pi_{n-1} - \Pi_n - \Pi_{n-2})}$$

The procedure is summarized below:

- Solve a series of meshes with solutions given by $\vec{u}_1, \vec{u}_2, ..., \vec{u}_n$. Each mesh has a characteristic element size h.
- For each mesh find the total potential energy:

$$\Pi_h = -\frac{1}{2} U^T K U$$

- Using the most refined meshes compute the potential energy for the exact solution:

$$\Pi_{Exa} = \frac{\Pi_{n-1}^2 - \Pi_n \prod_{n-2}}{2\Pi_{n-1} - \Pi_n - \Pi_{n-2}}$$

- For each mesh compute:

$$\frac{\|\vec{u}_{Exa} - \vec{u}_h\|}{\|\vec{u}_{Exa}\|} = \left[\frac{\Pi_{Exa} - \Pi_h}{\Pi_{Exa}}\right]^{1/2}$$

  and fill out the following table:

h	$\prod_{FE}$	$\|\vec{u}_{Exa} - \vec{u}_{FE}\|$	$\frac{\|\vec{u}_{Exa} - \vec{u}_{FE}\|}{\|\vec{u}_{Exa}\|}$
1.0			
0.5			
0.25			
0.125			
0.0625			

Table C.1. Convergence of anlysis results

- Plot the values of $\log\left(\frac{\|\vec{u}_{Exa} - \vec{u}_h\|}{\|\vec{u}_{Exa}\|}\right)$ vs $\log h$ and determine the slope which upon convergence must be close to the order of the complete polynomial used in the discretization.

Exampla: bar in compression Figure C.4 shows a tappered bar under a compressive uniformly distributed load of total magnitude $P = 0.5$. The bar is of length $l = 10$ and its large and short ends given by $h_1 = 2.0$ and $h_2 = 0.5$ respectively. The material properties correspond to a Poisson's ratio and Young modulos $\nu = 0.30$ and $E = 1.0$. We want to find the converged solution for the bar after using 3-noded triangular elements.

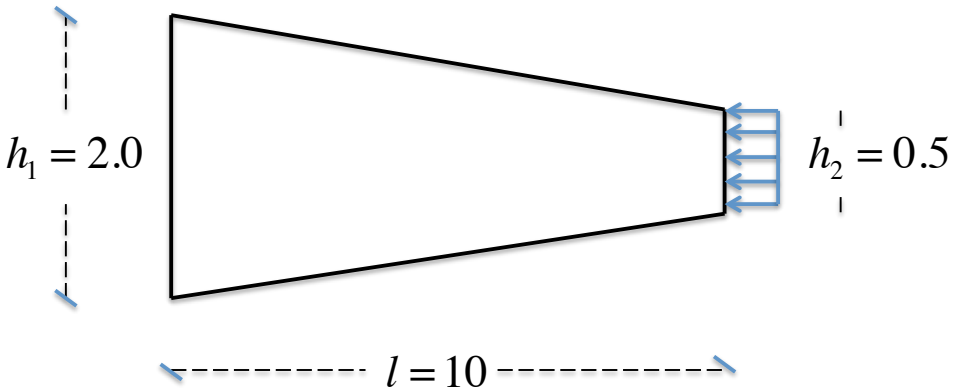


Figure C.3. Tappered bar under compressive load at the tip.

Figure C.4 displays 4 consecutive meshes with decreasing element size namely $h = [1.0, 0.5, 0.25, 0.125]$. While fig. C.5 displays the shear strain contour maps for the finite element solutions corresponding to the coarse and refined meshes. It is clear how these contours become smooth as the mesh is refined. This approach is sometimes used as an empirical test of convergence.

To measure the finite element convergence we first compute the total potential energy in each mesh ccording to:

$$\Pi_{FE} = -\frac{1}{2} U^T K U.$$

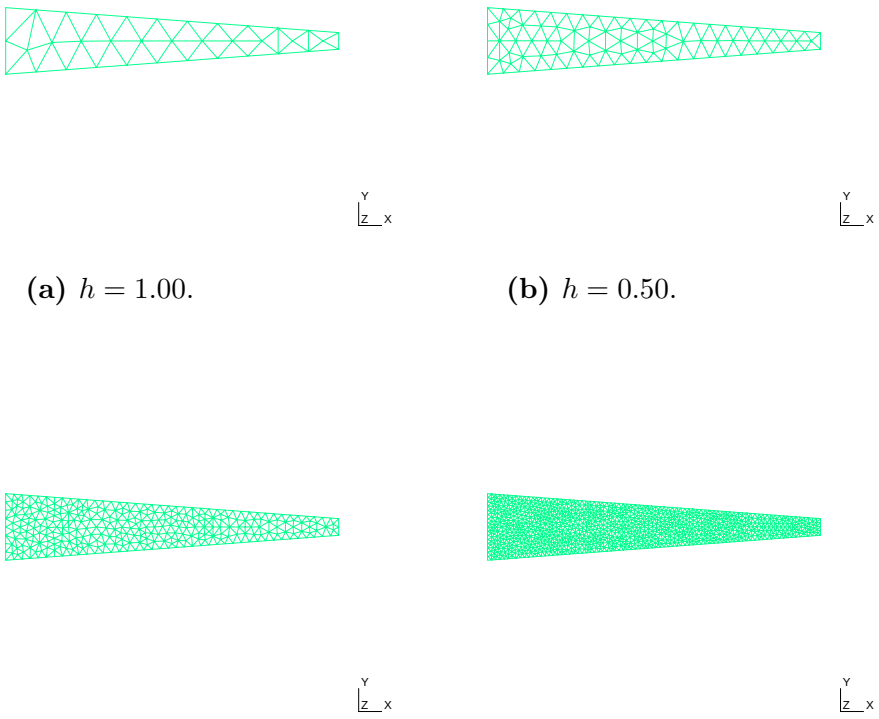
Now assuming we have consecutive meshes each one obtained after halving the elements in the previous mesh we have the following approximation for the exact total potential energy of the system, computed the last most refined meshes:

$$\Pi_{Exa} = \frac{1.161^2 - (-1.160)(-1.155)}{-2(1.161) - (-1.160) - (-1.155)} = -1.160.$$

To compute the energy norm of the error we use:

$$\frac{\|\vec{u}_{Exa} - \vec{u}_{FE}\|}{\|\vec{u}_{Exa}\|} = \left[\frac{\Pi_{Exa} - \prod_{FE}}{\Pi_{Exa}}\right]^{1/2}.$$

The analysis results are reported in table C.2.



(a) $h = 1.00$. **(b)** $h = 0.50$.

(c) $h = 0.250$. **(d)** $h = 0.125$.

Figure C.4. Refined meshes for a tappered bar.

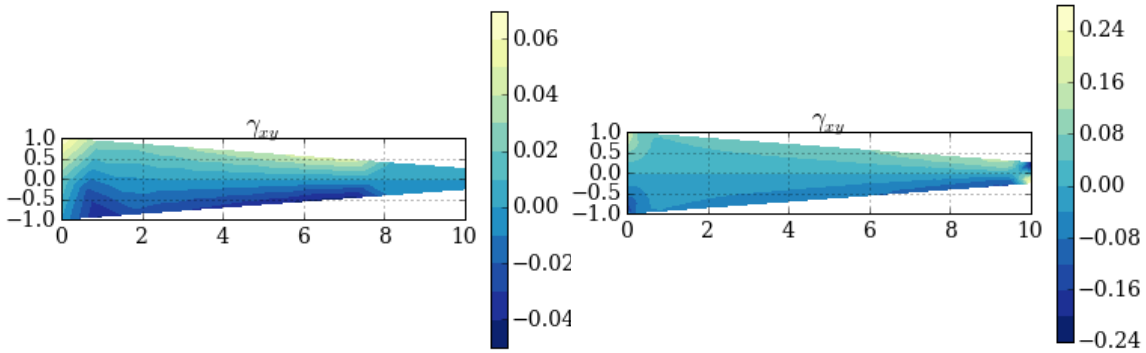


Figure C.5. Shear strain distribution for the coarse and fine mesh.

h	Π_{FE}	$\|\vec{u}_{Exa} - \vec{u}_{FE}\|$	$\frac{\|\vec{u}_{Exa} - \vec{u}_{FE}\|}{\|\vec{u}_{Exa}\|}$
1.0	−1.151	0.095	0.088
0.5	−1.155	0.071	0.066
0.25	−1.161	0.032	0.030
0.125	−1.160	0.001	0.001

Table C.2. Convergence of anlysis results

To measure convergence we plot:

$$\log\left(\|\vec{u}_{Exa} - \vec{u}_{FE}\|\right) = \log c + k \log h$$

leading to fig. C.6 from which $k \approx 1.14$.

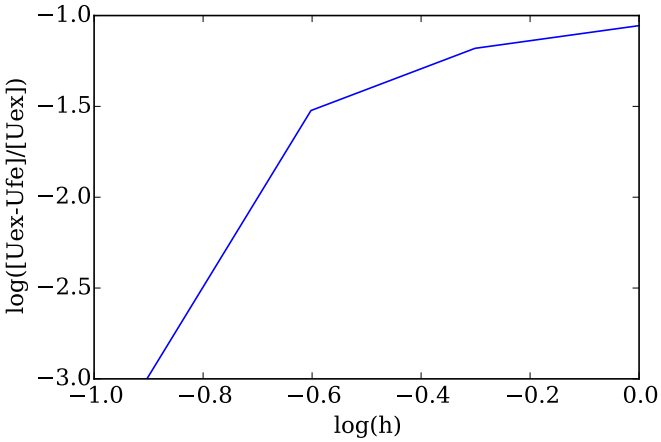


Figure C.6. Energy norm of the error

Example: cantilever beam Consider the cantilever beam shown in fig. C.7

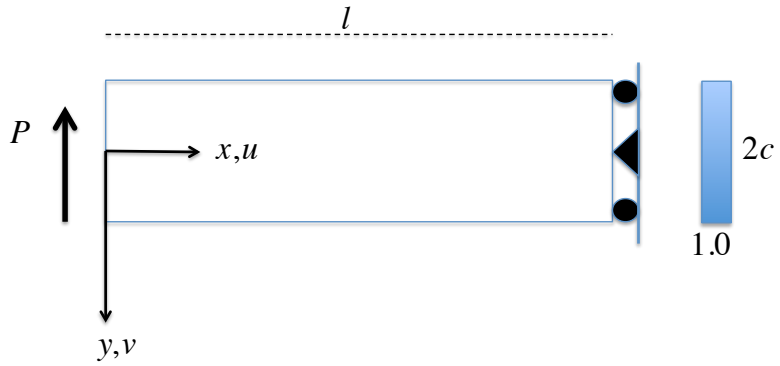


Figure C.7. Cantelever beam.

with analytic solution [7] given by:

$$
\begin{aligned}
u &= -\frac{P}{2EI}x^2 y - \frac{\nu P}{6EI}y^3 + \frac{P}{2IG}y^3 + \left(\frac{Pl^2}{2EI} - \frac{Pc^2}{2IG}\right) y\,, \\
v &= \frac{\nu P}{2EI}xy^2 + \frac{P}{6EI}x^3 - \frac{Pl^2}{2EI}x + \frac{Pl^3}{3EI}\,, \\
\varepsilon_{xx} &= \frac{\partial u}{\partial x} \equiv -\frac{P}{EI}xy\,, \\
\varepsilon_{yy} &= \frac{\partial v}{\partial y} \equiv \frac{\nu P}{EI}xy\,, \\
\gamma_{xy} &= \frac{\partial u}{\partial y} + \frac{\partial v}{\partial x} \equiv \frac{P}{2IG}(y^2 - c^2)\,.
\end{aligned}
$$

The horizontal and vertical displacement field corresponding to the particular values of $E = 1000.0$, $\nu = 0.30$ for the material parameters; $l = 24$ and $2c = 8$ for the geometry and $P = 50$ (upwards) for the load is shown below:

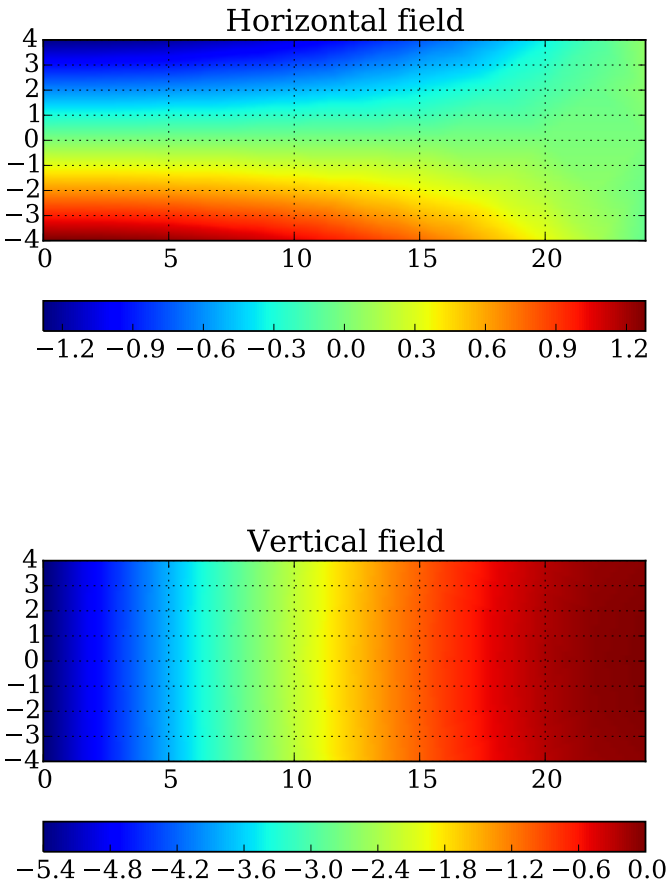


Figure C.8. Displacement field for the cantilever beam.

Perform a finite element simulation using a series of refined meshes with charateristic element size corresponding to $h = [6.0, 3.0, 1.5, 0.75, 0.375]$ and show that convergence is achieved. To conduct the finite element analysis fill out table C.3

h	$\prod_{FE}$	$\|\vec{u}_{Exa} - \vec{u}_{FE}\|$	$\frac{\|\vec{u}_{Exa} - \vec{u}_{FE}\|}{\|\vec{u}_{Exa}\|}$
6.0			
3.0			
1.5			
0.75			
0.375			

Table C.3. Convergence of anlysis results