A finite element elasticity programming in Mathematica software

Article *in* Computer Applications in Engineering Education · October 2017

DOI: 10.1002/cae.21958

2 authors, including:

Diogo L. Cecílio
Universidade Federal do Rio Grande do Sul
5 PUBLICATIONS **8** CITATIONS

Received: 27 October 2017 | Accepted: 26 March 2018
DOI: 10.1002/cae.21958

RESEARCH ARTICLE

WILEY

A finite element elasticity programming in Mathematica software

Diogo L. Cecílio[1] | **Thiago D. dos Santos**[2]

[1] University of São Paulo, São Carlos, São Paulo, Brazil

[2] University of Campinas, Campinas, São Paulo, Brazill

Correspondence
Diogo L. Cecílio, University of São Paulo, São Carlos, São Paulo, Brazil.
Email: cecilio.diogo@gmail.com

Abstract

Modern software's allow computers to perform symbolic calculations in addition to simple numerical computations for which they were originally designed. This fact opens new possibilities in numerical simulation. In this paper a finite element implementation using the commercial software *Wolfram Mathematica* is used to solve a plane stress elasticity problem. The code adaptability from the provided examples to more challenging problems can easily be performed using the present code. Numerical examples with post-processing illustrates the new tool and its flexibility. It has been proven in this study the feasibility of *Wolfram Mathematica* in learning and programming finite element methods for the solution of two-dimensional problems in elasticity.

KEYWORDS

elasticity, finite elements, plane stress, symbolic software, *Wolfram Mathematica*

1 | INTRODUCTION

The Finite Element Method (FEM) is a well known and powerful numerical method for solving computation problems in engineering and physics, such as structural analysis (e.g., elasticity), fluid flow, and heat transfer [3,4,13]. However, a clear step-by-step program implementation in a high-level programming environment has not been reported. Low-level environments such as FORTRAN and C are not iterative languages and users have to learn how to code, compile, debug, and execute their programs. These kind of environments do not have built-in graphic interface and the computation results must be exported to other programs for graphic interpretation. Additionally, environments like FORTRAN are not coding-efficient and do not offer functions to handle matrices in a formal manner.

According to Eriksson and Pacoste [6], symbolic tools improve the efficiency and documentation in the development of procedures and allow easy comparison between different assumptions in the formulations. According to Hakula and Tuominen [7], the implementation of the FEM on a symbolic environment is useful for prototyping new algorithms and ideas and serves as a testing ground for interesting programming techniques.

Limited documentation and process details have been reported on the application of the FEM using symbolic programing environments. Alberty et al. [1] implemented the FEM on MATLAB to solve 2D and 3D problems in linear elasticity. Ioakimidis [8] solved an elasticity problem to obtain a solution in terms of a symbolic parameter. Choi and Nomura [5] presented an application of *Wolfram Mathematica* to solve a two-dimensional elasticity problems. Additionally, other works have been presented in which closed-form integration of the stiffness matrix were obtained [2,11,17–19]. Recently, closed-form of elastic constants in frame-like periodic cellular solids have been analytically derived using symbolic objetc-oriented finite element program in MATLAB [14,16]. A version of hp-FEM

algorithm is implemented by Hakula and Tuominen [7] using the software *Wolfram Mathematica* as the programming environment.

In terms of FEM code dedicated to educational environment, we can cite the works of [9,10,12,15]. Mueller [12] described an efficient approach to introducing FEM to undegratuate students using MATLAB programming. The author described and provided two-dimensional beam problem codes. Jiang and Wang [9] written in *Wolfram Mathematica* a program to help students to understand how FEM deals with plasticity. Kosasih [10] describe the advantage of programming projects to help students understand the theory of FEM in the authos's class. The author also provide an example of MATLAB code for space truss problem. Siswanto and Darmawan [15] presented an strategy of teaching structural lines elements through the open source FreeMat, a free environment similar to MATLAB.

In the present paper an elasticity problem was implemented using quadrilateral and triangular elements in the *Wolfram Mathematica* environment, as an effort to support students and researchers. Unlike complex black-box commercial softwares, this paper provides a simple and short open-box program for plane linear elasticity problems. Instead of covering all kinds of possible problems in one program, the proposed tool aims to be easy to understand and modify. Therefore, only simple models are included to be adapted to specific needs.

The structure of the paper is as follows. An introduction to the FEM formulation applied to elastic problems is presented, followed by a detailed program description. Some selected problems are solved using the developed tool.

2 | FINITE ELEMENT FORMULATION

The mechanical problem consists in finding the displacement field $\vec{u}$ that is the solution of the following problem:

$$\begin{cases} \text{div}(\boldsymbol{\sigma}) + \overrightarrow{b} = \vec{0} & \text{in } \Omega \\ \overrightarrow{u} = \vec{0} & \text{on } \Gamma_D, \\ \boldsymbol{\sigma} \cdot \overrightarrow{n} = \overrightarrow{t} & \text{on } \Gamma_N \end{cases} \tag{1}$$

where Ω is the material domain, Γ_D is boundary part of Ω in which the displacement is zero (null *Dirichlet* boundary condition), Γ_N is the boundary part of Ω in which the traction is known (*Neumann* boundary condition), $\vec{b}$ are the body force, known in Ω, and $\vec{t}$ is the traction force known at the boundary Γ_N.

2.1 | Elasticity variational form

Considering the elastic stress-strain relationship given by

$$\boldsymbol{\sigma} = \lambda \; tr\,(\varepsilon)\; \boldsymbol{I} + 2\; \mu\; \varepsilon, \tag{2}$$

and the infinitesimal strains

$$\varepsilon(\vec{u}) = \frac{1}{2}\left(\nabla\vec{u} + \nabla\vec{u}^{\mathbf{T}}\right), \tag{3}$$

the weak form of the equilibrium equation is obtained by multiplying the Equation 1 by a trial function $\vec{v} \in \mathbf{V}$, that is:

$$\int_\Omega - \text{div}(\boldsymbol{\sigma})\vec{v}\, d\omega - \int_\Omega \vec{b} \cdot\; \vec{v}\, d\omega = 0, \tag{4}$$

in which

$$\mathbf{V} = \left\{\vec{v} \in \left[H^1(\Omega)\right]^2 \text{ such that } \vec{v} = \vec{0} \text{ in } \Gamma_D\right\}, \tag{5}$$

where $\left[H^1(\Omega)\right]^2$ denotes the vectorial space of functions which is square integrable. Using the divergence theorem (integration by parts) at the Equation 4, we have:

$$\int_\Omega \boldsymbol{\sigma} : \nabla\vec{v}\, d\omega - \int_\Omega \vec{b}\cdot\vec{v}\, d\omega - \int_{\Gamma_N} \vec{t}\cdot\vec{v}\, ds = 0,\ \forall\ \vec{v} \in \mathbf{V}, \tag{6}$$

where the problem consists in find $\vec{u} \in \mathbf{V}$ with boundary conditions defined in Equation 1.

2.2 | Plane stress elasticity

To derive the discretized form of the virtual work equation it is convenient to introduce the standard matrix notation that follows. The general form of the constitutive relation is expressed as:

$$\boldsymbol{\sigma} = \mathbf{C}\; \varepsilon, \tag{7}$$

where $\mathbf{C}$ is a fourth-order stiffness tensor. Each term in Equation 7 has a form that depends upon the elasticity problem to be solved. For plane stress elasticity problems considered in this work, Equation 7 simplify drastically and can assume a vectorized form:

$$\vec{\sigma} = \; \mathbf{C}\; \vec{\varepsilon}, \tag{8}$$

where $\vec{\sigma}$ and $\vec{\varepsilon}$ are vectors and $\mathbf{C}$ is a second-order tensor also called as constitutive matrix, that is:

$$\begin{bmatrix}\sigma_{xx}\\ \sigma_{yy}\\ \sigma_{xy}\end{bmatrix} = \begin{bmatrix}\frac{E}{1-\nu^2} & \frac{\nu E}{1-\nu^2} & 0\\ \frac{\nu E}{1-\nu^2} & \frac{E}{1-\nu^2} & 0\\ 0 & 0 & \frac{E}{2(1+\nu)}\end{bmatrix}\begin{bmatrix}\varepsilon_{xx}\\ \varepsilon_{yy}\\ \varepsilon_{xy}\end{bmatrix}, \quad (9)$$

where E is the Young's modulus and v is the Poisson's ratio. The relation between the strains and displacements is written as follows:

$$\begin{bmatrix}\varepsilon_{xx}\\ \varepsilon_{yy}\\ \varepsilon_{xy}\end{bmatrix} = \begin{bmatrix}\frac{\partial}{\partial x} & 0\\ 0 & \frac{\partial}{\partial y}\\ \frac{\partial}{\partial y} & \frac{\partial}{\partial x}\end{bmatrix}\begin{bmatrix}u_x\\ u_y\end{bmatrix}, \quad (10)$$

where u_x and u_y are the in-plane displacements. In matrix notation, Equation 10 are written as:

$$\vec{\varepsilon} = \mathbf{D}\ \vec{u}. \quad (11)$$

Using matrix notation, the equations of motion for plane stress elasticity can be written as:

$$\mathbf{D^T}\vec{\sigma} + \vec{b} = \vec{0}, \quad (12)$$

where $\vec{b} = \begin{bmatrix} b_x\ b_y\end{bmatrix}^{\mathbf{T}}$ are the external forces. Equation 12 can be expressed in terms of the displacements u_x and u_y, by using Equations 8 and 11:

$$\mathbf{D^T}\ \mathbf{C}\,\mathbf{D}\ \vec{u} + \vec{b} = \vec{0}. \quad (13)$$

Finally, following the steps to deduce Equation 6, the weak form of Equation 13 is:

$$\int_\Omega (\mathbf{D}\,\vec{v})\cdot(\mathbf{C}\,\mathbf{D}\vec{u})\ d\omega - \int_\Omega \vec{b}\cdot\vec{v}\ d\omega - \int_{\Gamma_N}\vec{t}\cdot\vec{v}\ ds = 0, \quad \forall\ \vec{v} = \begin{bmatrix} v_x\, v_y\end{bmatrix}^{\mathbf{T}} \in \mathbf{V}, \quad (14)$$

2.3 | Shape functions

The most common finite element technique is based on *shape functions*, that is, functions that are used to describe the geometry and the solution for a given problem. The shape functions are built in a parametric and normalized space, or simply, *master element* ($\hat{\Omega}_e$), which boundaries are ± 1. The master element domain for quadrilateral elements is in $-1 \le \xi \le 1$ and $-1 \le \eta \le 1$. For triangular elements, the domain is $0 \le \xi \le 1$ and $0 \le \eta \le 1 - \xi$. One property of all shape functions is to have a unitary value in one node and zero in the others.

Here, we exemplify tree different types of elements: quadrilateral with four-node and nine-node and triangle with three-node. The four-node quadrilateral shape functions in master element domain $\hat{\Omega}_e$ are defined in Equation 15 and illustrated in Figure 1. Figure 2 illustrates a geometric description of a given element (distorted) using coordinate transformations of the shape functions defined in the master element.

$$\begin{aligned}\hat{\psi}_1 &= 0.25(1-\eta)(1-\xi)\\ \hat{\psi}_2 &= 0.25(1-\eta)(1+\xi)\\ \hat{\psi}_3 &= 0.25(1+\eta)(1+\xi)\\ \hat{\psi}_4 &= 0.25(1+\eta)(1-\xi)\end{aligned} \quad (15)$$

The three-node triangular elements shape functions are described by the Equation 16.

$$\begin{aligned}\hat{\psi}_1 &= 1-\xi-\eta\\ \hat{\psi}_2 &= \xi\\ \hat{\psi}_3 &= \eta\end{aligned} \quad (16)$$

Finally, the nine-node quadrilateral elements shape functions are described by the Equation 17.

$$\begin{aligned}\hat{\psi}_1 &= \xi\eta(\xi-1)(\eta-1)/4\\ \hat{\psi}_2 &= \xi\eta(\xi+1)(\eta-1)/4\\ \hat{\psi}_3 &= \xi\eta(\xi+1)(\eta+1)/4\\ \hat{\psi}_4 &= \xi\eta(\xi-1)(\eta+1)/4\\ \hat{\psi}_5 &= -\eta(\xi+1)(\xi-1)(\eta-1)/2\\ \hat{\psi}_6 &= -\xi(\xi+1)(\eta+1)(\eta-1)/2\\ \hat{\psi}_7 &= -\eta(\xi+1)(\xi-1)(\eta+1)/2\\ \hat{\psi}_8 &= -\xi(\xi-1)(\eta+1)(\eta-1)/2\\ \hat{\psi}_9 &= (\xi+1)(\xi-1)(\eta+1)(\eta-1)\end{aligned} \quad (17)$$

2.4 | Integration in the master element

Numerical integration is used extensively in finite elements analysis. Integration techniques (e.g., the trapezoidal rule) often assume equally spaced data and are somewhat limited in applicability and accuracy when used in finite element analysis. The Gaussian quadrature has been widely included in finite element software's as the numerical integration approximation. In this method, an integral is evaluated as a summation over all the integration points (ξ_i, η_i), as presented in Equation 18. The order of the Gaussian quadrature is given

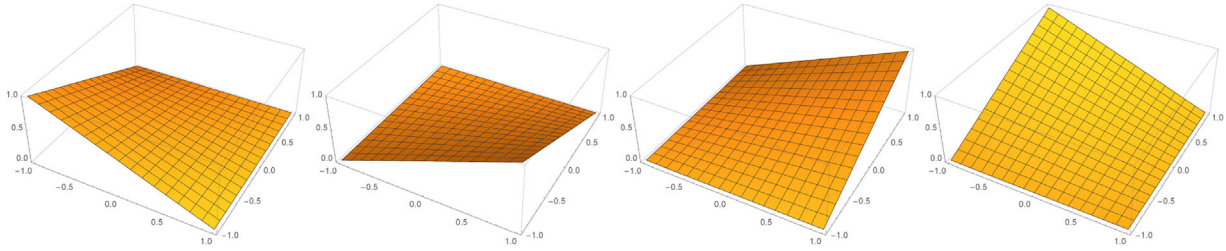


FIGURE 1 Quadrilateral four-node shape functions

by N. A Gaussian quadrature using N points in each k directions (in ξ and η, for quadrilateral elements, e.g.) can provide the exact integral if the function to be integrated is a polynomial of degree $2N-1$ or less.

$$I = \int_{-1}^{1}\int_{-1}^{1} g(\xi,\eta)\omega_i d\xi d\eta = \sum_{i=1}^{N} g(\xi_i,\eta_i)\omega_i. \tag{18}$$

Figure 2 illustrates four integration points (order $N=2$) in a quadrilateral four-node element defined in $\hat{\Omega}_e$. These four integration points allow the exact integration of the first order polynomial described in Equation 15.

2.5 | Coordinate transformation and jacobian

In the isoparametric formulation of the finite element method, the local geometry of a given element is transformed (Figure 2) in a normalized space using the same shape functions that describe the solution. The element in the (undistorted) normalized space, using ξ, η coordinates, is referred as the master element (see Figure 2). The derivatives is also evaluated in the master element and requires a formal mathematical transformation to back to the original coordinates. Describing each element geometry (and solution) in a master element space is a powerful tool to deal with problems defined in a general geometry.

In the distorted element, the geometry (i.e., local coordinates) is defined in ξ, η using coordinates transformation:

$$\begin{aligned} x(\xi,\ \eta) &= \textstyle\sum_i^n \hat{\psi}_i x_i \\ y(\xi,\ \eta) &= \textstyle\sum_i^n \hat{\psi}_i y_i \end{aligned}, \tag{19}$$

where n is the number of shape functions (or nodes). The derivatives of the shape functions can be written as follows, using the chain rule:

$$\begin{aligned} \frac{\partial\hat{\psi}_i}{\partial\xi} &= \frac{\partial\hat{\psi}_i}{\partial x}\frac{\partial x}{\partial\xi} + \frac{\partial\hat{\psi}_i}{\partial y}\frac{\partial y}{\partial\xi} \\ \frac{\partial\hat{\psi}_i}{\partial\eta} &= \frac{\partial\hat{\psi}_i}{\partial x}\frac{\partial x}{\partial\eta} + \frac{\partial\hat{\psi}_i}{\partial y}\frac{\partial y}{\partial\eta} \end{aligned}, \tag{20}$$

or, in matrix form:

$$\begin{bmatrix} \frac{\partial\hat{\psi}_i}{\partial\xi} \\ \frac{\partial\hat{\psi}_i}{\partial\eta} \end{bmatrix} = \begin{bmatrix} \frac{\partial x}{\partial\xi} & \frac{\partial y}{\partial\xi} \\ \frac{\partial x}{\partial\eta} & \frac{\partial y}{\partial\eta} \end{bmatrix} \begin{bmatrix} \frac{\partial\hat{\psi}_i}{\partial x} \\ \frac{\partial\hat{\psi}_i}{\partial y} \end{bmatrix}. \tag{21}$$

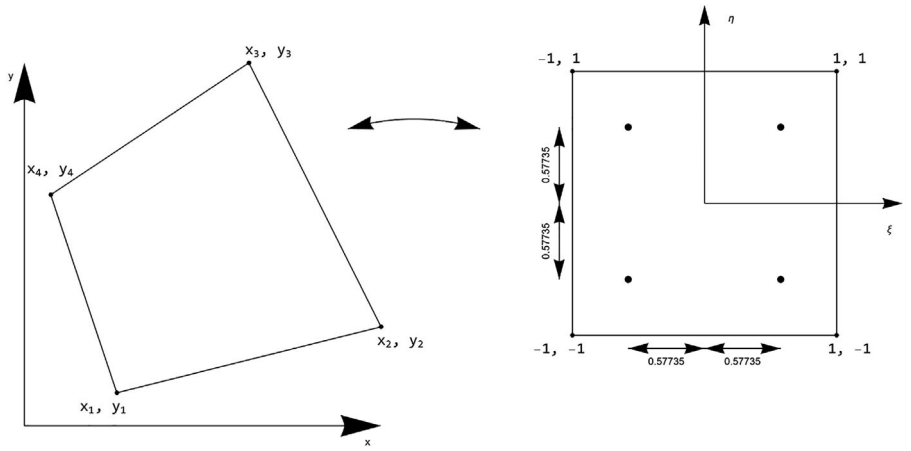


FIGURE 2 Distorted element defined in terms of ξ, η using coordinate transformation. Right side shows the four integration points for a Gaussian quadrature rule of order $N=2$

The first matrix on the right-rand side is defined as $\mathbf{J}$ and is referred as the *jacobian matrix*:

$$\mathbf{J} = \begin{bmatrix} \frac{\partial x}{\partial \xi} & \frac{\partial y}{\partial \xi} \\ \frac{\partial x}{\partial \eta} & \frac{\partial y}{\partial \eta} \end{bmatrix}. \tag{22}$$

Multiplying Equation 21 by $\mathbf{J}^{-1}$ gives the form of the shape functions derivatives in the x, y (local) coordinate system:

$$\begin{bmatrix} \frac{\partial \hat{\psi}_i}{\partial x} \\ \frac{\partial \hat{\psi}_i}{\partial y} \end{bmatrix} = \mathbf{J}^{-1} \begin{bmatrix} \frac{\partial \hat{\psi}_i}{\partial \xi} \\ \frac{\partial \hat{\psi}_i}{\partial \eta} \end{bmatrix}. \tag{23}$$

2.6 | Matrix finite element formulation

Using the definitions introduced in last sections, we can rewrite the weak formulation described by Equation 14 in a matrix form, which can be implemented and solved in computer systems. The element displacements $\vec{u} = \begin{bmatrix} u_x & u_y \end{bmatrix}^{\mathbf{T}}$ are approximated in terms of the shape functions such as:

$$\begin{matrix} u_x \approx \sum_i^n \hat{\psi}_i u_{x,i} \\ u_y \approx \sum_i^n \hat{\psi}_i u_{y,i} \end{matrix}, \tag{24}$$

where n is the number of element nodes and $u_{x,i}$ and $u_{y,i}$ are the nodal displacements of the element. The vector of nodal displacements can be defined as:

$$\vec{u}_n = \begin{bmatrix} u_{x,1} & u_{y,1} & u_{x,2} & u_{y,2} & \dots & u_{x,n} & u_{y,n} \end{bmatrix}^{\mathbf{T}}. \tag{25}$$

Using Equations 24 and 25, the vector of element displacements can be written as:

$$\vec{u} = \Psi \ \vec{u}_n, \tag{26}$$

where Ψ is a $2x2n$ matrix of shape functions, defined as:

$$\Psi = \begin{bmatrix} \hat{\psi}_1 & 0 & \hat{\psi}_2 & 0 & \dots & \hat{\psi}_n & 0 \\ 0 & \hat{\psi}_1 & 0 & \hat{\psi}_2 & \dots & 0 & \hat{\psi}_n \end{bmatrix}. \tag{27}$$

Similarly, the vector $\vec{v}$ is also written in terms of shape functions:

$$\vec{v} = \Psi \ \vec{v}_n, \tag{28}$$

where $\vec{v}_n$ are arbitrary nodal displacements. Applying the differential operator $\mathbf{D}$ in the displacements $\vec{u}$, we obtain:

$$\mathbf{D}\vec{u} = \mathbf{D}\Psi\vec{u}_n = \mathbf{B}\vec{u}_n, \tag{29}$$

where $\mathbf{B}$ is a $3x2n$ matrix:

$$\mathbf{D}\ \Psi = \mathbf{B} = \begin{bmatrix} \frac{\partial \hat{\psi}_1}{\partial x} & 0 & \frac{\partial \hat{\psi}_2}{\partial x} & 0 & \dots & \frac{\partial \hat{\psi}_n}{\partial x} & 0 \\ 0 & \frac{\partial \hat{\psi}_1}{\partial y} & 0 & \frac{\partial \hat{\psi}_2}{\partial y} & \dots & 0 & \frac{\partial \hat{\psi}_n}{\partial y} \\ \frac{\partial \hat{\psi}_1}{\partial y} & \frac{\partial \hat{\psi}_1}{\partial x} & \frac{\partial \hat{\psi}_2}{\partial y} & \frac{\partial \hat{\psi}_2}{\partial x} & \dots & \frac{\partial \hat{\psi}_n}{\partial y} & \frac{\partial \hat{\psi}_n}{\partial x} \end{bmatrix}. \tag{30}$$

The integration of Equation 14 is realized on the volume and surface contour of the material domain Ω. Dividing the domain Ω in elements such that $\Omega = \sum_e \Omega_e$, the volume V_e and the surface contour S_e of a given element Ω_e is:

$$\begin{matrix} V_e = h_e A_e \\ S_e = h_e s_e \end{matrix}, \tag{31}$$

where h_e is the element thickness, A_e is the xy-plane area (i.e., $dA = dxdy$) and s_e is the unitary element contour. Replacing Equations 26 and 28 in the plane stress weak formulation, and using the relations described by Equations 29 and 31, we can rewrite the Equation 14 as:

$$\sum_e \int_{\Omega_e} (\mathbf{B}\vec{v}_n) \cdot (\mathbf{C}\mathbf{B}\vec{u}_n)\ h_e\ dA_e - \sum_e \int_{\Omega_e} \vec{b} \cdot (\Psi\vec{v}_n)\ h_e\ dA_e - \sum_e \int_{\Gamma_{e,N}} \vec{t} \cdot (\Psi\vec{v}_n)\ h_e\ ds_e = \vec{0}. \tag{32}$$

The nodal vector $\vec{v}_n$ is arbitrary and it can be dropped in Equation 32. We can rewrite Equation 32 as:

$$\sum_e \int_{\Omega_e} \mathbf{B}^{\mathbf{T}}\ \mathbf{C}\ \mathbf{B}\vec{u}_n h_e\ dA_e - \sum_e \int_{\Omega_e} \Psi^{\mathbf{T}}\ \vec{b}\ h_e\ dA_e - \sum_e \int_{\Gamma_{e,N}} \Psi^{\mathbf{T}}\ \vec{t}\ h_e\ ds_e = \vec{0}, \tag{33}$$

or, in a condensed matrix form:

$$\mathbf{K}\ \mathbf{u} = \mathbf{F}, \tag{34}$$

where $\mathbf{K}$ is the global stiffness matrix, $\mathbf{u}$ is the unknown nodal displacements of all elements and $\mathbf{F}$ is the external global load vector. The element stiffness matrix and the element load vector are defined, respectively, as:

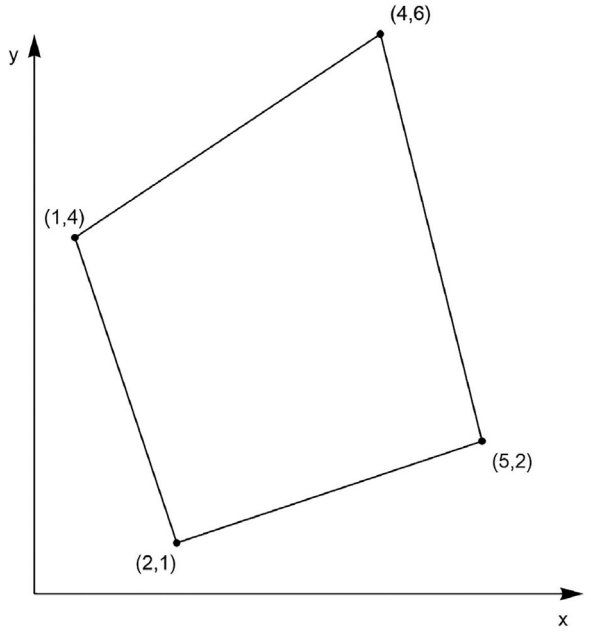


FIGURE 3 Linear rectangular isoparametric element mesh

$$\mathbf{K}_e = \int_{\Omega_e} \mathbf{B}^{\mathbf{T}}\mathbf{C}\mathbf{B} h_e dA_e, \tag{35}$$

$$\mathbf{F}_e = \int_{\Omega_e} \Psi^{\mathbf{T}}\, \vec{b}\, h_e\, dA_e + \int_{\Gamma_{e,N}} \Psi^{\mathbf{T}}\, \vec{t} h_e\, ds_e. \tag{36}$$

The sum in the Equation 33 represents the assembly or contribution of each element stiffness matrix and load vector in the global stiffness matrix and load vector, respectively. The integrations of Equations 35 and 36 are performed by Gaussian quadrature (e.g., Equation 18) in the master element using coordinate transformation.

3 | PROGRAM DESCRIPTION

A description of the code is presented in this section. The structure of the code is modular and based on functions which are added in the main function, *Assemble*. This makes the code cleaner and allows changes or improvements. The description here includes the following features (functions): implementation of the shape functions, numerical integration using a Gaussian quadrature, contributions of each integration point to calculate the element stiffness matrix, assembly of the global stiffness matrix and the definition of boundary conditions. The codes of all of these features are transcript in appendix. Additionally, also in appendix are the codes of geometric mesh generation and post processing. Once the purpose of this work is the FEM programming, we do not write a linear solver. Instead of and to keep the code clean and simple, we use the default linear solver package provided by *Wolfrom Mathematica*.

3.1 | Compute 2D shape

This function computes the rectangular and the triangular shape functions. The function has as input the polynomial order of the shape functions (e.g., 1 or 2) and the type of elements, being 1 for quadrilateral and 2 for triangular elements; the output consists of the shape functions and their derivatives. The code is illustrated in appendix.

3.2 | Integration rule

The function determines the Gaussian quadrature points and corresponding weights. This function has as input the polynomial order (e.g., 1 or 2) and the type of elements that needs to be integrated quadrilateral or triangular elements. The outputs are the quadrature points and its respective weights. The code is illustrated in appendix.

3.3 | Contribute

This function computes the contribution of each integration point to the element stiffness matrix and element load vector. The formulation implemented here is the plane stress elasticity in the matrix form, Equations 35 and 36. The term $\mathbf{B}_i^{\mathbf{T}}\mathbf{C}\ \mathbf{B}_i$ of Equation 35 is calculated for each $1 \le i \le N$, were N is the number of integration points. The coordinate transformations of $x(\xi, \eta)$, $y(\xi, \eta)$:

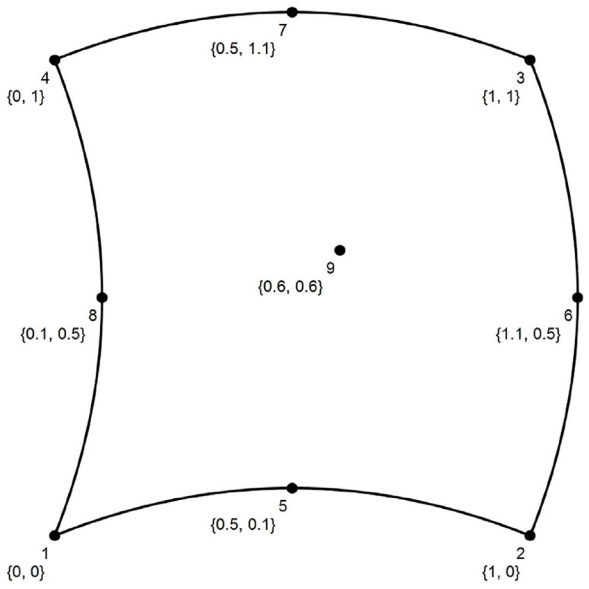


FIGURE 4 Quadratic nine-node rectangular isoparametric element mesh

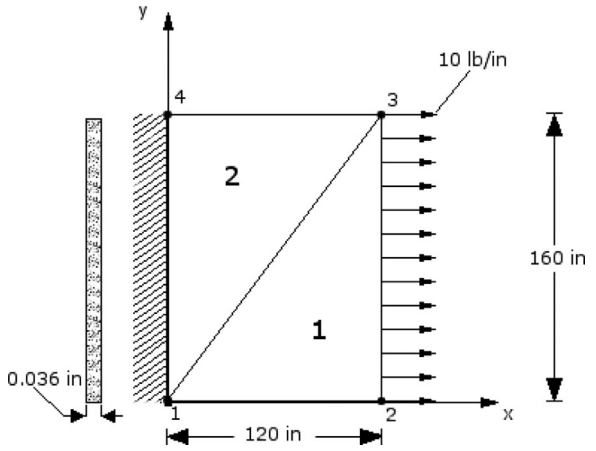


FIGURE 5 Finite element mesh and geometry of the plane elasticity problem solved in example 3

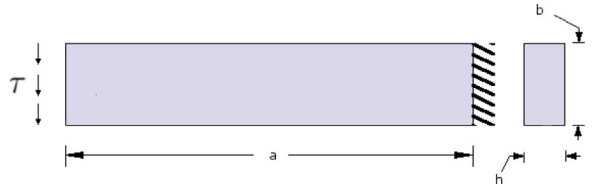


FIGURE 6 Cantilever beam adopted in example 4

$$x(\xi,\eta) = \sum_i^n \hat{\psi}_i x_i$$
$$y(\xi,\eta) = \sum_i^n \hat{\psi}_i y_i,$$

and the *jacobian*:

$$\mathbf{J} = \begin{bmatrix} \frac{\partial x}{\partial \xi} & \frac{\partial y}{\partial \xi} \\ \frac{\partial x}{\partial \eta} & \frac{\partial y}{\partial \eta} \end{bmatrix},$$

are also performed in this function, once the integration is performed in the master element. The code is illustrated in appendix. Here the advantages in using *Wolfram Mathematica* are clear. The matrix weak formulation can be easily implemented using the built-in matrix product, and another functions such as matrix inversion and determinant calculations can also be used, making the programming task focused in solving the problem and not in implementing other functions.

3.4 | CalcStiff

To compute the element stiffness matrix, a loop over all integration points of the element must be conducted. The loop over all integration points is performed by *CalcStiff* function, as illustrated in appendix. For each integration point the function *Contribute* is called. The sum of all integration points contributions is conducted to obtain the final element stiffness matrix (from Equation 35):

$$\int_{\Omega_e} \mathbf{B}^{\mathbf{T}}\ \mathbf{C}\ \mathbf{B}\ h_e\ dA_e = \sum_i^N \left(\mathbf{B}_i^{\mathbf{T}}\ \mathbf{C}\ \mathbf{B}_i\right) h_e\ w_i\ \det\ \mathbf{J}_i. \quad (37)$$

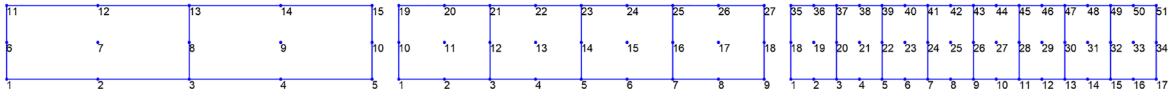
3.5 | Assemble

This is the main function, where the global stiffness matrix **K** and the global load vector **F** are assembled. Here a loop over all the elements of the mesh is conducted to calculate the each element stiffness matrix and load vector, by the function *CalcStiff*. Basically, the inputs of this function are the information of the mesh, type of element and the polynomial order. The code is illustrated appendix.

3.6 | Apply Node BC Displacement and Apply Node BC Force

Once the global stiffness matrix and global load vetor are assembled, the last step is the boundary condition (BC) definition. Imposing the boundary conditions is an important step in solving finite element problems. There are two types of

(a) 4×2 four-node linear elements mesh. (b) 8×2 four-node linear elements mesh. (c) 16×2 four-node linear elements mesh.



(d) 2×1 nine-node quadratic elements mesh. (e) 4×1 nine-node quadratic elements mesh. (f) 8×1 nine-node quadratic elements mesh.

FIGURE 7 Finite element meshes for an end-loaded cantilever beam

BC's: *Dirichlet* and *Neumann*. In plane elasticity when the boundary displacements are known, the BC is named *Dirichlet*; when the boundary forces are known, it is called *Neumann*.

To impose null displacements, the stiffness matrix position of interest is multiplied by a comparatively big number (e.g., 10^{12}). This procedure is performed by the function *ApplyNodeBCDisplacement*. To specify the boundary forces the values are directly placed in the force vector, which is done by the *ApplyNodeBCForce* function. The functions that impose the BC's are illustrated in the appendix.

TABLE 1 Comparison of the finite element solution with elasticity solution for a cantilever beam subjected to a uniform shear load at the free end

Tip deflection, $-u_y \times 10^-2$					
	Element type				
	Four-node linear		Nine-node quadratic		
Mesh number of nodes	Present study	Reddy [13]	Present study	Reddy [13]	Analytical
15	0.3104	0.3134	0.4973	0.5031	0.5188
27	0.4350	0.4388	0.5086	0.5129	
51	0.4840	0.4878	0.5104	0.5137	

4 | EXAMPLES

Five application examples are discussed in this section. Example 4.1 derives the plane stress elasticity stiffness matrix using the linear isoparametric formulation for a quadrilateral element of four nodes, as described by Buchanan [4]. Example 4.2 derives the plane stress elasticity stiffness matrix using a quadratic element of nine nodes. Example 4.3 computes the nodal displacements in a plate discretized by two linear triangular elements as detailed in Reddy [13]. Example 4.4 computes a tip deflection in an end-loaded beam for comparative discussion with previous works [13]. Finally, in example 4.5, a plate with a pressurized hole is solved using quadratic nine-node elements. The post-processed results are shown as a color map plots. These results were compared to those obtained from commercial finite element software *ANSYS*.

The code of all examples are detailed in the appendix. If the reader is interested in reproduce the results, the code is also available to download at the link https://github.com/diogocecilio/FEM.[1]

[1]The code was written the *Wolfram Mathematica* 11.0.1.0 version.

4.1 | Example 1

In this example the elasticity stiffness matrix for the simple linear four-node rectangular element illustrated in Figure 3 is evaluated. This example can be found in Buchanan [4] (chapter 6 page 184 example 6.11). Unit thickness, a Young's modulus $E = 1.0$, and a Poisson's coefficient $\nu = 0.25$ are assumed. The mesh of this example is illustrated in Figure 3. A $2x2$ Gaussian quadrature is used. For each Gauss point the function *Contribute* is applied and the matrix form $\mathbf{B^TCB}$ is computed. The final element stiffness matrix (38) is the sum of the four contributions:

$$\boldsymbol{K} = \begin{bmatrix} 0.407005 & 0.104972 & -0.300742 & -0.139621 & -0.0902343 & -0.115496 & -0.0160288 & 0.150145 \\ 0.104972 & 0.545515 & -0.0729543 & -0.0286192 & -0.115496 & -0.277013 & 0.0834783 & -0.239883 \\ -0.300742 & -0.0729543 & 0.702964 & -0.142679 & 0.0518507 & 0.0812231 & -0.454073 & 0.13441 \\ -0.139621 & -0.0286192 & -0.142679 & 0.393915 & 0.14789 & -0.182347 & 0.13441 & -0.182949 \\ -0.0902343 & -0.115496 & 0.0518507 & 0.14789 & 0.316499 & 0.107979 & -0.278116 & -0.140373 \\ -0.115496 & -0.277013 & 0.0812231 & -0.182347 & 0.107979 & 0.4688 & -0.073706 & -0.00944048 \\ -0.0160288 & 0.0834783 & -0.454073 & 0.13441 & -0.278116 & -0.073706 & 0.748217 & -0.144182 \\ 0.150145 & -0.239883 & 0.13441 & -0.182949 & -0.140373 & -0.00944048 & -0.144182 & 0.432273 \end{bmatrix}. \tag{38}$$

4.2 | Example 2

The elasticity stiffness matrix in a rectangular isoparametric element is evaluated. Unit thickness, Young's modulus E of 1.0, and a Poisson's coefficient v of 0.25 are assumed. The mesh of this example is illustrated in Figure 4. A $3x3$ Gaussian quadrature is used. For each gauss point the routine *Contribute* is called and the matrix form B^TCB is computed, and the final stiffness matrix (39) is the sum of the nine contributions.

$$
\boldsymbol{K} = \begin{bmatrix}
0.30 & 0.064 & -0.0041 & 0.0058 & -0.018 & -0.019 & -0.071 & -0.016 & -0.30 & -0.019 & 0.048 & 0.067 & 0.062 & 0.067 & 0.015 & 0.070 & -0.034 & -0.22 \\
0.064 & 0.30 & -0.016 & -0.071 & -0.019 & -0.018 & 0.0058 & -0.0041 & 0.070 & 0.015 & 0.067 & 0.062 & 0.067 & 0.048 & -0.019 & -0.30 & -0.22 & -0.034 \\
-0.0041 & -0.016 & 0.38 & -0.14 & -0.057 & 0.022 & -0.018 & 0.019 & -0.28 & 0.15 & -0.022 & -0.16 & 0.060 & -0.068 & 0.093 & -0.083 & -0.15 & 0.28 \\
0.0058 & -0.071 & -0.14 & 0.56 & -0.00055 & 0.026 & 0.019 & -0.018 & 0.058 & -0.0018 & -0.066 & -0.32 & -0.068 & 0.047 & -0.083 & 0.16 & 0.28 & -0.38 \\
-0.018 & -0.019 & -0.057 & -0.00055 & 0.78 & 0.33 & 0.026 & 0.022 & 0.16 & 0.082 & -0.019 & 0.0055 & -0.37 & -0.083 & 0.092 & 0.082 & -0.59 & -0.42 \\
-0.019 & -0.018 & 0.022 & 0.026 & 0.33 & 0.78 & -0.00055 & -0.057 & 0.082 & 0.092 & -0.083 & -0.37 & 0.0055 & -0.019 & 0.082 & 0.16 & -0.42 & -0.59 \\
-0.071 & 0.0058 & -0.018 & 0.019 & 0.026 & -0.00055 & 0.56 & -0.14 & 0.16 & -0.083 & 0.047 & -0.068 & -0.32 & -0.066 & -0.0018 & 0.058 & -0.38 & 0.28 \\
-0.016 & -0.0041 & 0.019 & -0.018 & 0.022 & -0.057 & -0.14 & 0.38 & -0.083 & 0.093 & -0.068 & 0.060 & -0.16 & -0.022 & 0.15 & -0.28 & 0.28 & -0.15 \\
-0.30 & 0.070 & -0.28 & 0.058 & 0.16 & 0.082 & 0.16 & -0.083 & 1.6 & -0.19 & -0.38 & -0.27 & -0.12 & -0.0038 & -0.59 & 0.42 & -0.20 & -0.085 \\
-0.019 & 0.015 & 0.15 & -0.0018 & 0.082 & 0.092 & -0.083 & 0.093 & -0.19 & 1.8 & -0.27 & -0.15 & -0.0038 & 0.10 & 0.42 & -0.59 & -0.085 & -1.4 \\
0.048 & 0.067 & -0.022 & -0.066 & -0.019 & -0.083 & 0.047 & -0.068 & -0.38 & -0.27 & 1.6 & 0.10 & -0.030 & 0.22 & 0.10 & -0.0038 & -1.3 & 0.10 \\
0.067 & 0.062 & -0.16 & -0.32 & 0.0055 & -0.37 & -0.068 & 0.060 & -0.27 & -0.15 & 0.10 & 1.1 & 0.22 & -0.030 & -0.0038 & -0.12 & 0.10 & -0.19 \\
0.062 & 0.067 & 0.060 & -0.068 & -0.37 & 0.0055 & -0.32 & -0.16 & -0.12 & -0.0038 & -0.030 & 0.22 & 1.1 & 0.10 & -0.15 & -0.27 & -0.19 & 0.10 \\
0.067 & 0.048 & -0.068 & 0.047 & -0.083 & -0.019 & -0.066 & -0.022 & -0.0038 & 0.10 & 0.22 & -0.030 & 0.10 & 1.6 & -0.27 & -0.38 & 0.10 & -1.3 \\
0.015 & -0.019 & 0.093 & -0.083 & 0.092 & 0.082 & -0.0018 & 0.15 & -0.59 & 0.42 & 0.10 & -0.0038 & -0.15 & -0.27 & 1.8 & -0.19 & -1.4 & -0.085 \\
0.070 & -0.30 & -0.083 & 0.16 & 0.082 & 0.16 & 0.058 & -0.28 & 0.42 & -0.59 & -0.0038 & -0.12 & -0.27 & -0.38 & -0.19 & 1.6 & -0.085 & -0.20 \\
-0.034 & -0.22 & -0.15 & 0.28 & -0.59 & -0.42 & -0.38 & 0.28 & -0.20 & -0.085 & -1.3 & 0.10 & -0.19 & 0.10 & -1.4 & -0.085 & 4.2 & 0.061 \\
-0.22 & -0.034 & 0.28 & -0.38 & -0.42 & -0.59 & 0.28 & -0.15 & -0.085 & -1.4 & 0.10 & -0.19 & 0.10 & -1.3 & -0.085 & -0.20 & 0.061 & 4.2
\end{bmatrix}. \tag{39}
$$

4.3 | Example 3

Consider a thin elastic plate subjected to uniformly distributed edge load, as shown in Figure 5. The problem consists in finding the plate nodal displacements. The example is adopted from Reddy [13] (chapter 11 page 622). The plate thickness, height, and length are $h = 0.036$ in, $b = 160$ in, $a = 120$ in, respectively, as shown in Figure 5. The material properties are $E = 3 \times 10^7$ psi and $\nu = 0.25$.

To impose the null displacements due to the problem constraints in nodes 1 and 2, the global stiffness matrix diagonal is multiplied by a big number (10^{12}) in these positions. Another way to do this is to put "one" in the main diagonal and zero in the associate rows and columns. The final global system of equations after the application of the boundary conditions is detailed in (40).

$$
10^5 \begin{bmatrix}
9.3 \times 10^{12} & 0 & -7.68 & 1.44 & 0 & -3.60 & -1.62 & 2.16 \\
0 & 7.2 \times 10^{12} & 2.16 & -2.88 & -3.60 & 0 & 1.44 & -4.32 \\
-7.68 & 2.16 & 9.30 & -3.60 & -1.62 & 1.44 & 0 & 0 \\
1.44 & -2.88 & -3.60 & 7.20 & 2.16 & -4.32 & 0 & 0 \\
0 & -3.60 & -1.62 & 2.16 & 9.30 & 0 & -7.68 & 1.44 \\
-3.60 & 0 & 1.44 & -4.32 & 0 & 7.20 & 2.16 & -2.88 \\
-1.62 & 1.44 & 0 & 0 & -7.68 & 2.16 & 9.3 \times 10^{12} & -3.60 \\
2.16 & -4.32 & 0 & 0 & 1.44 & -2.88 & -3.60 & 7.2 \times 10^{12}
\end{bmatrix}
\begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \\ u_5 \\ u_6 \\ u_7 \\ u_8 \end{bmatrix}
=
\begin{bmatrix} 0 \\ 0 \\ 800 \\ 0 \\ 800 \\ 0 \\ 0 \\ 0 \end{bmatrix}. \tag{40}
$$

and the final displacement vector represented in (41).

$$
\begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \\ u_5 \\ u_6 \\ u_7 \\ u_8 \end{bmatrix}
= 10^{-4}
\begin{bmatrix} 0 \\ 0 \\ 11.3 \\ 2.0 \\ 10.1 \\ -1.08 \\ 0 \\ 0 \end{bmatrix} \text{in.} \tag{41}
$$

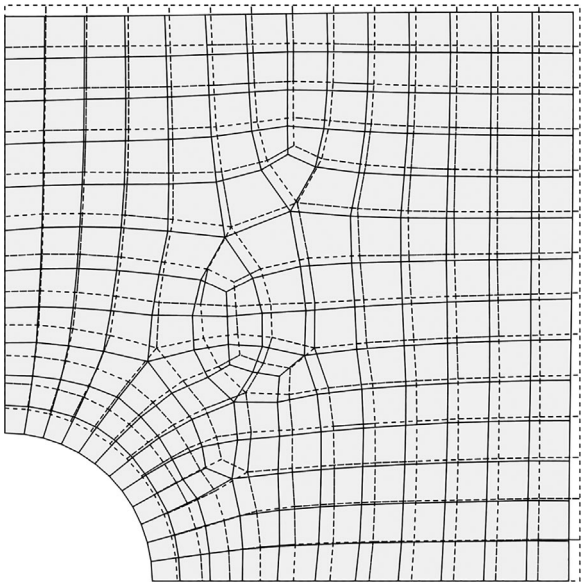
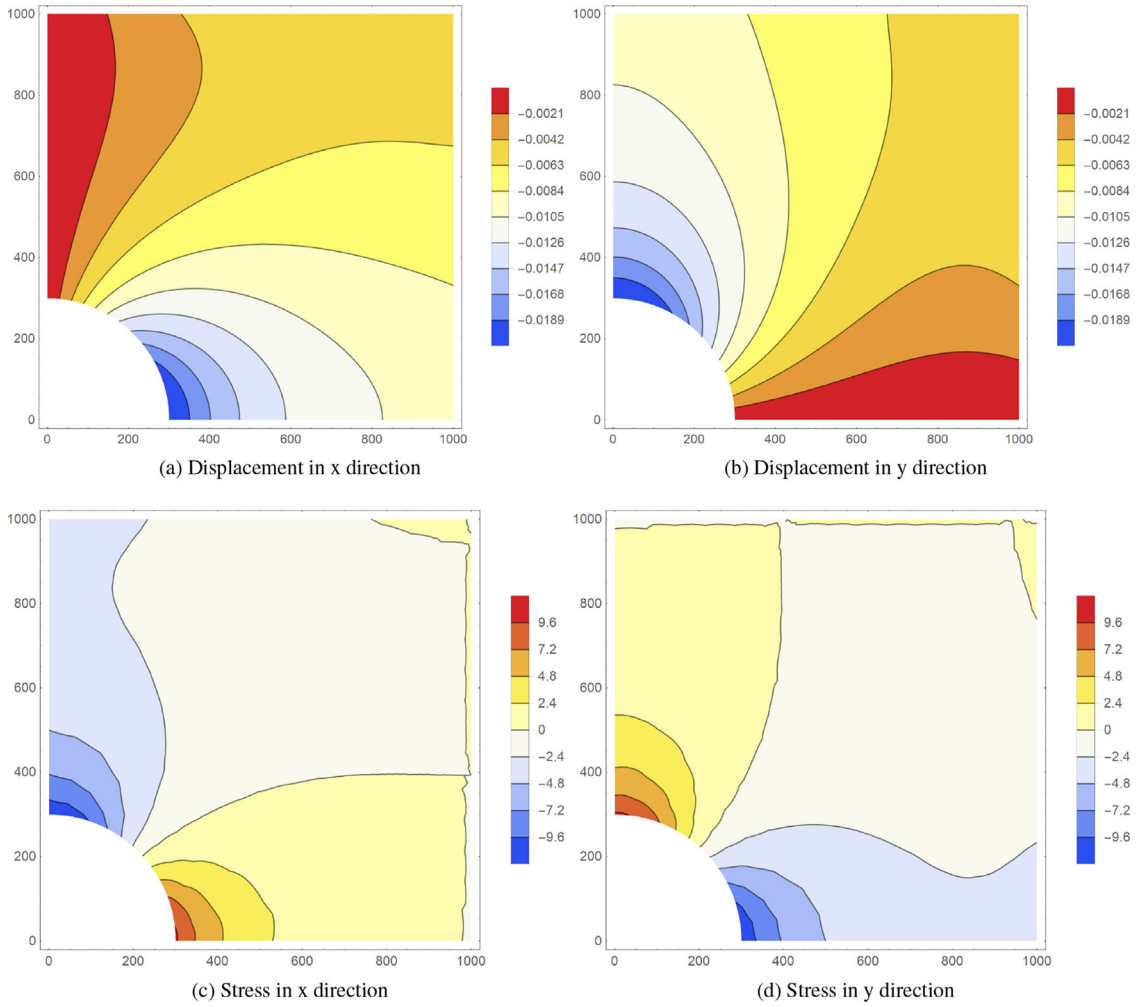


FIGURE 8 Deformed and undeformed mesh illustrated together

4.4 | Example 4

As an application exercise, the solution obtained by [13] (chapter 11 page 625) is reproduced. Consider the cantilever beam shown in Figure 6 ($E = 30 \times 10^6$ psi, $v = 0.25$, $a = 10$ in, $b = 2$ in, $h = 1$ n). The problem consists in finding the tip deflection when the beam is subjected to a uniform distributed shear stress $\tau = 150$ psi. To solve the problem and see the effect of refining the mesh on the final solution, six different meshes are considered (see Figure 7). A coarse 15 node mesh, a more refined 27 node mesh, a 51 nodes mesh are used. Tree of these meshes are made of linear four-node elements and another tree are made of quadratic nine-node elements.

The obtained results are very close to the analytical elastic solution described in [13]. It is clear from the results detailed in Table 1, that the 15 nodes mesh with four-node linear elements has a very low accuracy when compared with the nine-node quadratic elements mesh. The nine-node elements produces a good approximation even when the mesh is coarse.



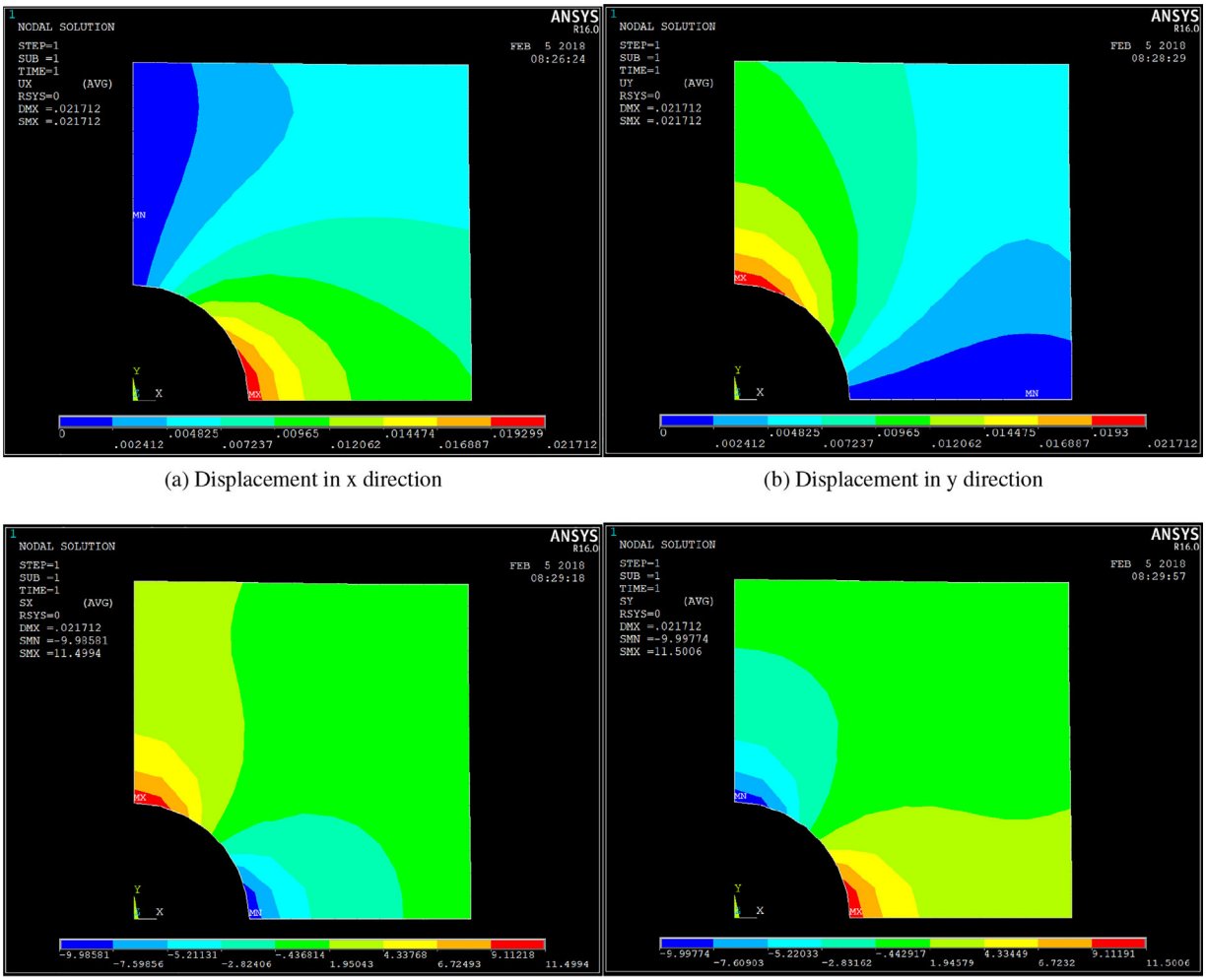
(a) Displacement in x direction

(b) Displacement in y direction

(c) Stress in x direction

(d) Stress in y direction

FIGURE 9 Displacements and stresses post-process visualized as a color map computed with *Wolfram Mathematica*



(a) Displacement in x direction

(b) Displacement in y direction

(c) Stress in x direction

(d) Stress in y direction

FIGURE 10 Displacements and stresses post-process visualized as a color map computed with *ANSYS*

4.5 | Example 5

In this example, a steel plate with a hole under plane stress conditions and unitary thickness is simulated. Due the symmetry (respect to x and to y) only a quarter of the plate were modeled. The hole ($r = 300$ mm) is located in the center of the plate. The length of each side of the plate quarter is 1000 mm. The *Dirichlet* boundary conditions consists in restricting the displacement vertically in the bottom line and the horizontally in the left line. The *Neumann* boundary condition consists in a pressure of 10 N/mm at the hole boundary.

The mesh is automatically generated by the pre processor *GID*.[2] A total of 202 nine nodes quadrilateral elements composes the final mesh. The Young's modulus and the Poisson's coefficient have the values $E = 205000$ N/mm^2 and $v = 0.30$, respectively.

The final assembled global matrix has 1,742 degrees of freedom. To simplify we opt to not show the code used to generate this example. The code will be available at www.github.com for the interested reader. After solving the problem, the deformed mesh can be obtained. The deformed and undeformed mesh are illustrated together in Figure 8.

Wolfram Mathematica has a powerfull postprocess toolkit. The displacements and stresses can be visualized as a color map using the *Wolfram Mathematica* native function *ListContourPlot*. This is ilustrated in Figure 9. For comparison purposes the same problem was simulated using the commercial software *ANSYS*, and the solution can be seen in Figure 10. Observing the results it is conclusive that the solutions corresponds.

[2]More informatio about *GID* can be found in the homepage: https://www.gidhome.com

5 | CONCLUSION

The focus of this work is to help and estimulate students in programming finite element method. To illustrate a code programming, a plane stress elasticity problem is used. The presentation of the weak formulation is performed before computational details. Important computational features of the finite element method as shape functions, numerical integration, and coordinate transformations are presented and used to generate the matrix form to the plane stress elasticity. The implementations of all the features and the matrix form using the commercial software *Wolfram Mathematica* are also presented and detailed. The modular structure makes the code clean and simple, and allows future changes. The code adaptability from the provided examples to more challenging problems are also shown. Numerical examples with post-processing have been shown. It ha1s been proven in this study the feasibility of *Wolfram Mathematica* in programming finite element methods for the solution of problems in elasticity. The shape function, its derivatives, Jacobian and the strain-displacement matrix *B* for each element are computed symbolically and stored in closed form. The program has a good readability and makes it self-evident. The program may be easily generalized. The semi-symbolic finite element programming is a good compromise between the computational efficiency and human efforts in developing FEM programs. This paper shows that symbolic algebra systems such as *Wolfram Mathematica* can be a useful and powerful tool in learning and programming the finite element method.

ORCID

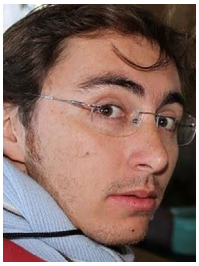
Diogo L. Cecílio http://orcid.org/0000-0001-7376-5707

REFERENCES

1. J. Alberty, C. Carstensen, and S. Funken, *Matlab implementation of the finite element method in elasticity*, Computing **690** (2018), 239–263. https://doi.org/10.1007/s00607-002-1459-8
2. G. Alotta, G. Failla, and M. Zingales, *Finite element method for a nonlocal timoshenko beam model*, Finite Elem. Anal. Des. **89** (2014) 77–92. https://doi.org/10.1016/j.finel.2014.05.011
3. E. B. Becker, G. F. Carey, and J. T. Oden, *Finite elements: An introduction*, vol I. Prentice-Hall, Englewood Cliffs, NJ, 1981.
4. G. R. Buchanan, *Schaunm's outline of theory and problems of finite element analysis*, MaCGraw-Hill, Cookeville, TN, 1995.
5. D. K. Choi and S. Nomura, *Application of symbolic computation to two-dimensional elasticity*, Comput. Struct. **430** (1992), 645–649. https://doi.org/10.1016/0045-7949(92)90505-T
6. A. Eriksson and C. Pacoste, *Symbolic software tools in the development of finite elements*, Comput. Struct. **720**, (1999): 579–593. https://doi.org/10.1016/S0045-7949(98)00333-2
7. H. Hakula and T. Tuominen, *Mathematica implementation of the high order finite element method applied to eigenproblems*, Computing **950** (2013), 277–301. https://doi.org/10.1007/s00607-012-0262-4
8. N. I. Ioakimidis, *Elementary applications of mathematica to the solution of elasticity problems by the finite element method*, Comput. Methods Appl. Mech. Eng. **1020** (1993), 29–40. https://doi.org/10.1016/0045-7825(93)90139-O
9. Y. Jiang and C. Wang, *On teaching finite element method in plasticity with mathematica*, Comput. Appl. Eng. Educ. **160** (2008), 233–242. https://doi.org/10.1002/cae.20135.
10. P. B. Kosasih, *Learning finite element methods by building applications*, Int. J. Mech. Engng. Educ. **380** (2010), 167–184. https://doi.org/10.7227/IJMEE.38.2.7
11. C. K. Lee and R. E. Hobbs, *Closed form stiffness matrix solutions for some commonly used hybrid finite elements*, Comput. Struct. **670** (1998), 463–482. https://doi.org/10.1016/S0045-7949(98)00055-8
12. D. W. Mueller, *An introduction to the finite element method using matlab*, Int. J. Mech. Engng. Educ. **330** (2005), 260–277. https://doi.org/10.7227/IJMEE.33.3.8
13. J. N. Reddy, *An introduction to the finite element method*, 3rd ed., MaCGraw-Hill, 2006. ISBN 007-124473-5
14. P. Sam et al., *Closed-form effective elastic constants of frame-like periodic cellular solids by a symbolic object-oriented finite element program*, Int. J. Mech. Mater. Des. **130** (2017), 363–383. https://doi.org/10.1007/s10999-016-9342-5
15. W. A. Siswanto and A. S. Darmawan, *Teaching finite element method of structural line elements assisted by open source freemat*, Res. J. Appl. Sci. Eng. Technol. **40** (2012), 1277–1286.
16. K. Theerakittayakorn et al., *Exact forms of effective elastic properties of frame-like periodic cellular solids*, Arch. Appl. Mech. **860** (2016), 1465–1482. https://doi.org/10.1007/s00419-016-1129-5
17. L. Videla, M. Cerrolaza, and N. Aparicio, *Explicit integration of the stiffness matrix of a four-noded-plane-elasticity finite element*, Commun. Numer. Methods Eng. **120** (1996), 731–743. https://doi.org/10.1002/(SICI)1099-0887(199611)12:11i731::AID-CNM9¿3.0.CO;2-K
18. L. Videla et al., *Exact integration of the stiffness matrix of an 8-node plane elastic finite element by symbolic computation*, Numer. Methods Partial. Differ. Equ. **240** (2008), 240–261. https://doi.org/10.1002/num.20274
19. C. K. Yew, J. T. Bole, and D. MacKenzie, *Closed form integration of element stiffness matrices using a computer algebra system*, Comput. Struct. **560** (1995), 529–539. https://doi.org/10.1016/0045-7949(94)00549-I

D. L. CECÍLIO is a civil engineer with master in Structural Engineering from UNICAMP in 2011 and PhD at Mechanical Engineering from UNICAMP in 2014. Currently, he is postdoctoral researcher at EESC-USP. Has experience in: numerical simulation, constitutive

modeling, elastoplasticity, finite deformations, and structural reliability.



T. D. DOS SANTOS is a PhD candidate at University of Campinas—UNICAMP, Brazil. He received his degree in Civil Engineering in 2007 and his master of Civil Engineering from UNICAMP in 2012. He has been working with numerical simulation for more than 10 years. His current research activity is adaptive mesh refinement applied to ice sheet simulation.

How to cite this article: Cecílio DL, dos Santos TD. A finite element elasticity programming in Mathematica software. *Comput Appl Eng Educ*. 2018;1–18. https://doi.org/10.1002/cae.21958

APPENDIX

Wolfram Mathematica code

```
(*Impose Boundary Forces*)
ApplyNodeBCForce[Fglob_,nodes_,efs_]:=Block[{position,i,j,fglob=Fglob,node,k,ef},
For[k=1,k≤Length[nodes],k++,node=nodes[[k]];
ef=efs[[k]];
position=node 2-1;
For[i=1,i≤2,i++,fglob[[position+i-1]]+=ef[[i]];];];
fglob
]
(*Impose Null Displacements*)
ApplyNodeBCDisplacement[Kglob_,nodes_,eks_]:=Block[{position,i,j,kglob=Kglob,ek,node,k,ef},
For[k=1,k≤Length[nodes],k++,node=nodes[[k]];
ek=eks[[k]];
position=node 2-1;
(*Dirichlet-Specified Solution*)
For[i=1,i≤2,i++,For[j=1,j≤2,j++,kglob[[position+i-1,position+j-1]]*=ek[[i,j]];];];];
kglob]
```

Figure A1 *ApplyNodeBCForce* and *ApplyNodeBCDisplacement*: function that imposes the boundary conditions

```
Compute2DShape[order_,eltype_]:=Block[{zeta1,zeta2,zeta3,a,b,xi,eta,psis,dpsis},
If[eltype==1,
If[order==1,
psis={
1/4(1-xi)(1-eta),1/4(1+xi)(1-eta),
1/4 (1+xi)(1+eta),1/4(1-xi)(1+eta)
};
,
psis={
xi eta(xi-1)(eta-1)/4,xi eta(xi+1)(eta-1)/4,
xi eta(xi+1)(eta+1)/4,xi eta(xi-1)(eta+1)/4,
-eta(xi+1)(xi-1)(eta-1)/2,-xi(xi+1)(eta+1)(eta-1)/2,
-eta(xi+1)(xi-1)(eta+1)/2,-xi(xi-1)(eta+1)(eta-1)/2,
(xi+1)(xi-1)(eta+1)(eta-1)
};
];
,
zeta1=1-xi-eta;
zeta2=xi;
zeta3=eta;
If[order==1,
psis={zeta1,zeta2, zeta3};
,
psis={
2*zeta1*(zeta1-1/2),2*zeta2*(zeta2-1/2),
2*zeta3*(zeta3-1/2),4*zeta1*zeta2,
4*zeta2*zeta3,4*zeta3*zeta1
};
];
];
dpsis=Transpose[Table[{D[psis[[i]],xi],D[psis[[i]],eta]},{i,1,Length[psis]}]];
{psis,dpsis}
];
```

Figure A2 *Compute2DShape*: function that computes the shape functions for three-node and six-node triangular elements and four-node and nine-node quadrilateral elements

```
<<NumericalDifferentialEquationAnalysis`
IntegrationRule[porder_,eltype_]:=Block[{pts,w,npts,matpsts},
If[eltype==1,
{pts,w}=Transpose[GaussianQuadratureWeights[porder+1,-1,1]];
npts=Length[pts];
matpsts=Flatten[Table[{pts[[j]],pts[[i]],w[[j]]w[[i]]},{i,npts,1,-1},{j,1,npts}],1];
,
If[porder==1,
matpsts={{1/3,1/3,1/2}};
];
If[porder==2,
matpsts={{1/6,1/6,1/2 1/3},{2/3,1/6,1/2 1/3},{1/6,2/3,1/2 1/3}};
];
];
N[matpsts]
]
```

Figure A3 *IntegrationRule*: function that computes the Gaussian quadrature points and weights for triangular and quadrilateral elements

```
Contribute[data_]:=Block[{psis,GradPsi,Jac,x,y,nnodes,GradPhi,DetJ,InvJac,B,C,ek,ef,elcoords,weight},
{psis,GradPsi,elcoords,weight}=data;
{x,y}=psis.elcoords;
Jac=GradPsi.elcoords;
nnodes=Length[psis];
DetJ=Det[Jac];
InvJac=Inverse[Jac];
GradPhi=InvJac.GradPsi;
B={
Flatten[Table[{GradPhi[[1,i]],0},{i,1,nnodes}],1],
Flatten[Table[{0,GradPhi[[2,i]]},{i,1,nnodes}],1],
Flatten[Table[{GradPhi[[2,i]],GradPhi[[1,i]]},{i,1,nnodes}],1]
};
nusqr=nu nu;
C={{young/(1-nusqr),nu young/(1-nusqr),0},{nu young/(1-nusqr),young/(1-nusqr),0},
{0,0,young/(2 (1+nu))}};
ek=(Transpose[B].C.B)weight DetJ;
ek
];
```

Figure A4 *Contribute*: function that implements the differential equation in weak formulation

```
CalcStiff[order_,elcoords_,eltype_]:=Block[{nnodes,ek,ef,intrule,psis,GradPsi,xi,eta,data,w,npts},
nnodes=Length[elcoords];
ek=Table[0,{nnodes 2},{nnodes 2}];
ef=Table[0,{nnodes 2}];
intrule=IntegrationRule[order,eltype];
{psis,GradPsi}=Compute2DShape[order,eltype];
npts=Length[intrule];
Table[
{xi,eta,w}=intrule[[i]];
data={psis,GradPsi,elcoords,w};
ek+=Contribute[data];
,{i,1,npts}
];
{ek,ef}
]
```

Figure A5 *CalcStiff*: function that calculates the element stiffness matrix

```
Assemble[allcoords_,nnodes_,topol_,order_,eltype_]:=Block[{nels,rows,sz,cols,Kglob,Fglob,co,Ke,Fe,fu,rowglob,colglob},
nels=Length[allcoords];
rows=Length[allcoords[[1]]];
sz=2 Length[nnodes];
cols=rows;
Kglob=Table[0,{sz},{sz}];
Fglob=Table[0,{sz}];
Table[
co=allcoords[[k]]//N;
{Ke,Fe}=CalcStiff[order,co,eltype];
fu=-1;
Table[
rowglob=topol[[k,i]];
Table[
colglob=topol[[k,j]];
Kglob[[2 rowglob+fu,2 colglob+fu]]+=Ke[[2 i+fu,2 j+fu]];
Kglob[[2rowglob+fu,2colglob+1+fu]]+=Ke[[2i+fu,2j+1+fu]];
Kglob[[2rowglob+1+fu,2colglob+fu]]+=Ke[[2i+1+fu,2j+fu]];
Kglob[[2rowglob+1+fu,2colglob+1+fu]]+=Ke[[2i+1+fu,2j+1+fu]];
,
{j,1,cols}
];
Fglob[[2rowglob+fu]]+=Fe[[i]];,
{i,1,rows}
];
,
{k,1,nels}
];
{Kglob,Fglob}
]
```

Figure A6 *Assemble*: function that assembles the global stiffness matrix

```
GenerateGridMesh[a_,b_,nx_,ny_,order_]:=Block[{x=0.,y=0.,dx,dy,meshnodes={},i,j,
meshtopology={},allcoords,k,topolsz,l},
k=0;
If[order==2,
topolsz=9;
meshnodes=Flatten[Table[Table[{x,y},{x,0,a,a/(nx order-order)}],{y,0,b,b/(ny order-order)}],1];
For[i=1 ,i<ny,i++,
l=1;
For[j=1,j<nx,j++,
AppendTo[meshtopology,{l+k,l+2+k,4 nx+l+k,4 nx-2+l+k,l+1+k,l+1+nx 2+k,l+nx 4-1+k,2 nx+ l-1+k,2 nx+l+k}];
l+=2;
];
k+=4 nx-2;
];
,
topolsz=4;
meshnodes=Flatten[Table[Table[{x,y},{y,0,b,b/(ny order-order)}],{x,0,a,a/(nx order-order)}],1];
meshtopology=Flatten[Table[Table[{i+j,i+j+ny,i+j+ny+1,i+j+1},{i,1,nx ny-ny,ny}],{j,0,ny-2}],1];
];
allcoords=Table[meshnodes[[meshtopology[[i,j]]]],{i,1,Length[meshtopology]},{j,1,topolsz}];
{allcoords,meshnodes,meshtopology}
]
```

Figure A7 *GenerateGridMesh*: function that generates a mesh of rectangular four-node or nine-node elements

```
GenerateGraphics[nodes_,topology_,order_]:=Block[{meshvis,nodevis,v},
If[order==1,v={1,2,3,4},v={1,5,2,6,3,7,4,8}];
meshvis=Graphics[{FaceForm[],EdgeForm[Blue],GraphicsComplex[nodes,Polygon[topology[[All,v]]]]}];
nodevis=Graphics[{MapIndexed[Text[#2[[1]],#1,{-1,1}]&,nodes],{Blue,Point[nodes]}}];
{meshvis,nodevis}
];
```

Figure A8 *GenerateGraphics*: function that plots the mesh elements and node numbers

```
order=1;
young=1;
nu=0.25;
eltype=1;
mu=young/(2(1+nu));
lambda=young nu/((1+nu)(1-2nu));
allcoords={{{2,1},{5,2},{4,6},{1,4}}}
topol={{1,2,3,4}}
eltype=1;
forcing=0.;
nnodes={{2,1},{5,2},{4,6},{1,4}};
meshVis1=Graphics[{FaceForm[],EdgeForm[Black],GraphicsComplex[nnodes,Polygon[topol[[All,{1,2,3,4}]]]]}];
nodeVis=Graphics[{Black,PointSize[Medium],Point[nnodes]}];
nodeVis2=Graphics[{MapIndexed[Text[#2[[1]],#1,{2,2}]&,nnodes],{Black,Point[nnodes]}}];
Show[meshVis1,nodeVis,nodeVis2]
{kE,fE}=Assemble[allcoords,nnodes,topol,order,eltype];
MatrixForm[kE]
```

Figure A9 Code of the Example 4.1

```
order=2;
young=1;
eltype=1;
nu=0.25;
mu=young/(2(1+nu));
lambda=young nu/((1+nu)(1-2nu));
allcoords={{{0,0},{1,0},{1,1},{0,1},{0.5,0.1},{1.1,0.5},{0.5,1.1},{0.1,0.5},{0.6,0.6}}};
nnodes=Flatten[allcoords,1];
topol={{1,2,3,4,5,6,7,8,9}};
forcing=0.;
{KE,FE}=Assemble[allcoords,nnodes,topol,order,eltype];
MatrixForm[KE]
```

Figure A10 Code of the Example 4.2

```
order=1;
h=0.036;(*inch*)
young=30 10^6   h;
nu=0.25;
mu=young/(2(1+nu));
lambda=young nu/((1+nu)(1-2nu));
allcoords={{{0,0},{120,0},{120,160}},{{0,0},{120,160},{0,160}}}(*inch*)
topol={{1,2,3},{1,3,4}}
eltype=2;
forcing=0.;
nnodes={{0,0},{120,0},{120,160},{0,160}}
meshVis1=Graphics[{FaceForm[],EdgeForm[Blue],GraphicsComplex[nnodes,Polygon[topol[[All,{1,2,3}]]]]}];
nodeVis=Graphics[{MapIndexed[Text[#2[[1]],#1,{-1,1}]&,nnodes],{Blue,Point[nnodes]}}];
mesh1=Show[meshVis1,nodeVis]
{KE,FE}=Assemble[allcoords,nnodes,topol,order,eltype];

KE=ApplyNodeBCDisplacement[KE,{1,4},{{BIG,1},{1,BIG}},{{BIG,1},{1,BIG}}];
FE=ApplyNodeBCForce[FE,{2,3},{{800,0},{800,0}}];
MatrixPlot[KE]
sol=LinearSolve[KE,FE,Method->"Cholesky"];
scale=8000;

deformed=(Flatten[nnodes]+scale sol);
tabdeformed=Table[{ deformed[[i]], deformed[[i+1]]},{i,1,Length[deformed],2}];
meshVisDef=Graphics[{FaceForm[{LightRed,Opacity[3]}],EdgeForm[Red],GraphicsComplex[tabdeformed,Polygon[topol[[All,{1,2,3}]]]]}];
Show[meshVisDef,meshVis1]
10^4 sol//Chop
```

Figure A11 Code of the Example 4.3

```
eltype=1;
order=1;h=1;a=10;b=2;young=30 10^6;(*Young modulus-psi*)nu=0.25;mu=young/(2(1+nu));lambda=young nu/((1+nu)(1-2nu));(*Lamé Constants-psi*)
(*Mesh Generation and visualization*)
order1=1;
{allcoords1,meshnodes1,meshtopology1}=GenerateGridMesh[a,b,5,3,order1];
{allcoords2,meshnodes2,meshtopology2}=GenerateGridMesh[a,b,9,3,order1];
{allcoords3,meshnodes3,meshtopology3}=GenerateGridMesh[a,b,17,3,order1];
{meshvis1,nodevis1}=GenerateGraphics[meshnodes1,meshtopology1,order1];
{meshvis2,nodevis2}=GenerateGraphics[meshnodes2,meshtopology2,order1];
{meshvis3,nodevis3}=GenerateGraphics[meshnodes3,meshtopology3,order1];
Show[meshvis1,nodevis1]
Show[meshvis2,nodevis2]
Show[meshvis3,nodevis3]

order2=2;
{allcoords4,meshnodes4,meshtopology4}=GenerateGridMesh[a,b,3,2,order2];
{allcoords5,meshnodes5,meshtopology5}=GenerateGridMesh[a,b,5,2,order2];
{allcoords6,meshnodes6,meshtopology6}=GenerateGridMesh[a,b,9,2,order2];
{meshvis4,nodevis4}=GenerateGraphics[meshnodes4,meshtopology4,order2];
{meshvis5,nodevis5}=GenerateGraphics[meshnodes5,meshtopology5,order2];
{meshvis6,nodevis6}=GenerateGraphics[meshnodes6,meshtopology6,order2];
Show[meshvis4,nodevis4]
Show[meshvis5,nodevis5]
Show[meshvis6,nodevis6]

(*Stiffness Matrix and Load Vector Assembling*)
{K1,F1}=Assemble[allcoords1,meshnodes1,meshtopology1,order1,eltype];
{K2,F2}=Assemble[allcoords2,meshnodes2,meshtopology2,order1,eltype];
{K3,F3}=Assemble[allcoords3,meshnodes3,meshtopology3,order1,eltype];
{K4,F4}=Assemble[allcoords4,meshnodes4,meshtopology4,order2,eltype];
{K5,F5}=Assemble[allcoords5,meshnodes5,meshtopology5,order2,eltype];
{K6,F6}=Assemble[allcoords6,meshnodes6,meshtopology6,order2,eltype];

(*Imposing Boundary Conditions*)
(*Force Vector*)
F1=ApplyNodeBCForce[F1,{1,2,3},{{0,-75},{0,-150},{0,-75}}];
F2=ApplyNodeBCForce[F2,{1,2,3},{{0,-75},{0,-150},{0,-75}}];
F3=ApplyNodeBCForce[F3,{1,2,3},{{0,-75},{0,-150},{0,-75}}];

F4=ApplyNodeBCForce[F4,{1,6,11},{{0,-50},{0,-200},{0,-50}}];
F5=ApplyNodeBCForce[F5,{1,10,19},{{0,-50},{0,-200},{0,-50}}];
F6=ApplyNodeBCForce[F6,{1,18,35},{{0,-50},{0,-200},{0,-50}}];

(*Null Displacement*)
BIG=10^12;
K1=ApplyNodeBCDisplacement[K1,{13,14,15},{{{BIG,1},{1,BIG}},{{BIG,1},{1,BIG}},{{BIG,1},{1,BIG}}}];
K2=ApplyNodeBCDisplacement[K2,{25,26,27},{{{BIG,1},{1,BIG}},{{BIG,1},{1,BIG}},{{BIG,1},{1,BIG}}}];
K3=ApplyNodeBCDisplacement[K3,{49,50,51},{{{BIG,1},{1,BIG}},{{BIG,1},{1,BIG}},{{BIG,1},{1,BIG}}}];

K4=ApplyNodeBCDisplacement[K4,{5,10,15},{{{BIG,1},{1,BIG}},{{BIG,1},{1,BIG}},{{BIG,1},{1,BIG}}}];
K5=ApplyNodeBCDisplacement[K5,{9,18,27},{{{BIG,1},{1,BIG}},{{BIG,1},{1,BIG}},{{BIG,1},{1,BIG}}}];
K6=ApplyNodeBCDisplacement[K6,{17,34,51},{{{BIG,1},{1,BIG}},{{BIG,1},{1,BIG}},{{BIG,1},{1,BIG}}}];
(*Solving Linear System*)
sol1=LinearSolve[K1,F1,Method->"Cholesky"];sol2=LinearSolve[K2,F2,Method->"Cholesky"];sol3=LinearSolve[K3,F3,Method->"Cholesky"];
sol4=LinearSolve[K4,F4,Method->"Cholesky"];sol5=LinearSolve[K5,F5,Method->"Cholesky"];sol6=LinearSolve[K6,F6,Method->"Cholesky"];
TableForm[{{"","Linear","Reference","Quadratic","Reference"},
{"15 nodes",SetPrecision[-sol1[[2 2]]100,4],0.3134,SetPrecision[-sol4[[6 2]]100,4],0.5031},
{"27 nodes",SetPrecision[-sol2[[2 2]]100,4],0.4388,SetPrecision[-sol5[[10 2]]100,4],0.5129},
{"51 nodes",SetPrecision[-sol3[[2 2]]100,4],0.4878,SetPrecision[-sol6[[18 2]]100,4],0.5137}}]
```

Figure A12 Code of the Example 4.4