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

Chapter 1

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 rigour 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.

1.1 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. 1.1 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.

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[1](fig. 1.1) 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. 1.2).

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. 1.2 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.

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

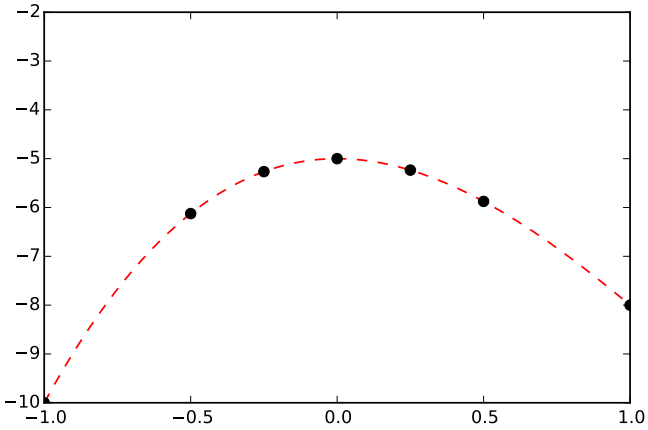


Figure 1.1. The black points represent known values of an otherwise unknown function. The dashed line represents an approximation to the unknown function in terms of a polyonmial of order $n-1$.

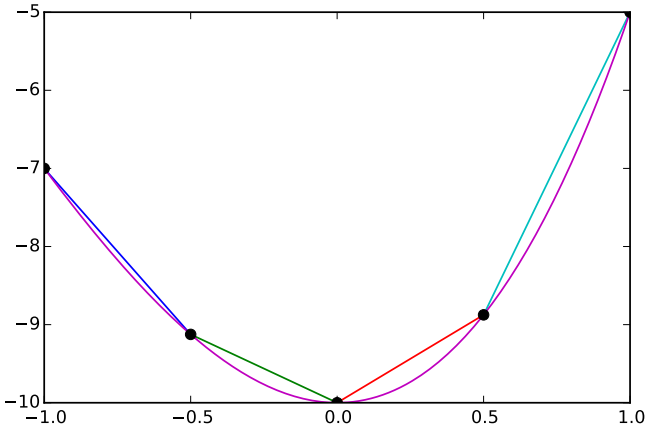


Figure 1.2. Interpolation takes place inside each sub-interval producing a piecewise interpolation approximation to the unknown function

1.2 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;

$$p(x) = L^1(x)f(x^1) + L^2(x)f(x^2) + \ldots + L^n(x)f(x^n) \tag{1.1}$$

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

$$L^I(x) = \prod_{\substack{J=1 \\ I \neq J}}^{n} \frac{(x - x^J)}{(x^I - x^J)}. \tag{1.2}$$

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. (1.1). 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. (1.1) can be equivalently written like:

$$p(x^I) = L^I(x)f(x^I) \tag{1.3}$$

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.

1.3 Interpolation of a function using 3 data points

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 i,t appears

Table 1.1 contains the exact values for 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 $[-1, 1]$. Use a Lagrange interpolation scheme to find an interpolating polynomial that approximates the function and its first derivative at $x = 0.7$. Assume that the values of the first derivative at the nodal points are unknown.

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

Table 1.1. 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. (1.2) the interpolation functions, shown in fig. 1.3, are:

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

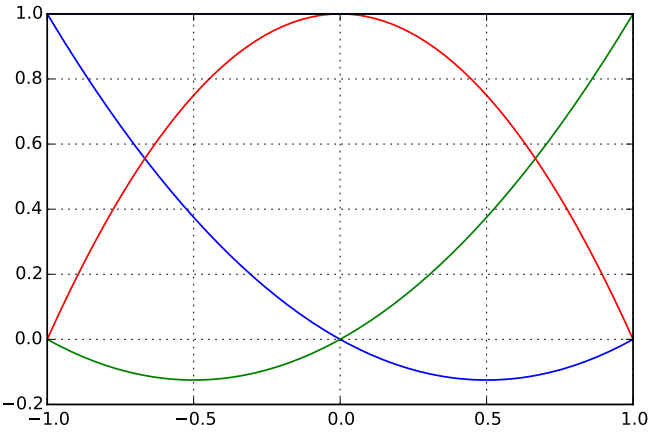


Figure 1.3. (top) Interpolation polynomials $L^1(x)$, $L^2(x)$ and $L^3(x)$ as per eq. (1.2) computed for a second order scheme with 3 nodal points.

The interpolating polynomial approximating the function is obtained using eq. (1.3) 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. 1.4. 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.

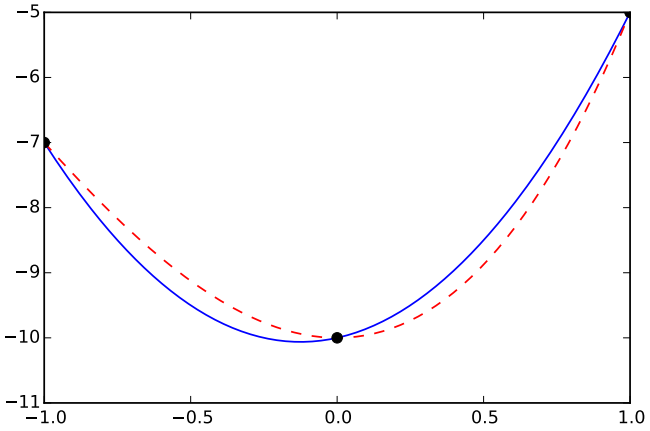


Figure 1.4. Resulting interpolating function $p(x)$ as per eq. (1.3) compared with the exact function $f(x) = x^3 + 4x^2 - 10$.

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:

$$\frac{dp(x)}{dx} = \frac{dL^1(x)}{dx} f^1 + \frac{dL^2(x)}{dx}(x) f^2 + \frac{dL^3(x)}{dx} f^3 \tag{1.4}$$

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 $\frac{dL^I(x^J)}{dx} = \delta^{IJ}$.

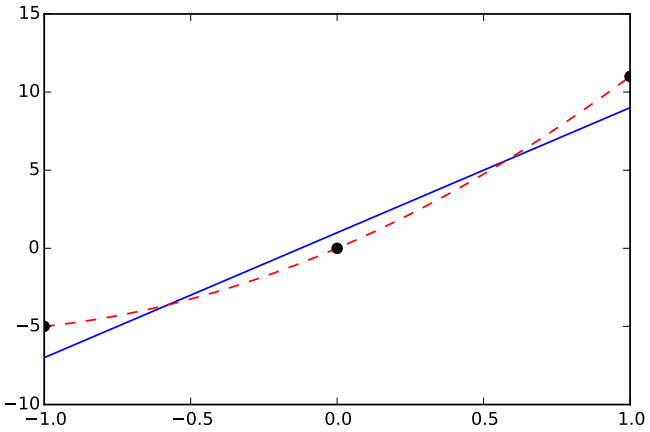


Figure 1.5. Comparison between the first order derivative of $p(x)$ with the closed-form result of $\frac{df(x)}{dx}$

Figure 1.6 shows the approximation to $f'(x)$ when $p(x)$ is computed using 4th-order polynomials.

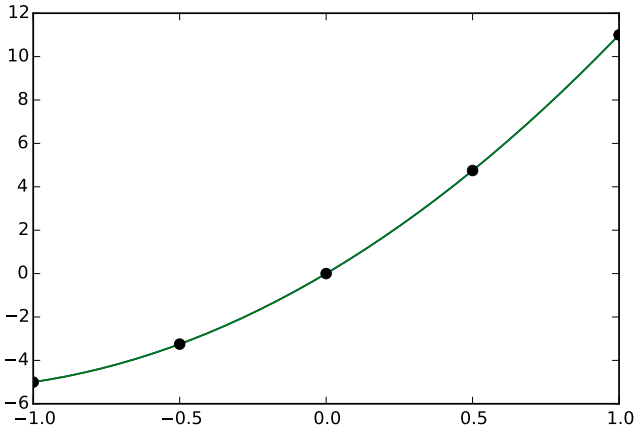
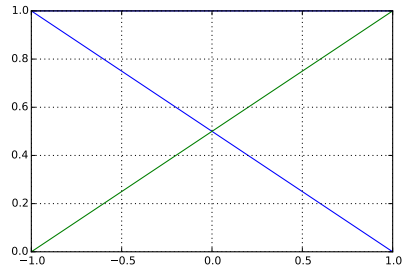
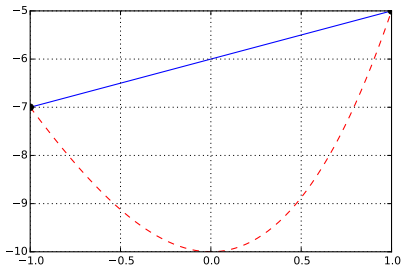


Figure 1.6. Comparison between the first order derivative of $p(x)$ computed using 4th-order polynomials with the closed-form result of $\frac{df(x)}{dx}$

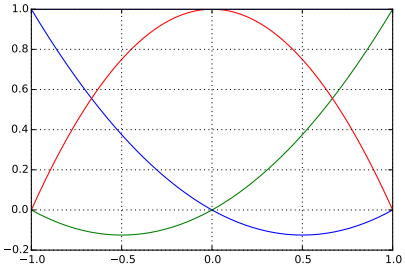
To identify the variation in the solution with different interpolation schemes fig. 1.7 compares the exact and interpolated solution for polynomials of order 1, 2 and 4 respectively. The left column displays the interpolation polynomials $L^I(x)$.



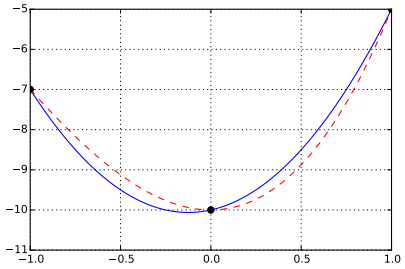
(a) First order.



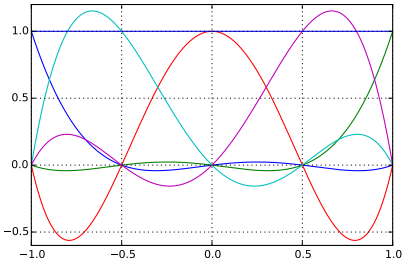
(b) Actual and interpolated function.



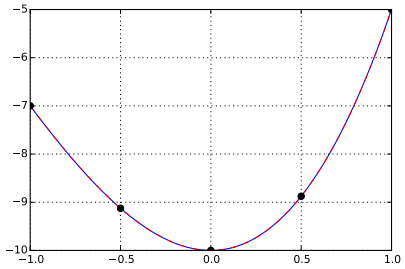
(c) Second order.



(d) Actual and interpolated function.



(e) Fourth order.



(f) Actual and interpolated function.

Figure 1.7. Interpolation of the function $f(x) = x^3 + 4x^2 - 10$ using Lagrange polynomials of increasing order.

1.3.1 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 1.2 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 1.2. Partition of the interval $[-1.0, 1.0]$ into subdomains

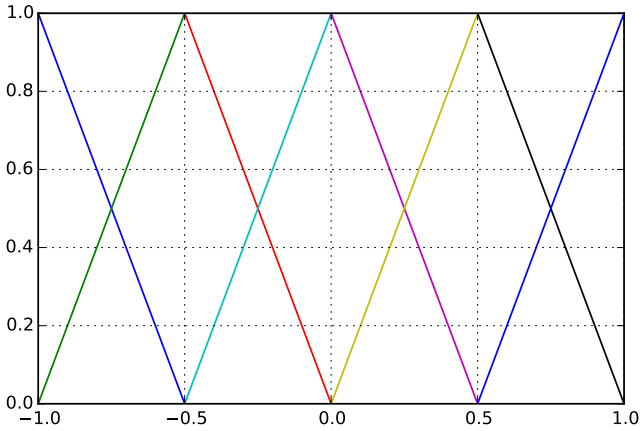
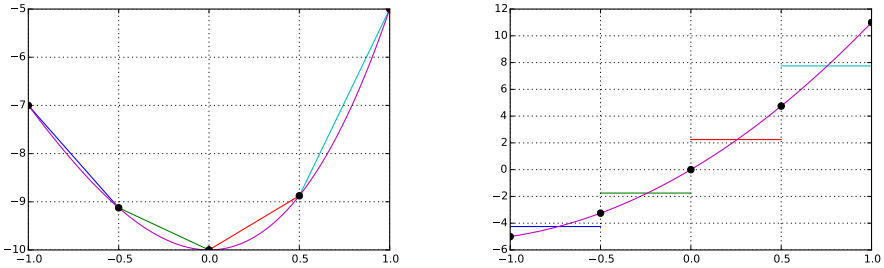


Figure 1.8. Local interpolating polynomials.

This approach results in an interpolating polynomial as shown in fig. 1.9 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 1.9. Locally based interpolation scheme for the function $f(x) = x^3 + 4x^2 - 10$.

1.3.2 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 considered 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. 1.10.

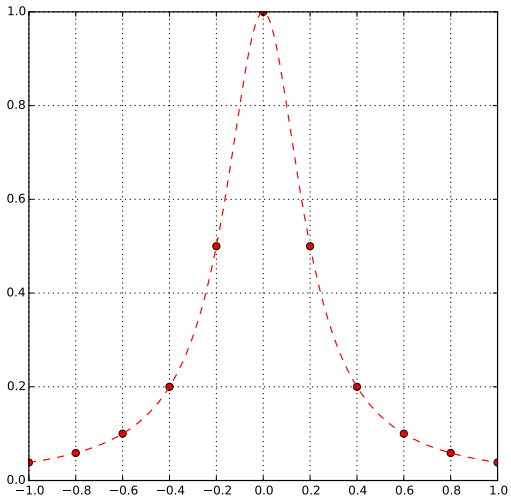
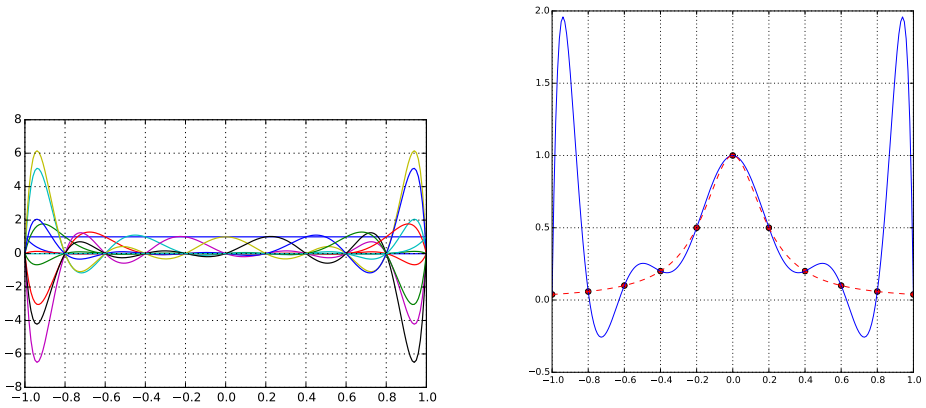


Figure 1.10. Runge function $f(x) = \frac{1}{1+25x^2}$.

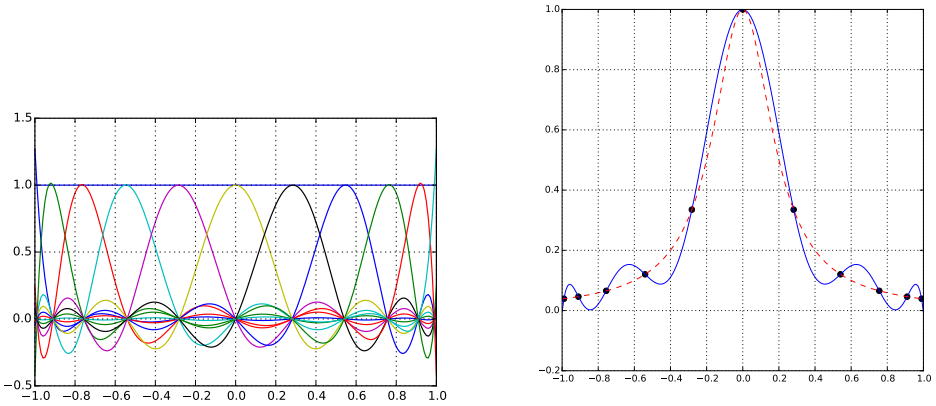
This function shows a strong spatial variation requiring an interpolating polynomial of larger order and as a result a larger number of nodal points. The 11 black dots shown in the figure represent nodal points equally spaced at $\Delta x = 0.2$ and where the function is assumed to be known. We wish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

Figure 1.11 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. 1.12 where there is a large concentration of nodal points along the edges.



(a) **(b)**

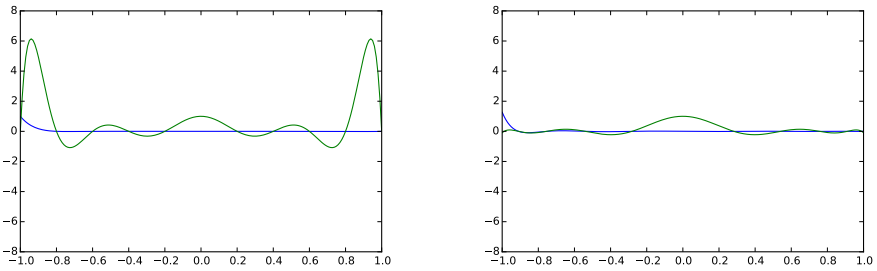
Figure 1.11. (a) Lagrange interpolation polynomials of order 10 associated to the 11 sampling points for fig. 1.10. (b) Interpolating polynomial to approximate the Runge function with the order-10 polynomials from part (a).



(a) **(b)**

Figure 1.12. (a) Polinomios de Lagrange de orden 10 asociados con puntos de muestreo no equi-distantes. (b) función de interpolación para aproximar la función de Runge construida con los polinomios de orden 10 de la parte(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. 1.11) as shown in part(a) of fig. 1.13. 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 1.13. (a) Equidistant nodal points (b) Non-uniform nodal points.

Homework activities prior to the class

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 $[-1, 1]$.

Class activity 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 $f(x)$.

Class activity For the Runge function defined by:

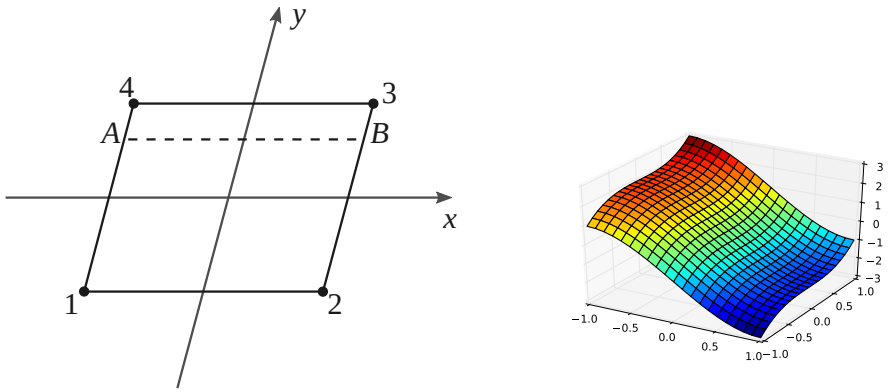
$$f(x) = \frac{1}{1 + 25x^2}$$

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.

Class activity 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.

1.4 Extension to 2D 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-points $\{(\overrightarrow{x}^1, f^1), \cdots, (\overrightarrow{x}^n, f^n)\}$. The domain and the visualization of the function are shown in fig. 1.14.



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

Figure 1.14. 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 naturally 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. 1.14 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. 1.15. 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. (1.2).

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:

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

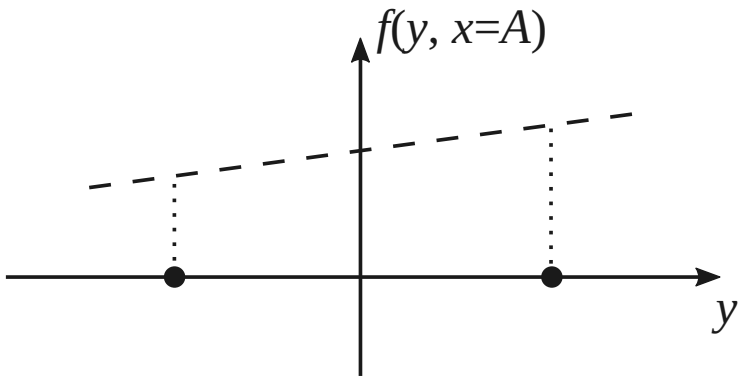


Figure 1.15. Interpolation along the y-direction

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

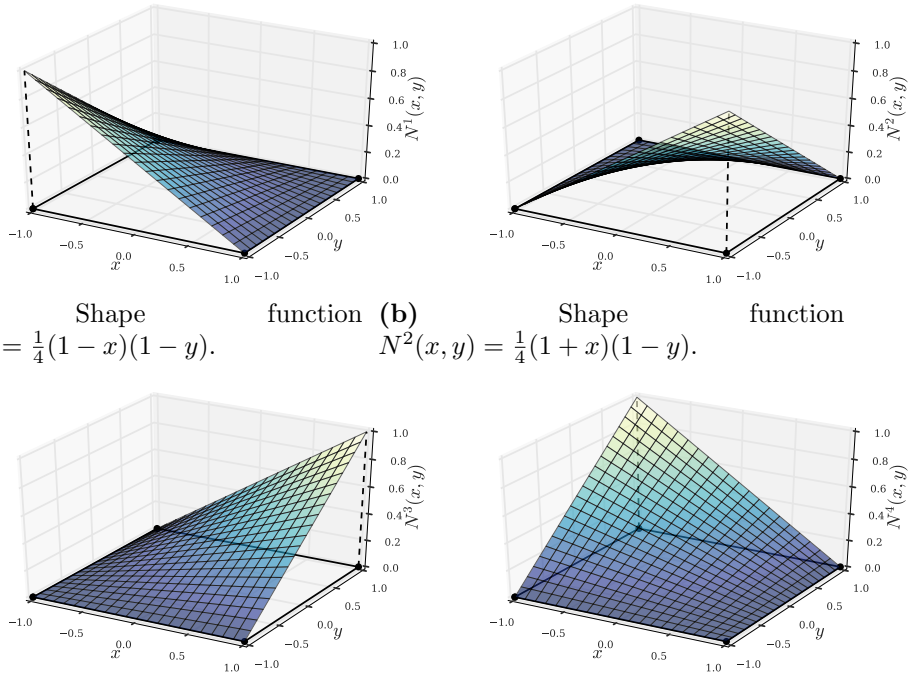
$$
\begin{aligned}
L^A(x) &\equiv L^1(x) \\
L^B(x) &\equiv L^2(x) \\
L^1(y) &\equiv L^1(y) \\
L^2(y) &\equiv L^1(y) \\
L^3(y) &\equiv L^2(y) \\
L^4(y) &\equiv L^2(x) \quad .
\end{aligned}
$$

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

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

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 1.16 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 $N^Q(x, y)$. The shape functions for a 9-noded element are displayed in fig. 1.17.



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

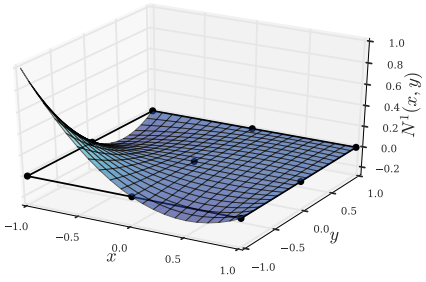
(b) Shape function $N^2(x,y) = \frac{1}{4}(1+x)(1-y)$.

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

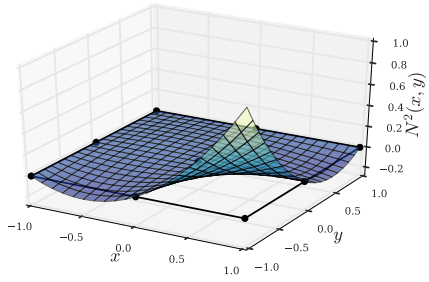
(d) Shape function $N^4(x,y) = \frac{1}{4}(1-x)(1+y)$.

Figure 1.16. Shape functions for a 4-nodes element.

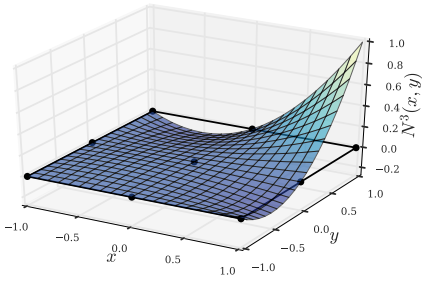
Finite element mesh Spatial-discretization of the computational domain is in the core of finite element analysis. This corresponds to the partition of the whole domain into **finite elements**. The complete set of finite elements, and its defining attributes, corresponding to a particular domain is termed 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.



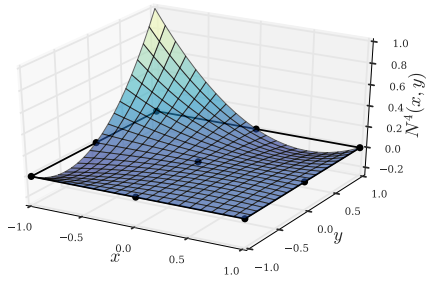
(a) Shape function $N^1(x, y)$.



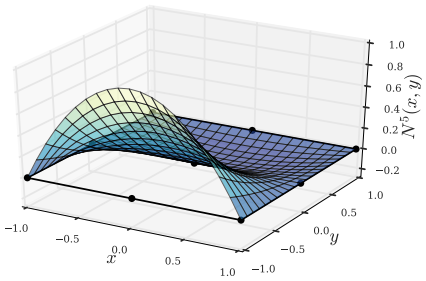
(b) Shape function $N^2(x, y)$.



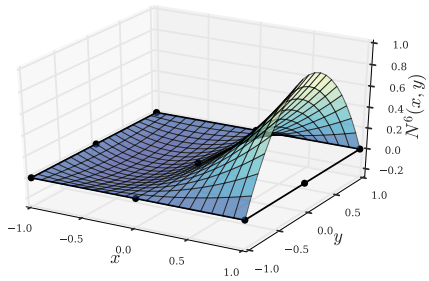
(c) Shape function $N^3(x, y)$.



(d) Shape function $N^4(x, y)$.

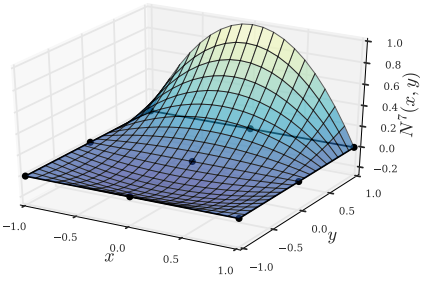


(e) Shape function $N^5(x, y)$.

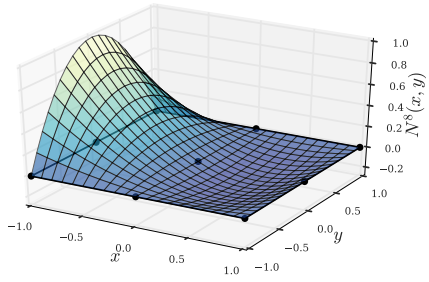


(f) Shape function $N^6(x, y)$.

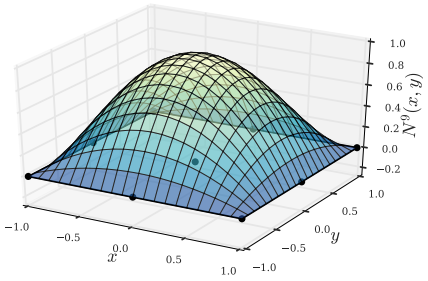
Figure 1.17. Shape functions for a 9-nodes element.



(g) Shape function $N^7(x, y)$.



(h) Shape function $N^8(x, y)$.



(i) Shape function $N^9(x, y)$.

Figure 1.17. Shape functions for a 9-nodes element. (Continued)

Class activity: Interpolation of a vector valued function in a $2D$ space. 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}.

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:

$$\delta W = \int_V \sigma_{ij} \delta u_{i,j} \, \mathrm{d}V \tag{1.5}$$

and then proceed to introduce discretization or approximations via interpolation theory. In this case it is useful to combine the index notation to describe the physical tensorial fields and at the same time the superposition implicit in interpolation schemes. For instance in the representation of a vector valued function using index notation a subscript like i in the representation u_i refers to the cartesian components of the vector field. If this vector valued function is also approximated in terms of an interpolating polynomial built in terms of interpolation (or shape) functions $L^Q(x)$ or $N^Q(x, y)$ then the approximated vector field can be written like:

$$u\_i \, (x, y, z) \; = \; N_i^Q(x, y, z) u^Q.$$

In this expression the subscript i, corresponding to the the physical components of the vector field u_i has been carried out as a subscript to the shape function 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. Condensing also the space dependence of the involved functions after using $\overrightarrow{x} = x\widehat{i} \; + \; y\widehat{j} \; + \; z\widehat{k}$ the above can be written like:

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

The main advantage in this notation is the possibility of conducting further operations, as required in the derivation of the algorithm, while combining physical and discrete information. To clarify, consider the term $\delta u_{i,j}$ in eq. (1.5) which corresponds to the spatial derivatives of the vector field δu_i leading to the an interpolated version of the second order tensor field $\delta u_{i,j}$. Clearly this term can be written like:

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

Deriving this expression to arrive at $\delta u_{i,j}$ we have

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

where it must be observed that the derivative has been carried out to the shape function since the δu^Qs are just constants corresponding to values of δu_i at the nodal points Q. Making

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

the above can be written like:

$$\delta u_{i,j}(\overrightarrow{x}) = B_{ij}^Q(\overrightarrow{x})\delta u^Q. \tag{1.6}$$

In the resulting final expression, eq. (1.6), 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 $N^Q(x, y)$.

To further explain the use of the combined notation consider 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

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

Using the above set of results in eq. (1.5) gives:

$$\delta W = \delta u^Q \int_V H^Q_{ij} C_{ijkl} H^P_{kl} \, \mathrm{d}\, V u^P \tag{1.7}$$

which is the final discrete version of eq. (1.5).

Class activity: 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:

$$\int_V \sigma_{ij}\delta u_{i,j} \, \mathrm{d}V - \int_V f_i \delta u_i \, \mathrm{d}V - \int_{S_t} t^n_i \delta u_i \, \mathrm{d}S = 0. \tag{1.8}$$

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:

- Write the discrete version of eq. (1.8)
- 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.

1.5 Interpolation over distorted domains

So far all the 2D interpolation operations have taken place over perfectly square domains where the interpolation functions are known at least in terms of the side parameter h. In many cases and particularly in finite element algorithms it is common to find distorted (although still quadrilateral) interpolation domains 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. In this case the solution approach is analogous to the scaling operation conducted previously when the domain had a side different than h. In the case of the distorted domain the trick is to use also a canonical element with prescribed interpolation functions and transform between both spaces using also interpolation. This idea is explained in fig. 1.18. 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. Similarly, the space of the canonical element, with position vector $\overrightarrow{r}$ is termed the natural space. For reasons that will be explained later, it is convenient to have canonical elements of side $h = 2.0$.

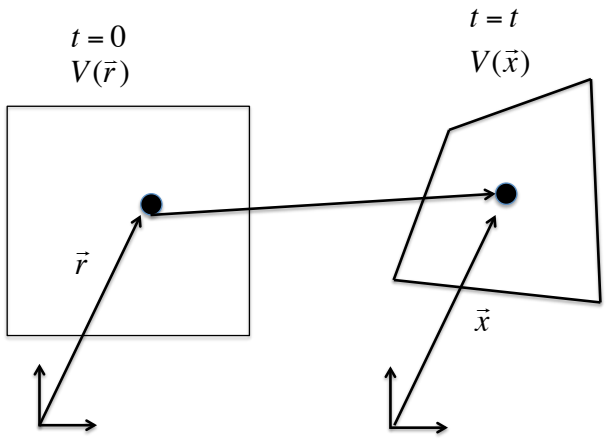


Figure 1.18. Definition of the natural domain

Mathematically, the connection between both spaces is written in

$$\begin{aligned} x_i &= x_i(\overrightarrow{r}) \\ r_I &= r_I(\overrightarrow{x}). \end{aligned} \tag{1.9}$$

In particular 1.9(a) can be written using:

$$x_i = N^Q(\overrightarrow{r})x^Q. \tag{1.10}$$

These relationships establish a one-to-one connection between each point in the physical space to a corresponding point in the canonical space. The first expression provides the position vector x_i 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 r_I in the canonical space for a point that occupies the position vector $\overrightarrow{x}$ in the physical space. The

transformation between the physical and canonical can be used to transform functions.

Assume $f = f(\overrightarrow{x})$ to be the function to be interpolated representing a physical variable that varies with $\overrightarrow{x}$. Using 1.9(a) we have:

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

where now $F = F(\overrightarrow{r})$ represents the same physical variable but expressed in terms of the position vector in the canonical space where it can be approximated via interpolation using pre-defined interpolation functions as given by:

$$F(\overrightarrow{r}) = N^Q(\overrightarrow{r})f^Q. \tag{1.11}$$

Note that as a result of the space transformation 1.9, 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.

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

- Use external software[2] 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.

[2]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.