ENGR7019 Engineering Dissertation Project Final Report

Development of Dynamic FE code in MATLAB (for planar quad elements)

by

Marcela Alessandra Vázquez Pérez
19193557

MSc MOTORSPORT ENGINEERING
Module: Dissertation
Academic year: 2023

Word count	Main Part	Appendix
	6,833	3,304
Number of illustrations	Main Part	Appendix
	11	0

School of Engineering, Computing and Mathematics
Oxford Brookes University

Statement of originality

Except for those parts in which it is explicitly stated to the contrary, this project is my own work. It has not been submitted for any degree at this or any other academic or professional institution.

Signature of Author Date

Regulations Governing the Deposit and Use of Master of Science Dissertations in the School of Engineering, computing and Mathematics,

Oxford Brookes University.

1. The 'top' copies of projects submitted in fulfilment of Master of Science course requirements shall normally be kept by the Department.
2. The author shall sign a declaration agreeing that, at the supervisor's discretion, the dissertation may be submitted in electronic form to any plagiarism checking service or tool.
3. The author shall sign a declaration agreeing that the dissertation be available for reading and copying in any form at the discretion of either the project supervisor or in their absence the Head of Postgraduate Programmes, in accordance with 5 below.
4. The project supervisor shall safeguard the interests of the author by requiring persons who consult the dissertation to sign a declaration acknowledging the author's copyright.
5. Permission for anyone other than the author to reproduce in any form or photocopy any part of the dissertation must be obtained from the project supervisor, or in their absence the Head of Postgraduate Programmes, who will give his/her permission for such reproduction only to the extent which he/she considers to be fair and reasonable.

I agree that this dissertation may be submitted in electronic form to any plagiarism checking service or tool at the discretion of my project supervisor in accordance with regulation 2 above.

I agree that this dissertation may be available for reading and photocopying at the discretion of my project supervisor or the Head of Postgraduate Programmes in accordance with regulation 5 above.

Signature of Author Date

Abstract

In the present research, a solid MATLAB-based method for dynamic finite element analysis with an interactive interface for structural problems is presented. The process is started by the user providing important parameters including element and node counts, material properties, and nodal positions. The behaviour of an element is defined by natural shape functions, which results in the numerical calculation of the strain displacement matrix. This code accurately calculates the element stiffness matrix for each element while taking into account conditions of plane stress or plane strain. Then the creation of a global stiffness matrix through matrix assembly is performed which was the main challenge.

Then boundary conditions are also provided by the user and are applied to the vectors U and F, after which the matrix is solved by partitioning and Gaussian elimination to identify unknown displacements and reactions. Stress vectors are recovered during post-processing for each element, making it easier to calculate principal stresses and angles. Validation was completed comparing the results given by the code with specialized books who had problems and answers already developed.

The successful completion of this project places the code as a great teaching tool for students and teachers wishing to study or teach the Finite Element Method (FEM), providing practical insights into structural problem-solving.

Highlights

- This effectiveness highlights MATLAB's efficacy as a programming environment for finite element method implementations.
- Offering a user-friendly interface for inputting problem parameters in Finite Element Method (FEM) is extremely beneficial in guiding students.
- The successful application of Gaussian integration to improve dynamic analysis accuracy. This not only demonstrates the project's sophisticated skills, but also its potential for precisely tackling complicated real-world structural challenges.
- The incorporation of several numerical calculations into a single comprehensive code to make it more accessible and user-friendly for both students and engineers.

Table of Contents of Main Part

List of Figures of Main part

List of Tables of Main part

List of Symbols and Abbreviations

η	Natural coordinates
$[B]$	Strain Displacement Matrix
$[D]$	Material Matrix
$[K]$	Global Stiffness Matrix
σ	Plane Stress
τ_{xy}	Shearing stress acting in the direction of the x axis on a surface perpendicular to the y axis
υ	Poisson's ratio
ε	Plane Strain
ξ	Natural coordinates
E	Young's modulus
t	Element Thickness

1 Introduction

1.1 Concise background information, the project aim and objectives

The aim of this project is to develop an interactive interface that allows users to provide material properties, specify input loads, and determine if the problem refers to either plane stress or plane strain. The code will then solve for strains and stresses within the structure.

Objectives:

1. Create a MATLAB program that uses natural shape functions to determine the stiffness matrix for a single planar quadrilateral element.
2. Create a global stiffness matrix by combining the stiffness matrices and extending the MATLAB method to solve for two elements initially.
3. Improve the accuracy of the dynamic analysis code by using numerical integration techniques like Gaussian integration.
4. Create a user-friendly interactive interface for mid-level users to make it easier to utilize the code as a teaching tool for finite element analysis.
5. Create code documentation.

1.2 Summary of literature review

1.2.1 Introduction

The development of dynamic finite element algorithms for planar quadrilateral elements has received substantial interest in recent years due to its vast applications in structural analysis and design. These algorithms strive to provide accurate predictions of strains and stresses in various engineering materials under dynamic loading situations. One critical feature of such codes is their ability to handle both plane stress and plane strain problems effectively. This literature review delves into the notions of background of FEM, plane stress and strain, the formulation of the stiffness matrix, and the significance of four-node quadrilateral elements in dynamic finite element analysis.

1.2.2 Main Body

Our environment is more complicated than what the human mind can process in a single effort. Therefore, complicated systems are broken down into their individual components or more manageable subdomains called elements, whose behaviours are simpler to understand. The original system can then be recreated using these components, allowing one to investigate and analyse its general behaviour. This approach makes it possible to explore complex systems in a way that is more intuitive and effective, promoting perceptive comprehension and well-informed decision making (Zienkiewicz et al. 2013). The Finite Element Analysis (FEA) is an effective computational technique that uses the Finite Element Method (FEM) to solve challenging engineering systems (Simon-Marinica, Adrian Bogdan et al. 2021). The FEM uses Partial Differential Equations (PDEs) to analyze finite elements in any given phenomenon (Obumneme et al. 2022). With this approach, the system is discretized into a set of smaller, manageable components and the equations of motion for each of these components are then solved. These equations are then combined to give a solution for the entire system.

The origins of FEM can be dated back to the mid-twentieth century. The works of Hrennikoff (1941), Courant (1943), Argyris & Kelsey (1954), Turner et al. (1956), Clough (1960) and Zienkiewicz (1977), highlight significant aspects of the FEM and are considered important in the development thereof. The term "finite element" was first used by Ray W. Clough (Selleri 2022). The contributions of these pioneering academics established the framework for the development and widespread use of the FEM, which has since evolved into a versatile and powerful tool for tackling complex engineering problems across multiple disciplines. The main principle behind the finite element approach is to identify an approximate solution to any complex real-life engineering problem by replacing it with a simpler one (Rao 2018).

The 4-node quadrilateral was first created by Argyris in 1954 as a rectangular panel with reinforced edges. Taig & Kerr (1964) published a conforming generalisation to arbitrary geometry using quadrilateral-fitted coordinates already denoted as ξ , η, but running from 0 to 1. Quadrilateral elements are a type of planar element used in FEA. These elements are isoparametric, which means they can have curved boundaries and offer more flexibility. In terms of meshing and accuracy, bilinear quadrilateral elements are thought to be superior to straightforward linear triangular elements in 2-dimensional analysis (Perumal & Mon 2011).

The element stiffness matrix contains all of a finite element's essential properties. For a structural finite element, the stiffness matrix holds the geometric and material behaviour details that show the element's resistance to deformation under loading (Hutton 2004). It connects nodal forces to displacements and takes on a different shape depending on the number of degrees of freedom for the element in question (Rees 1997). We are taking into account a quadrilateral element that is defined by four nodes in natural coordinates (ξ,η) and we are also considering natural shape functions. These shape functions can be used to determine the stiffness matrix for the element, as well as to interpolate the nodal values within the element. Natural coordinates make identifying the element stiffness matrix and putting together the global element matrix easier (Ferreira 2008).

Two fundamental presumptions used in structural analysis to reduce the complexity of three-dimensional problems are plane stress and plane strain. The plane stress problems are those of thin plates loaded over their lateral boundaries by tractions that are uniform across the plate's thickness or symmetric with respect to its mid-plane z = 0. On the other hand, the plane strain problems involve long cylindrical bodies with homogeneous cross sections that are loaded by tractions which are orthogonal to the body's longitudinal (z) axis and for each cross section z = const (Lubarda & Lubarda 2020). These assumptions allow for more efficient and simpler dynamic load analysis of structures.

There is a scarcity of research or material addressing the construction of a dynamic finite element code in MATLAB for planar quad elements with the precise features and aims mentioned in this project. While there may be existing literature on the subject, it is possible that the combination of elements such as an interactive interface, user-defined material properties and loads, and consideration of plane stress or plane strain concerns has not been substantially researched or recorded. As a result, the research and development of a dynamic finite element code that incorporates these specific features and objectives in the context of planar quad elements is required to close the knowledge gap. This may entail researching and developing appropriate algorithms, data structures, and numerical approaches to handle dynamic analysis, user interaction, and the needed features. The advantage with MATLAB is that the extensive mathematical and graphical functionalities eliminate the need to create these functions from scratch or look for appropriate pre-existing libraries. Due to this, even very straightforward two-dimensional finite element programs in MATLAB can be effectively stated in a few hundred lines of code, as opposed to the possibly thousands of lines needed in languages like Fortran or C++.

Furthermore, comparing the performance, accuracy, and efficiency of the produced code to existing commercial software or analytical solutions will help to close this knowledge gap.

1.2.3 Conclusion

In conclusion, reliable strain and stress predictions in engineering materials under dynamic loading circumstances depend greatly on the development of dynamic finite element codes for planar quad elements. The above review of the literature has given a general overview of the assumptions relating to plane stress and strain, the creation of the stiffness matrix, and the relevance of planar quad elements in dynamic finite element analysis. For a dynamic finite element code to be successfully implemented and be able to handle issues involving both plane stress and plane strain, it is essential to comprehend these ideas.

By undertaking this project, I have the opportunity to contribute new perspectives, innovative approaches, and feasible solutions to the field of dynamic finite element analysis for planar quad elements, reducing the knowledge gap that currently exists and perhaps even enhancing the capabilities of finite element analysis software in this particular area.

1.3 Originality and Contribution

This research project makes an important contribution to the field of Finite Element Analysis (FEA), notably in the context of dynamic analysis for planar quad elements. My supervisor laid the groundwork for the project by developing a code capable of generating the stiffness matrix for a single element. However, the challenge of extending this capability to additional elements remained unknown territory.

This project's key innovation is in successfully overcoming this obstacle. I improved the first code to handle many elements concurrently, which greatly improves its practical utility. By doing so, I enabled engineers and academics to study complicated systems more accurately and efficiently, which was previously unavailable in the literature.

Furthermore, my efforts extended beyond code development. Recognizing the need for user-friendly educational tools in the field of FEA, I turned the code into an interactive interface. Students and practitioners can use this interface to enter system parameters, define material properties, provide loads, and differentiate between plane stress and plane strain conditions. The stiffness matrix for each element is then generated, the global stiffness matrix is assembled, boundary conditions are applied, and strains and stresses are computed. This instructional tool not only helps students grasp the fundamentals of FEA, but it also simplifies the actual application of dynamic analysis.

1.4 The approach

The successful implementation of the project's objectives needed an organized and systematic strategy that included both code development and user interface design. The following section provides a quick description of the tactics and methodologies used to attain each objective:

Objective 1: Create a MATLAB program for a single element stiffness matrix: To achieve the first objective, a MATLAB program was meticulously constructed to compute the stiffness matrix for a single planar quadrilateral element. The method involved the use of natural shape functions, which allowed for the precise estimation of the stiffness matrix of the element. The basis for all later goals was laid in this first step.

Objective 2: Create a Global Stiffness Matrix for Multiple Elements: The next goal was to design a process for assembling a global stiffness matrix, building on the success of the single element stiffness matrix. This was accomplished by extending the MATLAB program to handle several elements. Individual element stiffness matrices were combined to provide a full global stiffness matrix, which is required for dynamic finite element analysis.

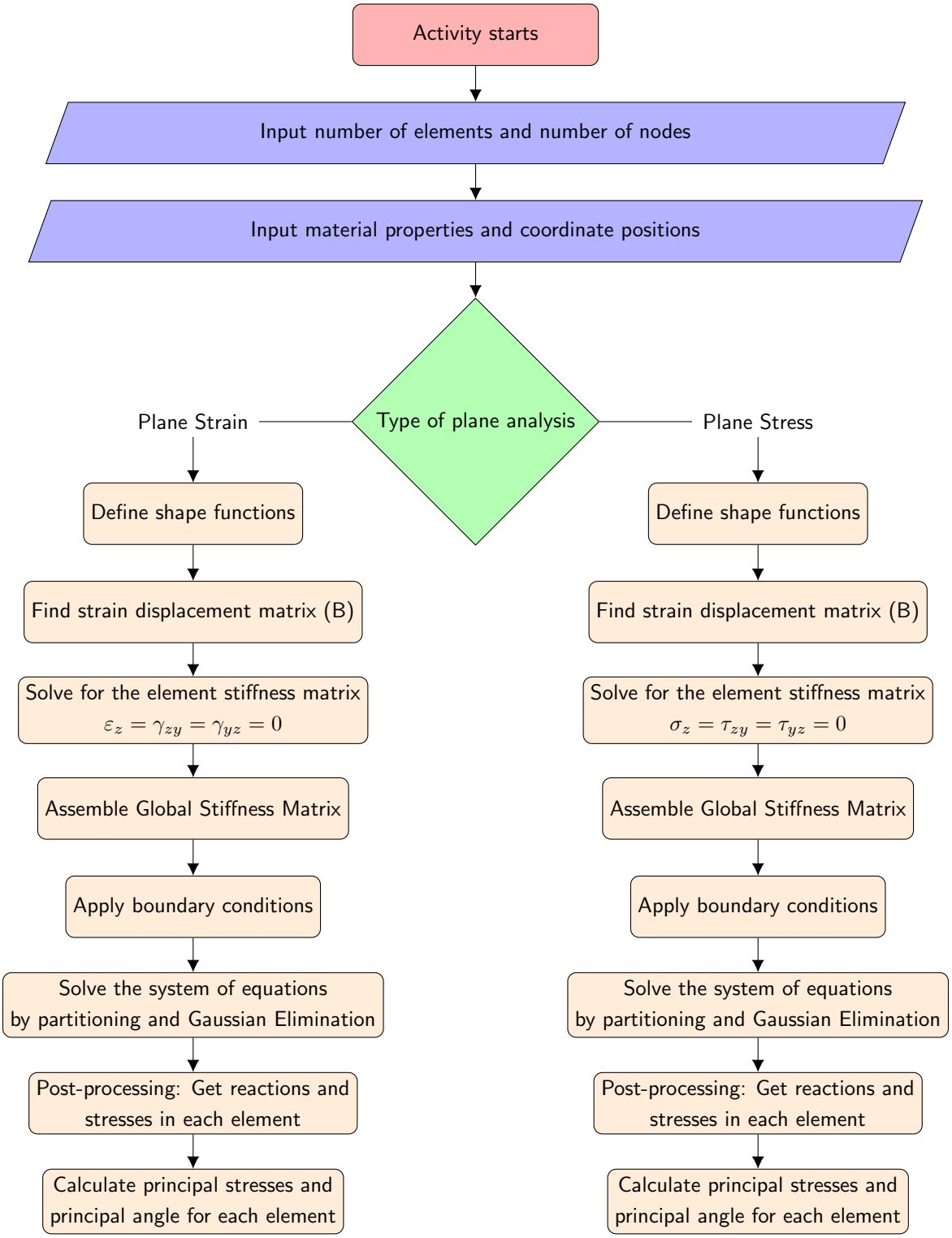
Objective 3: Improve Accuracy Using Numerical Integration Techniques: In FEM, numerical integration is widely used. To improve the accuracy of the dynamic analysis code, numerical integration techniques, particularly Gaussian integration, were used. These strategies were added into the code to execute integration over the element domain. This process considerably enhanced the precision of strain and stress estimates, resulting in reliable data for engineering analysis.

Objective 4: Create a User-Friendly Interactive Interface: An interactive interface was created because it was crucial to have tools that were easy to use for instruction and practical use. Mid-level users can interactively input system parameters, specify loads, define material attributes, and decide between conditions for plane stress and plane strain thanks to this user-friendly interface.

Objective 5: Documentation: Creating documentation that not only assists users in using the code but also educates them on the underlying processes. This dual-purpose documentation seeks to provide users with the information and skills needed to utilize the code for engineering analysis while also developing a greater understanding of finite element analysis's complexities.

2 Finite Element Analysis Methodology

2.1 Flow Chart



The flowchart above describes the methodical approach taken to develop a dynamic Finite Element Analysis code specifically designed for planar quadrilateral elements. With the help of this code, structural behavior under dynamic loading situations can be precisely predicted. The flowchart provides a concise, step-by-step picture of the code's execution for both Plane Stress and Plane Strain analyses based on the user's inputs.

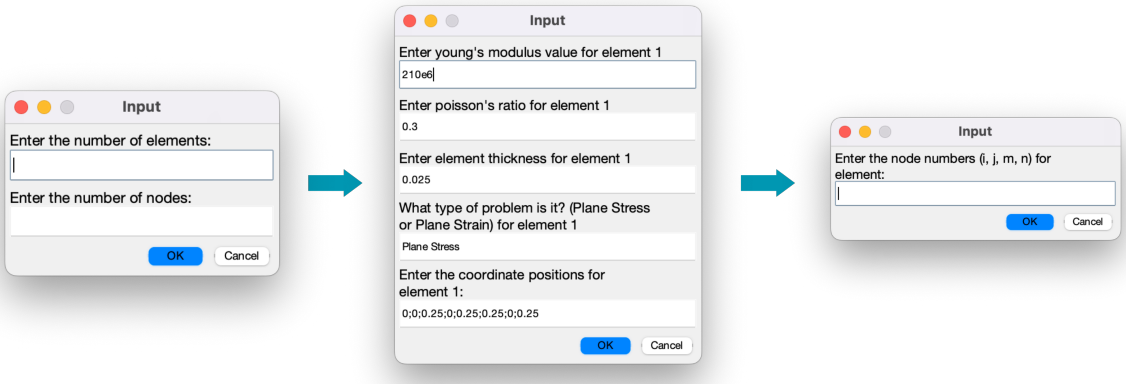


Figure 1: Prompts for user inputs at the beginning of the code

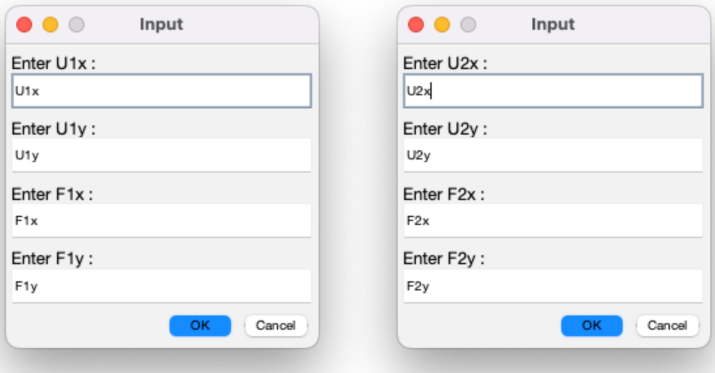


Figure 2: Prompts for boundary conditions for the number of nodes given by the user

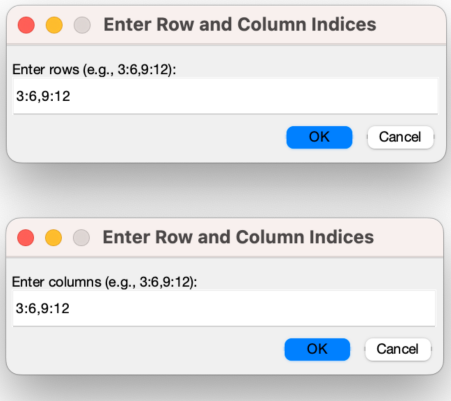


Figure 3: Prompt user for the rows and columns to extract submatrix

1. Input parameters: The user starts the process by specifying the number of elements and nodes, as well as the properties of the material and coordinate positions as shown in Figure 1.
2. Shape Function Definition: Natural shape functions are used to describe the behavior of the elements
3. The strain displacement matrix (B) is calculated by the code to link strains to displacements within the element.
4. Element Stiffness Matrix: The code solves the element stiffness matrix while taking plane stress or plane strain conditions into account.
5. Matrix Assembly: The component stiffness matrices are combined to form a global stiffness matrix.
6. Applying the boundary conditions to the vectors U and F.
7. Solve equations: The matrix will be solved by partitioning the global stiffness matrix and Gaussian elimination to find unknown displacements and reactions.
8. Post-processing: The stress vector is obtained for each element and with that we can get the principal stresses and angle for each element.

2.2 Schematic arrangement of the system being analyzed

A schematic diagram of the quadrilateral element is shown in Figure 4 (Qianwei et al. 2019). Nodal points are labelled in an anticlockwise direction and in ascending order. The global coordinates of the four nodes are given by $(x_1, y_1), (x_2, y_2), (x_3, y_3), (x_4, y_4)$ and can be seen in Figure 4b. Local coordinates are used to define element interpolation functions that meet special continuity requirements that may not be met by global coordinate interpolation (Akin 1994). The element is mapped to a rectangle through the use of the natural coordinates ξ and η as shown in Figure 4a (Kattan 2008)

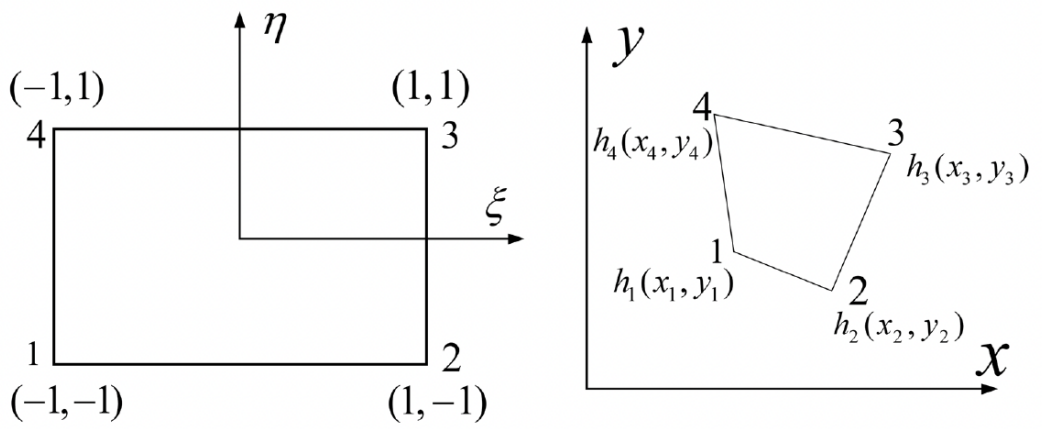


Figure 4: Schematic diagram of the quadrilateral element (a) Quadrilateral master/parent element in $\xi\eta$-plane (left). (b) Quadrilateral element in xy-plane (right).

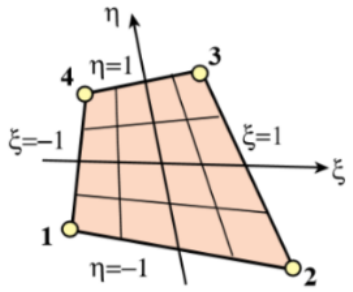


Figure 5: 4 Node Bilinear Quadrilateral

2.3 Justification for the methods used

In this section, a thorough justification for the approaches and techniques used in the construction of the dynamic finite element algorithm for planar quadrilateral components using MATLAB is presented. The methodology employed and its relevance to the research objectives is fully addressed.

2.3.1 Method Selection

The creation of this code needs careful consideration of the methodologies utilized for modeling, analysis, and implementation. An increasing number of engineering applications use mathematical models. Various models are designed in the field of Elasticity Theory to analyze the behavior of structures in terms of forces, deformations, and displacements. One-dimensional, two-dimensional, or three-dimensional problem scenarios can all be addressed using these models. In this project our focus is in the two-dimensional problems where partial differential equations are used. The difficulty of solving the established model arises as a result of the attempt to build a model to understand a particular physical occurrence, emphasizing the crucial role of differential equations in this quest for understanding and resolution. Analytical and/or numerical methods of resolution are both possible. The Finite Element Method stands out among the other numerical methods for the scenario stated (Reis & Júnior 2018). This method was chosen based on its suitability for fulfilling the research objectives:

2.3.2 Finite Element Method (FEM)

For modeling and simulating structural behavior, the FEM was chosen as the fundamental technique. The power of the FEM as a numerical analysis tool can be emphasised because with it, it is possible to simulate physical issues with more complicated boundary conditions. It is a well-known engineering approach, and its ability to handle complex geometries is one of its greatest advantages. FEM has the ability to accurately estimate solutions to partial differential equations, making it an excellent choice for dynamic analysis. Furthermore, it may be used to solve complicated problems in science and engineering, demonstrating its adaptability and usefulness (Buchanan 1995).

2.3.3 MATLAB Programming

The programming environment was chosen because of MATLAB's adaptability and robust support for numerical computing. For the purpose of implementing the FEM, MATLAB offers a sizable library of functions for matrix operations, linear algebra, and optimization. In this particular case, the use of MATLAB Symbolic Math Toolbox, has proven to be a valuable asset in the development of the code for this dissertation project. The integration of the MATLAB Symbolic Math Toolbox proves to be extremely beneficial, providing a wide variety of benefits that reach far beyond the scope of simple computation. This computational tool is an essential component in both teaching and learning techniques, allowing for a deeper understanding of complicated mathematical concepts. It enables us to effortlessly switch between symbolic and numerical computation, greatly lowering the laborious load of manual calculations. By utilizing MATLAB's features, we can shift our attention to important areas of our research, like figuring out the complex connections between mathematical models and their counterparts in the real world. MATLAB not only simplifies the computational process, but it also allows us to go deeper into the core ideas and concepts underlying our dissertation, thereby improving the rigorousness and accuracy of our study (Ortigoza & Ponce De La Cruz Herrera 2023).

2.3.4 Relevance

The development of this code is especially important given the absence of comprehensive educational resources for cultivating an adequate knowledge of the Finite Element Method. Students may find FEM challenging since it frequently requires a strong mathematical foundation, especially when they face FEA and FEM modules. Many people have trouble remembering linear algebra concepts or understanding the abstract nature and importance of the equations. This instructional tool fills in this knowledge gap by offering an approachable framework for understanding the complexities of FEM. Despite the abundance of study and publications on this topic, existing codes in MATLAB and other programming languages are frequently dispersed and difficult to understand. As a result, the main goal here is to simplify the learning process and encourage a deeper comprehension of FEM principles, making them more accessible to both students and researchers.

2.3.5 Availability of Resources

MATLAB is a great option for implementation because it is widely accessible and simple to use. Academics and engineers can effectively utilize the code thanks to its user-friendly interface, which promotes wider acceptance among the engineering community.

2.3.6 Method Validation

To verify the resilience and dependability of the generated code, the validation procedure was carefully carried out. In addition to developing a comparative code in Python, particular problem parameters for planar quadrilateral elements were gathered from authoritative textbooks that gave answers to these difficult problems. The accuracy and consistency of the MATLAB and Python implementations were then evaluated systematically against these known parameter ranges. This validation technique not only strengthens the code's legitimacy, but also establishes its ability to handle complex planar quad element challenges.

2.3.7 Conclusion

In conclusion, the approaches used to construct the dynamic finite element code for planar quadrilateral components using MATLAB are well-justified. The finite element method has been chosen as the main computational method since it is inherently relevant to this research project. A variety of complicated problems involving forces, deformations, and structural displacements have been successfully addressed by this highly recognized structural mechanics method. It is a great option for fulfilling the projects objectives due to its versatility to various material models, boundary conditions, and element types. The use of the MATLAB programming language in this project provides useful insights into FEM programming (Chessa 2002). On top of that, the synergy between MATLAB's large mathematical libraries and FEM's programming capabilities makes it the ideal mix for creating a powerful teaching tool for students. By utilizing this synergy, this initiative advances structural analysis while also making it easier to understand the mathematical foundations of FEM, making it relevant and approachable to the educational community.

2.4 Important mathematical expressions used

The interpolation functions for the coordinates are:

$$x = \sum_{i=1}^{4} N_i x_i; \quad y = \sum_{i=1}^{4} N_i y_i; \tag{1}$$

Since the element is isoparametric, the following equations describe the relationship between local and global coordinate systems:

$$x = N_1 x_1 + N_2 x_2 + N_3 x_3 + N_4 x_4 \tag{2}$$

$$y = N_1 y_1 + N_2 y_2 + N_3 y_3 + N_4 y_4 \tag{3}$$

The shape functions for a bilinear quadrilateral element in terms of natural coordinates given by (Kattan 2008) are:

$$N_1 = \frac{1}{4}(1-\xi)(1-\eta) \tag{4}$$

$$N_2 = \frac{1}{4}(1+\xi)(1-\eta) \tag{5}$$

$$N_3 = \frac{1}{4}(1+\xi)(1+\eta) \tag{6}$$

$$N_4 = \frac{1}{4}(1-\xi)(1+\eta) \tag{7}$$

The Strain Displacement Matrix $[B]$ is given by:

$$[B] = \frac{1}{|J|}\left[B_1, B_2, B_3, B_4\right] \tag{8}$$

The determinant $|J|$ is given by:

$$|J| = \frac{1}{8}\left[x_1, x_2, x_3, x_4\right]\begin{bmatrix} 0 & 1-\eta & \eta-\xi & \xi-1 \\ \eta-1 & 0 & \xi+1 & -\xi-\eta \\ \xi-\eta & -\xi-1 & 0 & \eta+1 \\ 1-\xi & \xi+\eta & -\eta-1 & 0 \end{bmatrix} \tag{9}$$

Where $[B_i]$ is given by:

$$[B_i] = \begin{bmatrix} a\frac{\delta N_i}{\delta\xi} & -b\frac{\delta N_i}{\delta\eta} & 0 & 0 \\ 0 & 0 & c\frac{\delta N_i}{\delta\eta} & -d\frac{\delta N_i}{\delta\xi} \\ c\frac{\delta N_i}{\delta\eta} & d\frac{\delta N_i}{\delta\xi} & a\frac{\delta N_i}{\delta\xi} & -b\frac{\delta N_i}{\delta\eta} \end{bmatrix} \tag{10}$$

The parameters a,b,c, and d are given by:

$$a = \frac{1}{4}\left[y_1(\,\xi-1) + y_2(-1-\xi) + y_3(1+\xi) + y_4(1-\xi)\right] \tag{11}$$

$$b = \frac{1}{4}\left[y_1(\,\eta-1) + y_2(1-\eta) + y_3(1+\eta) + y_4(-1-\eta)\right] \tag{12}$$

$$c = \frac{1}{4}\left[x_1(\,\eta-1) + x_2(1-\eta) + x_3(1+\eta) + x_4(-1-\eta)\right] \tag{13}$$

$$d = \frac{1}{4}\left[x_1(\,\xi-1) + x_2(-1-\xi) + x_3(1+\xi) + x_4(1-\xi)\right] \tag{14}$$

For plane stress the matrix [D] is given by:

$$[D] = \frac{E}{1-\nu^2}\begin{bmatrix} 1 & \nu & 0 \\ \nu & 1 & 0 \\ 0 & 0 & \frac{1-\nu}{2} \end{bmatrix} \tag{15}$$

For plane strain the matrix [D] is given by:

$$[D] = \frac{E}{(1+\nu)(1-2\nu)}\begin{bmatrix} 1-\nu & \nu & 0 \\ \nu & 1-\nu & 0 \\ 0 & 0 & \frac{1-2\nu}{2} \end{bmatrix} \tag{16}$$

To solve for the element stiffness matrix using natural coordinates, we must integrate the following expression:

$$[k] = t\int_{-1}^{1}\int_{-1}^{1}[B]^T[D][B]\,|J|\,d\xi d\eta \tag{17}$$

where $[B]^T$ is the transpose of the strain displacement matrix [B] and [D] is the material matrix. The strain

displacement matrix [B] and the Det(J)

The following structure equation results from obtaining the global stiffness matrix K:

$$[K]\{U\} = \{F\} \tag{18}$$

where U is the global nodal displacement vector and F is the global nodal force vector. The stress vector is produced for each element using the approach that follows after the unknown displacements and reactions have been identified:

$$\{\sigma\} = [D][B]\{u\} \tag{19}$$

where σ is the stress vector in the element (of size 3×1) and u is the 8×1 element displacement vector. The vector σ is written for each element as:

$$\{\sigma\} = \begin{bmatrix} \sigma_x \\ \sigma_y \\ \tau_{xy} \end{bmatrix} \tag{20}$$

2.5 Table of input parameters

Symbol	Parameter
E	Young's modulus
υ	Poisson's ratio
t	Element Thickness
$\varepsilon = \begin{Bmatrix} \varepsilon_x \\ \varepsilon_y \\ \gamma_{xy} \end{Bmatrix}$	Plane Strain
$\sigma = \begin{Bmatrix} \sigma_x \\ \sigma_y \\ \tau_{xy} \end{Bmatrix}$	Plane Stress

Table 1: Input Parameters

Node I	ξ_I	η_I
1	-1	-1
2	1	-1
3	1	1
4	-1	1

Table 2: Nodal coordinates in the parametric element domain (Fish & Belytschko 2007)

2.6 Table of alternative approaches

Alternative	Description	Justification of Approach
Alternative 1	Instead of using MATLAB, write the dynamic finite element code in Python. For this solution, it would be necessary to investigate the Python libraries and tools that are available for finite element analysis and modify the code implementation accordingly.	Python provides a variety of strong libraries and tools for scientific computing and numerical analysis, allowing for flexibility and possibly improving performance. It is very user friendly and intuitive. It is also open source which gives us a wide range of libraries we can used for the project (Nazaruddin & Siallagan 2021).
Alternative 2	Instead of writing new code, use a commercial finite element analysis programme that is already available. This method entails choosing an appropriate software package that supports planar quad elements and has the required features and functionalities.	Commercial software solutions frequently have a long history, have undergone significant testing, and include a wide range of capabilities for dynamic analysis. Utilising such software can speed up development, provide access to cutting-edge features, and provide technical assistance. (Khennane 2013)
Alternative 3	Examine the viability of implementing artificial intelligence or machine learning methods (Pan et al. 2021) into the dynamic finite element code. Investigating how these innovative techniques might improve the analysis's precision, effectiveness, and automation could potentially create new opportunities for the project.	The accuracy and effectiveness of the dynamic analysis may be improved by integrating machine learning or AI approaches, which can also lead to improvements in data-driven modelling, optimisation, and automation (Jung et al. 2022). This alternative could be complemented by the Alternative 1 of doing it in python.

Table 3: Table of alternative approaches

2.7 Table of limitations

Assumption/Limitation	Description/Impact of the assumption on outcomes
Limited to bilinear quadrilateral elements	The project might not be suitable to problems with three dimensions. For complex geometries or structures that planar quad elements are unable to effectively portray, the accuracy of the results may be affected. It is a very specific approach to a certain problem.
Assumes linear variation across the element	Linear variation across a bilinear quadrilateral element can be described by the following function in terms of Cartesian coordinate: $f(x, y) = a_1 + a_1x + a_3y + a_4xy$
In a FE analysis, the displacements are more accurate than the calculated stresses	The displacements are carefully calculated by inverting the stiffness matrix at the nodal positions (integration points). The shape function is used to approximate the displacements within the elements. Because strains and stresses are calculated from displacements, they are less precise.
Need smaller elements to get good stress results	If the element size is relatively large, the accuracy of the stress results may suffer. To more properly reflect localized stress variations, smaller elements may be needed.

Table 4: Table of limitations

3 Results and discussion

3.1 Results

The complete code is located in Appendix A and it is tested against problems found in (Kattan 2008) where the results are compared to make sure they are correct.

3.1.1 Problem 1

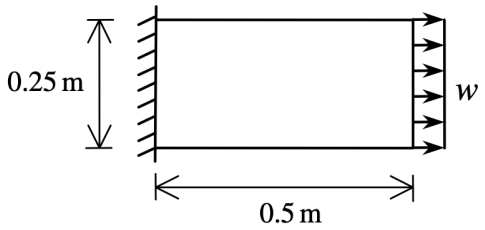


Figure 6: Thin plate for problem 1

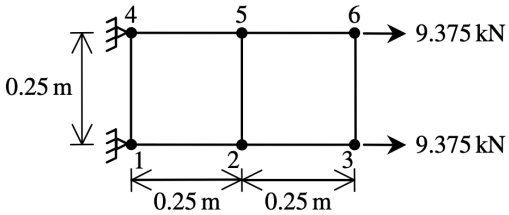


Figure 7: Discretization of Thin plate for problem 1 using two bilinear quadrilaterals

Given E = 210 GPa, $\upsilon = 0.3$, t = 0.025 m, and w = 3000 kN/m2 and the nodal positions of element 1 and element 2:

$$Element\,1 = \begin{pmatrix} x_1 \\ y_1 \\ x_2 \\ y_2 \\ x_3 \\ y_3 \\ x_4 \\ y_4 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0.25 \\ 0 \\ 0.25 \\ 0.25 \\ 0 \\ 0.25 \end{pmatrix} \qquad Element\,2 = \begin{pmatrix} x_1 \\ y_1 \\ x_2 \\ y_2 \\ x_3 \\ y_3 \\ x_4 \\ y_4 \end{pmatrix} = \begin{pmatrix} 0.25 \\ 0 \\ 0.5 \\ 0 \\ 0.5 \\ 0.25 \\ 0.25 \\ 0.25 \end{pmatrix}$$

Determine:

1. Global stiffness matrix [K]
2. Horizontal and vertical displacements at node 3 and 6 {U}
3. The reactions at nodes 1 and 4 {F}

Element number	Node i	Node j	Node m	Node n
1	1	2	5	4
2	2	3	6	5

Table 5: Element connectivity

4. The stresses in each element σ_x, σ_y, τ_{xy}
5. The principal stresses and principal angle for each element σ_1, σ_2, θ_p

```
K =

   1.0e+06 *

  Columns 1 through 9

    2.5962    0.9375   -1.5865   -0.0721         0         0    0.2885    0.0721   -1.2981
    0.9375    2.5962    0.0721    0.2885         0         0   -0.0721   -1.5865   -0.9375
   -1.5865    0.0721    5.1923         0   -1.5865   -0.0721   -1.2981    0.9375    0.5769
   -0.0721    0.2885         0    5.1923    0.0721    0.2885    0.9375   -1.2981         0
         0         0   -1.5865    0.0721    2.5962   -0.9375         0         0   -1.2981
         0         0   -0.0721    0.2885   -0.9375    2.5962         0         0    0.9375
    0.2885   -0.0721   -1.2981    0.9375         0         0    2.5962   -0.9375   -1.5865
    0.0721   -1.5865    0.9375   -1.2981         0         0   -0.9375    2.5962   -0.0721
   -1.2981   -0.9375    0.5769         0   -1.2981    0.9375   -1.5865   -0.0721    5.1923
   -0.9375   -1.2981         0   -3.1731    0.9375   -1.2981    0.0721    0.2885         0
         0         0   -1.2981   -0.9375    0.2885    0.0721         0         0   -1.5865
         0         0   -0.9375   -1.2981   -0.0721   -1.5865         0         0    0.0721

  Columns 10 through 12

   -0.9375         0         0
   -1.2981         0         0
         0   -1.2981   -0.9375
   -3.1731   -0.9375   -1.2981
    0.9375    0.2885   -0.0721
   -1.2981    0.0721   -1.5865
    0.0721         0         0
    0.2885         0         0
         0   -1.5865    0.0721
    5.1923   -0.0721    0.2885
   -0.0721    2.5962    0.9375
    0.2885    0.9375    2.5962
```

Figure 8: Global stiffness matrix for problem 1

The F global nodal force vector is:

```
F1x =  -9.3750
F1y =  -1.9741
F2x =  0.0000
F2y =  0.0000
F3x =  9.3750
F3y =  0.0000
F4x =  -9.3750
F4y =  1.9741
```

```
F5x = 0.0000
F5y = 0.0000
F6x = 9.3750
F6y = 0.0000
```

The U global nodal displacement vector is:

```
U1x = 0.0000e+00
U1y = 0.0000e+00
U2x = 3.4395e-06
U2y = 6.3181e-07
U3x = 7.0300e-06
U3y = 5.0321e-07
U4x = 0.0000e+00
U4y = 0.0000e+00
U5x = 3.4395e-06
U5y = -6.3181e-07
U6x = 7.0300e-06
U6y = -5.0321e-07
```

The element nodal displacement vectors u_1 and u_2 have the stresses σ_x, σ_y, τ_{xy}:

```
For u1:
sigma_x = 3000
sigma_y = 369.2797
tau_xy = 0
For u2:
sigma_x = 3000
sigma_y = -53.4178
tau_xy = 0
```

Principal Stresses for σ_1 and σ_2 and principal angle θ_p:

```
sigma1:
s1 = 3000
s2 = 369.2797
Theta = 0
sigma2:
s1 = 3000
s2 = -53.4178
Theta = 0
```

When these results are compared to those reported in the reference book, a significant similarity is noticed. This congruence confirms the robustness and accuracy of the developed code, certifying its functionality.

3.1.2 Problem 2

Given E = 210 GPa, $\upsilon = 0.3$, t = 0.025 m, and w = 100 kN/m2 for the three elements shown in Figure 9:

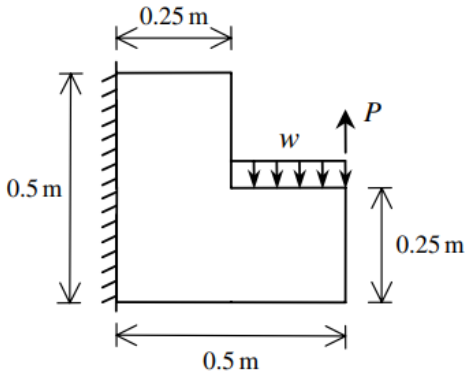


Figure 9: Thin Plate with a Distributed Load and a Concentrated Load for problem 2

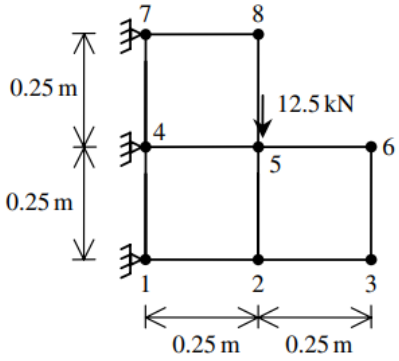


Figure 10: Discretization of Thin Plate Using Three Bilinear Quadrilaterals Elements for problem 2

$$Element\,1 = \begin{pmatrix} x_1 \\ y_1 \\ x_2 \\ y_2 \\ x_3 \\ y_3 \\ x_4 \\ y_4 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0.25 \\ 0 \\ 0.25 \\ 0.25 \\ 0 \\ 0.25 \end{pmatrix} \quad Element\,2 = \begin{pmatrix} x_1 \\ y_1 \\ x_2 \\ y_2 \\ x_3 \\ y_3 \\ x_4 \\ y_4 \end{pmatrix} = \begin{pmatrix} 0.25 \\ 0 \\ 0.5 \\ 0 \\ 0.5 \\ 0.25 \\ 0.25 \\ 0.25 \end{pmatrix} \quad Element\,3 = \begin{pmatrix} x_1 \\ y_1 \\ x_2 \\ y_2 \\ x_3 \\ y_3 \\ x_4 \\ y_4 \end{pmatrix} = \begin{pmatrix} 0 \\ 0.25 \\ 0.25 \\ 0.25 \\ 0.25 \\ 0.5 \\ 0 \\ 0.5 \end{pmatrix}$$

Element number	Node i	Node j	Node m	Node n
1	1	2	5	4
2	2	3	6	5
3	4	5	8	7

Table 6: Element connectivity

Determine:

1. Global stiffness matrix [K]
2. Horizontal and vertical displacements at each node {U}
3. The reactions at nodes 1,4 and 7 {F}
4. The stresses in each element σ_x, σ_y, τ_{xy}
5. The principal stresses and principal angle for each element σ_1, σ_2, θ_p

```
K =

   1.0e+06 *

  Columns 1 through 12

    2.5962    0.9375   -1.5865   -0.0721         0         0    0.2885    0.0721   -1.2981   -0.9375         0         0
    0.9375    2.5962    0.0721    0.2885         0         0   -0.0721   -1.5865   -0.9375   -1.2981         0         0
   -1.5865    0.0721    5.1923         0   -1.5865   -0.0721   -1.2981    0.9375    0.5769         0   -1.2981   -0.9375
   -0.0721    0.2885         0    5.1923    0.0721    0.2885    0.9375   -1.2981         0   -3.1731   -0.9375   -1.2981
         0         0   -1.5865    0.0721    2.5962   -0.9375         0         0   -1.2981    0.9375    0.2885   -0.0721
         0         0   -0.0721    0.2885   -0.9375    2.5962         0         0    0.9375   -1.2981    0.0721   -1.5865
    0.2885   -0.0721   -1.2981    0.9375         0         0    5.1923         0   -3.1731         0         0         0
    0.0721   -1.5865    0.9375   -1.2981         0         0         0    5.1923         0    0.5769         0         0
   -1.2981   -0.9375    0.5769         0   -1.2981    0.9375   -3.1731         0    7.7885   -0.9375   -1.5865    0.0721
   -0.9375   -1.2981         0   -3.1731    0.9375   -1.2981         0    0.5769   -0.9375    7.7885   -0.0721    0.2885
         0         0   -1.2981   -0.9375    0.2885    0.0721         0         0   -1.5865   -0.0721    2.5962    0.9375
         0         0   -0.9375   -1.2981   -0.0721   -1.5865         0         0    0.0721    0.2885    0.9375    2.5962
         0         0         0         0         0         0    0.2885   -0.0721   -1.2981    0.9375         0         0
         0         0         0         0         0         0    0.0721   -1.5865    0.9375   -1.2981         0         0
         0         0         0         0         0         0   -1.2981   -0.9375    0.2885    0.0721         0         0
         0         0         0         0         0         0   -0.9375   -1.2981   -0.0721   -1.5865         0         0

  Columns 13 through 16

         0         0         0         0
         0         0         0         0
         0         0         0         0
         0         0         0         0
         0         0         0         0
         0         0         0         0
    0.2885    0.0721   -1.2981   -0.9375
   -0.0721   -1.5865   -0.9375   -1.2981
   -1.2981    0.9375    0.2885   -0.0721
    0.9375   -1.2981    0.0721   -1.5865
         0         0         0         0
         0         0         0         0
    2.5962   -0.9375   -1.5865    0.0721
   -0.9375    2.5962   -0.0721    0.2885
   -1.5865   -0.0721    2.5962    0.9375
    0.0721    0.2885    0.9375    2.5962
```

Figure 11: Global stiffness matrix for problem 2

The F global nodal force vector is:

```
F1x = 6.3994
F1y = 4.3354
F2x = 0.0000
F2y = 0.0000
F3x = 0.0000
F3y = 0.0000
F4x = -0.2988
F4y = 3.6296
F5x = 0.0000
F5y = -12.5000
```

```
F6x = 0.0000
F6y = 0.0000
F7x = -6.1006
F7y = 4.5350
F8x = 0.0000
F8y = 0.0000
```

The U global nodal displacement vector is:

```
U1x = 0.0000e+00
U1y = 0.0000e+00
U2x = -1.3961e-06
U2y = -3.5363e-06
U3x = -1.3286e-06
U3y = -5.4055e-06
U4x = 0.0000e+00
U4y = 0.0000e+00
U5x = 1.8083e-08
U5y = -4.2163e-06
U6x = 8.5585e-08
U6y = -5.1755e-06
U7x = 0.0000e+00
U7y = 0.0000e+00
U8x = 1.2021e-06
U8y = -3.0103e-06
```

The element nodal displacement vectors u_1, u_2 and u_3 have the stresses σ_x, σ_y, τ_{xy}:

```
For u1:
sigma_x = -730.1786
sigma_y = -504.6593
tau_xy = -1023.9005
For u2:
sigma_x = 0
sigma_y = -189.0038
tau_xy = 0
For u3:
sigma_x = 730.1786
sigma_y = 725.6064
tau_xy = -976.0995
```

Principal Stresses for σ_1, σ_2 and σ_3 and principal angle θ_p:

```
sigma1:
s1 = 412.6718
s2 = -1647.5098
Theta = 41.8577
sigma2:
```

```
s1 = 0
s2 = -189.0038
Theta = 0
sigma3:
s1 = 1703.9946
s2 = -248.2097
Theta = -44.9329
```

The developed code was validated further by applying it to a different problem. The code's results revealed amazing consistency with the reference book, confirming its dependability and correctness in generating accurate solutions.

3.2 Discussion

The fundamental product of the project in this dissertation is the code developed, which reflects the realisation of our objectives. The code can be found in its totality under Appendix A. While we have previously presented an overview of the methodology process of FEM and what was aspired to achieve, it is important to recognise that the code's complexities and details surpass the conciseness of the previous explanations. In order to clarify and understand the specifics of the code's functionality, the code's internal workings are examined in this part.

The user is first prompted by the code for input values needed for the FE problem as shown in Figures 1,2 and 3. Users are asked to provide the total number of elements and nodes as well as some material characteristics, such as Young's modulus, Poisson's ratio, and element thickness. The user is also required to enter coordinates for the element nodes the problem type (Plane Stress or Plane Strain). Another important prompt is presented to the user which is to input the node numbers for each element. Remember that the planar quad element has four nodes and the order of the nodes for each element is important. The direction must be anticlockwise. To make things easier, default values are offered.

Meanwhile the global stiffness matrix K is initialised in the code as a zero matrix. The number of nodes determines the size of this matrix, K_size. Additionally, the stiffness matrices for each individual element are created in a cell array called $K\_elements$.

The function enters a loop with the user's inputs to handle each element. This loop begins by gathering data from the input such as material characteristics, element connection (nodal information), and coordinates for every element.

Each finite element's stiffness matrix is determined by the code. It gets the strain-displacement matrix (B) from the differentials of the natural shape functions, as indicated in eq. 8. Calculations are also made for the Jacobian's determinate. Based on the type of problem the user entered, the material matrix is generated. The final element stiffness matrix is computed with MATLAB using eq. 17. The stiffness matrix of each element is then stored in a dynamic variable and the cell array $K\_elements$.

The method enters a second loop after getting stiffness matrices for each element to combine these stiffness matrices into the K global stiffness matrix. It adds contributions from each element to the relevant positions in K using the function $assembleStiffness$ (Kattan 2008) which is defined at the end of the code.

The global stiffness matrix K is displayed. This matrix serves as a representation of the complete system and includes details on the connections between each node in the structure.

The user must enter both known and unknown nodal displacements (U) and forces (F) using the code as shown in Figure 2. Users are prompted to enter values directly for known values, whereas variable names like U1x, U1y, etc. are offered as placeholders for unknown values.

In order to facilitate subsequent calculations, the user-provided data for nodal displacements and forces are processed and arranged into cell arrays. This is carried out for both U (displacements) and F (forces).The code then generates reduced vectors for nodal displacements (U) by deleting all known values, resulting in just the unknown displacements being selected. These reduced vectors are shown.

To solve the system of equations, the code employs the global stiffness matrix K and the known forces (F). The user is asked to enter the rows and columns from K that must be extracted based on the unknown values in U. The system of equations is solved using Gaussian elimination. To find the unknown displacements, the backslash operator (\) is used in MATLAB.

The code displays the unknown displacements derived from solving the equation system. Variables in the workspace are assigned to the unknown displacements for the next calculations.

Subsequently, the code then enters a post-processing step in which it computes stresses for each element. It initially collects nodal displacement vectors before calculating stresses based on the material characteristics and geometry of the element. For readability, the code rounds small numbers to zero. Then it computes the principal stresses and angles for each element with the function *BilinearQuadElementPStresses* (Kattan 2008). The results are displayed with small numbers rounded to zero for clarity.

The actual code provided in Appendix A is commented so the user has better understanding on what is happening in each step. Having explained the code, the rest of the discussion will be organised around a few important matters.

3.2.1 Code Development and Unification

The development of a code for Finite Element specifically designed for planar quad elements was the foundational aspect of this dissertation topic. The code's conception was founded on a thorough analysis of the various publications listed throughout this dissertation. The aim was to construct a single, user-friendly MATLAB-based code that could be executed to efficiently address a variety of structural mechanics issues by combining these multiple sources.

3.2.2 Flexibility and Efficiency in Computational Methods

There are many different ways to approach a problem in the world of coding. Although the code is fully operational and capable of producing accurate results, it is critical to recognise that there are other functions, approaches or programming languages where this code can improve. As a result, even at this point, where the code is functional, there are still opportunities for refinement and optimisation. Finding and applying these improvements could make the code even more user-friendly and efficient in the future.

3.2.3 The code's accessibility and educational value

The potential of the generated FE code to be used as a teaching aid is one noteworthy feature. Recognising that not all users will be familiar with MATLAB or the Finite Element method for dealing with planar quad elements, the code was designed to provide insight into the underlying processes. Careful study of the code can give a coherent picture of the problem-solving steps involved, even to a student or rookie.

3.2.4 Combining Various Methodologies

Despite the fact that the FEM is a method that is well known and frequently used in the field of structural mechanics, it is important to emphasise how adaptable and versatile the method is. The majority of FEM materials offer a broad framework for problem-solving, however when using this approach with various element types, variations appear. In this dissertation approaches and insights were gathered from a variety of literature sources and combined into a single tool designed specifically for planar quad components. By doing this, we hoped to offer a comprehensive body of information that could be used for both academic and practical purposes. While the fundamental mathematics stay constant, the interpretations and explanations offered in books and papers change dramatically. Therefore, it seems like a good idea to compile this knowledge into a single, easily available resource that is aimed towards a level that is more understandable for students.

3.2.5 Conclusion

This dissertation, in retrospect, shows a thorough journey through the field of finite element analysis and the resolving of structural problems. The first objective was to create a customised MATLAB code for planar quadrilateral elements (starting with one element), which was successfully created. The challenge did not end there, though; it continued to include the creation of a global stiffness matrix combining more than one element, which enabled us to take on more complex structural problems. The dynamic analysis was remarkably precise thanks to the integration of numerical techniques like Gaussian integration.

The ability to turn our code into a user-friendly, interactive interface is one of the biggest accomplishments of this project. This change makes it a helpful educational tool for anyone attempting to understand the complexity of finite element method. Furthermore, careful consideration was given to readability and comprehension by including detailed code documentation.

It's important to note that people interacting with this code should ideally have a basic understanding of finite element methods due to the technical nature of the subject matter. This precondition is particularly important in parts where users are asked to provide rows and columns for the global stiffness matrix's submatrix extraction (e.g., % Prompt user for the rows and columns to extract sub-matrix using inputdlg). Although the code does shed light on how the U vector is built, users who are already familiar with matrices will be better able to identify the precise rows and columns required for extraction.

Finally, this work demonstrates the limitless possibilities of finite element methodologies. It not only fills the gap between theory and application in the real world, but it also encourages creativity in engineers, researchers, and students who are faced with difficult structural problems.

4 Discussion and future work

This chapter's conclusion serves as a summary analysis of the findings presented throughout the report. As expected, our main objective was accomplished; the created code correctly applies the finite element method to a wide range of planar quad element problems. This success demonstrates the code's usefulness in real world applications.

In the future, there is room for improvement. While the current code successfully communicates with users via dialogue boxes, the logical next step is to develop a specific MATLAB application. This would add to the code's strong mathematical base by providing a more user-friendly and visually appealing interface.

The automated partitioning stage is another area that could use improvement. The search for simpler, automated methods should nevertheless be pursued even though this currently requires user involvement. It is an opportunity that could further improve the code's effectiveness and usability even though it is not yet effectively implemented.

We also need to take into account the exciting possibilities of languages like Julia or Python. These languages are attractive choices for future development because they offer a flexible toolkit for creating interactive and user-friendly applications. Compared to MATLAB, their open-source nature provides global access, and their automation capabilities further increase their attractiveness.

The importance of giving consumers a greater knowledge of the underlying mathematics cannot be overstated, even though it is outside the scope of this study. By providing understanding of the intricate nature of computation, this might greatly increase the code's value and make it an even more useful teaching tool. The field's continual evolution is reflected in these potential developments, creating the opportunity for constant growth and enhancement.

References

Akin, J. E. (1994), *Finite elements for analysis and design*, Computational mathematics and applications, Academic Press, London ;. Section: xi, 548 pages : illustrations ; 24 cm + 1 computer disc (3 1/2 in.).

Argyris, J. & Kelsey, S. (1954), 'Energy Theorems and Structural Analysis', *Aircraft Engineering and Aerospace Technology* **26**(12), 410–422. Publisher: MCB UP Ltd.
URL: *https://doi.org/10.1108/eb032502*

Buchanan, G. R. (1995), *Schaum's outline of theory and problems of finite element analysis*, Schaum's outline series, McGraw-Hill, New York. Section: viii, 264 pages : illustrations ; 28 cm.
URL: *http://catdir.loc.gov/catdir/toc/mh022/94011362.html*

Chessa, J. (2002), 'Programing the Finite Element Method with Matlab'.

Clough, R. W. (1960), 'The Finite Element Method in Plane Stress Analysis'.

Courant, R. (1943), 'Variational methods for the solution of problems of equilibrium and vibrations', *Bulletin of the American Mathematical Society* **49**, 1–23.

Ferreira, A. (2008), *MATLAB Codes for Finite Element Analysis: Solids and Structures*, 1st edn, Springer Publishing Company, Incorporated.

Fish, J. & Belytschko, T. (2007), *A first course in finite elements*, John Wiley, Hoboken, NJ. Section: xiv, 319 p. : ill. (some col.) ; 25 cm.+ 1 computer optical disc (4 3/4 in.).
URL: *http://catdir.loc.gov/catdir/enhancements/fy0741/2007298648-b.html*

Hrennikoff, A. (1941), 'Solution of Problems of Elasticity by the Framework Method', *J. Appl. Mech.* .
URL: *https://cir.nii.ac.jp/crid/1570854175168115072*

Hutton, D. V. (2004), *Fundamentals of finite element analysis.*, 1st ed. edn, McGraw-Hill Higher Education, Boston. Section: xiv, 494 pages : illustrations ; 24 cm.

Jung, J., Jun, H. & Lee, P.-S. (2022), 'Self-updated four-node finite element using deep learning', *Computational Mechanics* **69**(1), 23–44.
URL: *https://doi.org/10.1007/s00466-021-02081-7*

Kattan, P. (2008), *MATLAB guide to finite elements: An interactive approach*, Springer.

Khennane, A. (2013), 'Introduction to finite element analysis using MATLAB and Abaqus', *Introduction to Finite Element Analysis Using MATLAB and Abaqus* . ISBN: 9781466580213.

Lubarda, M. V. & Lubarda, V. A. (2020), Two–Dimensional Problems of Elasticity, *in* 'Intermediate Solid Mechanics', Cambridge University Press, Cambridge, pp. 143–167.
URL: *https://www.cambridge.org/core/books/intermediate-solid-mechanics/twodimensional-problems-of-elasticity/5527EC009BF5F8D61CC482EA22E71C9D*

Nazaruddin, N. & Siallagan, R. (2021), 'Software Engineering Development of Finite Element Method Programming Applications in 2D Frame Structures Using Python Programs', *Journal of Physics: Conference Series* **2049**, 012031.

Obumneme, A. E., Chukwuebuka, A. O., Chukwudi, N. N., Oscar, N. C., Nelson, O. C., Keji, A. H., Ibeamaka, O. M., Sunday, O., Don-Ugbaga, C. & Ezeokpube, G. C. (2022), 'Finite Element Analysis of Continuous Plates Using a High-Performance Programming Language (MATLAB)', *Path of Science* .

Ortigoza, G. & Ponce De La Cruz Herrera, R. I. (2023), 'Resolviendo ecuaciones diferenciales ordinarias con Symbolic Math Toolbox™ (Matlab) y SymPy (Python)', *Revista Mexicana de Física E* **20**(2 Jul-Dec).
URL: *https://rmf.smf.mx/ojs/index.php/rmf-e/article/view/7012*

Pan, J., Huang, J., Wang, Y., Cheng, G. & Zeng, Y. (2021), 'A self-learning finite element extraction system based on reinforcement learning', *AI EDAM* **35**(2), 180–208. Edition: 2021/04/21 Publisher: Cambridge University Press.
URL: *https://www.cambridge.org/core/article/selflearning-finite-element-extraction-system-based-on-reinforcement-learning/75000DF8A16544C462457A78E1C12853*

Perumal, L. & Mon, D. T. T. (2011), 'Finite Elements for Engineering Analysis: A Brief Review'.

Qianwei, D., Yi, L., Zhang, B., Feng, D.-S., Wang, X. & Yin, X. (2019), 'A practical adaptive moving-mesh algorithm for solving unconfined seepage problem with Galerkin finite element method', *Scientific Reports* **9**, 6988.

Rao, S. S. . (2018), *The finite element method in engineering*, sixth edition. edn, Butterworth-Heinemann, an imprint of Elsevier, Kidlington, Oxford, United Kingdom.
URL: *https://www.sciencedirect.com/science/book/9780128117682*

Rees, 1947, D. W. A. D. W. A. (1997), *Basic solid mechanics*, Macmillan, Houndmills, Basingstoke, Hampshire. Section: xii, 396 pages : illustrations ; 24 cm.

Reis, J. P. C. d. & Júnior, P. A. A. M. (2018), 'Introduction to the Method of Finite Elements by a balance Sheet Problem: A Simplification for an Initial understanding of the Method', *International Journal of Advanced Engineering Research and Science* **5**, 1–4.
URL: *https://api.semanticscholar.org/CorpusID:59062596*

Selleri, S. (2022), 'A Brief History of Finite Element Method and Its Applications to Computational Electromagnetics', *The Applied Computational Electromagnetics Society Journal (ACES)* .

Simon-Marinica, Adrian Bogdan, Vlasin, Nicolae-Ioan, Manea, Florin & Florea, Gheorghe-Daniel (2021), 'Finite element method to solve engineering problems using ansys', *MATEC Web Conf.* **342**, 01015.
URL: *https://doi.org/10.1051/matecconf/202134201015*

Taig, I. & Kerr, R. (1964), 'Some problems in the discrete element representation of aircraft structures', *Matrix methods of structural analysis* pp. 267–315. Publisher: Pergamon Press London.

Turner, M. J., Clough, R. W., Martin, H. C. & Topp, L. J. (1956), 'Stiffness and Deflection Analysis of Complex Structures', *Journal of the Aeronautical Sciences (Institute of the Aeronautical Sciences)* **23**(9), 805–823.
URL: *https://app.dimensions.ai/details/publication/pub.1023945723*

Zienkiewicz, O. C. (1977), *The Finite Element Method (3rd edn)*, Vol. 60, McGraw-Hill, New York.

Zienkiewicz, O. C., Taylor, 1934, R. L. R. L. & Zhu, J. Z. (2013), *The finite element method : its basis and fundamentals*, seventh edition. edn, Butterworth-Heinemann, Oxford, UK ;.
URL: *http://www.books24x7.com/marc.asp?bookid=56579*

A Appendix - Code

The following code sets up and solves a FE problem for a 2D structure with user defined material properties, element connectivity, nodal displacements, and forces.

```
% Prompt user for both inputs in a single dialog box
prompt = {'\fontsize{13}Enter the number of elements:', '\fontsize{13}Enter the number of nodes:'};
dlgtitle = 'Input';
dims = [1 45];
definput = {'', ''}; % Default values (empty)
opts.Interpreter = 'tex';
answer = inputdlg(prompt, dlgtitle, dims, definput,opts);

if isempty(answer)
    error('User canceled the input.');
end

% Extract the inputs
num_elements = str2double(answer{1});
num_nodes = str2double(answer{2});

% Calculate the size of the global stiffness matrix K
K_size = 2 * num_nodes;

% Initialize the global stiffness matrix K as a zero matrix
K = zeros(K_size, K_size);

% Initialize a cell array to store stiffness matrices for each element
K_elements = cell(num_elements, 1);

% Loop through each element
for element_idx = 1:num_elements
    % Input for the current element
    prompt = ["\fontsize{13}Enter young's modulus value for element " + num2str(element_idx), ...
              "\fontsize{13}Enter poisson's ratio for element " + num2str(element_idx), ...
              "\fontsize{13}Enter element thickness for element " + num2str(element_idx), ...
              "\fontsize{13}What type of problem is it? (Plane Stress or Plane Strain) for element " + num2str(element_idx), ...
              "\fontsize{13}Enter the coordinate positions for element " + num2str(element_idx) + ": "];
    dims = [1 55];
    definput = {'210e6', '0.3', '0.025', 'Plane Stress', '
```

```
        0;0;0.25;0;0.25;0.25;0;0.25'};
36      opts.Interpreter = 'tex';
37      answer = inputdlg(prompt, dlgtitle, dims, definput, opts);
38
39      if isempty(answer)
40          error('User canceled the input.');
41      end
42
43      % Extract input values from the answer cell aray
44      E = str2double(answer{1});
45      nu = str2double(answer{2});
46      t = str2double(answer{3});
47      problem_type = answer{4};
48      P = str2num(answer{5}); %#ok<ST2NM> % Convert the string to a numeric
            vector
49
50       % Prompt the user for element connectivity (nodes i, j, m, and n)
51  %     connectivity_prompt = 'Enter the node numbers (i, j, m, n) for
        element:';
52  %     connectivity_answer = inputdlg(connectivity_prompt, dlgtitle, dims);
53      connectivity_prompt = {'\fontsize{13}Enter the node numbers (i, j, m,
            n) for element:'};
54      dlgtitle = 'Input';
55      dims = [1 55];
56      definput = {''}; % Default values (empty)
57      opts.Interpreter = 'tex';
58      connectivity_answer = inputdlg(connectivity_prompt, dlgtitle, dims,
            definput,opts);
59
60      if isempty(connectivity_answer)
61          error('User canceled the input.');
62      end
63
64      % Extract node numbers from the connectivity answer and convert to
            numeric values
65      node_inputs = str2num(connectivity_answer{1}); %#ok<ST2NM>
66
67      % Check if the number of inputs is valid
68      if numel(node_inputs) ~= 4
69          error('Invalid input for node numbers. Please provide four node
                numbers.');
70      end
71
72      % Assign node numbers to individual variables
73      i_node = node_inputs(1);
```

```
    j_node = node_inputs(2);
    m_node = node_inputs(3);
    n_node = node_inputs(4);

    % Store the connectivity information in the element_connectivity
       matrix
    element_connectivity(element_idx, :) = [i_node, j_node, m_node, n_node
       ];

    % element_connectivity(1, :) to access the nodes for each element

    % Assign nodal positions to individual variables
    x1 = P(1);
    y1 = P(2);
    x2 = P(3);
    y2 = P(4);
    x3 = P(5);
    y3 = P(6);
    x4 = P(7);
    y4 = P(8);

    % Display the input values
    fprintf("Young's Modulus: %.2f Pa\n",E);
    fprintf("Poisson's Ratio: %.2f\n",nu);
    fprintf("Element thickness: %.4f meters\n",t);
    fprintf("Problem Type: %s\n", problem_type);
    fprintf('Coordinate Positions:\n');
    fprintf('x1 = %.3f, y1 = %.3f\n', x1, y1);
    fprintf('x2 = %.3f, y2 = %.3f\n', x2, y2);
    fprintf('x3 = %.3f, y3 = %.3f\n', x3, y3);
    fprintf('x4 = %.3f, y4 = %.3f\n', x4, y4);


    % Determining the element stiffness matrix for a quadrilateral element
    % using natural shape functions (element edges aligned with axes)

    % The natural shape functions for a bilinear four node quadrilateral
    % element are:
    syms xi eta;
    N1 = (1-xi)*(1-eta)/4;
    N2 = (1+xi)*(1-eta)/4;
    N3 = (1+xi)*(1+eta)/4;
    N4 = (1-xi)*(1+eta)/4;

    % Find Strain Displacement Matrix (B)
```

```
    x = N1*x1 + N2*x2 + N3*x3 + N4*x4;
    y = N1*y1 + N2*y2 + N3*y3 + N4*y4;

    % These need to be differentiated with respect to xi and eta:
    xxi = diff(x,xi);
    xeta = diff(x,eta);
    yxi = diff(y,xi);
    yeta = diff(y,eta);

    % The determinate of the Jacobian (Det(J)) is given by:
    J = xxi*yeta - yxi*xeta;

    % The differentials of N1,N2,N3,N4 with respect to xi and eta are
       given by:
    N1xi = diff(N1,xi);
    N2xi = diff(N2,xi);
    N3xi = diff(N3,xi);
    N4xi = diff(N4,xi);
    N1eta = diff(N1,eta);
    N2eta = diff(N2,eta);
    N3eta = diff(N3,eta);
    N4eta = diff(N4,eta);

    % The strain displacement matrix is given by:
    B11 = yeta*N1xi - yxi*N1eta;
    B12 = 0;
    B13 = yeta*N2xi - yxi*N2eta;
    B14 = 0;
    B15 = yeta*N3xi - yxi*N3eta;
    B16 = 0;
    B17 = yeta*N4xi - yxi*N4eta;
    B18 = 0;
    B21 = 0;
    B22 = xxi*N1eta - xeta*N1xi;
    B23 = 0;
    B24 = xxi*N2eta - xeta*N2xi;
    B25 = 0;
    B26 = xxi*N3eta - xeta*N3xi;
    B27 = 0;
    B28 = xxi*N4eta - xeta*N4xi;
    B31 = xxi*N1eta - xeta*N1xi;
    B32 = yeta*N1xi - yxi*N1eta;
    B33 = xxi*N2eta - xeta*N2xi;
    B34 = yeta*N2xi - yxi*N2eta;
    B35 = xxi*N3eta - xeta*N3xi;
```

```
    B36 = yeta*N3xi - yxi*N3eta;
    B37 = xxi*N4eta - xeta*N4xi;
    B38 = yeta*N4xi - yxi*N4eta;
    B = [B11 B12 B13 B14 B15 B16 B17 B18 ;
       B21 B22 B23 B24 B25 B26 B27 B28 ;
       B31 B32 B33 B34 B35 B36 B37 B38];

    % Perform calculations based on the problem type
    % Get the material matrix:
    % Plane Stress
    if strcmpi(problem_type, 'Plane Stress')
       D = (E/(1-nu*nu))*[1, nu, 0 ; nu, 1, 0 ; 0, 0, (1-nu)/2];
    % Plane Strain
    elseif strcmpi(problem_type, 'Plane Strain')
       D = (E/(1+nu)/(1-2*nu))*[1-nu, nu, 0 ; nu, 1-nu, 0 ; 0, 0, (1-2*nu)
          /2];
    end

    % The final element stiffness matrix when integrated becomes:
    BD = transpose(B)*D*B/J;
    r = int(int(BD, eta, -1, 1), xi, -1, 1);
    z = t*r;
    w = double(z);

    % Store the stiffness matrix in a dynamic variable (k1, k2, k3, etc.)
    k_name = ['k', num2str(element_idx)]; % Generate the variable name
    eval([k_name, ' = w;']); % Assign the stiffness matrix to the variable

    % Store the stiffness matrix in the cell array
    K_elements{element_idx} = w;
end

% You now have stiffness matrices for all elements in K_elements cell
   array
% Access them using K_elements{element_idx}

% Now you have stiffness matrices for all elements in K_elements cell
   array


% Loop through each element to compute and assemble the stiffness matrix
for element_idx = 1:num_elements
    % Compute the stiffness matrix for the current element (k1, k2, etc.)
    k_name = ['k', num2str(element_idx)];
    k = eval(k_name); % Retrieve k1, k2, etc.
```

```
203
204     % Retrieve element connectivity information (I, j, m, n) for the
            current element
205     i = element_connectivity(element_idx, 1);
206     j = element_connectivity(element_idx, 2);
207     m = element_connectivity(element_idx, 3);
208     n = element_connectivity(element_idx, 4);
209
210     % Call the assembleStiffness function to update the global stiffness
            matrix K
211     K = assembleStiffness(K, k, i, j, m, n);
212 end
213
214 % Now, K contains the assembled global stiffness matrix
215 % Display the global stiffness matrix K
216 disp('Global Stiffness Matrix K:');
217 disp(K);
218
219 % Prompt user for known and unknown displacements (U) and forces (F) for
        each node
220 U = cell(2 * num_nodes, 1); % Initialize U as a cell array
221 F = cell(2 * num_nodes, 1); % Initialize F as a cell array
222
223 for node_idx = 1:num_nodes
224     % Prompt user for displacements Ux and Uy for each node
225     prompt = {['\fontsize{13}Enter ' 'U' num2str(node_idx) 'x' ' :'], ...
226               ['\fontsize{13}Enter ' 'U' num2str(node_idx) 'y' ' :'], ...
227               ['\fontsize{13}Enter ' 'F' num2str(node_idx) 'x' ' :'], ...
228               ['\fontsize{13}Enter ' 'F' num2str(node_idx) 'y' ' :']};
229     dims = [1 45];
230     definput = {['U' num2str(node_idx) 'x'], ['U' num2str(node_idx) 'y'],
            ...
231                 ['F' num2str(node_idx) 'x'], ['F' num2str(node_idx) 'y']};
232     opts.Interpreter = 'tex';
233     answer = inputdlg(prompt, dlgtitle, dims, definput, opts);
234
235     if isempty(answer)
236         error('User canceled the input.');
237     end
238
239     % Extract input values from the answer cell array
240     Ux = answer{1};
241     Uy = answer{2};
242     Fx = answer{3};
243     Fy = answer{4};
```

```
244
245
246     % Save the input values as variables
247     assignin('base', ['U' num2str(node_idx) 'x'], Ux);
248     assignin('base', ['U' num2str(node_idx) 'y'], Uy);
249     assignin('base', ['F' num2str(node_idx) 'x'], Fx);
250     assignin('base', ['F' num2str(node_idx) 'y'], Fy);
251
252     % Update the global nodal displacement vector (U) and force vector (F)
253     U{2 * node_idx - 1} = Ux;
254     U{2 * node_idx} = Uy;
255     F{2 * node_idx - 1} = Fx;
256     F{2 * node_idx} = Fy;
257 end
258
259 % Now, U contains the global nodal displacement vector and F contains the
        global nodal force vector
260 % Display the global nodal displacement vector and force vector
261 fprintf('Global Nodal Displacement Vector (U):\n');
262 num_nodes = numel(U) / 2;
263 for i = 1:num_nodes
264     node_idx = ceil(i);
265
266     % Display x component of displacement
267     fprintf('Node %dx: %s\n', node_idx, U{2*i - 1});
268
269     % Display y component of displacement
270     fprintf('Node %dy: %s\n', node_idx, U{2*i});
271 end
272
273 fprintf('Global Nodal Force Vector (F):\n');
274 for i = 1:num_nodes
275     node_idx = ceil(i);
276
277     % Display x component of force
278     fprintf('Node %dx: %s\n', node_idx, F{2*i - 1});
279
280     % Display y component of force
281     fprintf('Node %dy: %s\n', node_idx, F{2*i});
282 end
283
284 % Initialize a reduced U cell array
285 reduced_U = {};
286
287 % Iterate through the U vector
```

```
for i = 1:numel(U)
    % Check if the element starts with 'U' (indicating it's an unknown
       variable)
    if strncmp(U{i}, 'U', 1)
        reduced_U{end+1} = U{i};
    end
end

% transpose the reduced_U
reduced_U = reduced_U';

% Display the transposed reduced U vector
disp(reduced_U);

% Initialize the reduced force vector
reduced_F = [];

% Iterate through the F cell array
for i = 1:numel(F)
    % Check if the element is a numeric string
    if ~isnan(str2double(F{i}))
        % Convert the numeric string to a double and append to reduced_F
        reduced_F(end+1) = str2double(F{i});
    end
end

% Display the reduced_F vector
disp('Reduced Force Vector (F):');
disp(reduced_F);

% Prompt user for the rows and columns to extract sub-matrix using
   inputdlg
prompt_rows = {'Enter rows (e.g., 3:6,9:12):'};
prompt_cols = {'Enter columns (e.g., 3:6,9:12):'};
dlgtitle = 'Enter Row and Column Indices';
dims = [1 60];
definput = {'3:6,9:12', '3:6,9:12'}; % Default values (example)
opts.Interpreter = 'tex';

row_indices_str = inputdlg(prompt_rows, dlgtitle, dims, definput, opts);
col_indices_str = inputdlg(prompt_cols, dlgtitle, dims, definput, opts);

if isempty(row_indices_str) || isempty(col_indices_str)
    error('User canceled the input.');
end
```

```
331
332 % Convert the user input strings to numeric row and column indices
333 row_indices = eval(['[' row_indices_str{1} ']']);
334 col_indices = eval(['[' col_indices_str{1} ']']);
335
336 % Extract the sub-matrix from K based on the specified row and column
    indices
337 submatrix = K(row_indices, col_indices);
338
339 % Display the extracted sub-matrix
340 fprintf('Extracted Sub-Matrix from K:\n');
341 disp(submatrix);
342
343 % Solve the system of equations u = submatrix\reduced_F (gaussian
344 % elimination with MATLAB) The backlash operator "\" is used for that in
    MATLAB
345 % I transpose reduced_F to match the sizes of the matrices
346 u = submatrix \ reduced_F';
347
348
349 % show the solution for the system of equations
350 fprintf('The unknown displacements in reduced_U are:\n');
351
352 for i = 1:numel(reduced_U)
353     fprintf('%s = %.4e\n', reduced_U{i}, u(i));
354
355     % Create variables with the respective results
356     variable_name = reduced_U{i};
357     assignin('base', variable_name, u(i));
358 end
359
360
361 % Post processing
362
363 % Initialize a cell array to store the global nodal displacement vector U
364 U = cell(2 * num_nodes, 1);
365
366 % Loop through each node to construct the global nodal displacement vector
    U
367 for node_idx = 1:num_nodes
368     % Use evalin to fetch the known values for Ux and Uy from the
        workspace
369     Ux = evalin('base', ['U' num2str(node_idx) 'x']);
370     Uy = evalin('base', ['U' num2str(node_idx) 'y']);
371
```

```
    % Create cell array entries for Ux and Uy
    U{2 * node_idx - 1} = (Ux);
    U{2 * node_idx} = (Uy);
end

% Now, U contains the complete global nodal displacement vector
% Display the global nodal displacement vector
fprintf('Global Nodal Displacement Vector (U):\n');
num_nodes = numel(U) / 2;
for i = 1:num_nodes
    node_idx = ceil(i);

    % Display x component of displacement
    fprintf('Node %dx: %s\n', node_idx, U{2*i - 1});

    % Display y component of displacement
    fprintf('Node %dy: %s\n', node_idx, U{2*i});
end

% Initialize the numeric array
U_numeric = zeros(size(U));

% Loop through each element in the cell array
for i = 1:numel(U)
    % Check if it's a numeric value or a cell
    if isnumeric(U{i})
        U_numeric(i) = U{i};
    elseif iscell(U{i})
        % Assuming there's only one element in the cell
        inner_value = U{i}{1};

        if isnumeric(inner_value)
            U_numeric(i) = inner_value;
        else
            % Convert the string representation to a numeric value
            numeric_str = regexprep(inner_value, '[^\d.eE+-]', '');
            U_numeric(i) = str2double(numeric_str);
        end
    end
end

% Now, U_numeric should correctly represent numeric values

% Get unkonw forces with F = K*U
F = K*U_numeric;
```

```
417
418
419 % Define threshold for values close to zero
420 threshold = 1e-10;
421
422 % Define the unknown forces cell array
423 unknown_forces = cell(num_nodes, 2);
424
425 % Loop through each node to display and save the unknown forces
426 fprintf('The F global nodal force vector is:\n');
427 for node_idx = 1:num_nodes
428     force_x = F(2 * node_idx - 1);
429     force_y = F(2 * node_idx);
430
431     % Check if the values are close to zero and set them to zero
432     if abs(force_x) < threshold
433         force_x = 0;
434     end
435     if abs(force_y) < threshold
436         force_y = 0;
437     end
438
439     % Display and save the values to the workspace
440     fprintf('F%dx = %.4f\n', node_idx, force_x);
441     fprintf('F%dy = %.4f\n', node_idx, force_y);
442
443     % Create variables with the respective results and save them to the
        workspace
444     variable_name_x = ['F' num2str(node_idx) 'x'];
445     variable_name_y = ['F' num2str(node_idx) 'y'];
446     assignin('base', variable_name_x, force_x);
447     assignin('base', variable_name_y, force_y);
448
449     % Store the values in the unknown_forces cell array
450     unknown_forces{node_idx, 1} = variable_name_x;
451     unknown_forces{node_idx, 2} = variable_name_y;
452 end
453
454 % Now you have displayed, saved, and stored the unknown forces
455
456 % Loop through each node to display and save the unknown displacements
457 fprintf('The U global nodal displacement vector is:\n');
458 for node_idx = 1:num_nodes
459     disp_x = U_numeric(2 * node_idx - 1);
460     disp_y = U_numeric(2 * node_idx);
```

```
461
462     % Display and save the values to the workspace
463     fprintf('U%dx = %.4e\n', node_idx, disp_x);
464     fprintf('U%dy = %.4e\n', node_idx, disp_y);
465
466     % Create variables with the respective results and save them to the
            workspace
467     variable_name_x = ['U' num2str(node_idx) 'x'];
468     variable_name_y = ['U' num2str(node_idx) 'y'];
469     assignin('base', variable_name_x, disp_x);
470     assignin('base', variable_name_y, disp_y);
471
472     % Store the values in the unknown_disps cell array
473     unknown_disps{node_idx, 1} = variable_name_x;
474     unknown_disps{node_idx, 2} = variable_name_y; %#ok<*SAGROW>
475 end
476
477
478 % Find the stresses in each element setting the nodal displacement vectors
479 % to calculate the stresses sigman
480
481 % We already have the number of elements in the variable in num_elements
482 % Create nodal displacement vectors for each element
483
484 disp('The element nodal displacement vectors are: ');
485 % Determine the number of rows in element_connectivity
486 num_rows = size(element_connectivity, 1);
487
488 % Initialize a cell array to store the u vectors for each row
489 u_cell = cell(1, num_rows);
490
491 % Iterate through each row of element_connectivity
492 for row_to_extract = 1:num_rows
493     % Extract the node numbers for the current row
494     nodes = element_connectivity(row_to_extract, :);
495
496     % Initialize the u vector for the current row
497     u_row = zeros(2 * numel(nodes), 1);
498
499     % Construct the variable names and populate u_row
500     for i = 1:numel(nodes)
501         node_number = nodes(i);
502         % Construct the variable names based on node_number (e.g., U1x,
                U1y, U2x, U2y, etc.)
503         variable_name_x = ['U', num2str(node_number), 'x'];
```

```
        variable_name_y = ['U', num2str(node_number), 'y'];
        % Get the values from the workspace
        u_x = evalin('base', variable_name_x);
        u_y = evalin('base', variable_name_y);
        % Assign the values to u_row
        u_row(2 * i - 1) = u_x;
        u_row(2 * i) = u_y;
    end

    % Store the u vector for the current row in the cell array
    u_cell{row_to_extract} = u_row;

    % Assign u_row to individual variables (u1,u2,etc.)
    assignin('base',['u',num2str(row_to_extract)], u_row);

    % Display the u vector for the current row
    disp(['u', num2str(row_to_extract), ':']);
    disp(u_row);
end
% Now you have individual variables for each element nodal displacement
% vector
% Obtain the element stress vector for each element
syms xi eta;
a = (y1*(xi-1)+y2*(-1-xi)+y3*(1+xi)+y4*(1-xi))/4;
b = (y1*(eta-1)+y2*(1-eta)+y3*(1+eta)+y4*(-1-eta))/4;
c = (x1*(eta-1)+x2*(1-eta)+x3*(1+eta)+x4*(-1-eta))/4;
d = (x1*(xi-1)+x2*(-1-xi)+x3*(1+xi)+x4*(1-xi))/4;
B1 = [a*(eta-1)/4-b*(xi-1)/4 0 ; 0 c*(xi-1)/4-d*(eta-1)/4 ;
   c*(xi-1)/4-d*(eta-1)/4 a*(eta-1)/4-b*(xi-1)/4];
B2 = [a*(1-eta)/4-b*(-1-xi)/4 0 ; 0 c*(-1-xi)/4-d*(1-eta)/4 ;
   c*(-1-xi)/4-d*(1-eta)/4 a*(1-eta)/4-b*(-1-xi)/4];
B3 = [a*(eta+1)/4-b*(xi+1)/4 0 ; 0 c*(xi+1)/4-d*(eta+1)/4 ;
   c*(xi+1)/4-d*(eta+1)/4 a*(eta+1)/4-b*(xi+1)/4];
B4 = [a*(-1-eta)/4-b*(1-xi)/4 0 ; 0 c*(1-xi)/4-d*(-1-eta)/4 ;
   c*(1-xi)/4-d*(-1-eta)/4 a*(-1-eta)/4-b*(1-xi)/4];
Bfirst = [B1 B2 B3 B4];
Jfirst = [0 1-eta eta-xi xi-1 ; eta-1 0 xi+1 -xi-eta ;
   xi-eta -xi-1 0 eta+1 ; 1-xi xi+eta -eta-1 0];
J = [x1 x2 x3 x4]*Jfirst*[y1 ; y2 ; y3 ; y4]/8;
B = Bfirst/J;
if strcmpi(problem_type, 'Plane Stress')
   D = (E/(1-nu*nu))*[1, nu, 0 ; nu, 1, 0 ; 0, 0, (1-nu)/2];
elseif strcmpi(problem_type, 'Plane Strain')
   D = (E/(1+nu)/(1-2*nu))*[1-nu, nu, 0 ; nu, 1-nu, 0 ; 0, 0, (1-2*nu)/2];
end
```

```
549
550 %%
551 tolerance = 1e-10;
552 for i = 1:num_rows
553     % Create variable names like 'u1', 'u2', etc.
554     u_name = ['u' num2str(i)];
555
556     % Check if the variable exists in the workspace
557     if evalin('base', ['exist(''' u_name ''', ''var'')'])
558         % If the variable exists, assign it to 'u' and calculate 'wcent'
559         u = evalin('base', u_name);
560         w = D * B * u;
561
562         % Calculate and save the result in sigma1, sigma2, etc.
563         wcent = subs(w, {xi, eta}, {0, 0});
564         sigma_name = ['sigma' num2str(i)];
565          assignin('base', sigma_name, double(wcent));
566         sigma_values = double(wcent);
567
568          % Round values close to zero to zero
569         sigma_values(abs(sigma_values) < tolerance) = 0;
570
571         % Create variables for sigma_x, sigma_y, and tau_xy
572         sigma_x = sigma_values(1);
573         sigma_y = sigma_values(2);
574         tau_xy = sigma_values(3);
575
576         % Display the variable names and values
577         disp(['For ' u_name ':']);
578         disp(['sigma_x = ' num2str(sigma_x)]);
579         disp(['sigma_y = ' num2str(sigma_y)]);
580         disp(['tau_xy = ' num2str(tau_xy)]);
581     else
582         disp(['Variable ' u_name ' not found in the workspace.']);
583     end
584 end
585
586 % Calculate the principal stresses and principal angle for each element
587 for i = 1:num_rows
588     % Create variable names like 'sigma1', 'sigma2', etc.
589     sigma_name = ['sigma' num2str(i)];
590
591     % Check if the variable exists in the workspace
592     if evalin('base', ['exist(''' sigma_name ''', ''var'')'])
593         % If the variable exists, retrieve its value
```

```
        sigma = evalin('base', sigma_name);

        % Calculate principal stresses and angle using the function
        principal_results = BilinearQuadElementPStresses(sigma);

        % Round values close to zero to zero
        principal_results(abs(principal_results) < tolerance) = 0;

        % Save the results in variables s1, s2, theta, etc.
        s1 = principal_results(1);
        s2 = principal_results(2);
        theta = principal_results(3);

        % Check if the absolute value is less than the tolerance
        if abs(s1) < tolerance
            s1 = 0;
        end
        if abs(s2) < tolerance
            s2 = 0;
        end
        if abs(theta) < tolerance
            theta = 0;
        end

        % Display the results
        disp(['Principal Stresses for ' sigma_name ':']);
        disp(['s1 = ' num2str(s1)]);
        disp(['s2 = ' num2str(s2)]);
        disp(['Theta = ' num2str(theta)]);
    else
        disp(['Variable ' sigma_name ' not found in the workspace.']);
    end
end

% Define a function to assemble the element stiffness matrix k into the
   global stiffness matrix K
function K = assembleStiffness(K, k, i, j, m, n)
    % Update the global stiffness matrix using element stiffness matrix k
    K(2*i-1,2*i-1) = K(2*i-1,2*i-1) + k(1,1);
    K(2*i-1,2*i) = K(2*i-1,2*i) + k(1,2);
    K(2*i-1,2*j-1) = K(2*i-1,2*j-1) + k(1,3);
    K(2*i-1,2*j) = K(2*i-1,2*j) + k(1,4);
    K(2*i-1,2*m-1) = K(2*i-1,2*m-1) + k(1,5);
    K(2*i-1,2*m) = K(2*i-1,2*m) + k(1,6);
    K(2*i-1,2*n-1) = K(2*i-1,2*n-1) + k(1,7);
```

```
638     K(2*i-1,2*n) = K(2*i-1,2*n) + k(1,8);
639     K(2*i,2*i-1) = K(2*i,2*i-1) + k(2,1);
640     K(2*i,2*i) = K(2*i,2*i) + k(2,2);
641     K(2*i,2*j-1) = K(2*i,2*j-1) + k(2,3);
642     K(2*i,2*j) = K(2*i,2*j) + k(2,4);
643     K(2*i,2*m-1) = K(2*i,2*m-1) + k(2,5);
644     K(2*i,2*m) = K(2*i,2*m) + k(2,6);
645     K(2*i,2*n-1) = K(2*i,2*n-1) + k(2,7);
646     K(2*i,2*n) = K(2*i,2*n) + k(2,8);
647     K(2*j-1,2*i-1) = K(2*j-1,2*i-1) + k(3,1);
648     K(2*j-1,2*i) = K(2*j-1,2*i) + k(3,2);
649     K(2*j-1,2*j-1) = K(2*j-1,2*j-1) + k(3,3);
650     K(2*j-1,2*j) = K(2*j-1,2*j) + k(3,4);
651     K(2*j-1,2*m-1) = K(2*j-1,2*m-1) + k(3,5);
652     K(2*j-1,2*m) = K(2*j-1,2*m) + k(3,6);
653     K(2*j-1,2*n-1) = K(2*j-1,2*n-1) + k(3,7);
654     K(2*j-1,2*n) = K(2*j-1,2*n) + k(3,8);
655     K(2*j,2*i-1) = K(2*j,2*i-1) + k(4,1);
656     K(2*j,2*i) = K(2*j,2*i) + k(4,2);
657     K(2*j,2*j-1) = K(2*j,2*j-1) + k(4,3);
658     K(2*j,2*j) = K(2*j,2*j) + k(4,4);
659     K(2*j,2*m-1) = K(2*j,2*m-1) + k(4,5);
660     K(2*j,2*m) = K(2*j,2*m) + k(4,6);
661     K(2*j,2*n-1) = K(2*j,2*n-1) + k(4,7);
662     K(2*j,2*n) = K(2*j,2*n) + k(4,8);
663     K(2*m-1,2*i-1) = K(2*m-1,2*i-1) + k(5,1);
664     K(2*m-1,2*i) = K(2*m-1,2*i) + k(5,2);
665     K(2*m-1,2*j-1) = K(2*m-1,2*j-1) + k(5,3);
666     K(2*m-1,2*j) = K(2*m-1,2*j) + k(5,4);
667     K(2*m-1,2*m-1) = K(2*m-1,2*m-1) + k(5,5);
668     K(2*m-1,2*m) = K(2*m-1,2*m) + k(5,6);
669     K(2*m-1,2*n-1) = K(2*m-1,2*n-1) + k(5,7);
670     K(2*m-1,2*n) = K(2*m-1,2*n) + k(5,8);
671     K(2*m,2*i-1) = K(2*m,2*i-1) + k(6,1);
672     K(2*m,2*i) = K(2*m,2*i) + k(6,2);
673     K(2*m,2*j-1) = K(2*m,2*j-1) + k(6,3);
674     K(2*m,2*j) = K(2*m,2*j) + k(6,4);
675     K(2*m,2*m-1) = K(2*m,2*m-1) + k(6,5);
676     K(2*m,2*m) = K(2*m,2*m) + k(6,6);
677     K(2*m,2*n-1) = K(2*m,2*n-1) + k(6,7);
678     K(2*m,2*n) = K(2*m,2*n) + k(6,8);
679     K(2*n-1,2*i-1) = K(2*n-1,2*i-1) + k(7,1);
680     K(2*n-1,2*i) = K(2*n-1,2*i) + k(7,2);
681     K(2*n-1,2*j-1) = K(2*n-1,2*j-1) + k(7,3);
682     K(2*n-1,2*j) = K(2*n-1,2*j) + k(7,4);
```

```
683     K(2*n-1,2*m-1) = K(2*n-1,2*m-1) + k(7,5);
684     K(2*n-1,2*m) = K(2*n-1,2*m) + k(7,6);
685     K(2*n-1,2*n-1) = K(2*n-1,2*n-1) + k(7,7);
686     K(2*n-1,2*n) = K(2*n-1,2*n) + k(7,8);
687     K(2*n,2*i-1) = K(2*n,2*i-1) + k(8,1);
688     K(2*n,2*i) = K(2*n,2*i) + k(8,2);
689     K(2*n,2*j-1) = K(2*n,2*j-1) + k(8,3);
690     K(2*n,2*j) = K(2*n,2*j) + k(8,4);
691     K(2*n,2*m-1) = K(2*n,2*m-1) + k(8,5);
692     K(2*n,2*m) = K(2*n,2*m) + k(8,6);
693     K(2*n,2*n-1) = K(2*n,2*n-1) + k(8,7);
694     K(2*n,2*n) = K(2*n,2*n) + k(8,8);
695     y = K;
696 end
697
698 % This function returns the element principal stresses and their angle
       given the element stress vector.
699 function y = BilinearQuadElementPStresses(sigma)
700 R = (sigma(1) + sigma(2))/2;
701 Q = ((sigma(1) - sigma(2))/2)^2 + sigma(3)*sigma(3);
702 M = 2*sigma(3)/(sigma(1) - sigma(2));
703 s1 = R + sqrt(Q);
704 s2 = R - sqrt(Q);
705 theta = (atan(M)/2)*180/pi;
706 y = [s1 ; s2 ; theta];
707 end
```

Acknowledgements

I am grateful to my family for their unwavering support, which has been a continual source of strength and motivation throughout my academic path. Your confidence in me and my aspirations has been crucial to my success.

I would want to thank my dedicated supervisor, Dr. Neil Fellows, for his important guidance and continuous support. His expertise and advice were invaluable in steering this project in the proper direction, and I am grateful for his assistance.

Finally, I must acknowledge my role in this trip. This last year provided many challenges and obstacles, making completion of this project a genuinely tough venture. Despite these impediments, I persisted and eventually attained the goal I had set.

I am also very thankful for the opportunity to work on this project, which has expanded my knowledge and skills, and I look forward to what the future holds.