Chapter

A MATLAB-Based Symbolic Approach for the Quick Developing of Nonlinear Solid Mechanics Finite Elements

Antonio Bilotta

Abstract

A symbolic mathematical approach for the rapid early phase developing of finite elements is proposed. The algebraic manipulator adopted is MATLAB® and the applicative context is the analysis of hyperelastic solids or structures under the hypothesis of finite deformation kinematics. The work has been finalized through the production, in an object-oriented programming style, of three MATLAB® classes implementing a truss element, a tetrahedral element and plane element. The approach proposed, starting from the mathematical formulation and finishing with the code implementation, is described and its effectiveness, in terms of minimization of the gap between the theoretical formulation and its actual implementation, is highlighted.

Keywords: FEM, nonlinear solid mechanics, MATLAB®, Symbolic Math Toolbox™, object oriented

1. Introduction

The developing of finite element formulations, standard or new ones, requires a lengthy process which involves several steps.

The typical starting point is the formulation of a mathematical model where the main physical or real world phenomena to be described are established. At present time the definition of a mathematical model is at the basis of any serious attempt to obtain previsions in any engineering application [1–3], but not only in the engineering field [4, 5].

The subsequent step is the introduction of a numerical approximation technique. The most popular technique is the Finite Element Method (FEM), see [6–8], but now the number of computational approximation techniques is very large and a synthetic summary can be tried only by citing some of these less conventional methods: mixed finite element methods [9–13]; partition of unity-based discontinuous finite elements [14, 15]; meshless methods [16]; discontinuous Galerkin methods [17]. This operation leads to the identification of the needed discrete operators which define the computational model. For example, in the case of the analysis of solid mechanics problems by FEM, important discrete operators are the mechanical response vector of the finite element and its tangent stiffness matrix. This phase is characterised by the evaluation and the analysis of these operators through an algebraic manipulator such as MATLAB® [18]. MATLAB® is certainly one of the state-of-the-art mathematical softwares available for performing numeric or symbolic analyses, but it is not the only one and a quite long list, see [19], of packages offering very similar features is available.

The discrete model so defined is usually inserted into a prototype code, often by using again an algebraic manipulator but the use of compiled programming languages is also possible if not common. This prototype code allows to perform basic tests with the aim to check the effectiveness of the adopted model with respect to well known situations and to check for the presence of bugs. Often this phase can highlight also flaws in the mathematical model or in the discretization technique. In any case it is necessary to go back and to repeat the process just described.

The additional last step can be the production of an executable by using compiled programming languages such as C/C++ or fortran. This makes possible to extend the validation of the conceived numerical model by performing the analysis of larger sized problems.

As already said, the previously described work-flow is lengthy and it is often characterised by a gap between the theoretical formulation and its implementation in a numerical code. However some solutions capable to assist the developer in this process already exist ant it is worth to mention some ones. Such solutions typically refer to a specific context by keeping fixed the physical problem but letting open the specific instance of discretization technique which, however, is fixed too. This is the case of open source FEM libraries or commercial packages listed in [20]. In both cases the user must define the procedures or functions needed in order to assign the desired new finite element to be used within the analysis framework already available in the library. Other packages instead solves a generic system of Partial Differential Equations (PDEs) subjected to boundary and initial conditions. Inside this generic form the specific differential problem to be solved must to be fitted by the user, see for example [21, 22], but usually with no control over the discretization technique used by the solver.

The present work, quite far from being an alternative to the hugely developed and rich packages previously cited, proposes a basic approach for the quick early phase developing of solid mechanics finite elements formulation. Its intent is to show how to use MATLAB®, in particular by exploiting the capabilities of the Symbolic Math Toolbox™ [23], to produce numerical approximations of a given solid mechanics problem in a way that the usual gap between the theoretical formulation and its actual implementation in a code is not perceived. This result is obtained by condensing the development process going from the mathematical formulation to the prototype code implementation in a few lines of MATLAB® symbolic instructions. The applicative context is the nonlinear analysis of solids and structures, see [24, 25], by showing the formulation and the subsequent MATLAB® coding of some typical structural and solid finite elements. The mechanical formulation is based on the kinematics of finite deformation and, for the description of the material behaviour, on isotropic hyperelasticity, i.e. the stress solution is found as a derivative of some potential energy function. This allows to express the mechanical problem at hand in terms of the stationary condition of the Total Potential Energy (TPE). The stationary condition, assuming a fem discretization of the given domain, is then easily translated into a nonlinear algebraic problem whose unknowns are the position vectors of the nodes of the mesh used in the discretization.

The following finite elements are discussed: a truss element, a 3D tetrahedral element with four nodes and a four nodes quadrangular element subjected to plane strain condition. The choice allows to discuss gradually the main ingredients present in a finite element formulation and how these can be framed inside the proposed

MATLAB® approach. The latter is based on the definition of MATLAB® classes which share the same structuring and which differ only for the particular mechanical response to be implemented. In particular the generic class is structured as shown by the following instructions (Listing 1.1).

Listing 1.1. Generic class.

```
 classdef Element
          properties (SetAccess = private)
            % symbolic properties

            % numeric properties
          end

          methods
          function E = Element()
          end

          function E = Initialize (E,D, i)
          end

          function E = Compute(E)
          end

          function sig = Stress(E,gx)
          end
          end
 end
```

The properties section contains a group of symbolic properties devoted to handle the unknowns and the quantities depending on them used in the description of the element mechanical behaviour. The other group of numeric properties are used to handle quantities that are known and then they can have a numeric value. Beyond the constructor, that must to be present in any class, we have the function *Initialize,* belonging to the *pre-processing phase* of a FEM code, whose main task is the inizialization of the i-th element on the basis of the assigned data structure D. This is the moment also for evaluating the element operators, in a symbolic format, needed to the analysis. In the subsequent *analysis phase,* the function Compute evaluates the numeric instances of the symbolic operators previously prepared. The function Stress is typical of the *post-processing phase* of any FEM code and its task is to compute the stress solution inside the generic element starting from the kinematic global solution represented by vector gx.

Before proceeding with the description of the proposed work, it is noteworthy to observe that the use of MATLAB® to perform mathematical and numeric analyses is not certainly new and several books are dedicated to this subject, see [26–28] just to cite a few. Moreover the already cited book [24] employs MATLAB® for the implementation of a FEM software. However the present less comprehensive work is different because it carry out the formulation of the FEM operators by exploiting the potentiality of the symbolic manipulator and advising an object-oriented programming style.

A last further annotation regards the use of the symbolic approach which, with respect to the expected performance of final codes, represents a weakness. This aspect however is to be considered less important in a work regarding the early phase developing of a FEM formulation. Nevertheless techniques, [29–31], for the automatic generation of efficient and highly compressed code is a research theme which is attracting increasingly interest, making viable the up-scaling of the proposed approach.

The chapter is organised as follows. Section 2 presents the FEM formulation of the Total Potential Energy for a generic structure or solid, showing also the evaluation of the gradient needed to define the discrete equilibrium equations and the evaluation of the Jacobian necessary for their solution. Sections 3, 4 and 5 describes, respectively, the truss element, the tetrahedral element and plane quadrangular element. The closing section furnishes some additional final comments.

2. Total Potential Energy

An effective description of a generic mechanical problem can be obtained through the stationary condition of its Total Potential Energy (TPE) which, see for example [24], can be expressed as follows

$$\prod(\boldsymbol{x}) \equiv \prod_{\text{int}}(\boldsymbol{x}) + \prod_{\text{ext}}(\boldsymbol{x}) = \text{stat.} \tag{1}$$

$\boldsymbol{x}$ is the global vector of the current positions of the nodal points defining the mesh used to describe the geometry of the solid. $\prod_{\text{int}}(\boldsymbol{x})$, excluding dynamic and dissipative effects, is given only by the strain energy obtained by summing all the contribution from all the finite elements, i. e.

$$\prod_{\text{int}}(\boldsymbol{x}) = \sum_{\text{e}} \Psi_{\text{e}}(\boldsymbol{x}), \tag{2}$$

being $\Psi_{\text{e}}(\boldsymbol{x})$ the hyperelastic strain energy relative to the generic finite element. $\prod_{\text{ext}}(\boldsymbol{x})$ is the potential energy of external forces. For simplicity the case of a solid body subjected only to external punctual forces will be considered, in this case the potential energy can be written as

$$\prod_{\text{ext}}(\boldsymbol{x}) = -\boldsymbol{f} \cdot \boldsymbol{x}, \tag{3}$$

where $\boldsymbol{f}$ is the global vector of the applied forces in each node of the mesh. $\boldsymbol{f}$ has the same length of $\boldsymbol{x}$, however it is mainly composed by null entries.

On this basis the equilibrium equations can be easily formulated with respect the degrees of freedom involved in the FEM description of the body. In particular, by imposing the stationary condition (1), the equilibrium equations can be derived, obtaining

$$\underset{\text{e}}{\text{A}} \boldsymbol{g}_{\text{e}}(\boldsymbol{x}) - \boldsymbol{f} = \boldsymbol{0}. \tag{4}$$

where the assembly operator A is used to build up the global response vector by using the gradient vector of each finite element strain energy contribution, i.e.

$$\boldsymbol{g}_{\text{e}}(\boldsymbol{x}) = \frac{\partial \Psi_{\text{e}}(\boldsymbol{x})}{\partial \boldsymbol{x}}. \tag{5}$$

The solution of Eq. (4), a typically nonlinear algebraic system whose unknowns are the components of vector $\boldsymbol{x}$, is based on a Newton–Raphson iteration which can be formulated as follows

$$\underset{e}{\mathrm{A}}\boldsymbol{J}_{\mathrm{e}}(\boldsymbol{x}_j)(\boldsymbol{x}_{j+1}-\boldsymbol{x}_j) = -\left(\underset{e}{\mathrm{A}}\boldsymbol{g}_{\mathrm{e}}(\boldsymbol{x}_j) - \boldsymbol{f}\right), \tag{6}$$

where $\boldsymbol{x}_j$ and $\boldsymbol{x}_{j+1}$ are the estimated solutions at j-th and $(j+1)$-th iterations and $\boldsymbol{J}_{\mathrm{e}}(\boldsymbol{x})$ is the Jacobian matrix of the finite element given by

$$\boldsymbol{J}_{\mathrm{e}}(\boldsymbol{x}) = \frac{\partial \boldsymbol{g}_{\mathrm{e}}(\boldsymbol{x})}{\partial \boldsymbol{x}}. \tag{7}$$

The gradient vector $\boldsymbol{g}_{\mathrm{e}}(\boldsymbol{x})$ and the Jacobian matrix $\boldsymbol{J}_{\mathrm{e}}(\boldsymbol{x})$ can be used as basic building blocks for the finite element formulation. This is the approach at the basis of the MATLAB® implementations to be described in the following sections.

3. Truss element

The strain energy of the truss element is defined, see [24], as follows

$$\Psi_{\mathrm{e}}(\boldsymbol{x}) = \frac{1}{2}E\varepsilon^2 V, \varepsilon = \varepsilon(\boldsymbol{x}) = \ln\left(\frac{l}{L}\right), \tag{8}$$

where E is the Young modulus, L and V are the length and the volume of the bar in the reference configuration, l is the length of the bar in the current configuration. The geometric quantities just described are depicted in **Figure 1** where the coordinate vectors of the nodal points are also shown.

The implementation of the MATLAB® class Truss can stem from the properties reported in Listing 1.2. Some of them, those describing the reference configuration, can be numeric because are fixed. The other properties, which describe the current configuration, are expressed in symbolic form in order to be used as quantities whose the strain energy of truss element depends on.

Listing 1.2. Truss class: properties.

```
 properties (SetAccess = private)
          % symbolic properties
          xa % current coordinates of node a
          xb % current coordinates of node b
          xe % current element coordinates
          ge % gradient g (xe)
          Je % Jacobian J (xe)
          eps % strain eps (xe)
          N % axial force N (xe)

          % numeric properties
          a % node a global index
          b % node b global index
          Xe % reference element coordinates
          % ...
 end
```

During the pre-processing phase the numeric properties of the class are appropriately assigned and the symbolic properties are evaluated as shown in the following listing (Listing 1.3). This happens inside the function Initialize belonging to the methods of the class.

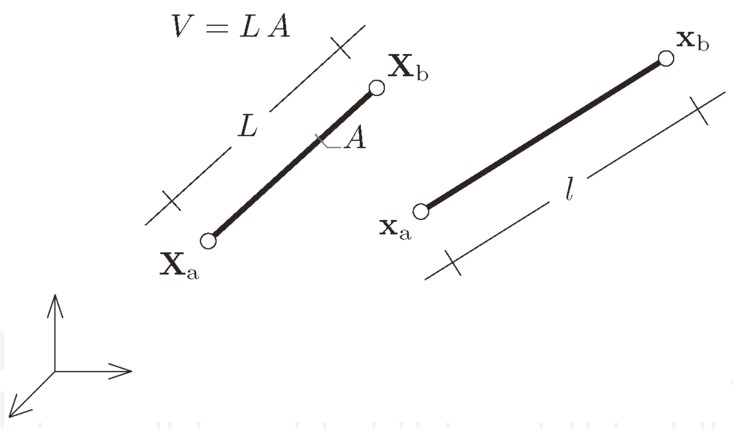

Figure 1.
Truss element: definition of the geometric quantities relative to the reference configuration (upper-case letters) and the current configuration (lower-case letters).

Listing 1.3. Truss class: function Initialize.

```
 function T = Initialize (T,D, i)
            % D brings all the problem data and its use
            % is not shown here
            T. xa = sym ('xa', [3 l], 'real');
            T. xb = sym ('xb', [3 1], 'rea1');
            T. xe = [T. xa; T. xb];

            % reference configuration
            L = dot(Xb–Xa, Xb–Xa);
            L = sqrt(L);

            % current configuration
            1 = dot (T. xb–T. xa, T. xb–T. xa);
            1 = sqrt(1);

            T. eps = log(1/L);
            Psi = 1/2 * E * T.eps^2 * (L*A);
            T. ge = gradient(Psi, T.xe);
            T. Je = jacobian (T.ge, T.xe);

            % preliminary symbolic evaluation of N
            T .N = E*A*T. eps;
 end
```

Listing 1.3 shows that, after the computation of the strain energy using Eq. (8), symbolic properties $\boldsymbol{g}_e$ and $\boldsymbol{J}_e$ are evaluated on the basis of Eqs. (5) and (7), respectively, by simply calling the function gradient and the function jacobian both belonging to the Symbolic Math Toolbox™. This highlights the *short distance* between the formulation and its code implementation.

After having prepared each Truss object in the way described above, it is possible to evaluate, whenever it is needed during the solution of the nonlinear equilibrium equations, the gradient and the Jacobian of the generic element with respect to estimated solution $\boldsymbol{x}_j$, see Eq. (6), by calling the following class method (Listing 1.4).

Listing 1.4. Truss class: function Compute.

```
 function T = Compute(T)
```

```
            T.se = subs (T.ge, T.xe, T.xxe);
            T.Ke = subs (T.Je, T.xe, T.xxe);
 end
```

The function Compute uses the numeric property T.xxe previously filled with the current nodal coordinates values and it stores the resulting numeric expressions of the gradient and Jacobian in the class properties se and Ke. The desired result is obtained by calling MATLAB® function subs which substitutes the symbolic variable T.xe with its numeric value T.xxe.

The post-processing phase of any FEM codes certainly comprehends the evaluation of the stress solution. In the case of the truss element, the axial force must to be computed with respect to each vector $\boldsymbol{x}$ calculated by the solution of the equilibrium Eqs. (4). Listing 1.5 shows the very simple function implementing the required computation.

Listing 1.5. Truss class: function Stress.

```
 function N = Stress (T,gx)
            % extraction of local vector lx from global gx
            N = subs(T.N, T.xe, lx);
 end
```

The complete listing of the class can be found in [32].

4. Tetrahedral element

The discussion of the implementation of a tetrahedral element, in particular a 4 nodes tetrahedron, allows to introduce an important ingredient of all finite element formulations: the interpolation chosen for the kinematic description. Standard approaches are hinged on the interpolation of the displacement field, in the present approach the focus is on the interpolation of the element coordinates in the reference configuration and in the current one. In the previous section regarding the truss element, this aspect remained hidden because the element elongation is easily formulated with respect to the element nodal coordinates.

Another important aspect which the tetrahedral element bring into play is the use of the continuum mechanics instruments, see [24, 25], and how these can be smoothly framed inside the proposed MATLAB® implementation.

Let us consider the geometry of the 4 node tetrahedron as illustrated in **Figure 2**. The description of the reference and current configurations of the tetrahedron are as follows.

$$\begin{aligned}\boldsymbol{X}(\zeta_1,\zeta_2,\zeta_3,\zeta_4) &= N_1\boldsymbol{X}_1 + N_2\boldsymbol{X}_2 + N_3\boldsymbol{X}_3 + N_4\boldsymbol{X}_4 \\ &= \zeta_1\boldsymbol{X}_1 + \zeta_2\boldsymbol{X}_2 + \zeta_3\boldsymbol{X}_3 + \zeta_4\boldsymbol{X}_4,\end{aligned} \tag{9}$$

$$\begin{aligned}\boldsymbol{x}(\zeta_1,\zeta_2,\zeta_3,\zeta_4) &= N_1\boldsymbol{x}_1 + N_2\boldsymbol{x}_2 + N_3\boldsymbol{x}_3 + N_4\boldsymbol{x}_4 \\ &= \zeta_1\boldsymbol{x}_1 + \zeta_2\boldsymbol{x}_2 + \zeta_3\boldsymbol{x}_3 + \zeta_4\boldsymbol{x}_4.\end{aligned} \tag{10}$$

The element local coordinates $\boldsymbol{\zeta} = [\zeta_1\ \zeta_2\ \zeta_3\ \zeta_4]^{\mathrm{T}}$ are the standard tetrahedral coordinates whose definition can be found in several resources, for example [6, 8]. On this basis the description of the deformation gradient over the tetrahedron can be formulated as follows

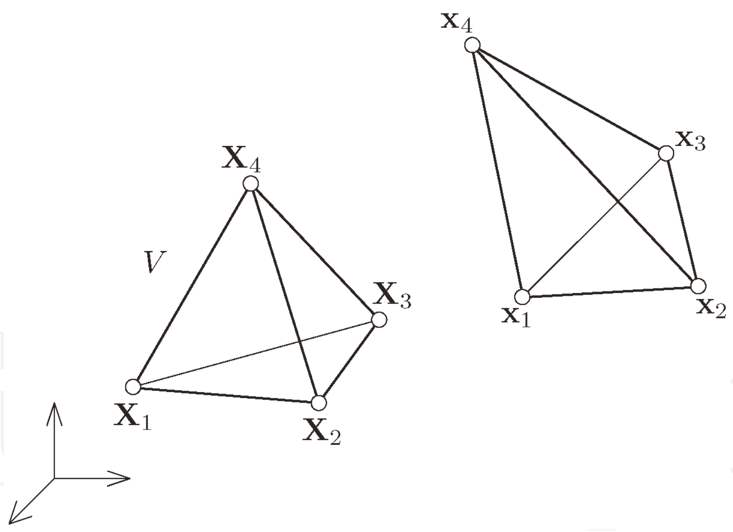

Figure 2.
Tetrahedral element: definition of the geometric quantities relative to the reference configuration (upper-case letters) and the current configuration (lower-case letters).

$$\boldsymbol{F} = \frac{\partial \boldsymbol{x}}{\partial \boldsymbol{X}} = \frac{\partial \boldsymbol{x}}{\partial \boldsymbol{\zeta}}\frac{\partial \boldsymbol{\zeta}}{\partial \boldsymbol{X}} = \frac{\partial \boldsymbol{x}}{\partial \boldsymbol{\zeta}}\left(\frac{\partial \boldsymbol{X}}{\partial \boldsymbol{\zeta}}\right)^{-1} = \boldsymbol{F}(\boldsymbol{x}), \tag{11}$$

with the operator

$$\frac{\partial \boldsymbol{x}}{\partial \boldsymbol{\zeta}} = [\,\boldsymbol{x}_1\boldsymbol{x}_2\boldsymbol{x}_3\boldsymbol{x}_4\,] \tag{12}$$

containing in its columns the four coordinate vectors relative to the current configuration, unknown vectors to be expressed in MATLAB® symbolic format, and the operator

$$\frac{\partial \boldsymbol{X}}{\partial \boldsymbol{\zeta}} = [\,\boldsymbol{X}_1\boldsymbol{X}_2\boldsymbol{X}_3\boldsymbol{X}_4] \tag{13}$$

containing the four coordinate vectors relative to the reference configuration to be evaluated numerically for each tetrahedron of the mesh. The apparent problem represented by the evaluation of inverse $\left(\frac{\partial \boldsymbol{X}}{\partial \zeta}\right)^{-1}$ starting from a 3×4 matrix is a standard matter in FEM procedures, see for example [6], and it can be easily calculated as shown in Appendix A.

The geometric formulation described above is directly inserted inside the Tetra4 class which can be implemented by following the same scheme already adopted for the class Truss. In particular the geometrical properties of the class are listed below (Listing 1.6).

Listing 1.6. Tetra4 class: properties.

```
properties (SetAccess = private)
        % symbolic propetries
        xl % current coordinates of node 1
        x2 % current coordinates of node 2
        x3 % current coordinates of node 3
        x4 % current coordinates of node 4
        xe % current element coordinates
        Fe % deformation gradient
```

```
            % numeric properties
            Xe % reference element coordinates
    end
```

On this basis the evaluation of the deformation gradient can be performed during the initialisation of the generic element by carrying out the following instructions (Listing 1.7).

Listing 1.7. Tetra4 class:function Initialize (evaluation of the deformation gradient).

```
 function T = Initialize (T,D, i)
            % D brings all the problem data and its use
            % is not shown here ...

            % reference configuration
            dXdzeta = [X1 X2 X3 X4];
            A = [1 1 1 1; dXdzeta];
            V = det(A)/6;
            iA = inv(A);
            dzetadX = iA(1:4 ,2:4);

            % current configuration (symbolic)
            dxdzeta = [T.xl T.x2 T.x3 T.x4];

            % deformation gradient
            T.Fe = dxdzeta*dzetadX;

            % ...
 end
```

It is now possibile to discuss the strain energy of the tetrahedral element. The choice is for a compressible neo-Hookean material, see [25], which allows to express the strain energy of the generic tetrahedron as follows

$$\Psi_e(\boldsymbol{x}) = \int_{\Omega_e} \Psi(\boldsymbol{C}) dV = \left(\frac{\mu}{2}(I_1 - 3) - \mu \ln J + \frac{\lambda}{2}(\ln J)^2 \right) V. \tag{14}$$

$\boldsymbol{C} = \boldsymbol{C}(\boldsymbol{x}) = \boldsymbol{F}^{\mathrm{T}}\boldsymbol{F}$ is the right Cauchy strain tensor and I_1 its first invariant, J = det $\boldsymbol{F}$, λ and μ are the Lamè parameters of the material. Thanks to the constant pattern of **F** over the element domain Ω_e, the strain energy of the element is simply given by the product between the strain energy density and the reference volume of the element. Such a evaluation, together with the derivation of the gradient vector and Jacobian matrix is implemented inside the function Initialize as shown in Listing 1.8.

Listing 1.8. Tetra4 class: function Initialize (strain energy).

```
 function T = Initialize (T,D, i )
            % ...
            C = T. Fe.'*T.Fe;
            Il = trace (C);
            J = det (T.Fe);

            Psi = (mi/2*(Il-3)-mi*log(J) + lam/2*log (J)^2) *V;
            T.ge = gradient(Psi, T.xe);
            T.Je = jacobian(T.ge, T.xe);
            % ...
    end
```

The symbolic gradient vector and Jacobian matrix evaluated in the initialization phase are then numerically computed during the analysis using a function identical to the function already presented in Listing 1.4 for the Truss class.

The basic operation of the post-processing is the computation of the Cauchy stress solution which, as **F**, is constant over the element domain. This step requires the evaluation of the second Piola-Kirchhoff stress tensor

$$\boldsymbol{S} = 2\frac{\partial\Psi}{\partial\boldsymbol{C}} = \boldsymbol{S}(\boldsymbol{C}) \tag{15}$$

and, by applying a push-forward operation to **S** [24, 25], the computation of the Cauchy stress tensor is

$$\sigma = J^{-1}\boldsymbol{F}\boldsymbol{S}(\boldsymbol{C})\boldsymbol{F}^{\mathrm{T}}. \tag{16}$$

The MATLAB® implementation of Eqs. (15) requires the introduction of a symbolic matrix for **C** to be used to perform another evaluation of the strain energy depending, this time, from the components of **C**. The obtained expression, $\Psi(\boldsymbol{C})$, can be derived in order obtain **S**. This step can be performed only one time during the initialisation of the Tetra4 class. Listing 1.9 shows these instructions together with the declaration of the necessary symbolic properties.

Listing 1.9. Tetra4 class: function Initialize (second Piola-Kirchhoff stress tensor).

```
            properties (SetAccess = private)
            % ...

            % symbolic properties
            Se % second Piola-Kirchhof stress tensor S(C)
            Ce % symbolic tensor C from which Se depends
    end

    function T = Initialize (T,D, i )
            % ...
            T.Ce = sym ('C', [3, 3], 'real');
            I1 = trace (T. Ce);
            I3 = det (T.Ce);
            Psi = mi/2*(Il – 3)–mi*log ( sqrt (I3))+ ...
                  lam/2*log (sqrt( I3))^2;
            T.Se = reshape (2* gradient (Psi ,T.Ce (:)) ,3 ,3 );
    end
```

Eq. (16) is used to compute the stress solution for each tetrahedron with respect to all the solutions $\boldsymbol{x}$ found by means of equilibrium equations (6). The class function implementing the required operations is reported in Listing 1.10.

Listing 1.10. Tetra4 class: function Stress.

```
    function sig = Stress (T,gx)
            % extraction of local vector lx from global gx
            F = subs (T.Fe, T.xe, lx);
            C = F.'*F;
            sig = F*subs (T.Se, T.Ce, C)*F.'/det(F);
    end
```

The complete listing of the class can be found in [33].

5. Plane strain 4 nodes element

The use of finite elements specifically formulated for the analysis of problems which admit a 2D reduction is very common and quadrangular elements play an important role in the case of simple geometries. In this section a 4 nodes quadrangular element subjected to plain strain condition is discussed. The element is very basic but it allows to discuss also the use of the Gauss integration points in the calculation of the required FEM operators. The use of the Gauss integration point is an important cornerstone for all finite element formulations.

Plane strain condition stems from the following assumption on the transformation defining the new configuration of each point of the body

$$x_1 = x_1(X_1, X_2), \tag{17}$$

$$x_2 = x_2(X_1, X_2), \tag{18}$$

$$x_3 = X_3. \tag{19}$$

Consequently, the associated deformation gradient takes the following form

$$\boldsymbol{F} = \begin{bmatrix} F_{11} & F_{12} & 0 \\ F_{21} & F_{22} & 0 \\ 0 & 0 & 1 \end{bmatrix}, \boldsymbol{F}_{2\times 2} \begin{bmatrix} F_{11} & F_{12} \\ F_{21} & F_{22} \end{bmatrix}. \tag{20}$$

Eqs. (18)–(20) allow the dealing with a 2D kinematic description. The stress solution, however, is not strictly plane because Eq. (19) constitutes an internal constraint determining also the presence of the component σ_{33}. This component anyway depends only from the 2D kinematic solution as it will be shown in the following.

The standard shape function of the four nodes plane element are

$$\begin{aligned} N_1 &= \frac{1}{4}(1-\zeta_1)(1-\zeta_2), N_2 = \frac{1}{4}(1+\zeta_1)(1-\zeta_2) \\ N_3 &= \frac{1}{4}(1+\zeta_1)(1+\zeta_2), N_4 = \frac{1}{4}(1-\zeta_1)(1+\zeta_2) \end{aligned} \tag{21}$$

being $\boldsymbol{\zeta} = [\zeta_1 \zeta_2]^{\mathrm{T}}$ the element local coordinates used for quadrangular elements, see for example [6]. Shape functions (21) can be properly used to describe, see **Figure 3**, the reference configuration and the current configuration of the element giving

$$\boldsymbol{X}(\zeta_1, \zeta_2) = N_1\boldsymbol{X}_1 + N_2\boldsymbol{X}_2 + N_3\boldsymbol{X}_3 + N_4\boldsymbol{X}_4. \tag{22}$$

$$\boldsymbol{x}(\zeta_1, \zeta_2) = N_1\boldsymbol{x}_1 + N_2\boldsymbol{x}_2 + N_3\boldsymbol{x}_3 + N_4\boldsymbol{x}_4. \tag{23}$$

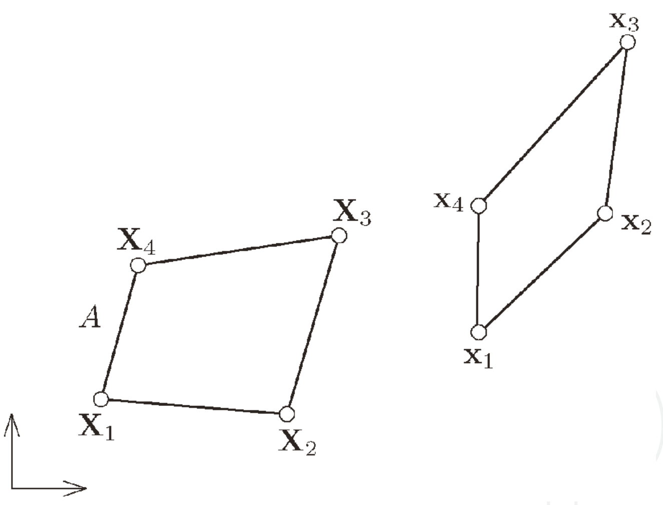

Figure 3.
Four nodes plane element: definition of the geometric quantities relative to the reference configuration (uppercase letters) and the current configuration (lower-case letters).

We have exactly the same pattern of the tetrahedron element, see Eqs. (9) and (10), except for the meaning of the shape function and the 2D dimension of the symbolic vectors $\boldsymbol{x}_i (i = 1 \dots 4)$ and numeric vectors $\boldsymbol{X}_i (i = 1 \dots 4)$. The deformation gradient can be evaluated using always Eq. (11) where now the operators are

$$\frac{\partial \boldsymbol{x}}{\partial \zeta} = \left[\left(-\frac{(1-\zeta_2)}{4}\boldsymbol{x}_1 + \frac{(1-\zeta_2)}{4}\boldsymbol{x}_2 + \frac{(1+\zeta_2)}{4}\boldsymbol{x}_3 - \frac{(1+\zeta_2)}{4}\boldsymbol{x}_4 \right) \left(-\frac{(1-\zeta_1)}{4}\boldsymbol{x}_1 - \frac{(1+\zeta_1)}{4}\boldsymbol{x}_2 + \frac{(1+\zeta_1)}{4}\boldsymbol{x}_3 + \frac{(1-\zeta_1)}{4}\boldsymbol{x}_4 \right) \right] \tag{24}$$

and

$$\frac{\partial \boldsymbol{X}}{\partial \zeta} = \left[\left(-\frac{(1-\zeta_2)}{4}\boldsymbol{X}_1 + \frac{(1-\zeta_2)}{4}\boldsymbol{X}_2 + \frac{(1+\zeta_2)}{4}\boldsymbol{X}_3 - \frac{(1+\zeta_2)}{4}\boldsymbol{X}_4 \right) \left(-\frac{(1-\zeta_1)}{4}\boldsymbol{X}_1 - \frac{(1+\zeta_1)}{4}\boldsymbol{X}_2 + \frac{(1+\zeta_1)}{4}\boldsymbol{X}_3 + \frac{(1-\zeta_1)}{4}\boldsymbol{X}_4 \right) \right] \tag{25}$$

are 2×2 matrices depending on the local coordinates of the element. Then the necessity to use the Gauss integration points in the evaluation of the strain energy of the element and, as a consequence, of the gradient and Jacobian of the element, see Eqs. (5) and (7). In particular four Gauss points are used, their coordinates and weights can be found in any FEM text book and are also shown in the complete listing of the class available in [34].

Previous discussion introduces the implementation details of the MATLAB® class PF4, PF stays for Plane **F**, whose kinematic properties, see Listing 1.11, are similar to those used for the class Tetra4 plus other properties required for the Gauss integration points. These properties are used to implement Eqs. (11), (24) and (25) which must to be evaluated in each Gauss point (bulky details are not shown but they can be found in the complete listing of the class, see [34]).

Listing 1.11. PF4 class: kinematic properties and evaluation of F.

```
properties (Constant)
```

```
            nG = 4;
            xiG = % Gauss coordinates , values not shown here
            wG = [1 1 1 1];
    end

    properties (SetAccess = private)
            % symbolic properties
            xl % current coordinates of node 1
            x2 % current coordinates of node 2
            x3 % current coordinates of node 3
            x4 % current coordinates of node 4
            xe % current element coordinates
            Fe % deformation gradient F(xe) (nGP times)

            % numeric properties
            Xe % reference element coordinates
    end

    function PF = Initialize (PF,D, i)
            % D brings all the problem data and its use
            % is not shown here

            PF . Fe = sym( zeros (2 ,2 ,PF .nG));
            for g = l:PF.nG
               % dzetadX evaluation in g
               % ...

               % dxdzeta evaluation in g
               % ...

               % F in g
               F = dxdzeta * dzetadX;
               PF.Fe(:, :, g) = F;

               % ...
            end
    end
```

In each Gauss integration point the strain energy, the compressible neo-Hookean form is used again, must to be evaluated by taking into account the simplification determined by the plane form assumed by tensor $\boldsymbol{F}$, then

$$J = \det\boldsymbol{F} = \det\boldsymbol{F}_{2\times2}, \tag{26}$$

and by tensor $\boldsymbol{C}$

$$\boldsymbol{C} = \begin{bmatrix} C_{11} & C_{12} & 0 \\ C_{21} & C_{22} & 0 \\ 0 & 0 & 1 \end{bmatrix}, \quad \boldsymbol{C}_{2\times2} = \begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix}, \tag{27}$$

from which

$$I_1 = \mathrm{tr}\boldsymbol{C} = \mathrm{tr}\boldsymbol{C}_{22} + 1. \tag{28}$$

Then the expression of the strain energy density valid for the plane strain condition is

$$\Psi_{PF} = \frac{\mu}{2}(I_1 - 2) - \mu \ln J + \frac{\lambda}{2}(\ln J)^2, \tag{29}$$

where I_1 and J are calculated on the basis of the plane form of kinematic tensors. The resulting strain energy of the generic element can be then evaluated by using the following formula

$$\Psi_e(\boldsymbol{x}) = \int_{\Omega_e} \Psi_{PF}\, dV = \sum_{g=1}^{4} [\Psi_{PF}] A_g\, th\, w_g = \sum_{g=1}^{4} \Psi_g, \tag{30}$$

where $A_g = \det\left[\frac{\partial \boldsymbol{X}}{\partial \zeta}\right]_g$ is the part of the reference domain pertaining to the Gauss point, w_g is the Gauss point weight and th is the domain thickness usually assumed unitary under plane strain condition. Using Eq. (30), (5), and (7) the following results are valid for the generic element

$$\boldsymbol{g}_e = \sum_{g=1}^{4} \frac{\partial \Psi_g}{\partial \boldsymbol{x}} = \sum_{g=1}^{4} \boldsymbol{g}_g, \quad \boldsymbol{J}_e = \sum_{g=1}^{4} \frac{\partial \boldsymbol{g}_g}{\partial \boldsymbol{x}} = \sum_{g=1}^{4} \boldsymbol{J}_g, \tag{31}$$

where $\boldsymbol{g}_g$ and $\boldsymbol{J}_g$ are the gradient and Jacobian, respectively, pertaining to the generic Gauss point.

The following MATLAB® instructions, Listing 1.12, implements, inside the function Initialize of PF4 class, the operations required by Eq. (31).

Listing 1.12. PF4 class: evaluation of ge and Je.

```
 function PF = Initialize (PF,D, i)
            % ...

            PF. Je = sym (zeros (8 ,8 , PF .nG));
            PF. Fe = sym ( zeros (2 ,2 ,PF .nG));
            for g = l:PF.nG
            % ...
            C = F.'*F;
            I1 = trace (C);
            J = det (F);
            Psi = (mi/2*(Il–2)–mi*log(J) + lam/2*log (J)^2) ...
            A*PF.wG(g)*th;
            PF.ge(: , 1, g) = gradient (Psi, PF. xe);
            PF.Je(:, :, g) = jacobian (PF. ge(: , 1, g), PF. xe);
            % ...
            end
            % ...
 end
```

During the analysis the main task to be performed by the element is the numerical evaluation of $\boldsymbol{g}_e$ and $\boldsymbol{J}_e$ that now must to be performed, see Eq. (31), on the basis of the following implementation of the class function Compute.

Listing 1.13. PF4 class:function Compute.

```
 function PF = Compute(PF)
            PF. se = zeros (8, 1);
            PF. Ke = zeros (8, 8);
            for g = l:PF.nG
              PF.se = PF.se + subs (PF.ge(:, 1, g), PF.xe, PF.xxe);
              PF.Ke = PF.Ke + subs (PF.Je(:, :, g), PF.xe, PF.xxe);
            end
 end
```

The last part of the class to be discussed regards the evaluation of stress solution. As already observed in the beginning of this section, Eq. (19) constitutes an internal constraint determining the presence of also the stress component σ_{33} to be evaluated together with the plane part of the stress tensor. The plane part can be calculated using Eqs. (15) and (16) where the plane version of C and F must be used starting from the strain energy expression given by Eq. (29). The σ_{33} component, stems from the plane solution, and is given by

$$\sigma_{33} = J^{-1}S_{33} = J^{-1}\frac{\lambda}{2}\ln\left(\det\boldsymbol{C}\right) \tag{32}$$

A simple derivation of this expression through MATLAB® is reported in Appendix A. The implementation of the operations required for the evaluation of the stress solution are reported below, Listing 1.14.

Listing 1.14. PF4 class: function Stress.

```
 function sig = Stress (T, gx)
            % retrieve local vector lx from global
            % solution gx
            sig = zeros(3, 3, PF.nG);
            for g = l:PF.nG
              F = subs (PF.Fe(:, :, g), PF.xe, lx);
              C = F.'*F;
               sig(l:2, l:2, g) = F*subs (PF. Se, PF.Ce,C)*F.'/det(F);
              sig (3, 3, g) = subs (PF.Se33, PF.Ce, C)/det (F);
            end
 end
```

Listing 1.14 shows the use of the symbolic properties PF.Se which is initialised in way similar to the property T.Se shown in Listing 1.9 for the tetrahedral element. Anyway the complete listing of the class can be found in [34].

6. Conclusions

The early phase developing of finite elements can be a lengthy and error prone processes involving the use of different tools. The MATLAB® symbolic approach here presented can be effectively used to test a produce new finite element formulation reducing a lot the distance between the formulation and its actual implementation. In order to be more illustrative the presentation regarded basic solid mechanics finite elements, a truss, tetrahedral and plane quadrangular element, but the developing of finite elements for more specific engineering applications is an objective worth to be pursued and it is the subject of the author’s current work.

The weakness of the proposed approach is the low performance of the final codes making difficult the analysis of real sized problems by using common hardware resources which, however, are adequate if small but significative test cases are chosen. A workaround, already tested by the author but not presented here, is the generation and storing on files of MATLAB® functions for the evaluation of the element operators. This must happens before, and one time for all, the execution of the analysis. The MATLAB® functions so obtained can be called during the analysis for evaluating the required finite element operators avoiding the calls to time-consuming function subs. Anyway the tuning of this operation is less automatic because the generation of the required MATLAB® functions can be, depending on the size of the operator to be translated into a MATLAB® function, time consuming, specially if the optimization flag is active. Then techniques quite common in the field of the symbolic and /or algorithmic differentiation should be exploited for the most intricate cases.

A. Appendix

A.1 Tetrahedron reference configuration operator inversion

The problem of the evaluation of the inverse of matrix $\frac{\partial \boldsymbol{X}}{\partial \zeta}$ present in Eq. (11) is circumvented by evaluating the Jacobian of the following system of equations

$$\begin{aligned} &1 = \zeta_1 + \zeta_2 + \zeta_3 + \zeta_4 \\ &\boldsymbol{X}(\zeta_1, \zeta_2, \zeta_3, \zeta_4) = \zeta_1 \boldsymbol{X}_1 + \zeta_2 \boldsymbol{X}_2 + \zeta_3 \boldsymbol{X}_3 + \zeta_4 \boldsymbol{X}_4 \end{aligned} \tag{33}$$

whose linearisation gives

$$\begin{bmatrix} 0 \\ d\boldsymbol{X} \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ \boldsymbol{X}_1 & \boldsymbol{X}_2 & \boldsymbol{X}_3 & \boldsymbol{X}_4 \end{bmatrix} [d\zeta]. \tag{34}$$

By inverting this relationship, i. e.

$$[d\zeta] = \begin{bmatrix} 1 & 1 & 1 & 1 \\ \boldsymbol{X}_1 & \boldsymbol{X}_2 & \boldsymbol{X}_3 & \boldsymbol{X}_4 \end{bmatrix}^{-1} \begin{bmatrix} 0 \\ d\boldsymbol{X} \end{bmatrix} = \begin{bmatrix} - & \frac{\partial \zeta_1}{\partial X_1} & \frac{\partial \zeta_1}{\partial X_2} & \frac{\partial \zeta_1}{\partial X_3} \\ - & \frac{\partial \zeta_2}{\partial X_1} & \frac{\partial \zeta_2}{\partial X_2} & \frac{\partial \zeta_2}{\partial X_3} \\ - & \frac{\partial \zeta_3}{\partial X_1} & \frac{\partial \zeta_3}{\partial X_2} & \frac{\partial \zeta_3}{\partial X_3} \\ - & \frac{\partial \zeta_4}{\partial X_1} & \frac{\partial \zeta_4}{\partial X_2} & \frac{\partial \zeta_4}{\partial X_3} \end{bmatrix} \begin{bmatrix} 0 \\ d\boldsymbol{X} \end{bmatrix}. \tag{35}$$

the evaluation of the desired 4×3 matrix, $\left(\frac{\partial \boldsymbol{X}}{\partial \zeta}\right)^{-1}$, is obtained. Moreover the volume of the tetrahedron in its reference configuration is an additional result thanks to relationship

$$6\,V = \det \begin{bmatrix} 1 & 1 & 1 & 1 \\ \boldsymbol{X}_1 & \boldsymbol{X}_2 & \boldsymbol{X}_3 & \boldsymbol{X}_4 \end{bmatrix}. \tag{36}$$

A.2 Out-of-plane normal component for the plane strain condition

The following MATLAB® instructions allow to find an explicit expression of the S_{33} component, from which $\sigma_{33} = J^{-1}S_{33}$.

```
syms lam mi 'real'
syms C [3 3] 'real'
C(l, 3) = 0; C(3, l) = 0; C(3, 2) = 0; C(2, 3) = 0;
I1 = trace(C); I3 = det(C);
Psi = mi/2*(I1–3)–mi*log(sqrt (I3))+ ...
                              lam/2*log (sqrt (13))^2;
S33 = simplify (2*diff (Psi ,C(3, 3)));
S33 = subs(S33, C(3, 3), 1);
```

Author details

Antonio Bilotta
Department of Informatics, Modeling, Electronics and System Engineering (DIMES), University of Calabria, Rende 87036, CS, Italy

*Address all correspondence to: antonio.bilotta@unical.it

References

[1] Cassel K W. Variational Methods with Applications in Science and Engineering. Cambridge University Press; 2013.

[2] Kreyszig E. Advanced Engineering Mathematics, 9th edition. John Wiley & Sons; 2006.

[3] Gu E Y L. A Journey from Robot to Digital Human (Mathematical Principles and Applications with MATLAB Programming). Springer-Verlag Berlin Heidelberg; 2013.

[4] Dym C L. Principles of Mathematical Modeling, 2nd edition. Elsevir Academic Press; 2004.

[5] Samarskii A A, Mikhailov A P. Principles of Mathematical Modeling (Ideas, Methods, Examples). CRC Press: Taylor & Francis Group; 2018.

[6] Zienkiewicz O C, Taylor R L, Zhu J Z. The Finite Element Method: Its Basis and Fundamentals (7th edition). Butterworth-Heinemann; 2013.

[7] Bathe K-J. Finite Element Procedures. Prentice-Hall; 1996.

[8] Hughes J R. The Finite Element Method. Dover Publications, Inc.; 2000.

[9] Brezzi F, Fortin M. Mixed and Hybrid Finite Element Methods. Springer-Verlag New York; 1991.

[10] Bilotta A, Casciaro R. Assumed stress formulation of high order quadrilateral elements with an improved in-plane bending behaviour. Comput. Methods Appl. Mech. Engrg. 2002; 191/15–16:1523–1540.

[11] Bilotta A, Casciaro R. A high performance element for the analysis of 2D elastoplastic continua. Computer Methods in Applied Mechanics and Engineering 2007; 196:818–828.

[12] Bilotta A, Leonetti L, Garcea G. Three field finite elements for the elastoplastic analysis of 2D continua. Finite Elements in Analysis & Design. 2011;47:1119–1130.

[13] Bilotta A, Turco E. Elastoplastic analysis of pressure-sensitive materials by an effective three-dimensional mixed finite element. ZAMM Zeitschrift fur Angewandte Mathematik und Mechanik 2017; 97(4): 382–396, doi 10.1002/zamm.201600051

[14] Simone A. Partition of unity-based discontinuous finite elements: GFEM, PUFEM, XFEM. Revue Europeenne de Genie Civil. 2007; 11/7–8:1045–1068. DOI: 10.1080/17747120.2007.9692976

[15] Fortino S, Bilotta A. Evaluation of the amount of crack growth in 2D LEFM problems. Engineering Fracture Mechanics. 2004; 71/9–10:1403–1419. DOI: 10.1016/S0013-7944(03)00161-9

[16] Belytschko T, Krongauz Y, Organ D, Fleming M, Krysl P. Meshless methods: An overview and recent developments. Computer Methods in Applied Mechanics and Engineering. 1996; 139/1: 3–47. DOI: 10.1016/S0045-7825(96)01078-X

[17] Cockburn B, Karniadakis G E, Shu C-W, editors. Discontinuous Galerkin Methods. Springer-Verlag Berlin Heidelberg; 2000.

[18] MATLAB® main documentation resource. Available from: https:// www.mathworks.com/help/matlab/ [Accessed: 2020-10-01]

[19] List of computer algebra systems. Available from: https://en.wikipedia.org/Eist_of_computer_algebra_systems [Accessed: 2020-10-01]

[20] List of finite element software packages. Available from: https://en.

wikipedia.org/wiki/List_of_finite_ ele ment_software_packages [Accessed: 2020-10-01]

[21] Partial Differential Toolbox. Available from: https://www. math works.com/products/pde.html [Accessed: 2020-10-01]

[22] Rackauckas C, A Comparison Between Differential Equation Solver Suites In MATLAB, R, Julia, Python, C, Mathematica, Maple, and Fortran. The Winnower 6:e153459.98975 (2018). DOI: 10.15200/winn.153459.98975

[23] Symbolic Math Toolbox™main documentation resource. Available from: https://www.mathworks.com/ products/symbolic.html [Accessed: 2020-10-01]

[24] Bonet J, Gil A J, Wood R D. Mechanics for finite element analysis: Statics. Cambridge University Press; 2016.

[25] Holzapfel G A. Nonlinear solid mechanics. John Wiley and Sons; 2000.

[26] Cooper J. A MATLAB® Companion for Multivariable Calculus. Harcourt/ Academic Press; 2001.

[27] Yang W Y, Choi Y K, Kim J, Kim M C, Kim H J, Im T. Engineering Mathematics with MATLAB®. CRC Press Taylor & Francis Group; 2018.

[28] Wilson H B, Turcotte L H, Halpern D. Advanced Mathematics and Mechanics Applications Using MATLAB®.

[29] Korelc J, Wrigger P. Automation of Finite Element Methods. Springer International Publishing Switzerland; 2016.

[30] Naumann U. The Art of Differentiating Computer Programs. Society for Industrial and Applied Mathematics; 2012.

[31] Griewank A, Walther A. Evaluating Derivatives. Society for Industrial and Applied Mathematics; 2008.

[32] Complete implementation of class Truss. Available from: https:// antoniob ilotta-structuralengineer.github.io/ MyWebSite/MATLAB/Truss.m [Uploaded: 2020-10-30]

[33] Complete implementation of class Tetra4. Available from: https:// a ntoniobilotta-structuralengineer.github. io/MyWebSite/MATLAB/Tetra4.m [Uploaded: 2020-10-30]

[34] Complete implementation of class PF4. Available from: https:// antoniob ilotta-structuralengineer.github.io/ MyWebSite/MATLAB/PF4. m [Uploaded: 2020-10-30]